POSIX

Commands
The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to: manuals@ts.fujitsu.com

Certified documentation according to DIN EN ISO 9001:2000

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2000.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Copyright © Fujitsu Technology Solutions GmbH 2009. All rights reserved. Delivery subject to availability; right of technical modifications reserved. All hardware and software names used are trademarks of their respective manufacturers.
Contents

1 Preface ........................................................................... 11

1.1 Structure of the POSIX documentation .......................... 12
1.2 Target group ................................................................. 12
1.3 Summary of contents .................................................... 13
1.4 Changes since the last version of the manual ................. 14
1.5 Notational conventions ............................................... 15

2 Working with the POSIX shell .......................................... 21

2.1 Accessing the POSIX shell ............................................. 23
2.2 Special features of POSIX shell operation .................... 27
2.3 Entering commands from the POSIX shell ................. 31
2.3.1 Chaining commands ............................................... 32
2.3.2 Further input after command invocation .................. 33
2.3.3 Command options ................................................. 34
2.3.4 Specifying file names ............................................. 35
2.3.5 Built-in POSIX commands ....................................... 36
2.3.6 I/O redirection ...................................................... 36
2.3.7 Shell procedures and processes ............................... 39
2.3.8 Processing commands using the POSIX shell .......... 40
2.3.9 Compound commands ........................................... 41
2.3.10 Comments .......................................................... 45
2.3.11 Aliasing ............................................................... 45
2.3.12 Tilde substitution ................................................ 46
2.3.13 Command substitution ......................................... 47
2.3.14 POSIX shell variables and parameter substitution ... 48
2.3.15 Blank interpretation .............................................. 53
2.3.16 File name generation ............................................ 54
2.3.17 Quoting metacharacters ........................................ 55
## Contents

2.3.18 Arithmetic evaluation ................................................. 56  
2.3.19 Conditional expressions ........................................... 57  
2.3.20 Environment ........................................................... 60  
2.3.21 Functions ............................................................... 61  
2.3.22 Jobs ........................................................................ 62  
2.3.23 Signals ..................................................................... 63  
2.3.24 Execution ................................................................. 64  
2.3.25 Command re-entry ...................................................... 65  
2.4 Command summary .......................................................... 66  
2.4.1 Summary of all commands of the POSIX shell ..................... 66  
2.4.2 Command summary according to functions ......................... 72  
2.5 Example ....................................................................... 82  
3 International environment (NLS locale) .................................. 85  
3.1 Native language support .................................................... 85  
3.2 Defining an internationalized environment .............................. 87  
3.2.1 The personal internationalized environment ......................... 87  
3.2.2 Precedence among environment variables .......................... 89  
3.2.3 Supplied locales ........................................................... 90  
3.2.4 Restrictions ................................................................ 90  
4 Commands .................................................................... 91  
adduser add individual user number ......................................... 91  
alias define or display aliases ................................................. 92  
ar archive maintainer ........................................................... 94  
asa interpret carriage-control characters .................................... 99  
at execute commands at a later time ....................................... 100  
awk pattern scanning and processing language .......................... 107  
basename return non-directory portion of path name ................... 149  
batch execute commands at a later time .................................... 151  
bc arbitrary-precision arithmetic language .............................. 154  
bg run jobs in the background ............................................... 166  
bs2cmd execute BS2000 command *(BS2000)* .......................... 167  
bs2cp copy POSIX/BS2000 files *(BS2000)* ............................ 169  
bs2do calling BS2000 procedures from the POSIX shell *(BS2000)* 183  
bs2file set BS2000 file attributes *(BS2000)* ......................... 189  
bs2lp send files to a printer *(BS2000)* ................................... 190  
bs2pkey set pkeys *(BS2000)* ............................................... 192
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cal</td>
<td>print calendar</td>
<td>193</td>
</tr>
<tr>
<td>cancel</td>
<td>cancel line printer requests</td>
<td>195</td>
</tr>
<tr>
<td>cat</td>
<td>concatenate and print files</td>
<td>197</td>
</tr>
<tr>
<td>cd</td>
<td>change working directory</td>
<td>199</td>
</tr>
<tr>
<td>chgrp</td>
<td>change file group ownership</td>
<td>202</td>
</tr>
<tr>
<td>chmod</td>
<td>change file modes</td>
<td>204</td>
</tr>
<tr>
<td>chown</td>
<td>change file ownership</td>
<td>210</td>
</tr>
<tr>
<td>cksum</td>
<td>write file checksums and sizes</td>
<td>212</td>
</tr>
<tr>
<td>cmp</td>
<td>compare two files</td>
<td>215</td>
</tr>
<tr>
<td>comm</td>
<td>select or reject lines common to two files</td>
<td>217</td>
</tr>
<tr>
<td>command</td>
<td>execute a simple command</td>
<td>220</td>
</tr>
<tr>
<td>compress</td>
<td>compress files</td>
<td>224</td>
</tr>
<tr>
<td>cp</td>
<td>copy files</td>
<td>227</td>
</tr>
<tr>
<td>cpio</td>
<td>copy in and out</td>
<td>233</td>
</tr>
<tr>
<td>crontab</td>
<td>schedule periodic background work</td>
<td>238</td>
</tr>
<tr>
<td>csplit</td>
<td>split files based on context</td>
<td>245</td>
</tr>
<tr>
<td>cut</td>
<td>cut out selected fields of each line of a file</td>
<td>250</td>
</tr>
<tr>
<td>date</td>
<td>write the date and time</td>
<td>254</td>
</tr>
<tr>
<td>dd</td>
<td>convert and copy a file</td>
<td>260</td>
</tr>
<tr>
<td>debug</td>
<td>program debugging in forked tasks</td>
<td>265</td>
</tr>
<tr>
<td>df</td>
<td>report free disk space</td>
<td>267</td>
</tr>
<tr>
<td>diff</td>
<td>compare two files</td>
<td>274</td>
</tr>
<tr>
<td>dirname</td>
<td>return directory portion of pathname</td>
<td>279</td>
</tr>
<tr>
<td>du</td>
<td>estimate file space usage</td>
<td>280</td>
</tr>
<tr>
<td>dumpfs</td>
<td>dump file system</td>
<td>282</td>
</tr>
<tr>
<td>echo</td>
<td>write arguments to standard output</td>
<td>284</td>
</tr>
<tr>
<td>ed</td>
<td>interactive line editor</td>
<td>289</td>
</tr>
<tr>
<td>edt</td>
<td>screen-oriented editor EDT (BS2000)</td>
<td>308</td>
</tr>
<tr>
<td>egrep</td>
<td>search a file with an ERE pattern</td>
<td>310</td>
</tr>
<tr>
<td>env</td>
<td>set environment for command execution</td>
<td>316</td>
</tr>
<tr>
<td>eval</td>
<td>construct command by concatenating arguments</td>
<td>320</td>
</tr>
<tr>
<td>ex</td>
<td>command and display editor</td>
<td>323</td>
</tr>
<tr>
<td>exec</td>
<td>execute commands and open, close or copy file descriptors</td>
<td>353</td>
</tr>
<tr>
<td>exit</td>
<td>cause the shell to exit</td>
<td>358</td>
</tr>
<tr>
<td>expand</td>
<td>convert tabs to spaces</td>
<td>361</td>
</tr>
<tr>
<td>export</td>
<td>set export attribute for variables</td>
<td>363</td>
</tr>
<tr>
<td>expr</td>
<td>evaluate arguments as an expression</td>
<td>366</td>
</tr>
<tr>
<td>false</td>
<td>return false value</td>
<td>371</td>
</tr>
<tr>
<td>fc</td>
<td>process command history list</td>
<td>372</td>
</tr>
<tr>
<td>fg</td>
<td>run jobs in the foreground</td>
<td>376</td>
</tr>
<tr>
<td>fgrep</td>
<td>search a file for a fixed-string pattern</td>
<td>377</td>
</tr>
<tr>
<td>file</td>
<td>determine file type</td>
<td>382</td>
</tr>
<tr>
<td>find</td>
<td>find files</td>
<td>385</td>
</tr>
<tr>
<td>fold</td>
<td>filter for folding lines</td>
<td>392</td>
</tr>
</tbody>
</table>
Contents

fsck    file system check .......................................................... 395
fsexpand expand existing file systems ........................................ 397
fyp     define file processing mode (BS2000) ................................. 399
gencat  generate a formatted message catalog ............................... 401
getconf get configuration values ................................................ 404
getopts parse utility options .................................................... 408
grep    search a file for a pattern ............................................ 411
hash    remember or report utility locations ............................... 416
hd      hex dump .................................................................... 420
head    copy the first part of files ............................................. 423
iconv   code set conversion ....................................................... 425
id      return user identity ..................................................... 427
info    online diagnostic tool ................................................... 430
ipcrm   remove inter-process communication facilities ..................... 432
ips     inter-process communication status .................................. 434
jobs    display status of jobs in the current session ...................... 439
join    relational database operator .......................................... 440
kill    terminate or signal processes ......................................... 444
let     integer arithmetic ....................................................... 448
lex     generate programs for lexical tasks ................................ 450
ln      link files ................................................................. 453
locale  get locale-specific information .................................... 462
localedef define local environment .......................................... 466
logger  log messages ................................................................ 470
logname return user's login name .............................................. 471
lp      send files to a printer .................................................... 472
lpstat  report line printer status information .............................. 477
ls      list directory contents .................................................. 480
mailx   interactive message processing system (mail extended) ....... 488
make    maintain, update and regenerate groups of programs .......... 526
man     display system documentation ....................................... 535
msg     permit or deny messages ............................................... 537
mkdir   make directories ......................................................... 539
mkfifo  make FIFO special files ............................................... 541
mkfs    make file system .......................................................... 543
mknod   make an inode ............................................................. 544
more    display files on a page-by-page basis .............................. 546
mount   mount a file system ..................................................... 551
mountall mount file systems ..................................................... 562
mv      move files .................................................................... 564
newgrp  change to a new group .................................................. 568
nice    invoke a utility with an altered system scheduling priority .... 571
nl      line numbering filter ..................................................... 572
nm      write the name list of an object file ................................. 579
Contents

nohup  invoke a utility immune to hangups .............................................. 582
od   dump files in various formats ......................................................... 584
paste  merge corresponding or subsequent lines of files ......................... 588
patch  apply changes to files .............................................................. 593
pathchk  check pathnames ................................................................. 597
pax   portable archive interchange ....................................................... 600
pdbl  set up and manage user-specific program cache ............................. 607
pkginfo  show information on software packages in POSIX ....................... 611
posdbl  set up and manage global program cache .................................... 614
pr    prepare files for printing .......................................................... 618
print  write arguments to standard output ............................................ 625
printf  formatted output ..................................................................... 626
ps    report process status .................................................................... 630
pwd   return working directory name ..................................................... 638
rcp   remote file copy ........................................................................... 639
read   read a line from standard input ................................................... 643
readonly  set read-only attributes for variables .................................. 647
renice  set system scheduling priorities of running processes ............... 649
rm    remove directory entries ............................................................... 650
rmdir  remove directories ....................................................................... 652
rmpart  remove partition ....................................................................... 654
sed   stream editor ................................................................................. 655
set    set or unset options and positional parameters ............................ 659
sh    shell, the standard command language interpreter ............................ 679
shift  shift positional parameters .......................................................... 682

show_pubset_export  show file system affected by pubset export ............... 684
sleep  suspend execution for an interval ................................................. 686
sort   sort, merge or sequence check text files ....................................... 688
split  split a file into pieces ................................................................... 695
start_bs2fsd  start copy daemons ............................................................. 698
strings  find printable strings in files ..................................................... 699
stty  check and change terminal options (set terminal type) ................... 701
sync  flush system buffers ....................................................................... 710
tabs   set terminal tabs ............................................................................ 713
tail   deliver the last part of a file ............................................................ 717
talk  talk to another user ......................................................................... 721
tar   file archiver ..................................................................................... 724
tee  join pipes and make copies of input .................................................. 730
test  evaluate expression ......................................................................... 732
time  time a simple command ................................................................. 739
Contents

times write process times .................................................. 741
touch change file access and modification times ........................... 742
tput change terminal characteristics ..................................... 746
tr translate characters ...................................................... 751
trap trap signals .............................................................. 757
true return true value ....................................................... 767
tsort topological sort .......................................................... 768
rty output path name of current terminal ................................. 771
type write a description of command type ................................. 772
typeset set attributes for variables ....................................... 773
ulimit set or report file size limit ........................................ 776
umask get or set the file mode creation mask ............................ 779
umount unmount a file system ............................................... 781
umountall unmount file systems ............................................ 782
unalias remove alias definitions ............................................ 784
uname return system name ................................................... 785
uncompress expand compressed files ....................................... 787
unexpand convert spaces to tabs ............................................ 789
uniq report or filter out repeated lines in a file ......................... 791
unset unset values and attributes of variables and functions .......... 794
usp set POSIX control parameters dynamically .......................... 796
uudecode decode a binary file .............................................. 798
uuencode encode a binary file .............................................. 800
uuname list names of known systems ..................................... 802
vi screen oriented (visual) display editor ................................. 803
wait await process completion .............................................. 804
wc word, line and byte or character count ............................... 806
whence query command type .................................................. 808
who display who is on the system ......................................... 809
write write to another user .................................................. 810
xargs construct argument list(s) and execute command .................. 811
yacc yet another compiler-compiler ....................................... 812
zcat expand and concatenate compressed data ............................. 813
: return true value ............................................................ 814
. execute commands in current environment ............................... 815
[...] evaluate expression ..................................................... 816
## Contents

5 Tables and directories ........................................... 873

5.1 Summary of command XPG4 conformity ......................... 873
5.2 Regular POSIX shell expressions ............................... 877
5.3 Metacharacters for the POSIX shell ......................... 884
5.4 ASCII character set (ISO 646) ............................ 889
5.5 EDF04 character set ........................................ 893

Related publications ............................................. 895

Index ............................................................ 899
1 Preface

POSIX (portable open system interface for UNIX) is a range of UNIX-based standards which ensure the compatibility and interoperability of applications in a heterogeneous network. A heterogeneous network consists of computers from different manufacturers, as well as system and application software from different software suppliers.

The POSIX standard was defined as the national American standard by the Institute of Electrical and Electronics Engineers (IEEE) in 1989. It was then extended by the X/OPEN consortium, and in 1990 became adopted as the international standard. (X/OPEN Portability Guide IV).

The X/OPEN Portability Guide IV, also known as XPG4 standard, comprises 7 volumes, including interface definitions for basic operating systems, programming languages, data management and networking. The BS2000/OSD operating system as of V2.0 supports the XPG4 standards which are contained in the first two volumes:

- Volume 1: System Interfaces and Headers (approx. 350 program interfaces)
- Volume 2: Commands and Utilities (approx. 200 user interfaces)

In order to support these interfaces, the POSIX functionality was integrated into BS2000/OSD. POSIX designates both the IEEE standard and the BS2000/OSD “POSIX” functionality. POSIX satisfies the requirements to allow its certification according to the XPG4 standard, which is carried out in two stages: at the end of 1995, BS2000/OSD received the “XPG4 base branding” (XPG4) from “The Open Group” (previously X/OPEN), and around mid 1997 it received branding according to the “XPG4 UNIX profile” (also known as XPG4.2 or UNIX95). In addition, BS2000/OSD with its POSIX subsystem has been certified as an internet server by “The Open Group” in 1999.

The kernel of the POSIX software product is implemented as a BS2000 subsystem. The library functions of the XPG4 standard are available to the user via a C library, and a defined set of commands is available via a shell (POSIX shell). The C library is a component of the product CRTE (Common RunTime Environment).

Application programs can be easily ported with POSIX, irrespective of the operating system being used. Programs consistent with XPG4 can therefore also run in BS2000/OSD following recompilation.

POSIX program interfaces are offered together with BS2000 program interfaces. It is possible to use a combination of both BS2000 and POSIX program interfaces in the same program, albeit with certain restrictions.
1.1 Structure of the POSIX documentation

The following documentation is available to help you to familiarize yourself with and work effectively in the POSIX subsystem in BS2000/OSD:

- The POSIX manual “Basics for Users and System Administrators” [1] provides an introduction into working with the POSIX subsystem. It also describes the administration tasks that will arise when dealing with POSIX, and gives information on which BS2000 software products you can use in conjunction with this subsystem.

- This manual, POSIX Commands, contains a description of the POSIX commands with which you can work in the POSIX shell.

POSIX documentation in the BS2000/OSD environment

The functionality of various software products in the BS2000/OSD environment is being expanded to enable you to combine these products with the use of the POSIX functions.

A series of utilities provide access to the POSIX file system. As a result, you can, for example, use EDT to process files from the POSIX file system.

Now that the CRTE (Common RunTime Environment) has been extended in accordance with the XPG4 standard, you can use the C Library Functions to write portable C programs irrespective of which operating system is running.

The manual “Basics for Users and System Administrators” [1] is the basic requirement for accessing POSIX functions from within other software products.

1.2 Target group

The POSIX commands represent a comprehensive set of versatile programs and procedures for process control.

This manual contains a description of POSIX commands. It is a fundamental reference work for all POSIX users who already possess some basic knowledge of POSIX operation. The level of knowledge required corresponds to the information presented in the POSIX manual “Basics for Users and System Administrators” [1]. For further literature, please refer to the section References.
1.3 Summary of contents

The chapter *Working with the POSIX Shell* provides basic information about POSIX shell operation. It also provides a function-specific command summary and an example of accessing and working in the POSIX shell.

The chapter *International Environment (NLS Locale)* tells you something about the Native Language System.

The chapter *Commands* presents the commands in alphabetical order. In order to help users locate the relevant information, the outer running title of each page specifies the name of the described command together with a supplementary keyword in the case of very long descriptions.

The chapter *Tables and directories* provides a summary of the XPG4 conformity of the POSIX commands. It also includes lists of the metacharacters and regular expressions of the POSIX shell and tables of the ASCII and EBCDIC character sets.

The index at the back of the book is intended to help you locate topics more quickly.

For technical reasons the printed manual is divided into two volumes.

**README file**

Information on functional changes and additions to the current product version described in this manual can be found in the product-specific README file. You will find the README file on your BS2000 computer under the file name SYSRME_product.version.language. The user ID under which the README file is cataloged can be obtained from your systems support. You can view the README file using the /SHOW-FILE command or an editor, and print it out on a standard printer using the following command:

```
/PRINT-DOCUMENT filename, LINE-SPACING=*BY–EBCDIC–CONTROL
```
1.4 Changes since the last version of the manual

This edition of the present manual features the following changes compared to the previous version (March 2007):

The following new commands were added to the Commands chapter:

- `show_pubset_export` - show file systems affected by EXPORT-PUBSET
- `start_bs2fsd` - start copy daemons

The following changes were made to the descriptions of the commands below:

- `df` - Support of bs2fs file systems
- `mount` - Support of bs2fs file systems
- `mountall` - Support of bs2fs file systems
- `pathchk` - Support of bs2fs file systems
- `umount` - Support of bs2fs file systems
- `umountall` - Support of bs2fs file systems
1.5 Notational conventions

As far as is possible, each command description is in a uniform pattern:

- Outer running title
- Inner running title (optional)
- Main title
- Description
- Synopsis
- Syntax description
- Exit status (optional)
- Error messages (optional)
- Files (optional)
- Environment variables (optional)
- Locale (optional)
- Examples (optional)
- See also (optional)

These elements are now described in detail.

Outer running title

The outer running title gives the command name. In the case of the larger commands it also acts as an orientation guide, the command name being followed by a keyword indicating the topic dealt with on the page.

Inner running title (optional)

The inner running title classifies certain commands for the sake of clarity, e.g. *Built-in sh command*.

Main title

The main title includes:
- the name of the command
- a short description of the command
- the suffix (*BS2000*) in the case of commands which are specially provided for operation with BS2000 as an extension to the XPG4 standard.
Description

Here the following are described:

- the functionality of the command
- the various purposes of individual formats if more than one is available
- the environment in which the command is to be used (e.g. entries in files, permissions, etc.)
- background information
- points to be observed before and after the command call.

In the case of commands which call complex programs (e.g. *awk, sh*), this section simply describes the program call. For further information, e.g. about *awk* operation, please refer to the associated syntax section.

Syntax

Syntax

```
  cmd[-a][-b][-c][-darg1][-f_arg2].file....
```

The meaning of the metasyntax is described below.

In the syntax:

**Boldface** characters

- Constants: These characters must be entered exactly as shown.

**Normal characters**

- Variables: These characters stand for other characters for which you may select and enter a suitable value.

[ ] Optional: arguments within square brackets are optional and may be omitted. The effects of these arguments are explained in the description of the options and arguments. You must not enter the square brackets themselves, unless specifically instructed to do so.

I A vertical bar identifies alternatives from which you may choose one only.

_ Indicates a mandatory blank (space).

... Ellipses indicate that the preceding argument can be repeated. If blanks that are not part of the argument have to be placed between repetitions, the ellipses are preceded by _ (space).
Example

`cmd[...-a][...-b][...-c][...-darg1][...-farg2].file......`

The following arguments are mandatory:

- `cmd`
- one or more `files`, each preceded by a blank.

The following arguments are optional extras:

- one or more of options `-a`, `-b`, `-c`; these options can either be listed separately:
  - `-a` `-b` `-c`
  - or grouped together:
  - `abc`
- option `-d`, where `arg1` must be replaced with an argument
- option `-f`, where `arg2` must be replaced with an argument.

In descriptive text:

No distinction is made between constants and variables in the descriptive text; all syntax elements as well as miscellaneous file names, path names, and commands are given in italics.

Input

In application examples, input into the system is identified by fixed-width, semibold type. Since all input lines are terminated with the `Ú` key at character-mode terminals, and with the keys `[EM][DUE]` at block-mode terminals, the keys at the ends of input lines are not specified.

Many inputs are terminal-dependent, they are different for block and character-mode terminals (see also “Terminal support” on page 27):

Output

Operating system output is shown in fixed-width font, except in descriptive text, where it is shown in italics.
Syntax description

option
(For a description of options see the section “Command options” on page 34)

argument
Description of the remaining arguments which you can pass when calling a
command, e.g. input files, output files, parameters, variables, field separators etc.

Exit status (optional)

The exit status is the value which a command returns to the calling process following
execution. The exit status provides information about command execution. The exit status
is a numerical value which is contained in the ? variable. You can enter the command
echo $? to query the exit status.

The exit status section is only included if the exit status deviates from following basic rule:

0     successful execution of the command
>0    the command has failed

Error messages (optional)

Important error messages are listed and explained here. Additional notes indicate how
errors can be avoided and recovered.

Unless otherwise specified, error messages are sent to the standard error output. Standard
error (stderr) is usually the screen.

Files (optional)

This section lists files that the command accesses or creates.

Environment variables (optional)

Some commands inspect the values of environment variables. These are listed here.

Locale (optional)

This section describes how NLS affects the command (see the chapter “International
environment (NLS locale)” on page 85).
Examples (optional)
Examples aim to illustrate:
- the main function of the command
- the use of the principal options
- acceptable complex combinations of options and arguments

See also (optional)
This section contains references to other commands that perform similar functions or work in conjunction with the described command. References to literature containing more information on the command is provided where appropriate.
System calls and library functions for C developers are indicated by a pair of parentheses, e.g. `chdir()`.

Notes and warnings

BS2000 Sections which refer to special characteristics of POSIX operation with the BS2000 are identified in this way.

⚠️ This symbol indicates important information which you must be careful to observe.

⚠️ This symbol precedes warnings which you should observe in the interests of system and operating security.

References
The following examples show how references to other sections of the text, manuals, commands, functions, system calls and file formats are presented.

see the section “Command summary” on page 66 Reference to the section “Function-specific command summary” in this manual
see “Basics for Users and System Administrators” [1] Reference to a manual which is listed as number [1] under Related publications at the back of this manual
see also `awk` Reference to the `awk` command in this manual
see also `chdir()` [4] Reference to the C function, system call or file format `chdir()` in the manual which is listed as number [4] under Related publications at the back of this manual
2 Working with the POSIX shell

The POSIX shell is the interface which connects you to the POSIX subsystem via the C runtime system/libraries.

The figure 1 illustrates the structure of POSIX in BS2000 and the integration of the POSIX shell.

The POSIX shell is a command interface which you can use in addition to the BS2000 command interface (see figure 2).
When you have successfully accessed the POSIX shell, you may use all the commands of the POSIX shell. After leaving the POSIX shell you may again enter BS2000 commands.

The POSIX shell reads commands from a terminal or from a file in the POSIX file system, interprets them in accordance with certain rules and is responsible for their execution. A file which contains commands for the POSIX shell is called a shell procedure (shell script).

The operation and performance of the POSIX shell depend on whether the terminal at which the user is working is a block or character-mode terminal.

The POSIX shell provides you with a comprehensive command language which you can use like a programming language. You can use the available commands to create your own programs which you can then run without the need for prior compilation.
2.1 Accessing the POSIX shell

You can access the POSIX shell in the following ways:
- via a BS2000 terminal (block-mode terminal)
- from a UNIX system (character-mode terminal)
- by means of an emulation

Figure 3: Ways of accessing the POSIX shell
Accessing the POSIX shell

Working with the POSIX shell

Access via a BS2000 terminal

After successfully logging on to BS2000, any BS2000 user can enter the /START-POSIX-SHELL command to call the POSIX shell:

START-POSIX-SHELL

VERSION = *STD / <product-version without-man-corr>
,MONJV = *NONE / <filename 1..54 without-gen-vers>
,CPU-LIMIT = *JOB-REST / <integer 1..32767>

VERSION = *STD / <product-version without-man-corr>
Version number of the program to be called (in this case the POSIX shell).
The default setting is *STD, which means that the version currently available is called.

MONJV = *NONE / <filename 1..54 without-gen-vers>
Name of the job variable which is to monitor the program. If this job variable does not yet exist, a new one is created. During the program run, the system sets the job variable to the appropriate values:

$R Program running
$T Program terminated
$A Program terminated abnormally

This operand is only available if the “Job Variables” software product is implemented (see also the “Job Variables” manual).

CPU-LIMIT = *JOB-REST / <integer 1..32767>
Maximum CPU time (in seconds) that the program can take to run. If no time limit was set when the job was started (CPU-LIMIT=*NO), then the program ignores this specification and does not observe a time limit when running.
If the program overruns the specified time, the program is interrupted during the dialog session and a corresponding message is output. Users can then either request a dump or abort or resume the program. If a STXIT routine has been defined in the program for the event that the CPU limit is reached, this routine runs and the program is terminated.
In batch mode, the program is terminated if the time limit is exceeded.

CPU-LIMIT = *JOB-REST
If the job was started with a time limit, the value defined during system generation is used for the program. If not, the program runs with no time limit.
Return codes

<table>
<thead>
<tr>
<th>(SC2)</th>
<th>SC1</th>
<th>Main code</th>
<th>Meaning/Guaranteed messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>0</td>
<td>CMD0001</td>
<td>No errors</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CCM0999</td>
<td>The POSIX shell could not be started.</td>
</tr>
</tbody>
</table>

Effect of the START-POSIX-SHELL command

This command sets up the POSIX environment and calls the program entered in the SYSSRPM file for the relevant user.

If you have entered the POSIX shell as a standard program in your user data, then after entering /START-POSIX-SHELL you can work interactively with the POSIX shell and you may use all the commands and functions of the POSIX shell. POSIX shell-specific commands are available for interaction between BS2000 and the POSIX subsystem.

Terminating the POSIX shell

In order to return to BS2000 you must first enter the \texttt{exit} command to close the POSIX shell.

Access from a character terminal

You can use the \texttt{rlogin} command to log on to a BS2000 host from a terminal of a UNIX system, provided you have the required authorization to access it. In other words, you will need to have a user ID (login name) that is authorized for POSIX \texttt{rlogin} access, the associated password, and an account number that can also be used for \texttt{rlogin} access accounting on the BS2000 host. After logging on, you can use POSIX as if in local mode.

In order to connect to BS2000, you must enter the following command in the POSIX shell:

\texttt{rlogin [-l <login-name>] <host>}

If you do not enter a login name then the login name under which you are logged on at the remote computer is used. If you use the \texttt{rlogin} command, the system asks for the password for the required login name. The password is verified by the BS2000 SRPM component (System \textbf{R}esources and \textbf{P}rivileges \textbf{M}anagement): The BS2000 login name and password specifications are checked against the access control attributes of the home pubset. If they match then the user is granted access to the POSIX subsystem. If the SECOS product is employed then the access control procedure can be further refined by means of LOGON protection.

If you use \texttt{rlogin} to access POSIX then you cannot use BS2000 commands.
Access via telnet

The `telnet` daemon `telnetd` provides direct access to BS2000 via the `telnet` protocol from the UNIX system and also directly from the PC via the `telnet` application, which appears to POSIX as a character terminal and behaves as such. Access control is handled in the same way as for `rlogin`, i.e. via BS2000 access mechanisms. Access without the entry of a password, as realized between UNIX systems (i.e. via an entry in the `.rhosts` file), is not supported.

Access via an emulation

The third way of accessing POSIX is by means of a terminal emulation. This means that you must first log on to a workstation or PC and then start a terminal emulation to emulate either a terminal of a UNIX system or a BS2000 terminal.

BS2000 terminal emulation

When you access POSIX via a BS2000 terminal emulation such as the EM9750 or MT9750, for example, your terminal acts as a block-type terminal, which means that you can enter BS2000 commands and `/START-POSIX-SHELL` as if at a BS2000 terminal (see page 24).

UNIX/SINIX terminal emulation for UNIX systems

UNIX/SINIX terminal emulations are available for workstations with UNIX system with graphical OSF/Motif-based interfaces and also for PCs running Windows (e.g. EM97801 and SINIX-TE). These emulations act as a character terminal of a UNIX system and enable you to enter commands such as `rlogin` (see page 25).
2.2 Special features of POSIX shell operation

Presettings in the user environment

When you have successfully accessed the POSIX subsystem, the POSIX shell is started. Before the POSIX shell reports that it is ready for operation, the following presettings are made in the user environment:

- The POSIX shell initializes the standard shell variables. It allocates default values to the following variables: HOME, LANG, LOGNAME, MAIL, PATH, SHELL, TTY, TERM, TZ and USER.
  If a variable has already been predefined by the SDF-P variable SYSPOSIX then this value is adopted. However, the shell variables USER, TERM, TYP, LOGNAME, HZ, HOME and MAIL may not be set by the user.
- The file /etc/profile is executed.
- If you have created it, the file $HOME/.profile is executed.

Terminal support

POSIX supports the block-mode terminals which are used in BS2000 as well as the character-mode terminals employed in UNIX systems. The latter are connected to UNIX multiuser systems and are operated by POSIX via networks. If you access POSIX via a workstation then a character-mode terminal is emulated. In the case of SINIX workstations, this can, for example, be a type 97801 terminal.

Block-mode and character-mode terminals operate differently:

- In the case of character-mode terminals each character that is input is immediately transferred to the UNIX system where it is transferred to the screen as a response to the input and displayed. Control functions such as cursor movement, uppercase/lowercase notation or the buffering of transfers are performed by the computer to which the terminal is connected. Screen-oriented applications are only permitted at character-mode terminals.
- In contrast, block-mode terminals transfer all the text entered at the screen to the computer in the form of a data block. Control functions are performed at the unit itself.

Block-mode terminals are directly connected to the BS2000 and POSIX. Both BS2000 and POSIX commands can be entered at this terminal type.

For BS2000 security reasons it is only allowed to write via block-mode terminals to terminals of the own process group. E.g. a `cp <file> /dev/term/xxx` processed for a block-mode terminal of another LOGON-Task will be rejected with permission denied.
Many inputs are terminal-dependent, i.e. they are different for block and character-mode terminals:

<table>
<thead>
<tr>
<th>Block-mode terminal</th>
<th>Character-mode terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@d</td>
<td>END</td>
</tr>
<tr>
<td>@@c</td>
<td>DEL</td>
</tr>
<tr>
<td>@@/</td>
<td>CTRL [j]</td>
</tr>
<tr>
<td>[EM] [JUE]</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>CTRL [S] / CTRL [Q] / CTRL [Z] ...</td>
</tr>
</tbody>
</table>

Special functions (P keys, Ctrl keys)

You can call the command `bs2pkey` to make the following settings for the \[P3\] and \[P4\] keys:

\[P3\] with @@c (CTRL [C])

\[P4\] with @@d (CTRL [D])

You may either call the program in the POSIX shell (without options) or enter it in the file `/etc/profile`. In the latter case the program is called each time the shell is activated.

File transfer

POSIX files contain no records. Instead, they are byte-oriented. In contrast, BS2000 files contain record-oriented and/or PAM block-oriented data.

By default, POSIX handles files in EBCDIC format, whereas UNIX systems, MS-DOS and Windows handle files in ASCII format. ASCII files stored in the POSIX file system can only be processed in the POSIX shell if they have first been converted.

Routines for the conversion of data between POSIX and BS2000 formats are available in order to permit the exchange of POSIX and BS2000 files. Files are converted from the EBCDIC DF 03 character set to ASCII ISO 7-bit code, and vice versa (see also chapter “Tables and directories” on page 873). Only pure text files can be converted.
Automatic conversion

The environment variable \texttt{IO\_CONVERSION} determines whether files accessed with POSIX commands (e.g. \texttt{awk}, \texttt{cat}, \texttt{grep}...) on mounted ASCII file systems are converted automatically. \texttt{IO\_CONVERSION} is set to the value "NO" by default, which means that no automatic conversion occurs. If desired, automatic conversion can be enabled with the following command:

\begin{verbatim}
export IO\_CONVERSION=\text{YES}
\end{verbatim}

If you want to set automatic conversion for a POSIX user as the default value directly on starting the POSIX shell, this \texttt{export} command must be entered in the \texttt{.profile} file in the HOME directory of that user.

Note that automatic conversion must not be enabled when using the following tools, since the tools themselves also perform the same conversion:
\begin{itemize}
  \item \texttt{dd}, \texttt{iconv}, \texttt{edt} with the \texttt{-k} option,
  \item \texttt{bs2cp} with the \texttt{-k} option.
\end{itemize}

Handling of archives/libraries:
\begin{itemize}
  \item \texttt{ar} does not convert automatically, since \texttt{ar} libraries often contain binary data.
  \item \texttt{pax} and \texttt{tar} convert automatically. However, a \texttt{pax} or \texttt{tar} archive must not be copied with \texttt{cp} if automatic conversion is enabled.
\end{itemize}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{BS2000} & \textbf{POSIX} \\
\hline
File structure: & Record-oriented & Byte-oriented \\
\hline
Coding: & EBCDIC & EBCDIC/ASCII \\
\hline
\begin{itemize}
  \item a) \texttt{bs2cp, edt}
  \item b) Files imported from other UNIX systems
  \item c) \texttt{iconv}
\end{itemize}
\end{tabular}
\caption{File transfer}
\end{table}
Special features of POSIX shell operation

Working with the POSIX shell

a) Transferring files from POSIX to BS2000 (as seen by the POSIX shell):

You use the POSIX command `bs2cp` or `edt` to transfer files from POSIX to BS2000. You do not need to specify the `-k` option if the files use the EBCDIC character set in the two file systems.

In addition, you may define file attributes for the BS2000 file. To do this you must use the POSIX command `bs2file` to define the BS2000 file attributes before issuing the "copy" command `bs2cp`. `bs2file` is mapped to the BS2000 command `SET-FILE-ATTRIBUTES`.

b) Transferring files from BS2000 to POSIX (as seen by the POSIX shell):

You use the POSIX command `bs2cp` or `edt` to transfer files from BS2000 to POSIX. You do not need to specify the `-k` option if the files use the EBCDIC character set in the two file systems.

Depending on the BS2000 file type (SAM, ISAM or PAM) the following applies:
- PAM files are always stored as binary files.
- ISAM files are generally stored as text files in the POSIX file system.
- In the case of SAM files and LMS elements (type S, J, X, D), you can choose whether the file/element should be placed in the POSIX file system as a text file, binary file or binary text file. To do this, you must use the POSIX command `ftype` to specify the file processing mode before issuing the "copy" command `bs2cp`.

c) Use the POSIX command `iconv` to perform file conversion within the POSIX file system. The file contents are converted.
2.3 Entering commands from the POSIX shell

When you have successfully accessed the POSIX subsystem, the POSIX shell is started. If you are using the POSIX shell in interactive mode, the POSIX shell outputs the value of the environment variable PS1 as a prompt before reading a command. By default, this is the dollar sign ($), or (#) for the privileged user, followed by a blank ( ).

Commands are entered in the following format:

```
command[...options][...parameters]...[\]
```

For command you must specify the name of the POSIX command or shell procedure which is to be executed. The options allow you to specify control instructions for command execution. Under parameters you can enter a call argument which the POSIX shell passes to the command. You may also enter multiple call arguments for certain commands.

At character-mode terminals you must use tabs or spaces to separate the command names and call arguments. After the final call argument you can conclude command input by pressing the [\] key (at character-mode terminals) or [EM] [DUE] (at block-mode terminals).

Pure BS2000 programs cannot be started from within the POSIX shell.

If the screen line is too short for the required input you may proceed in one of two ways:

- At the end of the line you may simply continue to type without pressing the [\] key. When you have entered the complete command, press the [\] key to terminate input.
- Enter [\] [\] to continue the line. The backslash (\) prevents the [\] key from terminating command input. You may then continue to enter the command. When you press [\] (without [\]), the command is executed.

Every POSIX command returns a value to the POSIX shell in which it was called. This is its exit status. If the command was executed correctly this value is 0; if errors were encountered its value is not equal to 0.

If a command outputs information on screen and the output exceeds the capacity of a screen page, then at character-mode terminals you can pause the output by pressing the key combination [CTRL] [s] and continue it by pressing [CTRL] [q]. This function is not supported at block-mode terminals.
2.3.1 Chaining commands

You can chain a set of commands together:

- You can use the `|` character to combine two commands to form a pipeline.
- You can use the characters `;` or `&` or `&&` or `||` to combine multiple commands or pipelines to form a command list.

Pipeline

A pipeline is a sequence of two or more commands each of which is linked to the preceding command by means of the `|` character. A pipeline between two commands redirects the standard output of the first command to the standard input of the second command (see the section “I/O redirection” on page 36). For this reason, commands which are chained in this way must fulfill the following conditions:

- The first command must write to the standard output.
- The next command in the pipeline must read from the standard input.
- If the pipeline contains more than two commands, then all the commands which come between the first and last command of the pipeline must read from the standard input and write to the standard output.

The POSIX shell starts all of the piped commands as independent parallel processes. However, in a monoprocessor system, the processor can only handle one of these processes during each time slice. For this reason, processor time is devoted to the individual processes in turn.

Only the process corresponding to the first command in the pipeline does not receive its input from another process.

Only the process corresponding to the last command in the pipeline does not route its output to another process.

The POSIX shell waits until the last of the piped commands terminates before processing any further inputs.

Command list

A command list is a sequence of one or more pipelines separated by the symbols `;` `&` `&&` `||` and optionally terminated by `;` `&` `&&` `||`. 
The symbols, ;, &, and |& have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e. the shell does not wait for that pipeline to finish). The symbol I& after a command list causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell.

In the following, the term command refers to a command or a pipeline. The effect of the various characters is specified in the table below:

<table>
<thead>
<tr>
<th>Character</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>;</td>
<td>The POSIX shell does not process the next command until the preceding command has terminated. It does not, however, issue a prompt.</td>
</tr>
<tr>
<td>&amp;</td>
<td>The POSIX shell starts the preceding command in the background and then immediately processes the following command. The POSIX shell redirects the standard input of all background commands to a file which has the properties of /dev/null. However, you may redirect the standard input of a background command to another file.</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>The POSIX shell does not execute the following command unless the preceding command returns the value 0 as its exit status.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
</tr>
</tbody>
</table>

2.3.2 Further input after command invocation

Some commands expect you to enter additional data from the keyboard after you have pressed ↵. In such cases the cursor is positioned at the start of the next line and either no prompt appears at all or a special prompt, varying according to the command, appears instead. Each of the lines you now enter must be terminated with ↵. To terminate data entry, you generally either press END (or use @ @d at block-mode terminals) or enter one of the special statements associated with your command.

If you terminate a command with ↵ leaving the command line incomplete, or if you escape ↵ with a backslash (\), the POSIX shell outputs the value of the shell variable PS2 (by default the "greater than" character > and a blank ...) as prompt.
2.3.3 Command options

You can invoke commands with one or more options or with none at all. Options modify the way in which a command functions. They generally consist of a single letter. In the command syntax they are either shown exactly as you have to enter them (in bold type) or are collectively identified as "option" (in normal type).

The following rules generally apply when you enter more than one option:

- The order in which you list the options is immaterial.
- Options without arguments can be entered in two ways:
  - singly, e.g.: cmd-a-b-c
  - grouped, e.g.: cmd-abc
- Options that require an argument, for example -d argument or -f argument, cannot be combined with those that do not. Such options must be entered individually, with blanks (,) to separate them:
  cmd-abc,-d-argument,-f-argument
- Options and option combinations are introduced by a minus sign (-): -abc.
- Two hyphens (--) may be used to indicate the end of the options. You need to use -- if the first argument begins with -.

Most commands output a usage message if they are called with an incorrect option. The usage message tells you which options and operands can be used in conjunction with a command call.

Example Output of a usage message when an incorrect option is specified

```
$ ls -y
ls: Illegal option -- y
ls: Usage: ls -lRadxmnlogrtucpFbqisfLe [files]
```
2.3.4 Specifying file names

The following conventions apply to file names:

● You may use any character with the exception of / and \0 (zero bytes terminating a character string).

● You should not enter the characters plus (+), minus (-) or period (.) at the beginning of a file name. The period is reserved for special files, e.g. for the file .profile.

Example of the use of a file name beginning with the minus sign (-):

\texttt{touch -- file1}  \hspace{1cm} (before the file name is entered, the (void) options must be terminated)

● For more information on the use of the characters *, ?, [...], see the section “File name generation“ on page 54.

● You should not use a number of special characters (see the chapter “Tables and directories“ on page 873).

● If the filename contains spaces or tabs you must place it inside quotes.

● Unlike BS2000, POSIX distinguishes between uppercase and lowercase in file names.

There are two ways of specifying the names of files and directories in the POSIX shell:

● You can specify a relative path name. Relative path names always start from the current directory.

● You can specify an absolute path name. Absolute path names start with a slash (/) and specify the name relative to the root directory.

● A file name must not exceed 1024 characters in length. This limit includes the directory names, the name of the file itself and the slashes which are used as separators.
2.3.5 Built-in POSIX commands

Built-in POSIX commands are commands which the POSIX shell interprets and processes itself without, consequently, generating a new process. They differ from all the other "external" commands in the following ways:

- As subprograms, they are components of the binary file /bin/sh. This means that you are not able to rename them. You can change access permissions for the /bin/sh file only, not for the individual built-in POSIX commands.
- The POSIX shell prioritizes execution of built-in POSIX commands. If a built-in POSIX command has the same name as an external command then the POSIX shell always executes the built-in POSIX command when this name is specified. The external command is executed if the call name contains a slash (/), for example if it is called with the corresponding absolute path name.
- Built-in POSIX commands are faster because they are executed by the POSIX shell itself.

2.3.6 I/O redirection

Before a command is executed, its input and output may be redirected by means of a special notation interpreted by the POSIX shell. The following arguments may appear anywhere in a simple command or may precede or follow a command. They are not passed on to the invoked command, but are interpreted by the shell. Command and parameter substitution occurs before the specified file or file descriptor is used. File name generation is performed only if the pattern matches a single file. Blank interpretation is not performed.

<file
Redirects the standard input (file descriptor 0) of a command to file, so that input to the command is read from file.

>file
Redirects the standard output (file descriptor 1) of a command to file, so that output from the command is written to file. If the file does not exist, it is created. If it does exist, is a regular file, and the noclobber option is set (see Built-in commands, set -o), this causes an error. If noclobber is not set, the file is truncated to zero length and its existing contents are lost.

>|file
Same as >file, except that it overrides the noclobber option.
Entering commands from the POSIX shell

I/O redirection

`>>file`
Redirects the standard output (file descriptor 1) of a command to `file`, so that output from the command is written to `file`. If the file exists, output is appended to it; otherwise, the file is created.

`<file`
Opens `file` for reading and writing as standard input.

`<<[-]string`
Introduces a "here document" (explained below). "Here documents" are used primarily in shell procedures to supply values to commands which can read from standard input. No parameter substitution, command substitution or file name generation is performed on `string`. Input to the POSIX shell is read up to (but excluding) a line that literally matches `string` or to an end-of-file. The resulting document, called a "here document", becomes the standard input for the command.

If one of the characters in `string` is quoted then all the characters of the "here document" are quoted for the POSIX shell.

The `string` marking the last line must not be quoted.

If no characters of `string` are quoted:
- parameter and command substitution is performed,
- escaped newlines (`\newline`) are ignored, and
- a backslash (`\`) must be used to quote backslashes (`\`), dollar signs (`$`), backquotes (`'`) and the first character of `string` if they are to be treated literally in the text.

If `-` is appended to `<<`, then all leading tabs are stripped from `string` and from each line in the here document.

`<&digit`
In the first form, the standard input is duplicated from file descriptor `digit`, i.e. is read from the file associated with `digit`. The second form does the same for the standard output.

`<&-`
The first form closes the standard input for the command, so that EOF is the only input to the command. The second form closes the standard output for the command, so that the command does not write any output.

`<&p`
The input from the co-process in a two-way pipeline is redirected to standard input. Similarly, the output to the co-process is redirected to standard output.
If one of the above redirections is preceded by a digit, the file descriptor number referred to is that specified by the digit (instead of the default 0 for stdin and 1 for stdout).

Example

To open file descriptor 2 (stderr) for writing as a duplicate of file descriptor 1 (stdout):

```
... 2>&1
```

Order of redirections and background commands

The order in which redirections are specified is significant. The POSIX shell evaluates each redirection in terms of the association (file descriptor, file) at the time of evaluation. For example:

```
... 1>file 2>&1
```

first associates standard output (file descriptor 1) with file and then associates standard error (file descriptor 2) with the file associated with file descriptor 1 (i.e. file). In other words, both the standard output and the error messages from the command are written to file. If the order were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been), file descriptor 1 with file. This would mean that only the standard output would be written to file, not the error messages.

If you use & to start a command in the background without active job control then the standard input is automatically associated with a file which possesses the same properties as /dev/null. If job control is active then the environment for command execution contains the same file descriptors, modified as appropriate by the input/output statements, as the executing POSIX shell.

You can set commands or pipelines to run in the background from within a script. They can then communicate with your program. To spawn a co-process of this type, you put the operator |& after the command. You can only use two-way pipes in scripts, not on the command line.

Command strings can be called as two-way pipes. If you abort the original process (e.g. with kill -9 PID) and then later attempt to write another command to a two-way pipe then a subshell is indeed called but the process is stopped. You receive the error message:

```
sh: bad file unit number.
```
2.3.7 Shell procedures and processes

Shell procedures are commands which have been grouped together to form a program. In order to create a shell procedure you must write the desired sequence of commands to a file. Enter `sh <file name>` to call a shell procedure. If the file possesses execution permission (`chmod +x <file name>`), you can call this command sequence directly, that is to say without `sh`.

This section describes the effect of the POSIX process structure when you are working with the POSIX shell, i.e. when you are entering commands or working with shell procedures.

Any input that you enter after `/START-POSIX-SHELL` is received by the POSIX shell and processed accordingly.

The most important commands are implemented as “built-in” commands. Some commands actually spawn a separate process, so the process in which the POSIX shell is running thus becomes a parent process.

Basically, any process can spawn a new process. Until the process spawns another process it is the child process of a parent process. However, as soon as it generates a new process it becomes the parent process of the new child process. It nevertheless retains its status as the child process of the process to which it is subordinate. In other words, a process can simultaneously be a parent process and a child process.

The parent-child process hierarchy is of great importance if you want to pass environment variables in shell procedures. All users can define the environment variables with which they wish their POSIX shell (= parent process) to operate. However, these environment variables are initially known only by the POSIX shell. If a child process is to operate with the environment variables used in the parent process (= POSIX shell) then these variables must be exported to the child process. The `export` command is available for this purpose.
2.3.8 Processing commands using the POSIX shell

Apart from the built-in POSIX commands, every command is linked to a file in the POSIX file system. The names of most command files are entered in the directory `/usr/bin`. The shell variable `$PATH` contains a list of directories in which commands are searched. When executing a command, the POSIX shell reads the command, searches in the directories contained in `$PATH` for a file name which is identical to the command name, and executes the command.

In order to execute a command that is not one of its built-in commands, the POSIX shell spawns a new process and waits until that process terminates. It then displays the setting of the shell variable `PS1` as a prompt (by default, this is `$` for non-privileged users or `#` for privileged users), which indicates that it is ready to process a new command. If you enter the command `date`, for example, the current date will immediately be displayed, and the POSIX shell will then issue the prompt so that you can enter a new command.

In the case of a command such as `date`, the period that elapses between command input and the POSIX shell reporting ready to process a new command is very short. However, this period may be substantially longer when a different command is input, for example when a file is compiled. This is because the POSIX shell does not report until execution of the command has been terminated. In order to avoid having to wait for extended periods at the terminal you can use the special character `&`. If you terminate command input with this character, the POSIX shell reports immediately with the screen prompt without waiting to terminate execution of the entered command. Commands which are terminated with `&` are started as background processes which you may leave to run without intervention.
### 2.3.9 Compound commands

**Syntax**

```bash
for .identifier[..in..word...]; do ..command_list; done
```

or

```bash
for .identifier[..in..word_list]
do ..command_list
done
```

The `for` command can be used to repeatedly execute `command_list`. Each time a `for` loop is executed, `identifier` is set to the next word taken from the `word_list`. If `word_list` is omitted, the `command_list` is executed once for each positional parameter ("$@") that is set. Execution ends when there are no more words in the `word_list`.

**Syntax**

```bash
select ..identifier[..in..word_list] do ..command_list; done
```

or

```bash
select ..identifier[..in..word_list]
do ..command_list
done
```

The `select` command can be used to repeatedly execute a given `command_list` under input control. `select` prints the set of words in `word_list` on standard output, each preceded by a number. The `PS3` prompt is then printed, and a line is read from the standard input. The content of the line read from standard input is saved in the variable `REPLY`. If this line matches the number of one of the listed words, then the value of the parameter `identifier` is set to the word corresponding to this number. If the line is empty, the `word_list` is printed again, and `identifier` is set to the name of the shell script. If `in word_list` is omitted, then the positional parameters are used instead (as is done with `for` loops). The `select` loop is executed repeatedly until a `break` built-in or end-of-file is encountered.

**Example**

The following POSIX shell script can be used to selectively print information on each individual file in the current directory:

```bash
select file in `ls`
do
if [ -z "$file" ]
then
echo "Select number"
else
ls -lsid $file
fi
done
```
**Compound commands**

The `case` command provides for conditional branching to a command list on the basis of a pattern. `case` executes the `command_list` associated with the first `pattern` that matches `word`. The form of the patterns is the same as that used for file name generation (see the section “File name generation” on page 54).

```
Syntax

```case word in
  [[(pattern[...])command_list;]...
esac
```

The `case` command can be used to conditionally execute a number of different command lists. The first list following `if (command_list1)` is executed as the condition, and if it returns a zero exit status (true), the list following the first `then` is executed `(command_list2)`. Otherwise, the list following `elif (command_list3)` is executed as the next condition, and if its value is zero, the list following the next `then (command_list4)` is executed. Failing that, the `else` list `(command_list5)` is executed. If no `else` list or `then` list is executed, the `if` command returns a zero exit status.

```
Syntax

```if .command_list1
  then .command_list2
  [elif .command_list3
   then .command_list4]
  [else .command_list5]
fi
```

The `while` and `until` commands define loops with exit conditions. A `while` command repeatedly executes the condition `command_list1`, and if the exit status of the last command in the list is zero (i.e. true), it executes the `do` list in the body of the loop `(command_list2); otherwise the loop terminates. If no commands in `command_list2` (the `do` list) are executed, the `while` command returns a zero exit status.

```
Syntax

```while .command_list1
  do .command_list2
done
until .command_list1
  do .command_list2
done
```
until may be used in place of while to negate the loop termination test. The until command tests whether the exit condition is false (non-zero) and terminates the loop as soon as it returns an exit status of zero (true).

Syntax

continue..number

continue continues with the next iteration of the superordinate for, while, until or select loop. If number has been specified, then the next iteration is performed in the superordinate loop which corresponds to number. If number is equal to 0, processing continues beyond the outermost loop. continue 1 has the same effect as continue.

Syntax

break..number

break aborts the superordinate for, while, until or select loop. If number has been specified, then execution continues in the superordinate loop which corresponds to number. If number is equal to 0, then the outermost loop is exited. break 1 has the same effect as break.

Syntax

return..number

If return is called in a function then the command returns to the calling procedure. The return status is determined by number or by the last command executed. If return is called outside of a function or in a procedure executed with a leading dot, then it has the same effect as a call to exit.

Syntax

(command_list)

Executes command_list in a separate environment.

i

If two consecutive left-hand parentheses ( are required to define nested commands then these must be separated by a space. Failure to do so may result in their being interpreted as an arithmetical expression (see Command substitution).

Syntax

{command_list;}

command_list is simply executed in the current POSIX shell.

i

The left brace '{' must be followed by a space!
Note that unlike the parentheses ( and ), the braces { and } are reserved words and must be typed at the beginning of a line or after a semicolon (;) in order to be recognized as such.
### Compound commands

**Syntax**

```bash
[[expression]]
```

Evaluates the conditional expression and returns a zero exit status if expression is true; otherwise, 1.

See *Conditional expressions* for a description of expression.

**Syntax**

```bash
function identifier {...command_list;}
identifier () {...command_list;}
```

Defines a function named identifier. This name is used to call the function like a command. The body of the function consists of the command_list that is enclosed within curly braces { and }.

*The left brace '{' must be followed by a space!*

The braces {} are unnecessary if

- the command list consists of a single `for`, `case`, `if`, `while`, `until` or `select` statement.

*Example*

```bash
function identifier case word in ...
identifier () for identifier ...
```

- the command list is executed in a subshell, i.e. () instead of {}.

*Example*

```bash
function identifier (command_list)
identifier () (command_list)
```

**Syntax**

```bash
time pipeline
```

The commands in *pipeline* are executed and the elapsed time and the user and system time are reported on standard error. The following reserved words are only recognized as the first word of a command and when not quoted:

```bash
if then elif else fi case esac for select while until do done { } function time [[ ]]
```
2.3.10 Comments

A word beginning with the hash character # causes that word and all the following characters in the line to be ignored.

2.3.11 Aliasing

If the first word of a command is a defined alias, it is replaced by the text of the alias. An alias name consists of any number of characters excluding metacharacters, quoting characters, file expansion characters, parameter and command substitution characters, and the equal sign =. The replacement string can contain any valid POSIX shell script including the metacharacters listed above. The first word of each command in the replaced text will in turn be tested for aliases. If the last character of the alias value is a blank, the word following the alias will also be checked for alias substitution.

Aliases can be used to redefine built-in commands but cannot be used to redefine the reserved words listed above. Aliases can be created, exported and listed on standard output with the alias command and can be removed with the unalias command. Exported aliases remain in effect for scripts invoked by name, but must be reinitialized for separate invocations of the POSIX shell.

Aliasing is performed when scripts are read, not while they are being executed. Thus for an alias to take effect the alias definition command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the built-in alias command allows the value of the alias to be automatically set to the full path name of the corresponding command. These aliases are called tracked aliases. The value of a tracked alias is defined the first time the corresponding command is looked up and becomes undefined each time the PATH variable is reset. These aliases remain tracked so that the next reference will redefine the value. Several tracked aliases are compiled into the POSIX shell. The -h option of the built-in set command makes each referenced command name into a tracked alias.
Tilde substitution

The following exported aliases are compiled into the POSIX shell but can be unset or redefined with `unalias`.

```plaintext
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup'
r='fc -e -'
true=':'
type='whence -v'
```

Example

The following aliases could be defined to construct `l`, `ll` and `lf`:

```plaintext
alias l='/bin/ls -m'
alias ll='/bin/ls -l'
alias lf='/bin/ls -CF'
```

2.3.12 Tilde substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~ (tilde). If it does, then the word up to the next / (slash) is checked to see if it matches a login name in the `/etc/passwd` file. If a match is found, the tilde and the matched login name are replaced by the login directory of the matched user. This is called a tilde substitution. If no match is found, the original text is left unchanged. A tilde by itself, or in front of a slash, is replaced by `$HOME`. A tilde followed by a plus sign or a minus sign is replaced by `$PWD` and `$OLDPWD`, respectively.

Example

The command `cat ~/.profile` allows you to edit the `.profile` file in your login directory from any location in your file tree.
2.3.13 Command substitution

There are two ways of substituting the standard output of a command for a part of a word or a full word. In the first (new) form, the command is enclosed in parentheses and preceded by a dollar sign: $\left(...\right)$. In the second (old) form, the command is enclosed within backquotes: ‘...’. If the second form is used, the string between the backquotes is processed for special quoting characters before the command is executed (see the section “Quoting metacharacters” on page 55). Trailing newlines are removed for both forms.

The command substitution $\left(cat \; file\right)$ can be replaced by the equivalent but faster $\left(<\; file\right)$. Command substitutions on built-in commands which do not perform I/O redirection are carried out without a separate process being spawned.

Example The following command can be used to edit all the files in a directory which have names ending in .c and contain the string include:

```
for name in $( grep -l include *.c )
do
    edt $name
done
```

`grep -l include *.c` scans all .c files for the string include. The -l option tells grep to list the names of all the files in which it finds the string.

An arithmetic expression enclosed in double parentheses and preceded by a dollar sign, i.e. $\left(...\right)$, is replaced by the value of the arithmetic expression within the double parentheses.

Example This command will print the second-last parameter in a shell script:

```
eval print \$$(( \$#-1 ))
```
2.3.14 POSIX shell variables and parameter substitution

A parameter is an identifier, one or more digits, or any of the following characters:
* @ # ? - $ !

A variable (a parameter denoted by an identifier) has a value and zero or more attributes. Variables can be assigned values and attributes with the built-in `typeset` command. The attributes supported by the POSIX shell are described under `typeset`. Exported parameters pass their values and attributes to the environment.

The POSIX shell supports one-dimensional arrays. An element of an array variable is referenced by a subscript. A subscript is denoted by a left square bracket [, followed by an arithmetic expression (see the section "Arithmetic evaluation" on page 56), followed by a right square bracket ].

To assign values to an array, you can use the built-in `set` command as follows:

```
set -A name value...
```

The value of all subscripts must be in the permissible range. Arrays need not be declared; any reference to a variable with a valid subscript is legal, and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing element zero.

```
name=value[...name=value]...
```

The value of a variable may be assigned in this form. If the integer attribute is set for a variable, its value can be used for arithmetic evaluations. Positional parameters (denoted by a number) may be assigned values with the built-in `set` command.

Parameter `$0` is set from argument zero when the shell is invoked.

```
${#parameter}
```

If `parameter` is defined by the character * or @, then all the positional parameters, starting with `$I`, are substituted. The first character of the `IFS` variable is used as the field separator (see below). If an array identifier with the subscript * or @ is used, the value for each of the elements is substituted (separated by the same field separator character).

```
${#parameter}
```

If `parameter` is * or @, the number of positional parameters is substituted. Otherwise, the length of the value of `parameter` is substituted.
Entering commands from the POSIX shell

Shell variables and parameter substitution

$\{\#identifier[*]\}

The number of elements in the array identifier is substituted.

$\{parameter:=-word\}

If parameter is set and is non-null, its value is substituted; otherwise, word is substituted.

$\{parameter:=word\}

If parameter is not set or is null, then it is set to word. The value of the parameter is then substituted.

Positional parameters may not be assigned values in this way.

$\{parameter:?word\}

If parameter is set and is non-null, its value is substituted; otherwise, word is printed and the shell exits. If word is omitted, a standard message is printed.

$\{parameter:+word\}

If parameter is set and is non-null, word is substituted; otherwise, the null string is substituted.

$\{parameter#pattern\}

$\{parameter##pattern\}

If the POSIX shell pattern matches the beginning of the value of parameter, then the value of this substitution is the value of parameter with the matched portion deleted; otherwise, the value of parameter is substituted. In the first form, the smallest matching pattern is deleted; in the second form, the largest matching pattern is deleted.

$\{parameter%pattern\}

$\{parameter%%pattern\}

If the POSIX shell pattern matches the end of the value of parameter, then the value of this substitution is the value of parameter with the matched portion deleted; otherwise, the value of parameter is substituted. In the first form, the smallest matching pattern is deleted; in the second form, the largest matching pattern is deleted.

In the previous 8 substitutions, word is not evaluated unless it is to be used as the substituted string.

Example

pwd is only executed if dir is not set or is equal to the null string.

echo \${dir:=-$(pwd)}

If the colon is omitted from the above expressions, the POSIX shell only checks whether or not parameter is set; it does not check whether it is equal to the null string.
The following parameters are automatically set by the POSIX shell:

#   The number of positional parameters (in decimal).
-   All options supplied to the POSIX shell on invocation or by the `set` command.
?   The exit status returned by the last executed command.
$   The process ID of the current POSIX shell.
 _  Initially, the value of _ (underscore) is an absolute path name of the POSIX shell or script being executed as passed in the environment. Thereafter, it is always assigned the last argument of the previous command. This parameter is not set for commands which run asynchronously. The underscore parameter is also used to hold the name of the matching `MAIL` file when checking for mail.

Example
The `tail` command uses $_ to access the last file of the preceding `cat` command.

```
cat /usr/tmp/mydir/xyz* > all
tail $_
```

!   The process number of the last command invoked as a background process.

**ERRNO**
The value of `errno` as set by the most recently failed system call. This value is system-dependent and is intended for debugging purposes.

**LINENO**
The line number of the current line within the script or function being executed.

**OLDPWD**
The previous working directory set by the `cd` command.

**OPTARG**
The value of the last option argument processed by the built-in `getopt` command.

**OPTIND**
The index of the last option argument processed by the built-in `getopt` command.

**PPID**
The process number of the parent of the current POSIX shell.

**PWD**
The present working directory set by the `cd` command.

**RANDOM**
Each time this variable is referenced, a random integer, uniformly distributed between 0 and 32767, is generated. You can initialize the sequence of random numbers by assigning a numeric value to `RANDOM`. 
REPLY
This variable is set by the `select` statement and by the built-in `read` command when no arguments are supplied.

SECONDS
Each time this variable is referenced, the number of seconds since the invocation of the POSIX shell is returned. If this variable is assigned a value, the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

The following variables are used by the POSIX shell:

CDPATH
The search path for the `cd` command.

COLUMNS
If this variable is set, then its value is used to define the width of the editing window when the POSIX shell is in edit mode as well as for outputting the `select` list. This variable is of use when the POSIX shell is accessed via `rlogin`.

EDITOR
If the value of this variable ends with `vi` and if the variable `VISUAL` is not set then the corresponding option is set (see `set`). This variable is of use when the POSIX shell is accessed via `rlogin`.

ENV
If this variable is set then it contains the path name of the procedure which is executed when the POSIX shell is called. This procedure is mostly used for function and alias definitions. The value of this variable is used to perform parameter substitution for file name generation.

FCEDIT
The name of the standard editor for the built-in `fc` command. The only value which is currently permissible for this variable is `edt`.

FPATH
The search path for function definitions. This path is used when a function having the attribute `-u` is accessed and no command is found. If an executable file is found then it is read and executed in the current environment.

HISTFILE
If this variable is set when the POSIX shell is invoked, its value is the path name of the file that will be used to store the command history (see the section “Command re-entry” on page 65).
HISTSIZE
If this variable is set when the POSIX shell is invoked, the shell will remember the commands you enter (command history). The number of previously entered commands that are accessible by this shell will be greater than or equal to the given number. The default is 128.

HOME
The default argument (home directory) for the cd command.

IFS
The POSIX shell's internal field separator, used to separate command words which result from command or parameter substitution and to separate words with the built-in read command. The value of IFS is normally set to space, tab, and newline. The first character of the IFS variable is used to separate arguments in "$*" substitutions (see the section “Quoting metacharacters” on page 55).

IO_CONVERSION
If this variable is set to YES, and if POSIX commands (e.g. awk, cat, grep etc.) are being used to access files in a (mounted) ASCII file system, then conversion is performed automatically.

LINES
If this variable is set, its value is to determine the column length for printing select lists. select lists will print vertically until about two-thirds of LINES lines are filled.

MAIL
If this variable is set to the name of a mail file and the MAILPATH variable is not set, the shell informs you of the arrival of mail in the specified file.

MAILCHECK
The value of this variable specifies how often (in seconds) the POSIX shell should check for changes in the modification time of any of the files specified by the MAILPATH or MAIL variables. The default value for MAILCHECK is 600 seconds. When the specified time has elapsed, the POSIX shell checks the mail files before issuing the next prompt.

MAILPATH
A colon-separated list of file names. If this variable is set, the shell informs you of any modifications to the specified files that have occurred within the last MAILCHECK seconds. Each file name may be followed in the list by a question mark and a message that is to be printed. The message will undergo parameter substitution, with the variable $_ defined as the name of the file that has changed. The default message is: "you have mail in $_".

PATH
The search path for commands (see Execution). You may not change the value of this variable if you are using a restricted POSIX shell.
Enetering commands from the POSIX shell

Shell variables and parameter substitution

**PS1**
The value of this variable is expanded for parameter substitution in order to define the prompt used by the POSIX shell. The default value is "$" or, in the case of privileged users, "# ". The use of an exclamation mark ! in the prompt is replaced by the command number (see the section "Command re-entry" on page 65).

**PS2**
Secondary prompt string displayed by the POSIX shell when further input is expected after a newline character. The default is "> ".

**PS3**
Prompt string used within a select loop to prompt for the desired number. The default is "#? ".

**PS4**
The value of this variable is expanded for parameter substitution and precedes each line of an execution trace. The default execution trace prompt is "+ ".

**SHELL**
The path name of the POSIX shell is stored in the environment. When called, the POSIX shell acts as a restricted shell if the pattern *r*sh* matches the file name component of the path name (see section "basename return non-directory portion of path name" on page 149).

**TMOUT**
If TMOUT is set to a value greater than zero, the POSIX shell will automatically terminate if a command is not entered within the prescribed number of seconds after issuing the PSI prompt.
(Caution: the POSIX shell may have been compiled with a maximum bound for this value which cannot be exceeded).

**VISUAL**
If the value of this variable ends with vi then the corresponding option is se (see section "Built-in POSIX commands" on page 36). This variable is of use when the POSIX shell is accessed via rlogin.

The POSIX shell assigns default values to the following variables:

PATH, PS1, PS2, PS3, PS4, MAILCHECK, TMOUT and IFS.

The variables HOME, MAIL and SHELL are set by the command /START-POSIX-SHELL.

### 2.3.15 Blank interpretation

After parameter and command substitution, the results of substitutions are scanned for field separator characters and split into distinct arguments where such characters are found. Explicit null arguments (e.g. " " or ") are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.
2.3.16 File name generation

After these various substitutions, each word is examined for the presence of an asterisk *, question mark ?, or left-hand square bracket [. However, this examination is only performed if the option `-f` (see `set`) has not been set. If one of these characters is found in a word then the word is considered as a pattern. The word is then replaced by lexicographically sorted file names which match the pattern. If no matching file name is found for the pattern then the word is retained unchanged.

If you use patterns for file name generation you must pay special attention to the characters period . and slash /: a period at the start of a file name or immediately after a /, as well as / itself must match explicitly. The special handling of these characters does not apply to other substitutions.

* matches any string, including the null string.

? matches any single character. (Slashes and dots are treated specially; see above.)

[..] matches any one of the characters enclosed within the square brackets.

A pair of characters separated by a dash (-) matches any character lexically between the pair, inclusive. A dash can be included in the set of characters to be matched as long as it is the first or last character.

If the first character following the left bracket is an exclamation point, the enclosed set of characters is negated: any character not enclosed is matched.

A pattern list is a list of one or more patterns separated from each other by a | (vertical bar). Composite patterns can be formed with one or more of the following constructs:

?(pattern-list)

Optionally matches any one occurrence of the given patterns.

*(pattern-list)

Matches zero or more occurrences of the given patterns.

+(pattern-list)

Matches one or more occurrences of the given patterns.

@(pattern-list)

Matches exactly one of the given patterns.

!(pattern-list)

Matches anything, except one of the given patterns.
2.3.17 Quoting metacharacters

A metacharacter is one of the following characters:

; & ( ) | < > space tab newline-character

A *blank* is either a tab or space. An identifier consists of a string of letters, digits and the underscore character _. This string must begin with a letter or underscore. Identifiers are used as names for functions and variables. A word is a string of characters which are separated by one or more metacharacters. Each of the metacharacters has a special meaning for the POSIX shell and is used as a word separator unless quoted.

<table>
<thead>
<tr>
<th>Quote</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>A metacharacter preceded by a backslash \ is quoted and therefore stands for itself. The pair <code>\newline-character</code> is ignored or deleted by the POSIX shell.</td>
</tr>
<tr>
<td><code>'...</code></td>
<td>All characters which are enclosed in apostrophes <code>...</code> are quoted. However, a pair of apostrophes may not enclose a single apostrophe.</td>
</tr>
<tr>
<td><code>&quot;...&quot;</code></td>
<td>Strings which are enclosed in quotes &quot;...&quot; are subject to parameter and command substitution. When accompanied by a backslash they can be used to quote the backslash \, backquote ``, quotation mark <code>'</code> and dollar sign $. $* and $@ mean the same thing provided that they are not enclosed in quotes or used as a file name or as a value for variable assignment. Their meanings differ if they are used alone in quotes or if they used as command arguments. &quot;$*&quot; corresponds to &quot;$1 $2 ...&quot; if $ is the value of the first character of the variable IFS, and &quot;$@&quot; stands for &quot;$1 $2 ...&quot;, that is to say that the individual call arguments are retained.</td>
</tr>
<tr>
<td><code>\</code>...`</td>
<td>Strings which are enclosed in backquotes <code>\</code>...<code>can be quoted by using the backslash \\, backquotes</code> <code>, and the dollar sign $. If the whole string needs to be enclosed in quotation marks &quot;...</code>...<code>&quot; you can use the backslash \\ to quote the quotation mark </code>&quot;`.</td>
</tr>
</tbody>
</table>

The special meaning of reserved words or aliases can be removed by quoting any character of the reserved word. However, it is also sufficient to entire a single "" before the name in order to quote the entire word (e.g. "while").

The recognition of function names or built-in command names cannot be suppressed in this way.
### 2.3.18 Arithmetic evaluation

The built-in `let` command provides a mechanism for performing integer arithmetic. Evaluations are performed using `long` arithmetic. Constants are of the form `[base#]n`, where `base` is a decimal number between two and thirty-six representing the arithmetic base: `n` is a number in that base. If `base#` is omitted, base 10 is used.

An arithmetic expression uses much the same syntax, precedence, and associativity as the C language. All the integral operators, other than `++`, `--`, `?:` and the comma are supported. Variables can be referenced by name within an arithmetic expression without using the parameter substitution syntax (the `$` character). When a variable is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a variable can be specified as an attribute with the `-i` option of the built-in `typeset` command. Arithmetic evaluation is performed on the value of each assignment to a variable with the `-i` attribute. If you do not specify an arithmetic base, the first assignment to the variable determines the arithmetic base. This base is used when parameter substitution is performed.

Since many of the arithmetic operators must be quoted for the POSIX shell, an alternative form of the `let` command is provided. For any command which begins with a double left parenthesis `((`, all the characters until a matching double right parenthesis `))` are treated as a quoted expression. Thus `((a=a+b))` is equivalent to `let "a=a+b"`.
2.3.19 Conditional expressions

A conditional expression can be used to test attributes of files and to compare algebraic expressions and strings. In the POSIX shell, conditional expressions are specified as part of a compound command of the form 

```
[ ...
```

Blank interpretation and file name generation are not performed on the words of the conditional expression between 

```
[ and ]
```

Each conditional expression can be constructed from one or more of the following unary or binary expressions:

In each of the expressions below, if `file` is of the form `/dev/fd/n`, where `fd` is the file descriptor and `n` is an integer, then the test is applied to the open file whose descriptor number is `n`.

- `-a..file` (access) True if `file` exists.
- `-b..file` (block device) True if `file` exists and is a block special file.
- `-c..file` (character device) True if `file` exists and is a character special file.
- `-d..file` (directory) True if `file` exists and is a directory.
- `-f..file` (file) True if `file` exists and is an ordinary file.
- `-g..file` (group ID) True if `file` exists and has its set-group-ID bit set.
- `-k..file` (sticky) True if `file` exists and has its sticky bit set.
- `-o..option` (option) True if the named `option` is turned on (`option` can be set with `set`). You must use the full option name, e.g. `errexit`.
- `-p..file` (pipe) True if `file` exists and is a FIFO special file or a pipe.
- `-r..file` (read) True if `file` exists and the current process has read permission for it.
- `-s..file` (size) True if `file` exists and has a size greater than zero.
- `-t..filedes` (terminal) True if file descriptor number `filedes` is open and is associated with a terminal.
- `-u..file` (user ID) True if `file` exists and has its set-user-ID bit set.
Strings, Algebraic integer comparisons

Entering commands from the POSIX shell

-\texttt{w..file}  
(\texttt{write}) True if \texttt{file} exists and the current process has write permission for it.

-\texttt{x..file}  
(\texttt{execute}) True if \texttt{file} exists and the current process has execute permission for it. If \texttt{file} exists and is a directory, then the current process must have permission to search in the directory.

-\texttt{G..file}  
(\texttt{group}) True if \texttt{file} exists and its group matches the effective group ID of the current process.

-\texttt{L..file}  
(\texttt{symbolic link}) True if \texttt{file} exists and is a symbolic link.

-\texttt{O..file}  
(\texttt{owner}) True if \texttt{file} exists and is owned by the effective user ID of the current process.

-\texttt{S..file}  
(\texttt{socket}) True if \texttt{file} exists and is a socket.

\texttt{file1..\texttt{-nt..file2}}  
\texttt{(newer than)} True if \texttt{file1} exists and is newer than \texttt{file2}.

\texttt{file1..\texttt{-ot..file2}}  
\texttt{(older than)} True if \texttt{file1} exists and is older than \texttt{file2}.

\texttt{file1..\texttt{-ef..file2}}  
\texttt{(equal file)} True if \texttt{file1} and \texttt{file2} exist and are links to the same file.

**String attributes and comparisons**

-\texttt{n..string}  
\texttt{(non-zero)} True if \texttt{string} exists and is not a null string, i.e. if the length of the string is greater than 0.

-\texttt{z..string}  
\texttt{(zero)} True if the specified \texttt{string} is a null string, i.e. if the length of the string is 0.

\texttt{string../=..pattern}  
\texttt{True if \texttt{string} matches \texttt{pattern}.}

\texttt{string..!=..pattern}  
\texttt{True if \texttt{string} does not match \texttt{pattern}.}

\texttt{string1..<..string2}  
\texttt{True if \texttt{string1} comes before \texttt{string2} in the EBCDIC collating sequence.}

\texttt{string1..>..string2}  
\texttt{True if \texttt{string1} comes after \texttt{string2} based in the EBCDIC collating sequence.}
Algebraic comparisons between integers

expr1 -eq expr2
  (equal) True if expr1 is equal to expr2.

expr1 -ne expr2
  (not equal) True if expr1 is not equal to expr2.

expr1 -lt expr2
  (less than) True if expr1 is less than expr2.

expr1 -gt expr2
  (greater than) True if expr1 is greater than expr2.

expr1 -le expr2
  (less than or equal) True if expr1 is less than or equal to expr2.

expr1 -ge expr2
  (greater than or equal) True if expr1 is greater than or equal to expr2.

Negated and compound expressions

A compound expression can be constructed from the above expressions with any of the following mechanisms. The mechanisms are listed in decreasing order of precedence:

(expression)
  True if expression is true. The enclosed expression can be a single expression or a group of concatenated expressions.

!expression
  Negation: true if expression is false.

expression1 && expression2
  Logical AND: True if expression1 and expression2 are both true.

expression1 || expression2
  Logical OR: True if either expression1 or expression2 is true.
2.3.20 Environment

The environment of a process consists of a list of name=value pairs that is passed to an executed program in the same way as a normal argument list. The names must be POSIX shell-style identifiers; the values must be character strings (including the null string).

The POSIX shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a variable for each name found, giving it the corresponding value and marking it for export. Executed commands inherit the environment. If the user modifies the values of these variables or creates new ones using the export or typeset -x commands, they become part of the environment. The environment seen by any executed command is thus composed of any name=value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions that have been marked in export or typeset -x commands.

The environment for any simple command or function may be augmented by prefixing it with one or more variable assignments. A variable assignment argument is a word of the form identifier=value.

command sees the following lines as equivalent:

TERM=450 command arguments
( export TERM ; TERM=450 ; command arguments )

If the -k option was set on POSIX shell invocation or with the built-in set command, all variable assignment arguments are exported to the environment, even if they occur after the command name.
2.3.21 Functions

The reserved word *function* (see section "Compound commands" on page 41) is used to define POSIX shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands, with arguments passed as positional parameters (see section "Execution" on page 64).

Functions execute in the same process as the caller and share all open files and the present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate; the condition is then passed on to the caller. A trap on *EXIT* set inside a function is executed after completion of the function in the environment of the caller.

Variables are usually shared between the calling program and the function. However, the built-in *typeset* command can be used within a function to define local variables whose scope includes the current function and all functions it calls.

The built-in *return* command is used to return from function calls. Errors that occur within functions return control to the caller.

Function identifiers or names can be listed with the *-f* or *+f* option of the built-in *typeset* command. The text of functions may also be listed with the *-f* option. Functions can be undefined with the *-f* option of the built-in *unset* command.

Functions are usually not accessible when the POSIX shell executes a shell script. The *-xf* option of the *typeset* command allows a function to be marked for export. These functions can then be used in scripts that are executed without a separate invocation of the POSIX shell. Functions that need to be defined across separate invocations of the shell should be specified in the *ENV* file with the *-xf* option of *typeset*.

**Example**

The following function named *lh* lists the top two levels of the directory hierarchy in which you are located or which you specify as an argument. Regular files are ignored as arguments.

```bash
define function lh
{
  for i in ${*:-.}
  do
    if [[ -d $i ]]
    then
      print $i:
      cd $i
      ls -CF $( ls )
      cd - >/dev/null
    fi
  done
}
```
The \texttt{for} loop processes each of the specified arguments in succession. If you have not specified an argument, $*$ is set to the current directory ($\). If a directory is present, its name and contents are listed as follows: \texttt{cd $i} switches to the directory to be listed; \texttt{$( ls )}$ is replaced by the content of the directory $i$, and \texttt{ls -CF directory_list} lists the files, followed by the contents of the directories, with flags to indicate executables and directories. The subsequent \texttt{cd} command silently returns to the initial directory (before the first \texttt{cd $i}$).

### 2.3.22 Jobs

If the \texttt{monitor} option of the built-in \texttt{set} command is turned on (\texttt{set -m}), an interactive POSIX shell associates a job with each pipeline. It maintains a table of current jobs, which you can written to standard output with the built-in \texttt{jobs} command, and assigns them small integer numbers. When a job is started asynchronously with &, the shell prints a line of the form:

\begin{verbatim}
[1] 1234
\end{verbatim}

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process ID was 1234.

If you use \texttt{rlogin} to access the POSIX shell (the following key combinations are supported only on character-mode terminals):

- If you want to start other processes while an existing process is being executed then you simply need to press the key combination \texttt{CTRL Z}. This causes a STOP signal to be sent to the process which is currently running. The POSIX shell then indicates that the job has been stopped and issues a prompt. You may then modify the status of this job: you can use the built-in command \texttt{bg} (background) to run the job in background mode, leave it stopped while you execute other commands or retrieve it to the foreground using the built-in command \texttt{fg} (foreground).

\texttt{CTRL Z} is processed as soon as it is pressed and its effect is similar to that of the interrupt key combination \texttt{CTRL D}. Inputs which have not yet been read and outputs which have not yet been displayed are discarded.

There are several ways to reference jobs in the shell. A job can be referenced by the process ID of any process or by one of the following expressions:

- \texttt{%number} the job with the given job \texttt{number}.

- \texttt{%string} any job whose command line begins with \texttt{string}.

- \texttt{%?string} any job whose command line includes \texttt{string}.
the current job.

%%

equivalent to %%

%- the previous job.

The POSIX shell registers changes in the status of jobs immediately. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

When the monitor option is turned on, each background job that completes triggers any trap set for CHLD.

If you try to exit the POSIX shell while jobs are running in the background or stopped, you will be warned with a message.

You may then use the jobs command to obtain an overview of the current situation. If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

### 2.3.23 Signals

The INT and QUIT signals for an invoked background command (&) are ignored if the job monitor option is not active. Otherwise, signals have the values inherited by the POSIX shell from its parent (see also the built-in trap command).
2.3.24 Execution

Each time a command is executed, the substitutions described above are carried out in the following order:

- quoting
- parameter substitution
- tilde substitution
- aliasing
- file name generation
- I/O redirection
- command substitution

If the command name matches the name of one of the built-in commands, it is executed within the current POSIX shell process. Next, the command name is checked to see if it matches one of the user-defined functions. If it does, the positional parameters are saved and then reset to the arguments of the function call. When the function completes or executes a built-in `return` command, the positional parameter list is restored and any trap set on `EXIT` within the function is executed. The exit status is that of the last command executed in the function. A function is also executed in the current POSIX shell process. If a command name is not a built-in command or a user-defined function, a process is spawned and an attempt is made to execute the command via the system call `exec`.

The shell variable `PATH` defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is `/usr/bin:` (specifying `/usr/bin` and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a slash, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an `a.out` file, it is assumed to contain a shell script. A subshell is spawned to read it. Any non-exported aliases, functions and variables are removed in this case. A parenthesized command is executed in a sub-shell without removing non-exported quantities.
2.3.25 Command re-entry

The text of the last HISTSIZE commands entered from a terminal is saved in a history file. The default value for HISTSIZE is 128. If the HISTFILE variable is not set or if the file denoted by its value is not writable, the file $HOME/.sh_history is used to save the command history.

A POSIX shell can access the commands of all interactive shells which use the same history file. The built-in fc command can be used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands may be specified.

If you do not specify an editor program as an argument to fc, the value of the variable FCEDIT is used. If FCEDIT is not defined, /usr/bin/ed is used. The edited commands are printed and re-executed when you leave the editor.

The editor name - (if FCEDIT=-) is used to skip the editing phase and to directly re-execute the command. In this case a substitution variable of the form old=new can be used to modify the command before execution. For example, if r is aliased to fc -e -, then typing r bad=good c will re-execute the most recent command which starts with the letter c, replacing the first occurrence of the string bad with the string good.
2.4 Command summary

The summaries below provide

- an overview of all POSIX commands in the POSIX shell
- an overview according to functions

There are also commands which can be entered from within the POSIX shell but which are not described in this manual, for example the commands `cc` or `c89` which are used to call a compiler (for a description see the “C/C++ (BS2000/OSD)” manual [5]).

2.4.1 Summary of all commands of the POSIX shell

The POSIX shell comprises the basic shell (POSIX-BC) and the extended shell (POSIX-SH). It contains the POSIX commands found in the following table.

Entries in the Type column describe the command type:

- bin separate module
- blt built-in in the shell
- scr script

Apart from these commands, there are also some additional built-in commands (such as for, while, if, break, etc.), which are described in the section “Compound commands” on page 41ff.

The column LFS describes whether the commands can process large POSIX files:

- A (large file aware): uses large files correctly
- S (large file safe): recognizes large files but rejects processing in a defined manner

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Type</th>
<th>Delivery</th>
<th>Description</th>
<th>LFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>adduser</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Assign individual user number</td>
<td></td>
</tr>
<tr>
<td>alias</td>
<td>/usr/bin</td>
<td>blt+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Define or display alias</td>
<td></td>
</tr>
<tr>
<td>ar</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Administer libraries</td>
<td></td>
</tr>
<tr>
<td>asa</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Convert control characters for positioning</td>
<td>S</td>
</tr>
<tr>
<td>at</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Execute commands at a later date</td>
<td></td>
</tr>
<tr>
<td>awk</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Programmable processing of text files</td>
<td>A</td>
</tr>
<tr>
<td>basename</td>
<td>/usr/bin</td>
<td>scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Separate file name from path</td>
<td></td>
</tr>
<tr>
<td>batch</td>
<td>/usr/bin</td>
<td>scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Execute commands at a later date</td>
<td></td>
</tr>
<tr>
<td>bc</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Arithmetic language</td>
<td></td>
</tr>
</tbody>
</table>
## Working with the POSIX shell

### Command summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Type</th>
<th>Delivery</th>
<th>Description</th>
<th>LFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>bg</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Process jobs in background</td>
<td></td>
</tr>
<tr>
<td>bs2cmd</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Execute BS2000 command</td>
<td></td>
</tr>
<tr>
<td>bs2cp</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Copy BS2000 files</td>
<td>A</td>
</tr>
<tr>
<td>bs2do</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Calling BS2000 procedures from POSIX</td>
<td></td>
</tr>
<tr>
<td>bs2file</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Define file attributes for BS2000 files</td>
<td></td>
</tr>
<tr>
<td>bs2lp</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Print files</td>
<td></td>
</tr>
<tr>
<td>bs2pkey</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Assign P keys</td>
<td></td>
</tr>
<tr>
<td>cal</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Display calendar</td>
<td></td>
</tr>
<tr>
<td>cancel</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Delete print jobs</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Concatenate and output files</td>
<td>A</td>
</tr>
<tr>
<td>cd</td>
<td>/usr/bin</td>
<td>blt+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Change current directory</td>
<td>A</td>
</tr>
<tr>
<td>chgrp</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Change group number of file</td>
<td>A</td>
</tr>
<tr>
<td>chmod</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Change access rights</td>
<td>A</td>
</tr>
<tr>
<td>chown</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Change owner of file</td>
<td>A</td>
</tr>
<tr>
<td>cksum</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Write checksums and file sizes</td>
<td>A</td>
</tr>
<tr>
<td>cmp</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Compare files character by character</td>
<td>A</td>
</tr>
<tr>
<td>comm</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Search for identical lines in two sorted files</td>
<td>S</td>
</tr>
<tr>
<td>command</td>
<td>/usr/bin</td>
<td>blt+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Execute simple command</td>
<td></td>
</tr>
<tr>
<td>compress</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Compress files</td>
<td>A</td>
</tr>
<tr>
<td>cp</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Copy files</td>
<td>A</td>
</tr>
<tr>
<td>cp</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Copy files</td>
<td>A</td>
</tr>
<tr>
<td>cpio</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Copy files</td>
<td>A</td>
</tr>
<tr>
<td>crontab</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Swap files and directories in and out</td>
<td>A</td>
</tr>
<tr>
<td>csplit</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Execute commands at regular intervals</td>
<td></td>
</tr>
<tr>
<td>cut</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Split file according to specific criteria</td>
<td>S</td>
</tr>
<tr>
<td>date</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Cut bytes, characters or fields from the lines of a file</td>
<td>S</td>
</tr>
<tr>
<td>dd</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Display time and date</td>
<td></td>
</tr>
<tr>
<td>debug</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Copy and convert files</td>
<td></td>
</tr>
<tr>
<td>df</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Test POSIX programs</td>
<td></td>
</tr>
<tr>
<td>diff</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Display number of free and occupied disk blocks</td>
<td>A</td>
</tr>
<tr>
<td>dirname</td>
<td>/usr/bin</td>
<td>scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Compare files line by line</td>
<td>A</td>
</tr>
<tr>
<td>du</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Display occupied memory space</td>
<td>A</td>
</tr>
<tr>
<td>dumpsf</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Display internal file system information</td>
<td></td>
</tr>
<tr>
<td>echo</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Output call arguments</td>
<td></td>
</tr>
<tr>
<td>ed</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Line editor in interactive mode</td>
<td></td>
</tr>
<tr>
<td>edt</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Call BS2000 file editor EDT</td>
<td>S</td>
</tr>
</tbody>
</table>
## Command summary

### Working with the POSIX shell

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Type</th>
<th>Delivery</th>
<th>Description</th>
<th>LFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>egrep</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Find pattern</td>
<td>S</td>
</tr>
<tr>
<td>env</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Change environment when executing commands</td>
<td></td>
</tr>
<tr>
<td>eval</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Process call arguments and execute them as commands</td>
<td></td>
</tr>
<tr>
<td>ex</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Line editor</td>
<td></td>
</tr>
<tr>
<td>exec</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Overlay current shell</td>
<td></td>
</tr>
<tr>
<td>exit</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Terminate shell procedure</td>
<td></td>
</tr>
<tr>
<td>expand</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Convert tab character to blanks</td>
<td>S</td>
</tr>
<tr>
<td>export</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Export shell variables</td>
<td></td>
</tr>
<tr>
<td>expr</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Evaluate expressions</td>
<td></td>
</tr>
<tr>
<td>expr</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Evaluate expressions</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>/usr/bin</td>
<td>alias+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Return end status not equal to 0</td>
<td></td>
</tr>
<tr>
<td>fc</td>
<td>/usr/bin</td>
<td>blt+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Access to history file</td>
<td>S</td>
</tr>
<tr>
<td>fg</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Bring jobs into foreground</td>
<td></td>
</tr>
<tr>
<td>fgrep</td>
<td>/usr/bin</td>
<td>blt+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Find strings</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Define file type</td>
<td>A</td>
</tr>
<tr>
<td>find</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Search directories</td>
<td>A</td>
</tr>
<tr>
<td>fold</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Split up long lines</td>
<td>S</td>
</tr>
<tr>
<td>fsck</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Check consistency of file system and correct interactively with user</td>
<td></td>
</tr>
<tr>
<td>fsexpand</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Expand existing file systems</td>
<td>A</td>
</tr>
<tr>
<td>ftyp</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Define types of file processing (BS2000)</td>
<td></td>
</tr>
<tr>
<td>gencat</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Generate binary coded message catalog</td>
<td></td>
</tr>
<tr>
<td>genso</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Generate shared object</td>
<td></td>
</tr>
<tr>
<td>getconf</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Call up configuration values</td>
<td>A</td>
</tr>
<tr>
<td>getopts</td>
<td>/usr/bin</td>
<td>blt+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Search procedure arguments for options</td>
<td></td>
</tr>
<tr>
<td>grep</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Get pattern</td>
<td>A</td>
</tr>
<tr>
<td>hash</td>
<td>/usr/bin</td>
<td>alias+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Process shell hash table</td>
<td></td>
</tr>
<tr>
<td>hd</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Hex dump</td>
<td>A</td>
</tr>
<tr>
<td>head</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Output start of a file</td>
<td>A</td>
</tr>
<tr>
<td>iconv</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Convert code</td>
<td>A</td>
</tr>
<tr>
<td>id</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Output user identification</td>
<td></td>
</tr>
<tr>
<td>inetd</td>
<td>/usr/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Daemon for internet services</td>
<td></td>
</tr>
<tr>
<td>info</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Online diagnostic tool</td>
<td></td>
</tr>
<tr>
<td>ipcrm</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Remove setup for interprocess communications</td>
<td></td>
</tr>
<tr>
<td>lpcm</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Output state of interprocess communications setup</td>
<td></td>
</tr>
</tbody>
</table>
## Command summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Type</th>
<th>Delivery</th>
<th>Description</th>
<th>LFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobs</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Output job information</td>
<td></td>
</tr>
<tr>
<td>join</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Merge two files according to matching fields</td>
<td>A</td>
</tr>
<tr>
<td>kill</td>
<td>/usr/bin</td>
<td>blt+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Send signals to processes</td>
<td></td>
</tr>
<tr>
<td>lc</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Output information on directories and files</td>
<td></td>
</tr>
<tr>
<td>let</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Integer arithmetic</td>
<td></td>
</tr>
<tr>
<td>ln</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Enter reference to file</td>
<td>A</td>
</tr>
<tr>
<td>locale</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Call up information on the locale</td>
<td></td>
</tr>
<tr>
<td>localedef</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Define locale</td>
<td></td>
</tr>
<tr>
<td>logger</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Log messages</td>
<td></td>
</tr>
<tr>
<td>logname</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Query login name</td>
<td></td>
</tr>
<tr>
<td>lp</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Print out files</td>
<td></td>
</tr>
<tr>
<td>lpsstat</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Output information on print jobs</td>
<td></td>
</tr>
<tr>
<td>ls</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Output information on directories and files</td>
<td>A</td>
</tr>
<tr>
<td>mailx</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Interactively process messages</td>
<td></td>
</tr>
<tr>
<td>make</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Manage groups of files</td>
<td></td>
</tr>
<tr>
<td>man</td>
<td>/usr/bin</td>
<td>scr</td>
<td>SINLIB.POSIX-SH</td>
<td>Use online documentation</td>
<td></td>
</tr>
<tr>
<td>msg</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Forbid or permit receipt of messages</td>
<td></td>
</tr>
<tr>
<td>mkdir</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Create directory</td>
<td></td>
</tr>
<tr>
<td>mkfifo</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Create FIFO</td>
<td>A</td>
</tr>
<tr>
<td>mkfs</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Create file system</td>
<td></td>
</tr>
<tr>
<td>mknod</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Create device file</td>
<td>A</td>
</tr>
<tr>
<td>more</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Control screen output</td>
<td>A</td>
</tr>
<tr>
<td>mount</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Mount file systems and remote resources</td>
<td></td>
</tr>
<tr>
<td>mountall</td>
<td>/sbin</td>
<td>scr</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Mount two or more file systems</td>
<td></td>
</tr>
<tr>
<td>mv</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Move or rename files</td>
<td>A</td>
</tr>
<tr>
<td>newgrp</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Modify group membership</td>
<td></td>
</tr>
<tr>
<td>nice</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Change priority of commands</td>
<td></td>
</tr>
<tr>
<td>nl</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Number text lines</td>
<td>S</td>
</tr>
<tr>
<td>nm</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Output an object file symbol table</td>
<td></td>
</tr>
<tr>
<td>noup</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Execute command and ignore signals</td>
<td></td>
</tr>
<tr>
<td>od</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Output file contents in octal format</td>
<td>S</td>
</tr>
<tr>
<td>paste</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Merge lines</td>
<td>S</td>
</tr>
<tr>
<td>patch</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Use difference report</td>
<td></td>
</tr>
<tr>
<td>patchchk</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Check path names</td>
<td></td>
</tr>
<tr>
<td>pax</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Process portable archives</td>
<td>A</td>
</tr>
</tbody>
</table>
## Command summary

**Working with the POSIX shell**

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Type</th>
<th>Delivery</th>
<th>Description</th>
<th>LFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdbl</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Administer private POSIX loader</td>
<td></td>
</tr>
<tr>
<td>pkginfo</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Display information about software packages</td>
<td></td>
</tr>
<tr>
<td>posdbi</td>
<td>/usr/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Administer POSIX loader</td>
<td></td>
</tr>
<tr>
<td>pr</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Format files and output to standard output</td>
<td></td>
</tr>
<tr>
<td>print</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Output mechanism similar to echo</td>
<td></td>
</tr>
<tr>
<td>printf</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Formatted output</td>
<td></td>
</tr>
<tr>
<td>ps</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Query process data</td>
<td></td>
</tr>
<tr>
<td>pwd</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Output path name of current working directory</td>
<td></td>
</tr>
<tr>
<td>rcp</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.INET</td>
<td>Copy files from or to a remote computer</td>
<td>A</td>
</tr>
<tr>
<td>read</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Read arguments of standard input and assign shell variables</td>
<td></td>
</tr>
<tr>
<td>readonly</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Protect shell variables</td>
<td></td>
</tr>
<tr>
<td>renice</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Change priority of current processes</td>
<td></td>
</tr>
<tr>
<td>rm</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Delete files</td>
<td>A</td>
</tr>
<tr>
<td>rmdir</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-B.C.SHELL</td>
<td>Delete directories</td>
<td>A</td>
</tr>
<tr>
<td>rmpart</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Remove partition</td>
<td></td>
</tr>
<tr>
<td>rsh</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.INET</td>
<td>Execute commands on remote computer</td>
<td></td>
</tr>
<tr>
<td>sed</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Editor in procedure mode</td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Set options or parameters, output variables</td>
<td></td>
</tr>
<tr>
<td>settle</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Set shell options or positional parameters</td>
<td></td>
</tr>
<tr>
<td>sh</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>POSIX shell command interpreter and programming language</td>
<td>A</td>
</tr>
<tr>
<td>shift</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Move values of positional parameters to the left</td>
<td></td>
</tr>
<tr>
<td>sleep</td>
<td>/sbin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Temporarily halt processes</td>
<td></td>
</tr>
<tr>
<td>show_pubset_export</td>
<td>/sbin</td>
<td>scr</td>
<td>SINLIB.POSIX.BC.ROOT</td>
<td>show file systems affected by EXPORT-PUBSET</td>
<td></td>
</tr>
<tr>
<td>sort</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Sort and/or merge files</td>
<td></td>
</tr>
<tr>
<td>split</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Distribute file across several files</td>
<td>A</td>
</tr>
<tr>
<td>start_bs2fsd</td>
<td>/sbin</td>
<td>scr</td>
<td>SINLIB.POSIX.BC.ROOT</td>
<td>start copy daemons</td>
<td></td>
</tr>
<tr>
<td>strings</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Find printable strings in object or binary files</td>
<td>S</td>
</tr>
<tr>
<td>styty</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX.BC.ROOT</td>
<td>Output or modify attributes of a data display station</td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Calculate checksum of a file</td>
<td>A</td>
</tr>
<tr>
<td>sync</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.B.CS.ROOT</td>
<td>Write back system cache</td>
<td></td>
</tr>
<tr>
<td>tabs</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Set tabulator stops</td>
<td></td>
</tr>
<tr>
<td>tail</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Output the last part of a file</td>
<td>A</td>
</tr>
<tr>
<td>talk</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Talk with another user</td>
<td></td>
</tr>
<tr>
<td>tar</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX.BC.SHELL</td>
<td>Archive files</td>
<td>A</td>
</tr>
<tr>
<td>Name</td>
<td>Location</td>
<td>Type</td>
<td>Delivery</td>
<td>Description</td>
<td>LFS</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>------</td>
<td>------------------</td>
<td>---------------------------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>tee</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Join pipes together and copy input</td>
<td></td>
</tr>
<tr>
<td>test</td>
<td>/usr/bin</td>
<td>blt+scrr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Check conditions</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Measure command runtime</td>
<td></td>
</tr>
<tr>
<td>times</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Output total runtime of processes started up to now</td>
<td></td>
</tr>
<tr>
<td>touch</td>
<td>/usr/bin</td>
<td>blt+bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Update modification and access times</td>
<td>A</td>
</tr>
<tr>
<td>tput</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Initialize terminal or query terminfo database</td>
<td></td>
</tr>
<tr>
<td>tr</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Replace or delete character</td>
<td>A</td>
</tr>
<tr>
<td>trap</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Modify signal handling</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>/usr/bin</td>
<td>alias+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Return end status 0</td>
<td></td>
</tr>
<tr>
<td>tsort</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Sort topologically</td>
<td></td>
</tr>
<tr>
<td>tty</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Output path name of current terminal</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>/usr/bin</td>
<td>alias+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Query command type</td>
<td></td>
</tr>
<tr>
<td>typeset</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Set attributes for shell variable</td>
<td></td>
</tr>
<tr>
<td>ulimit</td>
<td>/usr/bin</td>
<td>blt+scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Limit file size for writing or query current limit</td>
<td>A</td>
</tr>
<tr>
<td>umask</td>
<td>/usr/bin</td>
<td>blt-scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Output or modify default allocation of access rights</td>
<td></td>
</tr>
<tr>
<td>umount</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Unmount file systems and remote resources</td>
<td></td>
</tr>
<tr>
<td>umountall</td>
<td>/sbin</td>
<td>scr</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Unmount two or more file systems</td>
<td></td>
</tr>
<tr>
<td>unalias</td>
<td>/usr/bin</td>
<td>blt-scr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Delete variables from alias tables</td>
<td></td>
</tr>
<tr>
<td>uname</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Output basic data on the current operating system</td>
<td></td>
</tr>
<tr>
<td>unname</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Output basic data on the current operating system</td>
<td></td>
</tr>
<tr>
<td>uncompress</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Uncompress compressed files</td>
<td>A</td>
</tr>
<tr>
<td>unexpand</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Convert blanks to tab characters</td>
<td>S</td>
</tr>
<tr>
<td>uniq</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Search for repeated lines</td>
<td></td>
</tr>
<tr>
<td>uudecode</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Decode file after mailx transfer</td>
<td></td>
</tr>
<tr>
<td>uuencode</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Encode file for mailx transfer</td>
<td></td>
</tr>
<tr>
<td>uname</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>List names of system</td>
<td></td>
</tr>
<tr>
<td>usp</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-BC.ROOT</td>
<td>Dynamic setting of POSIX control parameters</td>
<td></td>
</tr>
<tr>
<td>vi</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Screen editor</td>
<td></td>
</tr>
<tr>
<td>wait</td>
<td>/usr/bin</td>
<td>blt+scrr</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Wait for termination of background processes</td>
<td></td>
</tr>
<tr>
<td>wc</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Count words, characters and lines</td>
<td>A</td>
</tr>
<tr>
<td>whence</td>
<td>-</td>
<td>blt</td>
<td>SINLIB.POSIX-BC.SHELL</td>
<td>Command localization</td>
<td></td>
</tr>
<tr>
<td>who</td>
<td>/sbin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Show active user IDs</td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Send message to a user</td>
<td></td>
</tr>
<tr>
<td>xargs</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Create argument list(s) and execute command</td>
<td></td>
</tr>
<tr>
<td>yacc</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Create parser</td>
<td></td>
</tr>
<tr>
<td>zcat</td>
<td>/usr/bin</td>
<td>bin</td>
<td>SINLIB.POSIX-SH</td>
<td>Output compressed files</td>
<td>A</td>
</tr>
</tbody>
</table>
2.4.2 Command summary according to functions

The following summary indicates the function-specific subdivision of POSIX commands. Since some commands can be allocated to more than one function, it is possible that names may be duplicated. For a detailed description of the commands in alphabetical order please refer to the chapter “Commands” on page 91.

The commands are allocated to the following main functions:

- Command interpreter
- Querying or modifying the user environment
- Managing and processing files and texts
  - outputting
  - processing
  - saving and archiving
  - compressing and restoring to uncompressed state
  - querying and modifying file properties
- modifying and managing the file system
  - modifying the file system
  - managing the file system
- Printing and print management
- Data display terminal
- Editors
- Auxiliary commands for shell procedures
- Reading, converting and outputting characters
- Querying and modifying user properties
- User administration
- Interuser communication
- Calendar functions and dates
- Arithmetic functions
- Compiler commands
- Program testing
- Job management
- Process information
- Process control
Working with the POSIX shell

Command summary

- Inter-process communication
- Terminal
- Checking storage availability
- Information on system data
- Online documentation
- Flushing the system buffer
- Calling BS2000 procedures
- Administer POSIX program cache
- Network commands
- NLS commands (Native Language System)

Command interpreter

sh  POSIX shell

Query or modify user environment

adduser  assign individual user IDs
cd  change working directory
env  set environment for command execution
id  return user identity
logname  display login name
ls  list directory contents
pwd  return working directory name
tty  output path name of current terminal
who  show who is on the system
Managing and processing files and texts

- output
  - cat: concatenate and print files
  - hd: hex dump
  - head: copy the first part of files
  - more: page through a text file
  - od: dump files in various formats
  - pr: prepare files for printing
  - strings: find printable strings in object or binary files
  - tail: copy the last part of a file
  - zcat: display compressed files in expanded form

- process
  - awk: pattern scanning and processing language
  - cksum: write file checksums and sizes
  - cmp: compare two files
  - comm: select or reject lines common to two sorted files
  - csplit: context split
  - cut: cut out selected fields of each line of a file
  - diff: differential file comparator
  - edt: process file with EDT (BS2000)
  - egrep: search a file with an ERE pattern
  - fgrep: search a file for a fixed-string pattern
  - find: find files
  - fold: filter for folding lines
  - grep: search a file for a pattern
  - join: relational database operator
  - nl: number lines
  - paste: merge corresponding or subsequent lines of files
  - patch: use diff list
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>sort, merge or sequence check text files</td>
</tr>
<tr>
<td>split</td>
<td>split a file into pieces</td>
</tr>
<tr>
<td>sum</td>
<td>print checksum and block count of a file</td>
</tr>
<tr>
<td>tr</td>
<td>transliterate characters</td>
</tr>
<tr>
<td>tsort</td>
<td>topological sort</td>
</tr>
<tr>
<td>uniq</td>
<td>report or filter out repeated lines in a file</td>
</tr>
<tr>
<td>wc</td>
<td>word, line and byte or character count</td>
</tr>
</tbody>
</table>

- **save and archive**
  - ar: manage libraries
  - cpio: copy in and out
  - dd: convert and copy files
  - iconv: codeset conversion
  - nm: convert and copy files
  - pax: portable archive processing
  - tar: file archiver

- **compressing and restoring to uncompressed state**
  - compress: compress files
  - uncompress: expand compressed files
  - zcat: display compressed files in expanded form

- **query and modify file properties**
  - chgrp: change file group ownership
  - chmod: change file modes
  - chown: change file ownership
  - file: check type of file
  - ls: list directory contents
  - touch: change file access and modification times
  - umask: get or set the file mode creation mask
### Modifying and managing the file system

- **modify file system**
  - bs2cp: copy BS2000 files
  - bs2file: set BS2000 file attributes
  - cp: copy files
  - csplit: context split
  - find: find files
  - fsexpand: Expand existing file systems
  - ftyp: determine processing mode for BS2000 files
  - ln: link files
  - make: maintain groups of files
  - mkdir: make directory
  - mv: move files
  - rm: remove directory entries
  - rmdir: remove directories
  - split: split a file into pieces

- **Manage file system**
  - dumpfs: dump file system
  - fsck: file system check
  - mkfifo: make FIFO special file
  - mknod: make an inode
  - mount: mount a file system
  - mountall: mount file systems
  - pathchk: check pathnames
  - show-pubset_export: show file systems affected by EXPORT-PUBSET
  - start_bs2fsd: start copy daemons
  - umount: unmount a file system
  - umountall: unmount file systems
Printing and print management

asa interpret carriage-control characters
bs2lp send files to a printer
cancel cancel requests to an LP print service
lp send requests to an LP print service
lpstat print information about the status of the LP print
pr prepare files for printing

Editors

ed interactive line editor
edt process file with EDT (BS2000)
ex command and display editor
sed stream editor
vi display editor (visual)

Auxiliary commands for shell procedures

basename return non-directory portion of path name
dirname return directory portion of path name
expr evaluate arguments as an expression
false return false value
getopts parse utility options
athchk check pathnames
sleep suspend execution for an interval
test evaluate expression
true return true value
xargs construct argument list(s) and execute command
[......] evaluate expression
**Command summary**

**Working with the POSIX shell**

**Reading, converting and outputting characters**
- **echo**: write arguments to standard output
- **hd**: hex dump
- **od**: dump files in various formats
- **print**: output mechanism similar to echo
- **printf**: formatted output
- **tee**: join pipes and make copies of input

**Querying and modifying user properties**
- **id**: return user identity
- **logname**: display login name
- **mesg**: permit or deny messages
- **newgrp**: change to a new group

**User administration**
- **who**: show who is on the system

**Interuser communication**
- **mailx**: interactive message processing system (mail extended)
- **mesg**: permit or deny messages
- **talk**: talk to another user
- **write**: write to another user

**Calendar functions and dates**
- **at**: execute commands at a later time
- **batch**: execute commands at a later time
- **cal**: print calendar
- **crontab**: regularly execute commands at specific times
- **date**: write the date and time
Arithmetic functions
bc arbitrary precision arithmetic language
expr evaluate arguments as an expression
let integer calculations

Compiler commands
lex generate programs for lexical tasks
yacc yet another compiler-compiler

Program testing
debug test POSIX programs

Job management
bg run jobs in the background
fg run jobs in the foreground
jobs display status of jobs in the current session

Process information
logger log messages
ps report process status
time time a simple command	
times write process times

Process control
at / batch execute commands at a later time
kill terminate or signal processes
nice run a command at low priority (be nice)
nohup run a command immune to hangups and quits
renice alter the scheduling priority of running processes
sleep suspend execution for an interval
wait await process completion
Inter-process communication
ipcrm  remove inter-process communication facilities
ipcs  inter-process communication status

Data display terminal
bs2pkey  set P keys
expand  convert tabs to spaces
stty  display or modify attributes of a data display station
tabs  set tabs on a terminal
tput  initialize a terminal or query the terminfo database
tty  output path name of current terminal
unexpand  convert spaces to tabs

Checking storage availability
df  report free disk space
du  estimate file space usage

Information on system data
info  Online diagnostic tool
pkginfo  Display informations about software packages
ps  report process status
uname  output basic data on the current operating system
who  show who is on the system

On-line pages
man  display on-line manual pages

Clearing the system buffer
sync  flush system buffer
**Calling BS2000 procedures**

bs2do  Calling BS2000 procedures from POSIX

**Administer POSIX program cache**

pdbl  Administer private POSIX loader

posdbl  Administer POSIX loader

**Networking commands**

rcp  remote file copy

rsh  remote shell

uudecode  decode file after mailing with *mailx*

uuencode  encode file for mailing with *mailx*

uuname  list names of UUCP systems

**NLS commands (Native Language System)**

gencat  generate a binary encoded message catalog

locale  call up information about the locale

localedef  define locale

**Subsystem administration**

usp  Dynamic setting of POSIX control parameters
2.5 Example

This section presents an example of operation using the POSIX shell. You log onto the BS2000, output the directory which corresponds to your user ID and start the POSIX shell. Once in the POSIX shell, you first create a .profile file in which you define alias variables to simplify operation and specify a new prompt which displays the current path at any given time as an aid to orientation. Once the .profile file has been executed, the definitions it contains take effect.

You then transfer a file from the BS2000 file system to the POSIX file system where you process it.

```
/set-logon-parameters user-id=user1,account=...
/show-file-attributes

% 114 :1OSN:$USER1.ANHANG.V2
%  3 :1OSN:$USER1.AVASQUER
%  78 :1OSN:$USER1.BIB.EXAMPLES.SDF
%   6 :1OSN:$USER1.DO.MSGCHECK
% 5007 :1OSN:$USER1.FS.USER1
%   3 :1OSN:$USER1.MSG.PROT
%   3 :1OSN:$USER1.OUTPU
%   3 :1OSN:$USER1.PROG.C
%   3 :1OSN:$USER1.SYS.SDF.LOGON.USERPROC
/start-posix-shell

POSIX Basisshell 07.0A41 created Jan 27 2009
POSIX Shell 07.0A41 created Jan 27 2009
Copyright (C) Fujitsu Technology Solutions 2009
All Rights reserved
$ edt .profile
```

(1) Log on to the BS2000 as usual.
(2) Use the BS2000 command SHOW-FILE-ATTRIBUTES to display the directory corresponding to your login name.
(3) Use the BS2000 command START-POSIX-SHELL to call the POSIX shell.
(4) You are accepted as a POSIX shell user.
(5) Create the .profile file with the POSIX editor edt. Since the file does not exist, edt creates a new file (see page 83).
Working with the POSIX shell

Example

1.00 alias ll='ls -l'
2.00 alias la='ls -al'
3.00 PS1='$PWD> ' 
4.00 5.00 6.00 7.00 8.00 9.00 10.00 11.00 12.00 13.00 14.00 15.00 16.00 17.00 18.00 19.00 20.00 21.00 22.00

POSIX editor ready for file .profile: new file

After creating the .profile file with ed and quitting the editor with the command return, you should evaluate the .profile file in the current shell. To do this, enter .profile.

The POSIX shell reports with the newly defined prompt which displays the current path /home/user1. You use the command for which the alias la has been defined to display all the files in the directory.

Change to the subdirectory c-source in which, for example, you store your C programs.
Example Working with the POSIX shell

(9) Copy the file `prog.c`, which is located in the BS2000 file system, to the POSIX file system. The file is written to the current directory, `c-source`.

(10) Use `cat` to output the contents of the file `prog.c`.

(11) Compile the file `prog.c` using the C compiler. You want the result of the compilation to be written to the file `prog`.

(12) Run the program `prog`. It outputs the string “hello world” on screen.

(13) Enter the command `exit` to terminate the POSIX shell.

(14) You may enter further BS2000 commands if you so wish.

(15) Log off from BS2000.
3 International environment (NLS locale)

3.1 Native language support

The most of the UNIX derivatives always used to be based on the ASCII coded character set, with American English as the language in which the user communicated with the computer. For the commercial market, however, it has always been essential to offer interactive programs communicating in the language of the program user (language in this sense being taken to include regional conventions such as currency formats); and as a result considerable extra effort was required to produce national variants of programs for other markets.

The increasing international popularity of UNIX systems has now made it necessary to enhance the systems to cater flexibly for the differing scripts, languages and cultural conventions of its users.

With NLS (the Native Language System) X/Open has defined an interface which makes it possible to

– develop applications which communicate with the user in various native languages and conform to the associated local customs. Programs of this type make no assumptions about the language of the user and keep the data specifications separate from the program logic. Hence they are referred to as internationalized programs.

– supply the correct native language and associated local customs to the runtime environment of an internationalized application. This technique is known as localization. Hence the set of conventions relating to local custom and language which applies in a given environment is referred to as the locale.

The tools which NLS provides for these purposes include:

– an announcement mechanism which enables users to identify their own native language, local custom and codeset requirements to programs at runtime. This mechanism works on the basis of environment variables.

– message catalogs which allow program messages to be held apart from the program logic, translated into other languages and bound to the application at runtime.
Native language support

– internationalized C library functions which make no assumptions about native language, territory and codeset and are thus capable of handling universal character classification, case conversion (up/downshifting), number format conversion and string collation (sorting).

– a set of C library functions which allow a personal language environment to be set, modified and disabled at program runtime.

All the POSIX commands described in this manual are 8-bit transparent. That includes all the command-line arguments and all the data and characters interpreted by the commands.

There are, however, still certain restrictions on the portability of 8-bit data between POSIX and other systems:

– Data interchange between systems over email links may be restricted to 7-bit data on account of the mail or networking protocols being used.

– 8-bit data and file names may only be portable to systems which comply with the X/Open system interface definition.

For detailed information on NLS refer to the “Programmer's Guide: Internationalization - Localization” [13].
3.2 Defining an internationalized environment

3.2.1 The personal internationalized environment

POSIX supports announcement mechanisms which allow settings for language, national conventions and character set to be defined both on a system-wide basis and for individual users or user groups.

You set your working environment by assigning the name of the required locale to a set of environment variables reserved for that purpose. Whenever an internationalized program is invoked, these environment variables are read and the information for the locale assigned to them is bound to the program’s runtime environment.

The environment variables which set the locale are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>affects:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LANG</td>
<td>Entire locale</td>
</tr>
<tr>
<td>LC_ALL</td>
<td>Entire locale</td>
</tr>
<tr>
<td>LC_CTYPE</td>
<td>Character classes and case conversion (shifting)</td>
</tr>
<tr>
<td>LC_COLLATE</td>
<td>Collating sequence</td>
</tr>
<tr>
<td>LC_TIME</td>
<td>Date and time formats</td>
</tr>
<tr>
<td>LC_MONETARY</td>
<td>Currency symbol and monetary value format</td>
</tr>
<tr>
<td>LC_NUMERIC</td>
<td>Representation of the radix character, exponent character and digit grouping symbol</td>
</tr>
<tr>
<td>LC_MESSAGES</td>
<td>Message texts and answers to yes/no questions</td>
</tr>
</tbody>
</table>

The precedence among these variables is as defined in section “Precedence among environment variables” on page 89.
Defining an internationalized environment

International environment (NLS locale)

These variables can be defined in one of the following formats:

Format 1: variable=locale-name

Format 2: variable=language[_territory][.codeset]

locale-name
language[_territory][.codeset]

name of a directory under /usr/lib/locale. The length of this string should not exceed \{NL_LANGMAX\}.

The LC_CTYPE, LC_COLLATE, LC_TIME, LC_MONETARY, LC_NUMERIC and LC_MESSAGES variables also support an additional \@modifier element specifying extra alternatives, for example a particular type of collating sequence:

LC_variable=language_territory.codeset@modifier

Example

In a German environment you can use \@modifier to choose between the standard collating sequence (dictionary order) and the collating sequence preferred in the German telephone directory. The collation rules here differ in respect of uppercase and lowercase, umlauts and special characters. To select the phone book collating sequence you use the assignment:

**LC_COLLATE=De_DE.88591@TE**

If any of the variables LC_CTYPE, LC_COLLATE, LC_TIME, LC_MONETARY, LC_NUMERIC or LC_MESSAGES is undefined or is assigned the null string, its value defaults to the value of LANG.

If any of the variables LC_CTYPE, LC_COLLATE, LC_TIME, LC_MONETARY, LC_NUMERIC and LC_MESSAGES has an invalid value, or if the LANG variable is undefined or null, the system acts as if it were not internationalized; in other words, it collates according to the ASCII table, uses the American date format, English day and month names and so on. This is the default setting.
3.2.2 Precedence among environment variables

Like the LANG variable, the LC_ALL environment variable provides a general announcement mechanism for the entire locale, and it uses the same syntax. Unlike LANG, LC_ALL has the top precedence, which means that it overrides all other international environment variables. Setting LC_ALL is sufficient if one preset locale satisfies all user requirements for program localization.

LANG has the lowest precedence; so if it is set, it is still possible to set other international environment variables to customize the working environment of individual users to meet their specific requirements.

A typical case not covered by either LANG or LC_ALL is where you want to communicate with the system in one language but sort text files, for example, in another. In cases of this type, you set the international environment variables which allow you to modify individual aspects (categories) of your locale.

Thus if you want to communicate with the system in British English but you also need to sort German text files, you should define the following settings for the LANG and LC_COLLATE environment variables:

`LANG=En`  
`LC_COLLATE=C`

By setting other variables you can set up a multi-language working environment.

Note that the LC_ALL environment variable must be left unset in such cases, as it overrides all other international environment variables. If LC_COLLATE and LC_ALL are both set, LC_COLLATE is ignored.
3.2.3 Supplied locales

POSIX provides different locales:

- C
- De
- De.EDF04
- De_DE.EDF04
- En_US.EDF04
- De.EDF04@euro
- De_DE.EDF04@EU
- POSIX

The “POSIX” or “C” locale is equivalent to the behavior defined in the XPG4 standard. The “De” locale is based on the German language, with sorting (collation) based on the standard sort order (as defined by Duden).

3.2.4 Restrictions

Commands which are not part of the XPG4 standard (see also page 875) are not internationalized.
4 Commands

adduser  add individual user number

After creating up a new BS2000 user, the POSIX administrator has to assign an individual
user number to the new user on systems which do not have SDF-P.

The `adduser` command can only be executed by the POSIX administrator, who must have
the group and user number 0.

**Syntax**

```
adduser ..user..uid
```

**user**

User ID for the new users. The user ID must be specified in uppercase letters and corre-
sponds to the BS2000 user ID. It must not be more than 8 bytes long.

**uid**

User number (≥100) assigned to the new user. The user number must not be the same
as the default value (see the BS2000 command SHOW-POSIX-USER-ATTRIBUTES
[1]).

The user numbers 0 to 99 inclusive are reserved for privileged users.

**See also**  `chmod`, `id`
alias define or display aliases

You can use the built-in alias command in the POSIX shell sh to redefine shell commands. However, you may not use it to redefine reserved words. You can use the alias command to define and export alias variables as well as to write them to standard output. The unalias command is used to delete alias variables. Exported alias variables remain effective for procedures which are called by name. However, they must be reinitialized for any new, explicit calls to the POSIX shell.

For the use of alias variables see the section “Aliasing” on page 45.

Syntax

Format 1: alias[\-t][\_name]
Format 2: alias[\-x][\_name=[value]]...

Format 1 alias[\-t][\_name]
No option specified

alias with no arguments writes the list of aliases in the form name=value to standard output.

\-t is used to set and list alias variables which possess a path specification (tracked alias). The value of this type of variable consists of the full path name as far as name. While changing the value of PATH invalidates the value definition, name retains the path specification. If option \-t is not set then the pair name=value is output for every name which has no associated value in the argument list.

\-t not specified:

alias outputs the pair name=value for every name which has no associated value in the argument list.

Format 2 alias[\-x][\_name=[value]]...

A space at the end of value causes the next word in the command line to be checked for alias substitution.

\-x is used to set and output exported alias variables. An exported alias variable is defined for procedures which are called by name.

Exit status

The exit status is non-zero if no value is defined for name.

Variable

PATH

Default variable for the POSIX shell. The absolute path name of the directories in which the POSIX shell is intended to search for commands is assigned to the variable.
Locale

The following environment variables affect the execution of `alias`:

- **LANG**: Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**: If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**: Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example

You can use the following aliases to simplify the `ls` command:

```bash
$ alias ll='ls -l'
$ alias la='ls -al'
```

See also `unalias`
**ar**

**archive maintainer**

*ar* is used to maintain archives (libraries).

In specific terms, you can use *ar* to perform the following file management functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Main keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create an archive file</td>
<td>-q or -r</td>
</tr>
<tr>
<td>Quickly append a file to the end of the archive</td>
<td>-q</td>
</tr>
<tr>
<td>Place/replace a file in the archive</td>
<td>-r</td>
</tr>
<tr>
<td>Delete a file from the archive</td>
<td>-d</td>
</tr>
<tr>
<td>Move a file within the archive</td>
<td>-m</td>
</tr>
<tr>
<td>Print the contents of a file</td>
<td>-p</td>
</tr>
<tr>
<td>List the files in the archive</td>
<td>-t</td>
</tr>
<tr>
<td>Copy (extract) a file from the archive</td>
<td>-x</td>
</tr>
<tr>
<td>Output symbols</td>
<td>-S</td>
</tr>
</tbody>
</table>

**Syntax**

```
ar[-V]..mainkey[modifier]...[posname]..afile[.filename]...
```

- **-V**  *ar* prints its version number to standard error output.

**Main keys**

When you call *ar*, you must specify precisely one main key (*-d, -m, -p, -q, -r, -t, -x or -S*), optionally followed by one or more arguments.

- **-d**  (*d - delete*) *ar* deletes the specified files from the archive. If no files are specified, no action is taken.

- **-m**  (*m - move*) *ar* moves the specified files within the archive.

  With *posname*:
  The files are placed after (*a*) or before (*b* or *i*) the file named *posname* (see *posname*).

  Without *posname*:
  The files are appended to the end of the archive.

- **-p**  (*p - print*) *ar* prints the contents of the specified files. If no files are specified, the contents of all files are printed.
-q  (q - quickly) *ar* adds the specified files to the archive "quickly", i.e.:
   - If the archive already exists, *ar* appends the named files to it without checking whether they are already present in the archive;
   - If the archive does not yet exist, it is created.
No *posname* arguments are permitted.
The -q option is useful when creating a large archive step by step.

-r  (r - replace) This option has three different effects, depending on whether the specified archive exists and whether it contains the named files:
   - If the archive exists and contains the files, *ar* replaces the named files.
   - If the archive exists but is missing one or more of the named files, *ar* adds the missing files to the archive.
   - If the archive does not exist, *ar* creates the archive from the named files.
With modifier *u*:
*ar* replaces a file in the archive only if the named file has a later date of last modification than the version already in the archive.

With *posname*:
Files not yet in the archive are placed after (a) or before (b or i) the file *posfile* (see "*posname* on page 96).

Without *posname*:
Files not yet in the archive are appended to the end of the archive.

-t  (t - table) *ar* prints a table of contents of the archive file. If no files are specified, all files in the archive are listed; otherwise, only the named files.

-x  (x - extract) *ar* copies (extracts) files from the archive. If no files are specified, all files in the archive are extracted. The archive itself remains unaltered.

-S  ((S – Symbols) *ar* lists all symbols of the specified files. If no files are specified, *ar* lists all symbols which are contained in the library. The library is not changed by this.

U22794-J-Z125-6-76  95
Key modifiers

**c** (c - create) The message that *ar* usually issues when creating an archive is suppressed.

**C** (C - Create) Extracted files are prevented from being overwritten by files of the same name in the file system. This option is useful when -T is also used, to prevent truncated filenames from replacing files with the same prefix in the file system.

**T** (T - truncate) Truncates the names of extracted files. When files are extracted from an archive, their archive names may be longer than the file system can support. By default an error message will be output and the file will not be extracted if the filename is too long.

**l** (l - local ) Can be specified, but is ignored.

**s** (s - symbol table) *ar* regenerates the archive symbol table, but only if new objects are included in the library or existing objects are replaced.

**u** (u - update) See main key -r.

**v** (v - verbose) *ar* reports each action it takes (adding files to the archive, moving them within the archive, extracting them from the archive, etc.). If *v* is used with the -t key, *ar* displays a detailed listing similar to that produced by the *ls -l* command (see *ls*).

posname

The *posname* argument specifies the position at which a file is inserted into the archive. *posname* may be:

**a**.posfile
*ar* inserts the file before the named *posfile*.

**b**.posfile
*i*.posfile
*ar* inserts the file before the named *posfile*.

*posfile* is the name of a file already in the archive.

**afile**
Name of the archive to be created or processed.

**file**
Name of the file to be listed, printed out, added to, extracted from, or moved within the archive.
More than one file may be named on the command line. If you specify the same file more than once when calling *ar* with the -q or -r key, *ar* will enter the named file into the archive as often as specified. If a number of identically named files are contained in the archive, and you use *ar* without specifying a position, the first file with the given name will be accessed (rather than the most recent or oldest file, for example).
Structure of an archive

An archive is a single file that combines several files into one. By convention, the name of an archive file ends with the suffix .a. The purpose of an archive file is to enable the collective maintenance of a group of related files. Archives make it easier to maintain files, since you will often only need to specify the archive file instead of individually listing all its elements.

The constituents of an archive are typically object modules that usually form part of the same program or suite of programs.

The magic string and the file headers used by ar consist of printable ASCII characters. If an archive is entirely composed of printable files, the entire archive is printable.

When ar creates an archive, it creates headers in a format that is portable across all machines.

If the archive contains at least one object file, ar creates and maintains an archive symbol table. This symbol table is used by the link editor ld to effect multiple passes over the archive in an efficient manner. The archive symbol table is stored in a specially named file that is always the first file in the archive. This file is never listed or printed out like other archive files and is not accessible to the user. Whenever the ar command is used to create or update the contents of an archive, the symbol table is rebuilt. The symbol table can also be rebuilt by calling ar with the -s option.

As far as possible the virtual main memory is used and the export of elements to temporary files is avoided. If this is not successful, temporary files are created, but this is only necessary in the case of the main option -m. The following priorities then apply for the storage location of the temporary files:

1. the current directory
2. the directory specified in the TMPDIR variable
3. the /tmp directory

Locale

The following environment variables affect the execution of ar:

LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).
**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**LC_TIME**
Determine the format of date and time strings in archive content listings produced in conjunction with the \( v \) modifier.

**NLSPATH**
Determine the location of message catalogs for the processing of \( LC_MESSAGES \).

**Example 1**
Quickly creating an archive file:

```bash
$ ar -qv afile.a atoi.o itoa.o
ar: creating afile.a
a - atoi.o
a - itoa.o
```

*ar* creates the archive *afile.a* from the files *atoi.o* and *itoa.o*. The \( v \) modifier tells *ar* to display the names of the files as they are added.

**Example 2**
Placing a new file at a specified position in an archive:

```bash
$ ar -rvb atoi.o afile.a atof.o
```

*ar* copies *atof.o* into the archive file *afile.a*, placing it before *atoi.o*.

**Example 3**
Printing the table of contents of an archive:

```bash
$ ar -tv afile.a
rw-r--r-- 104/     1      2276 Jul 13 12:17 2008 atof.o
rw-r--r-- 104/     1       759 Jul 13 12:17 2008 atoi.o
rw-r--r-- 104/     1      1280 Jul 13 12:17 2008 itoa.o
```

**Example 4**
Extracting (copying) a file from an archive:

```bash
$ ar -xv afile.a atoi.o
```

The file *atoi.o* is copied from the archive *afile.a* into the working directory. The copy is also called *atoi.o*.

See also *cpio*, *pax*, *tar*
asa interpret carriage-control characters

asa will write its input files to standard output, mapping carriage-control characters from the text files to line-printer control sequences. The first character of every line will be removed from the input, and the following actions will be performed:

If the character removed is:

<table>
<thead>
<tr>
<th>Character</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>The rest of the line will be output without change.</td>
</tr>
<tr>
<td>0</td>
<td>A newline character will be output, then the rest of the input line.</td>
</tr>
<tr>
<td>1</td>
<td>One or more characters that causes an advance to the next page will be output, followed by the rest of the input line.</td>
</tr>
<tr>
<td>+</td>
<td>The newline character of the previous line will be replaced with one or more characters that causes printing to return to column position 1, followed by the rest of the input line. If the + is the first character in the input, it will have the same effect as the space character.</td>
</tr>
</tbody>
</table>

Syntax

asa[_.file ...]

file
A pathname of a text file used for input.
If no file operands are specified, the standard input will be used.

Locale

The following environment variables affect the execution of asa:

LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH
Determine the location of message catalogs for the processing of LC_MESSAGES.

See also

lp
at

execute commands at a later time

The at command
- reads commands from standard input or from a shell script and executes them at a later time specified by the user. A command list created in this way runs under a job number. (Format 1 and 2)
- lists on standard output any jobs scheduled with at or batch (see batch) which have not yet been processed (Format 3 and 4)
- removes jobs previously scheduled with at or batch (Format 5).

The output from Format 1 and Format 2 is sent to the user by mailx unless the standard output and standard error output of the commands to be executed have been redirected.

The environment variables, the current directory, the permissions for new files (see page 778) and the maximum permissible file size (see page 775) are retained, but open files and priorities are lost, and the trap command (shell built-in for catching signals) is deactivated. at writes the job number and the schedule time to standard error.

Jobs scheduled with at are retained even if the user who scheduled them closes the POSIX shell with exit or if the POSIX subsystem is shut down. There is no need to reschedule the jobs.

Before the call

The user ID must have a standard account number for rlogin access. This standard account number can be assigned using the ADD-USER, MODIFY-USER-ATTRIBUTES or ADD-POSIX-USER command.

If the file /usr/lib/cron.d/at.allow exists, you can only use at if your login name appears in it.

If the file /usr/lib/cron.d/at.allow does not exist, you can only use at if your login name does not appear in the file /usr/lib/cron.d/at.deny.

If neither /usr/lib/cron.d/at.allow nor /usr/lib/cron.d/at.deny exists, only the POSIX administrator is allowed to use at.

If only an empty deny file exists, for example, everyone is allowed to use at.

Only the POSIX administrator is allowed to create and modify the allow/deny files. Each line in these files contains precisely one login name.

The cron daemon is started by means of an rc script.
Execute commands at a later time

Format 1: \texttt{at\ [-m][\ -f\ script][\ -q\ queue][\ -t\ time]}

\texttt{-f\ script}
\quad \texttt{at} reads the commands to be executed from the specified shell script.
\quad You can specify a number of commands, separated from one another by a semicolon
\quad (\;) or a newline character. A command list created in this way runs under a job number.
\quad You can exit the command list terminal-dependent with \texttt{@@d} or \texttt{[END]}.

\texttt{-f} not specified:
\quad \texttt{at} reads the commands to be executed from the standard input.

\texttt{-m} Sends mail to the user after a job has been completed, indicating that the job is finished.
\quad Mail is sent only if the job has not already generated a mail message.

\texttt{-q\ queue}
\quad The \texttt{-q} option is used to assign a job to a specific queue in the \texttt{/var/spool/cron} directory.
\quad The values accepted for \texttt{queue} are:
\quad a \quad for the default queue for jobs scheduled with \texttt{at}.
\quad b \quad for the default queue for jobs scheduled with \texttt{batch}.
\quad c \quad for the default queue for jobs scheduled with \texttt{crontab}.

\texttt{-t\ time}
\quad Specifies the execution time for the commands. \texttt{time} is specified as follows:
\quad \texttt{[[CC]YY]MMDDhhmm[.SS]}
\quad \texttt{CC} \quad The first two digits of a date (century).
\quad \texttt{CC} not specified:
\quad \quad If the two-digit date is greater than 68, the current century is assumed,
\quad \quad otherwise the following is used.
\quad \texttt{YY} \quad Two-digit year specification.
\quad \texttt{YY} not specified: The current year is assumed.
\quad \texttt{MM} \quad Two-digit month specification (01 bis 12)
DD  Two-digit day specification (01 bis 31)
hh  Two-digit hour specification (00 bis 23)
mm  Two-digit minute specification (00 bis 59)
SS  Two-digit second specification (00 bis 61)
The values 60 and 61 are intended for leap seconds.
SS not specified: The value 0 seconds is assumed.

Format 2  \texttt{at[-m][-f..script][-q..queue]..time}

The options are described under Format 1.

time  Specifies the execution time for the commands. \textit{time} is specified as follows:

\begin{verbatim}
time[..date][..+increment]
\end{verbatim}

time  digits[suffix] or special_name

digits:  \[h\]h  1- and 2-digit numbers are interpreted as hours.
         hhmm  4-digit numbers are interpreted as hours and minutes.
         [h][h][m]:[m]m  Digits separated by a colon are interpreted as hours
                        and minutes.

suffix:  \textit{am}  Interpreted as before 12 noon (dependent from the
                        locale \textit{LC\_TIME})
         \textit{pm}  Interpreted as after 12 noon (dependent from the
                        locale \textit{LC\_TIME})

am, pm omitted  Interpreted as 24-hour clock
         \textbf{zulu[am][pm]}  Interpreted as Greenwich Mean Time

special_name: \textbf{noon, midnight, now}

date  \textbf{month..day[.year] or}
       \textbf{weekday[..nextweek|next..week] or}
       \textbf{special..day}

month (dependent from the locale \textit{LC\_TIME}):

\begin{verbatim}
jan[uary],  feb[uary],  mar[ch],  apr[il],  may,  jun[e],  jul[y],  aug[ust],
sep[tember],  oct[ober],  nov[ember],  dec[ember],
nextmonth | next..month
\end{verbatim}

day  A number between 1 and 31, depending on how long the month is.
**at**

**year**  Number defining the year to which the date applies

`nextyear` | `next..year`: selects the following year

`year` not specified:
If the date given is after the current date, `at` assumes the next year;
otherwise, the current year.

**weekday** (dependent from the locale `LC_TIME`):

`mon[day]`, `tue[esday]`, `wed[nesday]`, `thu[r]sday`, `fri[day]`,
`sat[urday]`, `sun[day]`

`nextweek` | `next..week`: selects the following week

**special_day**: `today`, `tomorrow`, `nextday`, `next..day`

`nextday` (or `next..day`) means that the job is executed a full day later.
If the specified time lies before the current time, `at` interprets the following day
as the current day. For example, if the job `at 10 nextday` were scheduled at
11:00am on 7/1/95, the job would be executed at 10:00am on 7/3/95; however,
if `at 14 nextday` were specified at the same time, execution would begin at
14:00 hours on 7/2/95.

**date** not specified:
– corresponds to `today` if the specified time (rounded to the nearest minute) lies after
  the current time
– corresponds to `tomorrow` if the specified time (rounded to the nearest minute) lies
  after the current time
– corresponds to `now` if the specified time (rounded to the nearest minute) lies after
  the current time

`+increment`

`increment` is a positive integer that must be followed by one of the following units of
time:

`minute[s]`, `hour[s]`, `day[s]`, `week[s]`, `month[s]`, `year[s]`

**Examples**

`at` can typically be specified in the following ways:

- at 0815am jan 24
- at 8:15am jan 24
- at 5pm friday
- at now +1hour
at

List jobs yet processed

Format 3  \texttt{at\ldots-\textit{\ldots}[-\textit{\ldots}jobnumber]} \\

Format 4  \texttt{at\ldots-\textit{\ldots}-\textit{\ldots}queue} \\
\texttt{-I[-\textit{\ldots}jobnumber]} \\
\texttt{\textit{\ldots}at lists the specified jobnumber, if the corresponding job has not yet been processed. jobnumber is the number that is reported on standard error when a job is scheduled with at, batch or cron.} \\
\texttt{\textit{\ldots}jobnumber not specified: at lists all jobs that are yet to be processed, together with their job numbers.} \\
\texttt{-l\ldots-q\ldots\textit{\ldots}queue} \\
\texttt{It is possible to specify the queue as an alternative to the jobnumber. at lists all of the jobs in queue.}

Remove jobs

Format 5  \texttt{at\ldots-\textit{\ldots}jobnumber} \\
\texttt{-r\ldotsjobnumber} \\
\texttt{\textit{\ldots}at removes the job jobnumber, which was previously scheduled with at or batch. jobnumber is the number reported on standard error when a command job is scheduled with at or batch. You can specify more than one job number, using blanks to separate them. Only the POSIX administrator is authorized to remove another user's jobs.}

Exit status  0: \textit{at} executed successfully  \\
\neq 0: An error occurred while \textit{at} was executing

Error  The commonest error messages are: \\
\texttt{at: bad date specification} \\
\texttt{You have entered the date in an incorrect format.} \\
\texttt{at: too late} \\
\texttt{The date you specified has already passed.} \\
\texttt{at: This job may not be executed at the proper time time lies between "now" and "now+1hour".} \\
\texttt{at: you are not authorized to use at. Sorry.} \\
\texttt{You are not authorized to use at. Sorry. (See Before the call)
at

File

File

/usr/lib/cron/at.allow
List of login names with permission to use at. One login name is entered per line.

/usr/lib/cron/at.deny
List of login names explicitly denied permission to use at. One login name is entered per line.

/var/spool/cron/atjobs
Directory containing a separate file for each at job which has not yet been executed. Each at job is allocated a file of its own with the file name jobnumber.a.

Variable

SHELL
Determine a name of a command interpreter to be used to invoke the at job. If the variable is unset or null, sh will be used. If it is set to a value other than a name for sh, the implementation will do one of the following: use that shell; use sh; use the login shell from the user database.

TZ
Determine the timezone. The job will be submitted for execution at the time specified by timespec or -t time relative to the timezone specified by the TZ variable. If timespec specifies a timezone, it will override TZ. If timespec does not specify a timezone and TZ is unset or null, an unspecified default timezone will be used.

Locale

The following environment variables affect the execution of at:

LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

LC_TIME
Determine the format of date and time string.

NLSPATH
Determine the location of message catalogs for the processing of LC_MESSAGES.
Example  The current date and the string *April Fool!* are to be displayed on terminal tty013 at 9 o'clock in the morning on the 1st of April.

```bash
$ at 1pm apr 1
echo 'date":April Fool!" >> /dev/tty013
[END] or @@d
```

See also  *batch, crontab, date, kill, mailx, nice, ps, sh, sort*
awk  pattern scanning and processing language

awk is a programmable text manipulation system.
When you call awk you specify an awk program it is to execute and the files it is to process.
The actions defined in the program are then performed on the basis of the specified files.
awk does not alter its input files. The results of the actions it performs are by default written
on standard output.

awk offers the following advantages over text manipulation programs such as egrep and sed:

- awk operates on one record at a time. As with egrep and sed, an input record is defined
  as one line by default; but with awk you can change this setting and define some other
  unit of text as the record.
- Each input record is split into fields which can be accessed individually.
- A pattern (selection criterion) may be a condition defined by the logical combination of
  extended regular expressions and relational operators.
- You can program any actions that you require. awk is a high-level C-like programming
  language.

A detailed description of awk is provided below in the following sections:

- Typical awk applications (see page 110)
- Structure of an awk program (see page 112)
- Operation of the awk command (see page 113)
- The input file (records, fields, special variables) (see page 113)
- Basic elements of the awk language (comments, constants, variables) (see page 117)
- Expressions (see page 122)
- Patterns (see page 124)
- Actions (control-flow statements, functions) (see page 126).
awk

Syntax

**Format 1:** awk[...-F.ERE][...-v..initialization]...prog[...initialization]...[...file]...

**Format 2:** awk[...-F.ERE][...-f..progfile]...[...v..initialization]...[...initialization]...[...file]

**-F.ERE**

Defines the field separator character for the input record (input field separator).

**ERE**

Extended regular expression that defines a character to be interpreted as the input field separator. Separators do not form part of the fields.

---

To be able to use \( t \) as the input field separator, you must specify it as follows on the *awk* command line or in the BEGIN section of the *awk* program:

```bash
awk -F"[t]"... or BEGIN {FS="t"...}
```

- **-F.ERE** not specified:

Blanks and tabs act as field separators.

- **-v..initialization**

Assignments in the form `var=value`.

The `var` variable which appears in the program is initialized to `value`.

- **var**

Name of the variable to be initialized.

- **value**

Initial value to be assigned to `var`. `value` can be defined in exactly the same way as an environment variable on shell level.

There is no difference between the assignment of a value with `-v initialization` and with `initialization` (see below).

**prog**

*awk* program argument.

Possible forms for `prog` are:

- 'awk-program', i.e. an *awk* program written on the command line, or
- `-f progfile`, i.e. the name of a file containing an *awk* program.

'awk_program'

An *awk* program written on the command line.

You should always enclose the *awk* program in single quotes in order to prevent the shell from interpreting metacharacters. If the program is more than one line long, you must escape the newline character with a backslash.

**Example:**

Output all lines in the *input* file whose third field consists of the character '0'

```bash
$ awk '$3 == 0' input
```
-f progfile

The awk program is located in the file named progfile.

initialization

Assignments in the form: var=value

The var variable (whether it appears in the awk program or not) is initialized to value. initialization and file may be specified in any order. The assignment is made at the time when the named file is opened.

Thus, an assignment before the first file argument will be executed after the BEGIN actions (if any), while an assignment after the last file argument will occur before the END actions (if any).

Exception

The $ variables (see Basic elements) cannot be initialized in this way.

define -> value

var Name of the variable to be initialized. The name must not begin with $.

value Initial value to be assigned to var. value can be defined in exactly the same way as an environment variable on shell level.

file

Name of the text file to be processed. You may list more than one file if you wish. Files are read in the order in which they are listed. If file is a dash (-), awk reads from standard input.

file not specified:

awk reads from standard input. awk reads input one record at a time, processes it, and after each line outputs the result for that record. Hitting END or CTRL D or @@d terminates your input.
Typical awk applications

*awk* is a tool which makes text manipulation tasks easy to accomplish. Typical applications for *awk* include:

- selectively extracting data from files
- checking the contents of files
- performing calculations on the data in a file
- changing the format of input data.

Using four simple examples, this section demonstrates how *awk* can be used.

Example 1  A file called *supplies* contains a list of office supplies. It includes the name of each article, along with its quantity and unit price:

<table>
<thead>
<tr>
<th>Article</th>
<th>Quantity</th>
<th>Unit Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pencil</td>
<td>100</td>
<td>0.60</td>
</tr>
<tr>
<td>Table</td>
<td>5</td>
<td>345.00</td>
</tr>
<tr>
<td>Lamp</td>
<td>20</td>
<td>79.80</td>
</tr>
<tr>
<td>Paper</td>
<td>75</td>
<td>1.00</td>
</tr>
<tr>
<td>Diskette</td>
<td>1000</td>
<td>2.40</td>
</tr>
<tr>
<td>Envelope</td>
<td>1500</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Example 1 Select all articles with a quantity greater than 100:

```bash
$ awk '$2 > 100 {print}' supplies
Diskette   1000     2.40
Envelope   1500     0.20
```

With `$2` you access the second field of a line, which in this case is the quantity of each article. If the quantity is greater than 100, the condition is fulfilled, and the `print` function is executed. Since no arguments were specified for `print`, the whole line is output.

Example 2 Calculate the total price for all articles with a quantity greater than 100 and print this total along with the article name:

```bash
$ awk '$2 > 100 {print $1 "\t" $2*$3}' supplies
Diskette        2400
Envelope        300
```

Three arguments are entered for the `print` function in this example. The following is output:

- `$1`  article name (first field)
- `
`  tab character
- `$2*$3` quantity (second field) times unit price (third field)
Example 3  Include a heading in the output:

```
$ awk 'BEGIN    {print "Article \tTotal"}
   > $2 > 100 {print $1 "\t" $2*$3} supplies
 Article  Total
 Diskette  2400
 Envelope  300
```

This example illustrates the use of the BEGIN pattern. `awk` executes the action after BEGIN only once, i.e., when the program is started. The heading is therefore printed only once at the beginning.

Example 4  Print a grand total of all amounts at the end.

For this purpose we use a variable called `sum`, which is initialized to zero in the BEGIN pattern. The product of column 2 and column 3 is calculated for each line, and all the products are summed up:

```
$ awk 'BEGIN    {sum=0; print "Article \tTotal"}
   > $2 > 100 {print $1 "\t" $2*$3; sum += $2*$3}
   > END      {print "\nGrand total: " sum} supplies
 Article  Total
 Diskette  2400
 Envelope  300
 Grand total: 2700
```

This example demonstrates the use of the END pattern. `awk` executes the action after END only once, i.e., before termination of the program. The grand total of all subtotals is therefore printed just once at the end.
Structure of an awk program

An awk program can consist of a BEGIN section, a main section, and an END section, structured as shown below:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>BEGIN section</th>
<th>main section</th>
<th>END section</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ BEGIN {action} ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[[pattern] {action}]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ pattern [{action}]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ function_definition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ END {action} ]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

pattern
The pattern indicates which data is to be selected from the input files (see page 124).

action
The action indicates what to do with data that matches the pattern (see page 126).

function-definition
A function-definition enables you to define your own functions (see page 133).

At least one of the three sections (pattern, action or function-definition) must be present. In a pattern {action} pair, either the pattern or the action can be omitted. If the action is omitted, each line that matches the pattern is output; omitting the pattern causes the action to be performed on all lines.
The definition of a user-defined function may appear at any position in the main section. Each of the following ust be located at the start of a line (following any number of blanks or tabs):

- the BEGIN section
- the {pattern}{action} and pattern [{action}] pairs
- the function definitions
- the END section.
Operation of the awk command

`awk` executes the `awk` program that is specified by the user, proceeding in the following sequence:

1. Initial processing
   The first step performed by `awk` is to initialize any variables that may have been defined. If there is a `BEGIN` section including an `action`, `awk` then executes the action specified there. The action in the `BEGIN` section is executed just once, before the first line is processed.

2. File processing
   Next `awk` processes the specified input files by reading the input records sequentially. For each input record, `awk` tries to match each pattern in the order that is specified in the `awk` program. If a pattern is matched, i.e. the selection criterion is fulfilled, the associated action is performed. If no pattern is specified for an action, `awk` performs the action for every record. If no action is specified for a pattern, the default action is to output (print) the record. Multiple input files are processed in the specified order.

3. Final processing
   When all the specified files have been processed, `awk` performs the action in the `END` section, if one has been included. `awk` then exits.

The input file

An input file consists of records that are subdivided into fields.

- Records
  Records are separated by a record separator. The record separator does not form part of a record. By default, a record is one line, and the record separator is the newline character. However, you do have the option of changing this setup by assigning any single character to the special variable `RS` (Record Separator). If you specify a string of characters as a value for `RS`, only the first character will be taken into account. The ordinal number of the current record is available in the variable `NR` (Number of Record). If there is more than one input file, `NR` counts from the start of the first file to the end of the last one. The special variable `$0` addresses the whole of the current record. Further information on variables is provided in the section “Basic elements of the awk language” on page 117.
awk, Input file

– Fields

Each record is split into fields separated by one or more field separators. The default field separator is white space (any sequence of tabs and blanks), but you do have the option of changing this by assigning any other character to the special variable \texttt{FS} (Field Separator). You can make this assignment either in the \texttt{awk} program or by using option \texttt{-F} on the command line. The value assigned to \texttt{FS} is interpreted as an extended regular expression (see section “Regular POSIX shell expressions” on page 877).

Example 1  To define the characters \texttt{x} and \texttt{y} as alternate field separators:

\begin{verbatim}
syntax on the \texttt{awk} command line: \texttt{-F[xy]}
syntax in the \texttt{awk} program: \texttt{FS=[xy]}
\end{verbatim}

Example 2  To define the field separator as one or more occurrences of the character \texttt{x}:

\begin{verbatim}
syntax on the \texttt{awk} command line: \texttt{-Fx+}
syntax in the \texttt{awk} program: \texttt{FS=x+}
\end{verbatim}

The default setting (any sequence of blanks and tabs) can be expressed by the regular expression \texttt{[ \t\-]+}, where \texttt{\-} stands for a blank, and \texttt{\t} represents a tab.
Note that the newline character is always interpreted as a field separator, regardless of the value assigned to \texttt{FS}!

The number of fields in the current record is stored in the variable \texttt{NF} (Number of Fields). Individual fields of the current record are addressed by the predefined variables \$1, \$2, to \$NF. Further information on variables is provided in the section “Basic elements of the \texttt{awk} language” on page 117.

Example  Default setup

\begin{verbatim}
Field 1  Field 2   ...         Field 5 ...
This     is        the first   record             <--- Record 1
and      this      is the      second record.     <--- Record 2
\end{verbatim}

Customized setup: \texttt{RS=":\%; FS=\"; :\;}

\begin{verbatim}
Field 1  Field 2       Field 3
%Name  : Address      : Phone number      <--- Record 1
%SNI AG :81730 Munich : 089-636-1         <--- Record 2
\end{verbatim}
Rules for record and field separators

- Default settings for record separators
  - The default record separator is the newline character.
  - If the null string is assigned to RS (RS=""), the file is treated as a single record. If several files are specified, each file will consist of a single record (which means that the ultimate value of NR will be equal to the number of files).

- Default settings for field separators
  - If the record separator is newline, the field separator defaults to blanks and tabs.
  - If the record separator is not a newline, the newline character always counts as a field separator, regardless of which character has been explicitly defined as the field separator (see example 2 on page 114).
  - If you explicitly assign a blank to FS, either with -F " " on the awk command line or by using the assignment FS=" ", then blanks and tab characters are treated as field separators.
  - On the other hand, if you explicitly assign the tab character to FS (FS="\t"), then only the tab character is treated as the field separator and not the blank.

- Leading field separators and field separator strings
  - The following applies to blanks, tabs and newlines as field separators:
    - Leading field separators are ignored.
    - Multiple occurrences of a field separator are treated as a single field separator (see example 9 on page 147).
  - For all other field separators, leading field separators are counted. In multiple occurrences of a field separator, each character is counted separately. Thus two consecutive field separators are deemed to have an empty field between them (see example 10 on page 148).

- Changing separators:
  If you need a number of different record separators in one file, you can change RS within the awk program. The new record separator comes into effect as soon as the assignment to RS has been implemented. Similarly, you can change FS within the awk program, should you require a number of different field separators in one file. The new field separator comes into effect as soon as the assignment to FS has been implemented.
**awk, Input file**

**Special variables for the input file**

The following table contains all special *awk* variables pertaining to the input file. The value *awk* usually assigns to these variables is indicated in the second column.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value set by <em>awk</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>FILENAME</td>
<td>Name of the current input file, - for standard input</td>
</tr>
<tr>
<td>FS</td>
<td>Input field separator (default: any sequence of blanks and tabs)</td>
</tr>
<tr>
<td>NF</td>
<td>Number of fields in the current record</td>
</tr>
<tr>
<td>NR</td>
<td>Ordinal number of the current record from start of input</td>
</tr>
<tr>
<td>FNR</td>
<td>Ordinal number of the current record in the current file</td>
</tr>
<tr>
<td>RS</td>
<td>Input record separator (default: newline)</td>
</tr>
<tr>
<td>$0</td>
<td>Current record</td>
</tr>
<tr>
<td>$1</td>
<td>First field of the current record</td>
</tr>
<tr>
<td>$2</td>
<td>Second field of the current record</td>
</tr>
<tr>
<td>...</td>
<td>Last field of the current record</td>
</tr>
<tr>
<td>$NF</td>
<td>Last field of the current record</td>
</tr>
</tbody>
</table>

You can change these variables within an *awk* program if you wish. This does not alter the input file. Further information on variables is provided in the section “Basic elements of the *awk* language” on page 117.
Basic elements of the awk language

This section gives a syntax of the basic elements of the awk language. You will need these elements in order to define pattern and action pairs.

Comments
You can include comments in an awk program, as in a shell script. A comment begins with the # character and continues till the end of the line.

Constants
There are two types of constant:

number
A number (numeric constant) is a signed or unsigned integer or floating point number. awk does not check its format. If your number contains invalid characters, awk attempts to filter out a valid part and ignores the rest.

integer
An integer is a sequence of digits from 0 to 9.

floating point number
A floating point number consists of a mantissa with or without an exponent. The mantissa comprises an integer with or without a fractional part. The fractional part is represented by a radix character and an integer.

string
A string (alphanumeric constant) is a sequence of characters, enclosed in double quotes "...". If the double quotes are omitted, awk will interpret the string as a variable name, a number, or an operator.

character
A single character is also enclosed in double quotes "..." in order to prevent awk interpreting the character as a variable name. A character may be a displayable character from the character set which is currently in use (see section “EDF04 character set” on page 893) or one of the following special characters as represented in C:

<table>
<thead>
<tr>
<th>Char</th>
<th>Equivalent in C</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>for &quot;</td>
</tr>
<tr>
<td>\</td>
<td>for \</td>
</tr>
<tr>
<td>\a</td>
<td>for bell character</td>
</tr>
<tr>
<td>\n</td>
<td>for newline character</td>
</tr>
<tr>
<td>\t</td>
<td>for tab character</td>
</tr>
<tr>
<td>\v</td>
<td>for vertical tab</td>
</tr>
<tr>
<td>\b</td>
<td>for backspace</td>
</tr>
<tr>
<td>\r</td>
<td>for carriage return</td>
</tr>
<tr>
<td>\f</td>
<td>for page feed</td>
</tr>
</tbody>
</table>
Variables

`awk` allows you to use simple variables and arrays to store values. The special variables are predefined; others can be defined by the user.

Name of a variable

The name of a user-defined variable can be any string made up of underscores (_), uppercase and lowercase letters and digits, beginning with a letter or an underscore.

Data type

Variables do not have a data type. You can thus assign either a number or a string to any variable. If the context is clearly numeric, variables are treated as numeric; otherwise, they default to alphanumeric. Example:

```awk
x = "Miller";  # Variable x contains the string Miller
x = "3"+4 ;    # Variable x has a value of 7
```

Declaration

`awk` variables do not need to be explicitly declared. User-defined variables are automatically declared the first time they are used.

Initialization

Special variables are initialized to predefined values by `awk`. Depending on the context, user-defined variables are initialized by `awk` to the null string or to 0 by default. If you wish, you can specify other initial values when you call `awk`.

Exceptions:
- When i>NF, $i will not always be the null string.
- $ variables cannot be initialized on the command line.
**Special variables**

`awk` recognizes the special variables listed in the table below. The values `awk` usually assigns to these variables are indicated in the table. New values may be assigned to the variables by the user.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value set by <code>awk</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGV</td>
<td>Array holding the command line arguments (excluding options and the <code>prog</code> argument), numbered from 0 to ARGC-1</td>
</tr>
<tr>
<td>ARGV</td>
<td>Array holding the command line arguments (excluding options and the <code>prog</code> argument), numbered from 0 to ARGC-1</td>
</tr>
<tr>
<td>ENVIRON</td>
<td>Array holding the values of environment variables, where the indexes are the names of the variables</td>
</tr>
<tr>
<td>FILENAME</td>
<td>Name of the current input file, - for standard input</td>
</tr>
<tr>
<td>FS</td>
<td>Input field separator (default: any sequence of blanks and tabs)</td>
</tr>
<tr>
<td>NF</td>
<td>Number of fields in the current record</td>
</tr>
<tr>
<td>NR</td>
<td>Ordinal number of the current record from start of input</td>
</tr>
<tr>
<td>FNR</td>
<td>Ordinal number of the current record in the current file</td>
</tr>
<tr>
<td>OFS</td>
<td>Output field separator (default: one blank)</td>
</tr>
<tr>
<td>ORS</td>
<td>Output record separator (default: newline)</td>
</tr>
<tr>
<td>OFMT</td>
<td>Output format for floating point numbers (see page 140) (default: %.6g, up to 6 places after the decimal point)</td>
</tr>
<tr>
<td>RS</td>
<td>Input record separator (default: newline)</td>
</tr>
<tr>
<td>RLENGTH</td>
<td>Length of the string matched by the <code>match</code> function</td>
</tr>
<tr>
<td>RSTART</td>
<td>Starting position of the string matched by the <code>match</code> function. Numbering begins with 1. This value always corresponds to the value returned by the <code>match</code> function.</td>
</tr>
<tr>
<td>SUBSEP</td>
<td>Subscript string separator for multi-dimensional arrays. The default setting is 034.</td>
</tr>
<tr>
<td>$0</td>
<td>Current record</td>
</tr>
<tr>
<td>$n</td>
<td>Field $n$ of the current record</td>
</tr>
<tr>
<td>$NF</td>
<td>Last field of the current record</td>
</tr>
</tbody>
</table>
What is the effect of changing special variables?

Example

The assignment

```bash
$1 = "new";
```

assigns the string `new` to `$1`; but this does not actually alter the first field of the current input record.

This also applies to the following `awk` settings relating to the input file:

1. The current input file does not change when you assign a new name to `FILENAME`.
2. When you assign a value to a variable `$i` where `$i` is greater than `NF`, `NF` is assigned the value of `$i`.
3. If you assign a new value to `NR`, you only alter the number assigned to the current line; you do not move to a different line.

Example

The contents of `$0` remain the same even if `NR` is modified:

```bash
{print NR, $0; NR=NR+34; print NR, $0}
```

A typical output would then be:

```
10 This is the tenth line
44 This is the tenth line
```

When you assign a new value to a variable, its old value is deleted. Thus, if you change `NF`, for example, the information on the number of fields in the current record is lost.

Peculiarity of `$` variables:

You can specify the number of a `$` variable as a constant or as an expression which evaluates to the number.

Example

You can use `$(NF-1)` to access the second-last field.
Array
An array is a set of constants or variables.
An array element is addressed as follows:

Syntax
array_name[index]

array_name
Name of a variable.

index
A simple variable.
The index may be numeric or alphanumeric. The index you specify can therefore be a number, a string, or an expression that evaluates to an index value.

awk provides two special types of arrays:

- Dynamic arrays
  Arrays, like simple variables, do not need to be declared. Above all, there is no need to define dimensions. New array elements are created automatically as and when required.

- Associative arrays
  Individual array elements can be accessed via an alphanumeric index.
  A special control-flow statement is provided in order to process all elements of an associative array:
  
  for (index in array) statement

  index assumes the index values present to this point in random order, and the specified statement is executed once for each array element (see control-flow statement `for`).

Example
A file called expenses contains various expenses incurred. For each item of expenditure the file shows the date, month, amount, and a brief description, with a colon to separate them. For example:

```
01:January: 40.78:Supplies
05:January: 6789.00:Laser printer
23:March: 240.32:Lamps
11:January: 478.00:Chairs
01:February: 45.00:Journals
```

Using an associative array you can easily calculate total expenditure for each month from the data in this file. The program in the example uses an array called mexpenses and the names of the months as an alphanumeric index. For each line, the expenses in the third field ($3$) are summed up to produce total expenditure for each month appearing in the second field ($2$).
Expressions

An expression can be any of the following:

- constant
- variable
- function_call
- un_op expression
- expression bin_op expression
- (expression)
- expression ? expression : expression

constant
Numeric or alphanumeric constant (see “Basic elements of the awk language” on page 117).

variable
Variable (see “Basic elements of the awk language” on page 117).

function_call
Invocation of a predefined function (see page 133).

expression
Expression.

un_op
Unary operator (see “awk operators” on page 123).

bin_op
Binary operator (see “awk operators” on page 123).

Expressions are evaluated and return a value. They may appear both in patterns and in actions.
**awk, Expressions**

**awk operators**

`awk` recognizes all C operators plus the operators for pattern matching and string concatenation. The following table lists all `awk` operators in ascending order of precedence. Operators in the same line have the same precedence.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>assignment operator</td>
</tr>
<tr>
<td>+=  -=  *=  /=  %=</td>
<td>compound assignment operators as in C</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td>~ !~</td>
<td>pattern matching operators</td>
</tr>
<tr>
<td>&gt; &gt;= &lt; &lt;= != ==</td>
<td>relational operators</td>
</tr>
<tr>
<td>operand list</td>
<td>concatenation</td>
</tr>
<tr>
<td>+ -</td>
<td>plus, minus</td>
</tr>
<tr>
<td>* / %</td>
<td>multiply, divide, remainder</td>
</tr>
<tr>
<td>!</td>
<td>logical NOT</td>
</tr>
<tr>
<td>^ **</td>
<td>exponent</td>
</tr>
<tr>
<td>++ --</td>
<td>increment, decrement</td>
</tr>
</tbody>
</table>

**Evaluation of expressions**

Since no data type is prescribed for the operands, you can freely mix numeric and alphanumeric constants. `awk` determines from the context whether a numeric or alphanumeric operation is required. Please note that, as in C, there are no special truth values. Like C, `awk` treats a value of 0 as false and a non-zero value as true. This means that any non-zero value as an argument of a logical operation is held to be true. If the result of a logical operation is true, it is represented as 1.

**Example** \((2\&\&2)+3=4\)
**Patterns**

Patterns (selection criteria) are specified by the user as a means of indicating which data is to be selected from the input files. A pattern can have any of the following forms:

**Syntax**

- `/regexp/`
- `relexp`
- `matchexp`
- `pattern_range`
- `compound_pattern`

`/regexp/` - Regular expression

*awk* supports extended regular expressions (see section “Regular POSIX shell expressions” on page 877). A regular expression is enclosed in slashes `/.../`.

**Example**

A regular expression matching any number of occurrences of a, b or c:

```
/[abc]+/
```

**compare relexp**

`relexp` is an expression (see page 122) featuring relational operators. The operators and their meanings are:

- `a > b`  
  *a* greater than *b*?
- `a >= b`  
  *a* greater than or equal to *b*?
- `a < b`  
  *a* less than *b*?
- `a <= b`  
  *a* less than or equal to *b*?
- `a == b`  
  *a* equal to *b*?
- `a != b`  
  *a* not equal to *b*?

Operands *a* and *b* are any expressions. If both operands are numeric, the comparison is numeric; if not, it is alphanumeric.

**matchexp**

`matchexp` is an expression (see page 122) featuring pattern matching operators. It involves the comparison of a regular expression (pattern) with a string. The pattern matching operators and their meanings are:

- `str ~ p`  
  String *str* must match pattern *p*
- `str !~ p`  
  String *str* must not match pattern *p*

Using `matchexp` as a pattern allows you to select individual fields.
Example  Select all records with a first field starting with A or a:

\$1 ~ /^\[Aa\]/

The regular expression ^[Aa] represents strings that begin with A or a. The first field of the
record ($1) must match (~) the regular expression, i.e. begin with A or a.

pattern_range

A pattern range takes the form:

regexp, regexp

Specifying a range causes the associated action to be executed for all records that lie
within the range. The limits of the range (start and end) are defined by two regular
expressions. The range begins with the first record containing a string that matches the
first regular expression and ends with the first record containing a string that matches
the second regular expression.

Example  Select the range from the first line beginning with C to the first line beginning with K and
output the first field of every line in the selected range:

/^C/, /^K/  {print $1}

compound_pattern

Logical operators (see Expressions) can be used to negate patterns and to combine
several of them to form a single pattern. The logical operators and their meanings are:

!pat  Negation of pattern pat
pat1 Il pat2  pat1 or pat2.
The criterion is satisfied if pat1 or pat2 matches.
pat1 && pat2  pat1 and pat2.
The criterion is satisfied if both pat1 and pat2 match.

(pat)  Parentheses

A compound condition is evaluated from left to right.

Example  Match all records that have an even number of fields and a letter between M (inclusive) and
Q (exclusive) in the first field.

NF%2==0 && $1 >= "M" && $1 < "Q"
You can generally combine patterns in several ways in order to make the same selection. Thus, if the currently valid collating sequence defines the range [M-Q] as the uppercase letters M, N, O, P and Q, the above selection could also be made with pattern matching operators:

\[\text{NF%2==0 && $1 \sim /^[MNOP]/}\]

Since the first `awk` condition depends on the collating sequence of the currently valid character set, it may not return the same result in every case. The second `awk` line, by contrast, will always select only those records in which the first field begins with the letter M, N, O or P.

**Actions**

Actions indicate what to do when a pattern is matched. An action will typically involve processing one of the selected files. An action has to begin in the same line as the associated pattern. If this is not possible, the newline character must be escaped with a backslash. Blanks and tabs between the action and the pattern are ignored.

An action comprises one or more statements and must be enclosed in braces `{...}` as shown below:

**Syntax**

`{statement...;[statement]...}`

**Statements**

A statement can be any of the following:

- expression
- control_statement

**expression**

An expression is evaluated but is not put to any further use unless `expression` is in the form of an assignment, an increment or a decrement (see section “Expressions” on page 122).

**control_statement**

A `control_statement` allows you to control the flow of an `awk` program (see section “Control-flow statements” on page 127).

A single statement may be spread over several lines, in which case each line except the last must end with a backslash. The backslash escapes (cancels the effect of) the newline character.

**Multiple statements**

You can group together a number of statements within one pair of braces `{}`. Statements are delimited by means of:

- a semicolon `;`
- a right brace `}`
- a newline character.
Control-flow statements

Control-flow statements allow you to control the flow of an `awk` program. `awk` recognizes the following control-flow statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>terminate a loop</td>
</tr>
<tr>
<td>continue</td>
<td>skip remainder of loop</td>
</tr>
<tr>
<td>exit</td>
<td>terminate the <code>awk</code> program</td>
</tr>
<tr>
<td>for</td>
<td>loop counter and looping an array</td>
</tr>
<tr>
<td>if</td>
<td>conditional statement</td>
</tr>
<tr>
<td>next</td>
<td>skip to the next input record</td>
</tr>
<tr>
<td>while</td>
<td>execute iteratively</td>
</tr>
<tr>
<td>do</td>
<td>execute iteratively</td>
</tr>
<tr>
<td>delete array[i]</td>
<td>delete element $i$ of the named <code>array</code></td>
</tr>
<tr>
<td>return x</td>
<td>return from a function with a value</td>
</tr>
<tr>
<td>return</td>
<td>return from a function without a value</td>
</tr>
</tbody>
</table>

The control-flow statements are described below in alphabetical order.

**break - Terminate a loop**

`break` can be used in the body of a `for`, `while`, or `do` loop. `break` causes an immediate exit from the enclosing loop.

**Syntax**

`break`

**Example**

While records continue to start with a dot, keep reading in the next record. Terminate the loop if the second field of the retrieved record is greater than 1000.

```awk
{ while($1 ~ /^\./)
  {
    getline;
    if($2 > 1000) break;
  }
}
```
**continue - Skip remainder of loop**

`continue` can be used in the body of a `for`, `while` or `do` loop. The `continue` statement causes the current iteration to be terminated and the next one to begin.

**Syntax**

```
continue
```

**Example**

Print even fields only:

```
{i=
 i=1;
 while(i++ <= NF)
   |
     if(i%2) continue;
     else print $i
 }
```

**do - Execute iteratively**

The statement in a `do` loop (or a `do-while` loop) is executed iteratively while a specified condition continues to be satisfied. In contrast to the `while` loop, the statement in a `do` loop is always executed at least once.

**Syntax**

```
do...[anweisung]...while...(expression)
```

**statement**

Statement that is executed in each iteration of the loop. If several statements are to be executed, they have to be grouped together in braces (`{ }`) and separated by semicolons or linefeed characters.

**expression**

Expression (see page 122) that specifies the condition.

**Example**

Print out the individual fields of a record:

```
{ i=0; do {print $(++i)} while (i != NF) }
```
**exit - Terminate the awk program**

*exit* terminates the *awk* program.

If an END section is present, *awk* executes the action specified in it; if not, the program is terminated immediately.

**Syntax**

```
exit
```

**Example**

If the commercial at symbol @ appears in the input, print the result and terminate processing:

```
... 
/@/ {exit}
... 
END {print ergebnis}
```

**for - Loop counter**

The statement in a *for* loop is executed iteratively while a condition continues to be satisfied.

**Syntax**

```
for(expr1; expr2; expr3) statement
```

**expr1**

Expression (see page 122).

*expr1* is evaluated once at the start of the *for* statement. *expr1* is often used to initialize incrementing variables.

Example:

```
i=1
```

**expr2**

Expression (see page 122).

*expr2* is evaluated before each iteration. The specified *statement* is executed only if *expr2* is non-zero (true); otherwise, the loop is terminated.

Example:

```
i<10
```

**expr3**

Expression (see page 122).

*expr3* is evaluated after each iteration. When incrementing variables are used, *expr3* increments the variable.

Example:

```
i++
```

**statement**

Statement that is executed in each iteration of the loop. If several statements are to be executed, they have to be grouped together in braces {}.

**Example**

Print out the fields of the current record in reverse order.

```
{for(i=NF; i>0; i--) print $i}
```
for - Looping an array
This variant of the for statement is a special awk facility for the handling of arrays.

Syntax
```
for(index in array) statement
```

index
Variable (see Basic elements) that assumes all values of the elements of array in random order. The index can be numeric or alphanumeric.

array
Array to be processed.

statement
Statement to be executed for each array element. If several statements are to be executed, they have to be grouped together in braces {}.

Example
The array named month contains the number of days in each month. Each array element is subscripted with the name of the month, e.g.

```
month["January"] = 31;
```

The following awk program prints the name of each month together with the number of days in it.

```
$ awk ' BEGIN { month["January"] = 31;  
> month["February"] = 28; 
> month["March"] = 31;    
> month["April"] = 30;  
> month["May"] = 31;      
> month["June"] = 30;    
> month["July"] = 31;     
> month["August"] = 31 }  
> END { for(i in month) print i," has ",month[i]," days" } '
```

May has 31 days
August has 31 days
July has 31 days
April has 30 days
June has 30 days
January has 31 days
March has 31 days
February has 28 days
awk, Control-flow statements

if - Conditional statement
The statement in an if construct is executed if the specified condition is satisfied.

Syntax
```
if(expr)...statement1...[else...statement2]
```

expr
Expression (see page 122) that defines the condition to be satisfied. If expr is non-zero (true), statement1 is executed.

statement1
Statement to be executed if expr is true. If several statements are to be executed, they have to be grouped together in braces { }.

statement2
Statement to be executed if expr is false. If several statements are to be executed, they have to be grouped together in braces { }.

Example
If field 1 is greater than field 2, fields 2 and 3 are printed; if not, fields 4 and 5 are printed:
```
{ if($1 > 2) print $2, $3; else print $4, $5 }
```

next - Skip to the next input record
The next statement causes awk to suspend processing of the current record; statements that follow next are not applied to the current record. awk then reads the next input record. NR, NF, FNR, $0, and $1 to $NF are reset.

Difference between next and the getline function:
getline sets the current record to the next one. Statements that follow getline are executed using the next record’s values for the $ variables and for NR, NF, and FNR.

Syntax
```
next
```

Example
Records that begin with a dot are ignored:
```
{ if ($1 ~ /^\./) next }
```
**while - Execute iteratively**

The statement in a `while` loop is executed iteratively while a specified condition continues to be satisfied.

**Syntax**

```
while(expr)...statement
```

- `expr` Expression (see page 122) that specifies the condition.
- `statement` Statement that is executed in each iteration of the loop. If several statements are to be executed, they have to be grouped together in braces `{ }`.

**Example**

Print all input fields, writing each field in a separate output line:

```awk
{ i = 1;
  while (i <= NF) {
    print $i
    i++
  }
}
```
Functions

*awk* provides a wide range of built-in functions and also offers you the option of defining functions of your own:

The `{statements}` may be preceded by a newline character. There may also be blank lines within the braces `{...}`. A function definition has the same precedence as *pattern {action}* pairs in the main section of an *awk* program.

Within an action section, function calls can be entered anywhere in an expression, except before the function declaration. There must be no space between the function name and the left parenthesis when a function is called. Nested and recursive function calls are legal.

Though most functions do not require you to enclose arguments in parentheses, it is a good practice to use them as a means of increasing program transparency. When you pass an array as an argument, a pointer to the array is passed (call by reference), which means that you can change the elements of the array from the function. In the case of scalar variables, the value of the variable is copied and passed (call by value), which means that you cannot change the value of the variable from the function. The scope of function arguments is restricted to the local function, whereas the scope of all other variables is always global. If you need a local variable in a function, define it at the end of the argument list in the function definition. Any variable in the argument list for which no current argument exists is a local variable with a predefined value of 0.

As in C, some functions return a result (e.g. *exp*), while others are procedural in character (e.g. output functions). The *return* statement can be used with or without a return value or may be omitted entirely. In the latter case, the return value would be undefined if it were to be accessed.

Example

In the example below, the function named *search* looks for the string *who* in the array *allnames* and returns the index or -1. The third argument, *incr*, is used as a local variable.

```awk
... { print $1, search($1, allnames) } ...
function search(who, allnames, incr)
{
  for (incr=0; allnames[incr]; incr++)
    if (index(allnames[incr], who) == 1
      && length(allnames[incr]) == length(who))
      return incr
  return -1
}
```
### awk, Functions

#### Built-in functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input function</strong></td>
<td></td>
</tr>
<tr>
<td>getline</td>
<td>Read input record</td>
</tr>
<tr>
<td><strong>Output functions</strong></td>
<td></td>
</tr>
<tr>
<td>print([arg,...])</td>
<td>Standard output function</td>
</tr>
<tr>
<td>printf(format [arg,...])</td>
<td>Formatted output</td>
</tr>
<tr>
<td><strong>Arithmetic functions</strong></td>
<td></td>
</tr>
<tr>
<td>atan2(y,x)</td>
<td>Arc tangent of ( y/x )</td>
</tr>
<tr>
<td>cos(x)</td>
<td>Cosine</td>
</tr>
<tr>
<td>exp(x)</td>
<td>Exponential function</td>
</tr>
<tr>
<td>int(x)</td>
<td>Truncate to integer</td>
</tr>
<tr>
<td>log(x)</td>
<td>Natural logarithm</td>
</tr>
<tr>
<td>rand()</td>
<td>Return a random number</td>
</tr>
<tr>
<td>sin(x)</td>
<td>Sine</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Square root</td>
</tr>
<tr>
<td>srand([x])</td>
<td>Set the seed (initial value) for rand()</td>
</tr>
<tr>
<td><strong>String functions</strong></td>
<td></td>
</tr>
<tr>
<td>gsub(re, repl[, instr])</td>
<td>Global substitution function</td>
</tr>
<tr>
<td>index(str1, str2)</td>
<td>Return first occurrence of substring</td>
</tr>
<tr>
<td>length([str])</td>
<td>Return length of string</td>
</tr>
<tr>
<td>match(str, re)</td>
<td>Check whether string ( str ) matches regular expression</td>
</tr>
<tr>
<td>split(str, array[, sep])</td>
<td>Subdivide string</td>
</tr>
<tr>
<td>sprintf(format, e1, e2,...)</td>
<td>Return formatted output as string</td>
</tr>
<tr>
<td>sub(re, repl[, instr])</td>
<td>Substitution function</td>
</tr>
<tr>
<td>substr(str, m[, n])</td>
<td>Define substring</td>
</tr>
<tr>
<td>tolower(s)</td>
<td>Convert to lowercase</td>
</tr>
<tr>
<td>toupper(s)</td>
<td>Convert to uppercase</td>
</tr>
<tr>
<td><strong>General functions</strong></td>
<td></td>
</tr>
<tr>
<td>close(expr)</td>
<td>Close file or pipe</td>
</tr>
<tr>
<td>system(expr)</td>
<td>Call shell command</td>
</tr>
</tbody>
</table>
The following section describes each of these functions in alphabetical order together with the associated arguments. The argument you specify can either be a constant or an expression (see page 122). awk first evaluates the expression arguments and then applies the function to the computed results.

**atan2 - Arc tangent**

atan2 calculates the arc tangent of the quotient of two numbers. atan2(y,x) returns the arc tangent of y/x.

**Syntax**

```plaintext
atan2(y,x)
```

**y,x** Numbers that produce the quotient for which the arc tangent is to be calculated.

**close - Close file or pipe**

close closes the specified file or pipe.

**Syntax**

```plaintext
close(expr)
```

**expr**

Name of the file or pipe to be closed, see redirection under the section "printf - Formatted output" on page 140.

**cos - Cosine**

cos calculates the cosine of a number.

**Syntax**

```plaintext
cos(x)
```

**x** Number for which the cosine is to be calculated.

**exp - Exponential function**

exp calculates $e$ to the power of $x$.

**Syntax**

```plaintext
exp(x)
```

**x** Number for which $e^x$ is to be computed.
**getline - Read a record**

`awk` retrieves a record as directed (see also the control-flow statement `next` on page 131).

`getline` has several different formats, with the following return values:

1   successful execution
0   end-of-file
-1  error

**Syntax**

```
getline
```

`awk` reads the next input record from the input file into `$0`. `NR`, `NF`, `FNR`, `$0`, and `$1` to `$NF` are reset.

**Example**

If a record contains `%%%`, the next record is read. In other words, input records containing `%%%` are ignored.

```
/%%%/ {getline}
```

**Syntax**

```
getline...<file
```

`awk` reads a record from the named `file` into `$0`. `NF`, `$0`, and `$1` to `$NF` are reset.

**file**

Name of the file from which a record is to be read.

**Syntax**

```
getline...var
```

`awk` fetches the next input record from the input file and puts it into the variable `var`. `NR` and `FNR` are reset.

**var**

Variable into which the next record is to be read.

**Syntax**

```
getline...var...<file
```

`awk` fetches a record from the named `file` and puts it into the variable `var`. `NR`, `NF`, `FNR`, `$0`, and `$1` to `$NF` remain unchanged.

**var**

Variable into which the record is to be read.

**file**

Name of the file from which the record is to be read.
awk, Functions

Syntax

```
command...|getline...[var]
```

The output of the named `command` is redirected to `getline`. Each `getline` call in this format causes `awk` to read the next line from the output of `command` and write it into `$0` or the variable `var`. If `var` is specified, `NR`, `NF`, `FNR`, `$0`, and `$1` to `$NF` remain unchanged; if not, `NF`, `$0`, and `$1` to `$NF` are reset.

This construct is equivalent to calling the C function `popen()` with mode `r`.

```
var
  Variable into which the record is to be written.
  `var` not specified: The record is written into `$0`.
```

```
command
  Name of the command whose output is to be read.
```

gsub - Global substitution function

gsub globally substitutes the string `repl` for all strings in `$0` or `instr` that match the extended regular expression `RE`.

gsub returns the number of substitutions

Syntax

```
gsub(re,repl[,instr])
```

```
re
  Extended regular expression that specifies the pattern to be matched.
```

```
repl
  String to be substituted for the strings that match `re`.
```

```
instr
  String in which the substitution is to be made.
  `instr` not specified: Substitution is done in `$0`.
```

index - Search for substrings

`index` searches for a substring within a string. If the substring is present, `index` returns the starting character position (numbered from 1 onward) of its first occurrence in the string; if not, it returns a value of 0.

Syntax

```
index(str1,str2)
```

```
str1
  String in which `index` looks for the substring.
```

```
str2
  Substring that `index` looks for.
```
awk, Functions

Example: Comparing the string "ToTo-LoTo" with "To"

\[ \text{index('ToTo-LoTo','To')} \text{ returns } 1. \]

**int - Truncate to integer**

\[ \text{int} \text{ returns the largest integer equal to or smaller than the argument.} \]

Syntax:

\[ \text{int}(x) \]

\[ x \] Number that is to be truncated to its integer part.

**length - Return length**

\[ \text{length} \text{ returns the length of a string.} \]

Syntax:

\[ \text{length}[(\text{str})] \]

\[ \text{str} \] length returns the length of string \text{str}.

\[ \text{str not specified:} \]

\[ \text{length} \text{ returns the length of the current input record } \text{$0$.} \]

**log - Logarithm**

\[ \text{log} \text{ calculates the natural base } e \text{ logarithm.} \]

Syntax:

\[ \text{log}(x) \]

\[ x \] Number whose natural log is to be computed.

**match - Match regular expressions**

\[ \text{match} \text{ checks whether a string in } \text{str} \text{ matches the extended regular expression in } \text{re}. \]

If a matching string is found, \text{match} \text{ returns the character position in } \text{str} \text{ (numbered from 1 onward) at which the string begins; if not, it returns 0.} \]

The variable \text{RSTART} is set to the return value of \text{match}; \text{RLENGTH} is set to the length of the matching string (or -1 if no matching string is found).

Syntax:

\[ \text{match}(\text{str},\text{re}) \]

\[ \text{str} \] String in which the pattern is to be matched.

\[ \text{re} \] Extended regular expression.
**print - Standard output function**

`print` is the standard output function. `print` outputs either the current record or the specified arguments and terminates its output with the output record separator `ORS`. For further details refer to page 140.

### Syntax

```plaintext
print([arg1[,arg2]...])[redirection]
```

- **No argument specified:**
  - `print` writes the current input record on standard output.

- **Arguments that are to be printed:**
  - `print` evaluates the expression arguments and concatenates the results in the order in which the arguments are specified.

- **Arguments that are to be printed:**
  - `print` outputs the evaluated expression arguments in the specified order, separated by the output field separator `OFS` if they are separated by commas in the `print` statement.

### Redirection

Output can be redirected to a file or piped to a program. You can use up to 10 output files.

- **redirection** can be in the form of:

  - `>`, `>>`, name of program
  - `| prog`

- **>`
  - The output is written to the named file. The former contents of file are deleted the first time `print` is called. All subsequent `print` or `printf` outputs to file in the same awk program are appended to the end of file. Unless explicitly closed, file remains open until the end of the awk program.

- **`>> file`
  - The output is appended to the previous contents of file. Unless explicitly closed, file remains open until the end of the awk program.

- **`| prog`
  - The output is piped to the program named prog.

You are only permitted to open one pipe to prog within an awk program, but you can pipe any number of `print` or `printf` outputs to it. This construct is equivalent to calling the C function `popen()` [4] with mode `w`. Unless explicitly closed, the pipe remains open until the end of the awk program.

The file or program name can specified directly (enclosed in "...") or via a variable that evaluates to the file name.
Caution!
If you redirect output to the input file, the input file will be destroyed without any warning!

Output format

`print` outputs integers in decimal and prints strings at full length. Apart from that, the output format is contingent on the following predefined variables:

- **OFS** - output field separator
  - *OFS* is one space by default. If you wish, you can assign any one character to *OFS* to change the output field separator.

- **ORS** - output record separator
  - *ORS* is the newline character by default. If you wish, you can assign any one character to *ORS* to change the output record separator.

- **OFMT** - floating point output format
  - *OFMT* defines the output format for floating point values and is set to "%.6g" by default. This means that the fractional part of a floating point number is printed with a maximum of 6 places. If you wish, you can assign a different `printf` format for floating point numbers to *OFMT* (see "printf - Formatted output" below).

**Example**
Print the first and second fields, separated by a blank:

```
{print $1,$2}
```

**Example**
Concatenate the first and second fields without an output field separator:

```
{print $1$2}
```
or

```
{print $1 $2}
```

**printf - Formatted output**

`printf` is the output function for formatted output. The output format can be specified as in the standard `printf()` function in C.

**Syntax**

```
printf(format,arg,...)[redirection]
```

**Format**
String defining the output format. The output format comprises plain characters and format elements (conversion specifications). Printable characters are output unaltered. The special characters listed in the "Basic elements" section are converted immediately. For example, \n sets the position to the start of the next line.
All format elements begin with the percent sign. The most common format elements are presented in the following table:

<table>
<thead>
<tr>
<th>Format element</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%c</td>
<td>single character</td>
</tr>
<tr>
<td>%d</td>
<td>decimal integer</td>
</tr>
<tr>
<td>%e</td>
<td>floating point number in exponential notation, e.g. 5.234e+2</td>
</tr>
<tr>
<td>%f</td>
<td>floating point number, e.g. 52.34</td>
</tr>
<tr>
<td>%g</td>
<td>%e or %f, whichever is shorter</td>
</tr>
<tr>
<td>%o</td>
<td>octal integer (base 8)</td>
</tr>
<tr>
<td>%s</td>
<td>character string</td>
</tr>
<tr>
<td>%u</td>
<td>unsigned decimal integer</td>
</tr>
<tr>
<td>%x</td>
<td>hexadecimal integer (base 16)</td>
</tr>
</tbody>
</table>

arg
Arguments that are to be printed.
printf evaluates the expression arguments, allocates them in the given order to the specifications in format, and outputs them in the appropriate format.
- If the format element is incompatible with the argument, e.g. a numeric format specification for an alphanumeric argument, a 0 is printed.
- If there are more arguments than format elements, the excess arguments are ignored, i.e. not printed.
- If there are more format elements than arguments, an error message is issued.

redirection
Redirection is as for printf.
redirection not specified:
printf prints on standard output.

Example
Field 1 is printed as a decimal number with at least 2 positions, followed by ** as a separator, followed by field 2 as a string of at least 5 characters, followed by newline:

```c
{ printf("%2d**%5s
", $1,$2) }
```

**rand** - Return a random number

`rand` returns a random number `r`, where `0 <= r < 1`.

Syntax
`rand`
awk, Functions

**sin - Sine**

`sin` returns the sine of a number.

**Syntax**

```
sin(x)
```

- `x`: Number whose sine is to be computed.

**split - Subdivide strings**

`split` divides a string into substrings and stores each substring as an element in an array. The elements are subscripted in ascending order, starting with 1. `split` returns the number of array elements.

**Syntax**

```
split(str,array[,sep])
```

- `str`: String that is to be split.
- `array`: Name of the resulting array.
- `sep`: Extended regular expression specifying the characters that act as a separator between the substrings in `str`.
  - `sep` not specified: `FS` is used as the separator.

**Example**

The input:

```
{s=split("january:february:march", months, ":");
for(i=1; i<s; i++) print months[i];
}
```

produces the output

```
january
february
march
```

**sprintf - Return formatted output as a string**

`sprintf` formats in exactly the same way as `printf`, but there is no direct output. `sprintf` instead returns the formatted output as a string, which could then be assigned to a variable or used for a similar purpose.
awk, Functions

**Syntax**

```plaintext
sprintf(format,arg,...)
```

**format**
String defining the output format (see "printf - Formatted output" on page 140).

**arg**
Arguments that are to be output (see "printf - Formatted output" on page 140).

**Example**
The following awk program fragment produces the same output as the example given under `printf`.

```plaintext
{x = sprintf("%2d**%5s\n", $1,$2); print x }
```

**sqrt - Calculate the square root**

`sqrt` calculates the square root of a number.

**Syntax**

```plaintext
sqrt(x)
```

**x**
Number whose square root is to be computed.

**srand - Set the seed for the rand function**

`srand` sets the seed (starting point) for the `rand` function to the number `x`, or to the current time if no argument is specified.

**Syntax**

```plaintext
srand([x])
```

**x**
Number that is to serve as the seed for `rand`.

**sub - Substitution function**

`sub` substitutes the string `repl` for the first instance of a string in `$0` or `instr` that matches the extended regular expression `RE`.

`sub` returns the number of substitutions.

**Syntax**

```plaintext
sub(re,repl[,instr])
```

**re**
Extended regular expression that specifies the pattern to be matched.

**repl**
String to be substituted for the strings that match `re`.

**instr**
String in which the substitution is to be made.

```
instr not specified:
The substitution is done in `$0`.
```
**awk, Functions**

**substr - Define a substring**

`substr` extracts a substring from a string.

**Syntax**

```bash
substr(str,m[,n])
```

- **str**
  - String from which the substring is to be extracted.

- **m**
  - Position in `str` at which the substring begins. Character positions are numbered consecutively from left to right, starting with one.

- **n**
  - Maximum length of the substring.
  - `n` not specified: The substring extends to the end of `str`.

**Example**

The input

```bash
{x = substr("060789",3,2); print 'Month = 'x}
```

produces the output:

```
Month = 07
```

**system - Call shell command**

`system` executes the specified shell command and returns its exit status.

**Syntax**

```bash
system(command)
```

- **command**
  - Name of the shell command to be executed.

**Error**

If an `awk` program contains errors, `awk` issues corresponding error messages and exits immediately. The error messages indicate the cause of the error, if detectable by `awk`, and the `awk` program line in which `awk` thinks the error is to be found. Typical error messages are:

- `awk: syntax error at source line xxx`
- `Line xxx of the awk program contains a syntax error.`
- `awk: illegal statement source line number xxx`
- `Line xxx of the awk program contains an illegal statement.`
Locale

The following environment variables affect the execution of `awk`:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_COLLATE**
Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions and in comparisons of string values.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments) and input files, the behavior of character classes within regular expressions, the identification of characters as letters, and the mapping of upper- and lower-case characters for the `toupper` and `tolower` functions.

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**LC_NUMERIC**
Determine the representation of the radix character, the exponentiation symbol and the digit grouping character.

**NLSPATH**
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**awk EXAMPLES**

**Example 1**
Output all input lines in which field 3 is greater than field 5:

```
$ awk '$3 > $5' file
```

Since no action has been specified, `awk` prints the selected lines by default.

**Example 2**
Print every 10th line of a file:

```
$ awk '(NR % 10) == 0' file
```

**Example 3**
Print the second to last and the last field in each line, separated by a colon:

```
$ awk 'BEGIN {OFS=":\"} \\
>    {print $(NF-1), $NF}' file
```

If a line consists of a single field, the entire line is output twice, separated by a colon (first `$0`, then `$1`).
awk, Examples

Example 4  Add up the values of the first field of every line and print the total and average at the end:

```bash
$ awk '{s += $1} > END {print "Total: ", s, "Average: ", s/NR}' file
```

Example 5  Find a preprocessor if directive, i.e. a range of lines in which the first line begins with #if and the last line with endif:

```bash
$ awk '/^#if/, /^#endif/' file
```

Example 6  Print all lines in which the first field differs from that of the previous line:

```bash
$ awk '$1 != prev { print; prev = $1 }' file
```

Example 7  file contains a list of data about young people, with the second field containing one of the entries school, university, apprenticeship or elsewhere. For statistical purposes, you want to count how many are at school and university:

```bash
$ awk '$2 ~ /school/ {incr["school"]++} > $2 ~ /university/ {incr["university"]++} > END {print "school: ", incr["school"];} > print "university: ", incr["university"]} file
```

Example 8  The file contents lists the table of contents of a text. The table of contents is organized in decimal classification and has the format:

1. Foreword  
2. Introduction  
3. The Game of Chess  
3.1. History  
3.2. Rules  
3.2.1 Setting Up the Figures  
4. The Game of Checkers/Draughts  
4.1. History  
8. Index  

The following awk program can be used to give the list a more orderly format:

```bash
$ awk '($1=$1 " ":) > $1=substr($1,1,6); > print $0' contents >> con.form
```
The output lines are prepared in the following stages:

First, six blanks are added to the end of the first field ($1=$1 \ldots \ldots \ldots \ldots$). Then the first field is truncated to six characters. Thus the first field of each line is 6 characters long, and field 2 always starts at column 7. The output in the file `con.form` will be as follows:

1. Foreword
2. Introduction
3. The Game of Chess
   3.1. History
   3.2. Rules
   3.2.1 Setting Up the Figures
   .
   .
4. The Game of Checkers/Draughts
   4.1. History
   .
   .
8. Index

Example 9 The following `awk` program in the file `prog` prints the number of fields and the actual fields of each record. The record separator has been redefined as the dollar sign. The field separators are thus blanks, tabs, and the newline character:

```
BEGIN { RS='$'; printf "Record\tNum" }
{ printf ("%4d\t%3d\t", NR, NF);
  for(i=1;i<=NF; i++) printf "%s:\t", $i }
END { print"
"
}
```

The file `text` contains the following text:

```
first record$: second record: $ 
fourth and last record: $
```

The call:

```
$ awk -f prog text
```

returns:

```
Record Num
1 2 first:record: 
2 2 second:record: 
3 0 
4 4 fourth:and:last:record: 
5 0 
```
Example 10 You now change the file text to:

```
&&
first&&record$second record$$fourth
and&
last
record&
```

and call `awk` again, this time using the `-F` option to change the field separator to `&`.

```
$ awk -F"&" -f prog text
```

The output returned is:

```
Record  Num  Field  Field  Field
  1    6  ::::first::record:
  2    1  second record:
  3    0
  4    8  fourth:and::last::record:::
```

This example illustrates how fields are separated when a non-standard separator is used. The first line (&&) of the text file is a part of the first record and now yields 3 fields, for example, because each individual separator in a string of separators (&&) is counted, and the newline implicitly acts as a separator as well (2 & + 1 newline = 3).

See also  `egrep`, `fgrep`, `grep`, `lex`, `sed`
basename  return non-directory portion of path name

You can use basename to

– extract the basic file name (basename) from the full path name,
– strip any suffixes from the file name.

basename strips all characters up to and including the last / from the specified string and writes the result to standard output. The basic file name can thus be separated from its path prefix. If you also specify a string suffix as a command-line argument, basename will strip this suffix as well. basename is useful in shell scripts.

Syntax

```
basename[...string[...suffix]]
```

string

string can be any character string. basename deletes all characters up to and including the last / from string and writes the result to standard output. Strings that do not include a slash are output unmodified.

string not specified:

A period (dot) is written to standard output.

suffix

suffix can be any character string.

If the specified suffix matches the end of string, string is output without suffix.

Locale

The following environment variables affect the execution of basename:

LANG

Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL

If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH

Determine the location of message catalogs for the processing of LC_MESSAGES.
basename

Example 1  The name prog is to be generated from /home/catherine/program:

$ basename /home/catherine/program ramm
prog

Example 2  The following shell script compiles a C source program. basename generates the name of the compiled program from the file name used as a command-line argument for the shell script. The compiled program is stored in an executable file in the current working directory. The shell script is called compile.

Contents of compile:

c89 -o `basename $1 .c` $1

If you call compile as follows:

$ compile /home/anna/cprogs/tab.c

then the name of the C source file is passed to the command c89 (control program for the compiling and linking of C programs, see c89 [5]) in the positional parameter $1.

The shell replaces the operand for the c89 option -o with the result of the basename call. The name of the executable file is tab.

See also  dirname, ed
**batch**  execute commands at a later time

*batch* reads commands from standard input, puts them in a queue, and executes them when system load level permits.

Standard output and standard error output of the commands to be executed are sent to the user by `mailx` unless they are redirected elsewhere. The environment variables, the current directory, the permissions for new files (see section "umask get or set the file mode creation mask" on page 778) and the maximum permissible file size (see section "ulimit set or report file size limit" on page 775) are retained, but open files and priorities are lost, and the `trap` command (shell built-in for catching signals) is deactivated.

*batch* writes the job number and the schedule time to standard error.

Jobs scheduled with *batch* are retained even if the user who scheduled them closes the POSIX shell with `exit` or if the POSIX subsystem is shut down. There is no need to reschedule the jobs.

*batch* has exactly the same effect as `at -qb` with no further options.

**Before the call**

The user ID must have a standard account number for `rlogin` access. This standard account number can be assigned using the ADD-USER, MODIFY-USER-ATTRIBUTES or ADD-POSIX-USER command.

If the file `/usr/lib/cron.d/at.allow` exists, you can only use *batch* if your login name appears in it.

If the file `/usr/lib/cron.d/at.allow` does not exist, you can only use *batch* if your login name does not appear in the file `/usr/lib/cron.d/at.deny`.

If neither `/usr/lib/cron.d/at.allow` nor `/usr/lib/cron.d/at.deny` exists, only the POSIX administrator is allowed to use *batch*.

If only an empty deny file exists, for example, everyone is allowed to use *batch*.

Only the POSIX administrator is allowed to create and modify the allow and deny files. Each line in these files contains precisely one login name.

**Syntax**

```plaintext
batch
command ...
[END] or @@d
```

**command**

Any command or shell script. You can specify more than one `command` at a time, using semicolons or newlines to separate them. The resulting command list is executed under a single job number.
batch

at: you are not authorized to use at. Sorry.
Permission to use batch denied (see Before the call above).

File

/usr/lib/cron.d/at.allow
List of login names with permission to use batch. One login name is entered per line.

/usr/lib/cron.d/at.deny
List of login names explicitly denied permission to use batch. One login name is entered per line.

/var/spool/cron/atjobs
Directory containing a separate file for each batch job which has not yet been executed.
Each batch job is allocated a file of its own with the file name jobnumber.b.

Variable

SHELL
Determine a name of a command interpreter to be used to invoke the at job. If the variable
is unset or null, sh will be used. If it is set to a value other than a name for sh, the implementa-
tion will do one of the following: use that shell; use sh; use the login shell from the user
database.

TZ
Determine the timezone. The job will be submitted for execution at the time specified be
timespec or -t time relative to the timezone specified by the TZ variable. If timespec specifies
a timezone, it will override TZ. If timespec does not specify a timezone and TZ is unset or null,
an unspecified default timezone will be used.

Locale
The following environment variables affect the execution of batch:

LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implemen-
tation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the
variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other inter-
nationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data
as characters (for example, single- as opposed to multi-byte characters in
arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents
of diagnostic messages written to standard error.

LC_TIME
Determine the format and contents for date and time strings written by batch.
**batch**

**NLSPATH**  Determine the location of message catalogs for the processing of
**LC_MESSAGES**.

**Example**  In the following example, the standard input is redirected, and `batch` takes its work from the
file `jobs`:

```
$ batch < jobs
    job 604763316.b at Mon Mar  9 14:48:36 2009
```

The jobs contained in the file `jobs` are run in sequential order as background processes by
`batch`. When the job is complete, you can have the result displayed on the screen by `mailx`.

**See also**  `at`, `crontab`, `mailx`, `ulimit`
You can use \texttt{bc} to perform arithmetical calculations. \texttt{bc} is an interactive program for a C-like input language.

### Syntax

\texttt{bc}\[\text{-l}\][\text{...file...}]

- \texttt{-l} stands for /usr/lib/lib.b, which is a library containing \texttt{bc} programs for various mathematical functions. The \texttt{-l} option must be specified if you want to use any of the following functions:

- \texttt{s(x)} sine
- \texttt{c(x)} cosine
- \texttt{e(x)} base e exponential function
- \texttt{l(x)} natural logarithm
- \texttt{a(x)} arctangent
- \texttt{j(n,x)} n-th order Bessel function

### File

Name of file containing a \texttt{bc} program. You may specify more than one file. When all statements from all files have been processed, \texttt{bc} reads from standard input. You can then enter further statements.

\textit{file} not specified:

\texttt{bc} reads from standard input.

### Elements of \texttt{bc} programs

A \texttt{bc} program consists of
- definitions
- statements
- comments

The following symbols are used in defining the structure of \texttt{bc} programs:

- \texttt{L} (L - letter) stands for one of the letters a-z
- \texttt{E} (E - expression) stands for an expression
- \texttt{S} (S - statement) stands for a statement

### Comments

Comments are enclosed in /*...*/ as in C.
Statements

Statements in bc can be:

- Expressions (see Expressions)

  The value of a statement that is an expression is printed unless the main operator is an assignment.

- Blocks (grouped statements): {S; ...; S}

- Conditional statement: if (E) S

  If expression E is true, i.e., has a non-zero value, then statement S is executed.

- Iteration statements:
  - while (E) S
    Expression E is evaluated and if it has a non-zero value, statement S is executed. E is then evaluated again, and S is executed again if E is still non-zero. This process is repeated for as long as E has a non-zero value.
  - for (E; E; E) S
    First, the first expression is evaluated. Next, the second expression is evaluated and if it has a non-zero value, statement S is executed. Lastly, the third expression is evaluated. Then the second expression is evaluated again and statement S is again executed if it is non-zero, and so on. Unlike in C programs, a for statement must always contain three expressions.

- Jump statement: break

  The break statement can be used only within an iteration statement. It causes termination of the nearest while or for statement. Program execution continues with the statement that follows the terminated iteration statement.

- Termination statement: quit

  The quit statement stops execution of a bc program. The quit statement is interpreted as soon as it is read, not when the bc program is executed.

Example

  The following bc program terminates immediately without printing the value of a:

  ```
  a=5
  if (a>10) quit
  a
  ```

  You can separate statements from one another with a semicolon or a newline character.
Expressions

Expressions consist of operands and operators.
Operands are names or arbitrarily long numbers with optional sign and decimal point.

Names

- \( L \)  simple variables
- \( L \)  function names
- \( L[E] \)  array elements

- \( ibase \)  base (radix) for input numbers, default: 10
- \( obase \)  base (radix) for output numbers, default: 10
- \( scale \)  number of fractional digits, default: 0

If arrays are used as function arguments or defined as automatic variables, empty square brackets must follow the array name.

The same name may be used simultaneously for an array, a function, and a simple variable. All variables are global to a \( bc \) program.

Other operands

- \( (E) \)  result of \( E \)
- \( \sqrt{E} \)  square root of \( E \)
- \( \text{length}(E) \)  number of significant decimal digits in \( E \)
- \( \text{scale}(E) \)  number of fractional digits in \( E \)
- \( L(E, ...,E) \)

Operators

- \(+ - */\)  addition, subtraction, multiplication, division
- \(^\)  power operator
- \( \% \)  remainder of integer division (can now also be applied to floating-point numbers)
- \( ++ - - \)  increment and decrement operators, which can be applied to names in prefix or postfix notation
- \(< <= == >= >!=\)  relational operators (less than, less than or equal to, equal to, greater than or equal to, greater than, not equal to)
- \( = \)  assignment operator
compound assignment operators, where a=@b is the same as a=a@b. @ can be any of the operators + - * / ^ or %.

The logical operators && and || are not recognized by the bc command.

Function definition

```
define L (L, ...,L) {  
    auto L, ....L  
    S: ...;S  
    return (E)  
}
```

Example

```
define p(x) {  
    auto q  
    q = p * p  
    return (q)  
}
```

Declaring the identifiers of a function as auto restricts their scope to that function. All function arguments are passed by value.

Functions in the math library /usr/lib/lib.b

Definitions of the mathematical functions listed below are contained in the library /usr/lib/lib.b. The functions can be accessed by calling bc with option -l.

- \( s(x) \) sine
- \( c(x) \) cosine
- \( e(x) \) base e exponential function
- \( l(x) \) natural logarithm
- \( a(x) \) arctangent
- \( j(n,x) \) n-th order Bessel function

Values in mathematical functions must be specified in absolute radian measure.

If you have write permission for /usr/lib/lib.b, you can add definitions of further functions and also modify or delete existing ones.

Defining bases for input and output numbers

With ibase and obase you can specify the base used for interpreting input and output values (input and output number radix). The following rules apply:

1. If you do not explicitly assign values to ibase and obase, input numbers are interpreted as decimal and results are output in decimal.
2. If you have already defined the input base with an \texttt{ibase\!=\!n} statement, the number that you use to define the output base in an \texttt{obase\!=\!m} statement must be in input base \texttt{n} as well.

\textit{Example}

The input base is to be 2, the output base 16:

```
$ bc
ibase=2
obase=10000
10100000/1010
10
```

\textbf{Fractional digits}

Each expression \texttt{E} in \texttt{bc} is associated with a specific number of fractional digits. You can use the \texttt{scale} variable to inspect or change this number and the \texttt{scale(E)} function just to inspect it.

\textit{Example}

In the following example the value of operand \texttt{a} is divided by the value of operand \texttt{b} with the \texttt{scale} variable initially left unset: the result contains no fractional digits. Then \texttt{scale} is assigned a value of 8: the result of the division is now correct to 8 places after the decimal point. Finally we inspect the number of fractional digits in the result and the value of \texttt{scale}.

```
$ bc
a=15.0
b=7.8
a/b
1
scale=8
a/b
1.92307692
scale(a/b)
8
scale
8
@@d or END
$
```

If you join two expressions using an operator, the number of fractional digits associated with the result is governed by a rule specific to the operator you use. The rules for \texttt{bc} operators are described below. Various symbols are used in the descriptions:
\begin{verbatim}

a = first operand
b = second operand
R = number of fractional digits in the result of a calculation
A = scale(a)
B = scale(b)

- The unary minus sign and the increment and decrement operators ++ and -- (in prefix
  ++ and postfix notation) do not affect the number of fractional digits.

  \textbf{Rule:} \texttt{scale(E) = scale (-E) = scale(--E) = scale(++E) ...}

  \textit{Example}

  \texttt{a} is assigned a value with three fractional digits. The query function \texttt{scale(a)} here
  always returns 3, regardless of whether \texttt{a} has a -, -- or ++ operator and regardless
  of the fact that \texttt{scale} was previously assigned a different value:

  \begin{verbatim}
  $ bc
  scale=1
  a=1.123
  scale(a)
  3
  scale(-a)
  -3
  scale(a++)
  3
  scale(--a)
  3
  @@d or [END]
  $ 
  \end{verbatim}

- With the binary operators + and -, \( R \) is equal to the number of fractional digits in the
  operand with the most fractional digits, regardless of whether you have previously
  assigned \texttt{scale} some other value.

  \textbf{Rule:} \( R = \max(A,B) \)

  \textit{Example}

  The \texttt{scale} variable is assigned a value of 1. Operand \texttt{a} is assigned a value with
  2 fractional digits, \texttt{b} a value with three fractional digits. Thus \texttt{b} has more fractional
  digits than \texttt{a}, and also more than \texttt{scale}. The query function returns 3 for both
  operands, the greater number in \texttt{b} taking precedence.

  \begin{verbatim}
  $ bc
  scale=1
  a=0.12
  b=0.123
  $ 
  \end{verbatim}

\end{verbatim}
scale(a+b)
3
scale(b-a)
3
@@d or [END]
$

* With multiplications, a value previously assigned to scale is significant: bc first calculates max, which is the highest of the values scale, A and B. It then forms the sum of A and B and compares this value with max. R is then the lower (min) of these two values.

**Rule:** \( R = \min(A+B, \max(scale, A, B)) \)

**Example**

scale has a value of 9, A and B are both 1. Thus the highest of the three values is 9. The sum of the number of fractional digits in the two operands is 2. The number of fractional digits in the result of the multiplication is the lower from the comparison of max and this sum, which is 2.

$ bc
scale=9
a=0.1
b=0.1
scale(a*b)
2
@@d or [END]
$

/ With divisions, the precision of the result is equal to the value of scale:

**Rule:** \( R = scale \)

**Example**

scale is first given a value of 8. Then the operands are assigned integer values. The result is also an integer. In spite of that, the query function returns 8, and the result is shown correct to 8 places after the decimal point.

$ bc
scale=8
a=16
b=4
scale(a/b)
8
a/b
4.0000000
@@d or [END]
$

^ With the power operator, R is formed as follows:
– If the integer exponent \( e \) is equal to or greater than 0:
\( \text{bc} \) takes the higher (max) of the two values \( \text{scale} \) and \( A \). It then multiplies \( A \) by the absolute value \( m \) of the exponent, compares the result with \( \text{max} \), and takes the lower of the two values.

Rule: \( R = \min (A^m, \text{max (scale, A)}) \)

– If the integer exponent \( e \) is less than 0:
The precision of the result is equal to the value of \( \text{scale} \).

Rule: \( R = \text{scale} \)

Example 1

\( \text{scale} \) is given a value of 7. \( a \) has one fractional digit, and the absolute value of exponent \( e \) is 4, i.e. greater than 0. The higher value from the comparison of \( \text{scale} \) and \( a \) is 7. However, the result of multiplying \( a \) and \( m \) is 4. Thus the number of fractional digits after exponentiation is 4:

\[
\begin{align*}
&\text{scale}=7 \\
a &= 3.1 \\
e &= 4 \\
&\text{scale}(a^e) \\
&= 4 \\
a^e \\
&= 92.3512 \\
&@\text{d} \text{ or [END]}
\end{align*}
\]

Example 2

However, if you set the exponent \( e \) to -4, the number of fractional digits is equal to the value of \( \text{scale} \):

\[
\begin{align*}
&\text{scale}=7 \\
a &= 3.1 \\
e &= -4 \\
&\text{scale}(a^e) \\
&= 7 \\
a^e \\
&= 0.0108281 \\
&@\text{d} \text{ or [END]}
\end{align*}
\]
With the assignment operators, the value of \( R \) is equal to that of \( A \) after assignment. For a given compound operator \( =@ \), the rule for calculating the number of fractional digits is the same as for the corresponding simple operator \( @ \).

**Rules:** \( R = \text{scale}(b) \) and \( R = \text{scale}(a@b) \)

**Example 1**

\( a \) is assigned a number with one fractional digit, \( b \) a number with two fractional digits: \( \text{scale}(a) \) is 1 and \( \text{scale}(b) \) is 2. If you inspect \( \text{scale}(a) \) after assigning the operands, \( bc \) returns a value of 2, which is \( \text{scale}(b) \). If you then inspect \( a \) again, the value returned is the value assigned to \( b \):

```
$ bc
a=0.1
b=0.12
scale(a)
1
scale(b)
2
a=b
scale(a)
2
a
.12
@\@d or [END]
$
```

**Example 2**

\( a \) is assigned a number with two fractional digits, \( b \) a number with three. In the function call, \( a \) is assigned the value resulting from adding the two operands. \( bc \) sees the number of fractional digits in the result of an addition as being equal the number of fractional digits in the operand with the most fractional digits. After assignment \( a \) has the same number of digits, i.e. three:

```
$ bc
a=0.1
a=0.123
scale(a)
2
scale(b)
3
a=+b
scale(a)
3
```
With remaindering, if the value of scale is non-zero, the result is computed as follows:

\[ a \% b = a - (a / b) \times b \]

First the division is performed using the precision of `scale`. The precision of the multiplication by \( b \) is equal to:

\( \text{scale} + B \)

In other words the multiplication is performed with full precision. Thus here the \( \% \) operator can be used as a measure of the precision with which the division is performed.

Finally \( R \) is whichever is the higher of \( A \) and \((\text{scale} + B)\).

**Rule:** \( R = \max ((\text{scale} + B), A) \)

**Example**

`scale` has a value of 4, \( a \) and \( b \) each have one fractional digit. The result of remaindering has five fractional digits:

```bash
$ bc
scale=4
a=1.2
b=1.1
scale(a%b)
5
a%b
.00001
@@d or [END]
$
```

**File**

`/usr/lib/lib.b`

Math library

**Locale**

The following environment variables affect the execution of `bc`:

`LANG`

Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
**bc**

**LC_ALL**  
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

**LC_MESSAGES**  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1**  
Addition, subtraction, multiplication, and division of numbers. Non-integer results are to have 2 digits after the decimal point:

```
$ bc
scale=2
3+7
10
8-15
-7
7*6
42
3/5
.60
quit
```

**Example 2**  
Defining a function to compute an approximate value of the exponential function.

```
scale=20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
    for(i=1;i==1;i++){
        a = a*x
        b = b*i
        c = a/b
        if(c==0) return (s)
        s = s+c
    }
}
```
The termination criterion of the for loop is contained in the body of the loop \((i \neq c \Rightarrow 0)\) and is not given by the second expression of the for statement as is usually the case. In a C program, the second expression of the for statement would simply be omitted; but in a \texttt{bc} program the for statement must always have 3 expressions. The expression \(i == 1\), which is always true, has been inserted for this reason.

**Example 3**  Printing approximate values of the exponential function of the first ten integers.

\[
\text{for}(i=1; i \leq 10; i++) \ e(i)
\]

**Example 4**  Printing the squares of the integers 1 to 4:

\[
\text{for}(i=1; i < 5; i++) \ (i*i)
\]

\[
1
4
9
16
\text{quit}
\]

See also  \texttt{expr, let}
**bg**

**run jobs in the background**

You can use the built-in command `bg` in the POSIX shell `sh` to specify jobs for background processing.

You should only use this command if you have accessed the POSIX shell via `rlogin`.

**Syntax**

```
bg[...job-id...]
```

**job-id**

Each specified job is held for background processing. The section “Jobs” on page 62 describes the format of `job-id`.

*job-id not specified:*

The current job is specified for background processing.

**Locale**

The following environment variables have an effect on the execution of `bg`:

- **LANG**: Specifies a default value for the locale variable that is unset or null. If `LANG` is unset or null, the corresponding locale default value is used. If the locale variable contains an invalid setting, `locale` behaves as if no variables had been set.

- **LC_ALL**: If this value is set, i.e. is not empty, this value overwrites the values of all other locale variables.

- **LC_CTYPE**: Determines the locale for the interpretation of byte sequences as characters (e.g. single-byte as opposed to multibyte characters in arguments).

- **LC_MESSAGES**: Determines the format and content of error messages.

- **NLSPATH**: Determines the position of the message catalog for the processing of `LC_MESSAGES`.

**Example**

You wish to specify job `%1` for background processing:

```
$ bg %1
```

**See also**

`fg`, `jobs`, `kill`, `wait`
The POSIX command `bs2cmd` executes a BS2000 command.

**Syntax**

`bs2cmd [\-h] cmd`

- `-h` Displays the command syntax and explains the options.

  cmd

  BS2000 command:
  Any BS2000 command in accordance with the BS2000 syntax in SDF format may be specified here.
  (The ISP format is now only supported for compatibility reasons.)

  Special characters in the BS2000 command (e.g. $, *) must be escaped by a backslash (`\`). Alternatively, the entire cmd string must be enclosed in quotes.

  The command name, operands and operand values are converted to uppercase letters.
  For commands in the ISP format, only the command name is converted to uppercase.

**Hint**

The `bs2cmd` command is also supported with `rlogin` and `telnet` accesses to POSIX.

- SYSFILE environment:
  BS2000 commands that need the SYSFILE environment for execution can be currently executed only in the base shell, since the SYSFILE environment is not fully initialized in a subshell.

- Unloading the shell:
  When `bs2cmd` is called from the base shell, the unloading of the shell is prevented by specifying a corresponding BS2000 command (e.g. START-PROGRAM or a user-defined command whose CALL procedure terminates the running program).
  `bs2cmd` issues a message (SDP0250), and the BS2000 command is not executed.
  The unloading of the base shell cannot be currently prevented on calling `bs2cmd` from a subshell.
Example  The following command shows the catalog entries of the SYSRME files on the BS2000 ID $QM212:

```
/home/user1> bs2cmd 'show-file-attributes $qm212.sysrme.*'
%   15 :10SN:$QM212.SYSRME.ARCHIVE.090.D
%   15 :10SN:$QM212.SYSRME.ARCHIVE.090.E
%   12 :10SN:$QM212.SYSRME.POSIX-BC.070.D
%   12 :10SN:$QM212.SYSRME.POSIX-BC.070.E
%   18 :10SN:$QM212.SYSRME.POSIX-BC.080.D
%   18 :10SN:$QM212.SYSRME.POSIX-BC.080.E
%:10SN: PUBLIC:   15 FILES RES=     210 FREE=     18 REL=      0 PAGES
%:10SN: PUB/S2:    2 FILES RES=     123 FREE=     1  REL=      0 PAGES
```

/home/user1>

See also  bs2cp, bs2file
bs2cp copies files from the POSIX file system to BS2000 and vice versa. The command has four formats:

- Copying a single POSIX file to BS2000 and vice versa (see below)
- Copying POSIX files to BS2000 as DVS files (see page 172)
- Copying POSIX files to a BS2000 PLAM library (see page 173)
- Copying BS2000 files with wildcard syntax to a POSIX file directory (see page 174)

The command creates physical copies. Hence, after copying the files are physically present in both the POSIX and BS2000 file systems.

Format 1

Copying a single POSIX file to BS2000 and vice versa

Syntax

```
bs2cp [-k | -t table] [-f] [-l] [-h] bs2:file..filecopy
```

- `-k` The file content is converted during copying:
  - from ASCII to EBCDIC, if `file` is a file of the POSIX file system
  - from EBCDIC to ASCII, if `file` is a BS2000 file (`bs2:` entered).

  This option is ignored and no conversion is performed if the original file or the file copy is a PLAM library element of the `L (LLM)` type.

  Unlike control with the `IO_CONVERSION` environment variable, `-k` will convert independently of the POSIX file system type. Hence, it will also convert when the POSIX file system is an EBCDIC file system (i.e. not an ASCII file system or an NFS-mounted file system).

  `-k` is only meaningful with text files. The `b2cp` command does not perform any contents plausibility check for meaningful conversion.

- `-t table` The file content is converted during copying using the `table` file as conversion table.

  The options `-k` and `-t` are mutually exclusive.

  This is similar to the option `-k` in that the option will be ignored and no conversion will take place if the original file or the file copy is a PLAM library element of the `L (LLM)` type.
Properties of the table file:

- EBCDIC format.
- It must contain exactly 256 character pairs. They must be composed of the following characters: 0 to 9, a to f and A to F.
- All EBCDIC characters from X'00' to X'40' can be placed between the character pairs. These are for example blank spaces, tabulators or newline characters.
- The file can be of any size. However, only the first 8172 characters (max.) are evaluated.

If the absolute path name of the table file is not given, the command will look up in $BS2CPTABS. If this variable is not set or is empty, the command will search for the table file in the /usr/lib/bs2cp directory.

At first, a 256-byte conversion table is created from the table file during the conversion. This is done by compressing each of the 256 character bytes to a hex decimal character. After that, each of the characters from the file to be copied is replaced by a character from the conversion table which is addressed via the binary value of the input character.

The input character $M$ is for example substituted by byte 212 of the conversion table because $M$ has the EBCDIC value X'D4' and hence the decimal value 212.

An example of the structure of the code table can be found under “Example table for option -t” on page 177.

-f By default, stderr is checked during copying from POSIX to BS2000 and the user is prompted if existing BS2000 files/elements should be overwritten or not. This dialog prompt is suppressed with -f and existing files are overwritten.

This option is ignored (if indicated) during copying from BS2000 to POSIX. Existing POSIX files are always overwritten.

If the environment variable OV exists, the dialog query is omitted regardless of the -f option. If OV = "Y", existing BS2000 files/elements are overwritten, otherwise they are not overwritten and a message is issued instead.

-l Each successfully copied file is reported as follows:

bs2cp: copy from path name to path name done

-h Output of the command syntax with explanation of options.

bs2:

The file or filecopy entered with this option is a BS2000 file.

DVS files are addressed by their name. Special characters (e.g. $ in the BS2000 user ID) in file names must be masked by a preceding backslash character or by bracketing the character strings bs2:file or bs2:filecopy in single inverted commas.
A library element is called in the following form (also see example 3 on page 181):

\[ \text{\texttt{lib(elem [,type [,vers]})}} \]

where

- **lib** is the name of the PLAM library in BS2000.
- **elem** is the name of the element.
- **type** is the type of element. The element types \(S\), \(M\), \(J\), \(P\), \(D\), \(X\) and \(L\) are supported. Default value is \(S\).
- **vers** is the version of the element. Default is \(^*\text{HIGH}\).

One of the files (\textit{file} or \textit{filecopy}) must be a BS2000 file or library element. POSIX file directories cannot be copied into BS2000. Wildcard syntax and construction entries are not supported.

**file**

Name of the file to be copied.

**filecopy**

Name of the copy. If \textit{filecopy} is an existing BS2000 file or a library element, the user is prompted if it should be overwritten. The option \(-f\) and the environment variable \(OV=Y\) suppress this prompt.

The following is valid for a \textit{filecopy} that is a BS2000 DVS file:

- If \textit{filecopy} already exists, bs2cp will adopt the file attributes from the file catalog.
- If \textit{filecopy} does not yet exist, it will be created. Prior to the \texttt{bs2cp} call, the file type of the new file can be set with the \texttt{ftyp} command. If no file type is set, a SAM file with a variable record length will be created.

The following is valid if \textit{filecopy} is an element of a BS2000 PLAM library:

- An existing \textit{elem} library element will be overwritten if the option \(-f\) is set or if the environment variable \(OV=Y\).
- If the \textit{elem} library element does not yet exist, it will be created. The \texttt{ftyp} and \texttt{bs2file} commands can also be used to influence the file types of non \(L\)-type elements.
- If no PLAM library with the name \textit{lib} exists, it will be created.
bs2cp

Format 2  Copying POSIX files to BS2000 as DVS files

Syntax

bs2cp [-x [-k -t table]] [-f] [-h] [-p prefix] [-s suffix] file...

- **x**  Extended format of the bs2cp command.

- **k**  see format 1 (page 169).

- **f**  This is analogous to format 1. By default, stderr is checked during copying and the user is prompted if existing BS2000 files should be overwritten. However, the prompt offers several reply options:

  bs2cp: overwrite A ? y (yes), n (no), a (all) or q (quit)

  If q is selected, the command bs2cp is aborted with an exit status unequal to null.

  The dialog query is suppressed with -f. Already existing files are always overwritten.

  If the environment variable OV exists, the dialog query is omitted regardless of the -f op-

tion. If OV = "Y", existing BS2000 files/elements are overwritten, otherwise they are not overwritten and a message is issued instead.

- **l**  Each successfully copied file is reported as follows:

  bs2cp: copy from path name to path name done

- **h**  Output of the command syntax with explanation of options.

- **p**  prefix

  The prefix character string precedes the names of the copies.

- **s**  suffix

  The suffix character string is appended to the names of the copies.

file...

  is a list consisting of one or more POSIX files where shell special characters (wildcards) can be used to generate file names. file must not be a file directory.

bs2:

  The copies are placed in BS2000 as DVS files. The copies always receive the same plain file names as the originals written in upper case. They can, however, be expanded with prefixes and suffixes. Any underscore characters will be converted into dollar signs.

  If files already exist, the user will be prompted if they are to be overwritten. Moreover, the user can specify that all the following files will be overwritten without any prompt being displayed. It is also possible to cancel processing.

  By default, the files are copied to the BS2000 user ID of the POSIX user in the home pubset. The prefix entered with -p can be used to indicate another Cat ID or User ID.

  See example 4 on page 181.
bs2cp

Format 3 Copying POSIX files to a BS2000 PLAM library

Syntax

bs2cp [-x] [-k] [-t] [-f] [-I] [-h] [-p] [-s] [prefix] [suffix] [file...]

- x Extended format of the bs2cp command.

- k -t
  see format 1 on page 169.

- f By default, stderr is checked and the user is prompted if existing BS2000 elements should be overwritten or not. This dialog prompt is suppressed with -f and existing files will be overwritten.

  If the environment variable OV exists, the dialog query is omitted regardless of the -f option. If OV = "Y", existing BS2000 files/elements are overwritten, otherwise they are not overwritten and a message is issued instead.

- l Each successfully copied file is reported as follows:
  bs2cp: copy from path name to path name done

- h Output of the command syntax with explanation of options.

- p prefix
  The prefix character string precedes the names of the copies.

- s suffix
  The suffix character string is appended to the names of the copies.

file...
  is a list consisting of one or more POSIX files where shell special characters (wildcards) can be used to generate file names. file must not be a file directory.

'bs2:lib([,[type]],vers)]'
  The copies are created as elements of the BS2000 PLAM library lib. For type and vers see page 171. The element names are created from the original simple file names (in upper case). They can be expanded with prefixes and suffixes.

See example 5 on page 181.
bs2cp

Format 4  Copying BS2000 files with wildcard syntax to a POSIX file directory

Syntax


- **x**  Extended format of the *bs2cp* command.
- **-k**  see format 1 on page 169.
- **-t**  Each successfully copied file is reported as follows:
  bs2cp: copy from *path name* to *path name* done
- **-h**  Output of the command syntax with explanation of options.
- **-p**  prefix
  The *prefix* character string precedes the names of the copies.
- **-s**  suffix
  The *suffix* character string is appended to the names of the copies.

'**bs2:file**'

*file* is a fully or partially qualified DVS file name with wildcard syntax (special character "*" of the BS2000 command SHOW-FILE-ATT or FS). Only one *bs2:file* operand is permitted.

Important: If the name consists of blank spaces only or is missing altogether, this is equivalent to the "***" wildcard character.

'**bs2:lib(elem[,type][,vers])**'

*elem* is the fully qualified name of a PLAM element or the partially qualified name of several PLAM elements with LMS wildcard syntax. The following wildcard special characters are supported: * < : >. Other special characters for LMS wildcards are not guaranteed.

Special case: If *elem* is not specified, all elements of the corresponding type and version will be used (this is equivalent to the wildcard character "***").

Minimal entry: *bs2:lib* all elements of type *S* and of highest version are copied.

_file directory_

The BS2000 files/elements are copied to the POSIX file directory indicated. The common shell notation is permitted for file directories. For example:

. (period)  Current directory
~ (tilde)   Home directory
/dev/1/dvz2 absolute path
dvz1/dvz2 Relative path starting at the current directory

See example 6 on page 182 and example 7 on page 182.
**Hint**

**EXIT status in the extended formats 2 through 4**

If several files are to be copied with the same bs2cp call, the exit status will only be 0 if all copies were successfully completed. As soon as the first error occurs, the exit status will be unequal to 0 and the remaining files will not be copied.

**Copying library elements**

When library elements are copied to POSIX files or vice versa, the software product LMS must be installed.

**DVS file attributes supported**

bs2cp only supports file attributes which are supported by the C runtime system with STREAM I/O. Hence, SAM files with fixed record length can only be opened as binary type (ftyp binary).

ftyp binary supports the following file attributes:

<table>
<thead>
<tr>
<th>FCB-TYP</th>
<th>REC-FORM</th>
<th>BLKCTRL</th>
<th>BLKSIZE (STD,n)</th>
<th>RECSIZE (r Byte)</th>
<th>Max. number of data bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM</td>
<td>E</td>
<td>PAMKEY</td>
<td>1≤ n ≤ 16</td>
<td>1≤ r ≤ n*2048</td>
<td>RECSIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(2K)</td>
<td>1≤ n ≤ 16</td>
<td>1≤ r ≤ n*2048-16</td>
<td>RECSIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(4K)</td>
<td>2≤ n ≤ 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>V</td>
<td>PAMKEY</td>
<td>1≤ n ≤ 16</td>
<td>4≤ r ≤ n*2048</td>
<td>RECSIZE - 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(2K)</td>
<td>1≤ n ≤ 16</td>
<td>4≤ r ≤ n*2048-16</td>
<td>RECSIZE - 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(4K)</td>
<td>2≤ n ≤ 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>PAMKEY</td>
<td>1≤ n ≤ 16</td>
<td></td>
<td>BLKSIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(2K)</td>
<td>1≤ n ≤ 16</td>
<td></td>
<td>BLKSIZE - 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(4K)</td>
<td>2≤ n ≤ 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO(2K)</td>
<td>1≤ n ≤ 16</td>
<td></td>
<td>BLKSIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO(4K)</td>
<td>2≤ n ≤ 16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
*ftyp* text supports the following file attributes:

<table>
<thead>
<tr>
<th>FCB-TYP</th>
<th>REC-FORM</th>
<th>BLKCTRL</th>
<th>BLKSIZE (STD,n)</th>
<th>RECSIZE (r Byte)</th>
<th>Max. number of data bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM</td>
<td>V</td>
<td>PAMKEY</td>
<td>1 ≤ n ≤ 16</td>
<td>4 ≤ r ≤ n*2048-4</td>
<td>RECSIZE - 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(2K)</td>
<td>1 ≤ n ≤ 16</td>
<td>4 ≤ r ≤ n*2048-16</td>
<td>RECSIZE - 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(4K)</td>
<td>2 ≤ n ≤ 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>PAMKEY</td>
<td>1 ≤ n ≤ 16</td>
<td></td>
<td>BLKSIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(2K)</td>
<td>1 ≤ n ≤ 16</td>
<td></td>
<td>BLKSIZE - 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(4K)</td>
<td>2 ≤ n ≤ 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISAM</td>
<td>V</td>
<td>PAMKEY</td>
<td>1 ≤ n ≤ 16</td>
<td>12 ≤ r ≤ n*2048</td>
<td>RECSIZE - 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(2K)</td>
<td>1 ≤ n ≤ 16</td>
<td>12 ≤ r ≤ n*2048</td>
<td>RECSIZE - 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA(4K)</td>
<td>2 ≤ n ≤ 16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The record keys of ISAM files are not transferred. It is not possible to copy temporary files from BS2000 or via remote file access.
Example table for option -t

Structure of a EBCDIC standard table. A target file is created with a 1:1 copy of the source file:

```
    00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
   10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
   20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
   30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
   40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
   50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
   60  61  62  63  64  65  66  67  68  69  6A  6B  6C  6D  6E  6F
   70  71  72  73  74  75  76  77  78  79  7A  7B  7C  7D  7E  7F
   80  81  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  8F
   90  91  92  93  94  95  96  97  98  99  9A  9B  9C  9D  9E  9F
  A0  A1  A2  A3  A4  A5  A6  A7  A8  A9  AA on AC AD AE AF
  B0  B1  B2  B3  B4  B5  B6  B7  B8  B9  BA BB BC BD BE BF
  C0  C1  C2  C3  C4  C5  C6  C7  C8  C9  CA CB CC CD CE CF
  D0  D1  D2  D3  D4  D5  D6  D7  D8  D9  DA DB DC DD DE DF
  E0  E1  E2  E3  E4  E5  E6  E7  E8  E9  EA EB EC ED EE EF
  F0  F1  F2  F3  F4  F5  F6  F7  F8  F9  FA FB FC FD FE FF
```

Error

path name: Permission denied
You do not have read permission for this file or you do not have write permission for the file directory.

path name: no such file or directory
The file or file directory indicated does not exist.

path name: is a directory
The name entered describes a file directory and not a file.

path name: DMS FSTAT error xxx
Too many files have been entered.

invalid character pair: xx (valid only 0, ...9,A,...,F)
An erroneous conversion table was indicated with option -t.

incorrect number of character pairs: xxx (must be 256)
An erroneous conversion table was indicated with option -t.

more than 256 character pairs given
An erroneous conversion table was indicated with option -t.
do not use options -k and -t at the same time
The options -k and -t cannot be used at the same time.

file name name invalid in this case
A name was indicated after bs2: in format 2.

element name name invalid in this case
An element name was given in format 3.

invalid target name path name
The name of the target file contains an **.

lib(elem,type,vers) already exists
The output library element already exists. (The -f option was not specified, the environment variable OV does not exist, and the entry takes place from stdin (-).)

lib(elem,type,vers) not accessible
The output library element cannot be accessed. You either have no write authorization for the library or for the element, or lib is not a PLAM library.

file already exists
The BS2000 output file already exists. (The -f option was not specified, the environment variable OV does not exist, and the entry takes place from (-).)

path name: is a directory
The path name specified refers to a directory.

path name is not an executable file
An attempt was made to copy a POSIX file which does not have the format (UFS-LLM) of an executable file into a library as an L element.

closing bracket missing
The closing bracket is missing in the element specification. (An opening bracket was found.)

Copy from DMS file to DMS file not supported — use command /COPY-FILE f1,f2
Copying a BS2000 file to a BS2000 file is not supported. The /COPY-FILE command must be used for this purpose.

Copy from UFS file to UFS file not supported — use command cp f1,f2
Copying a POSIX file to a POSIX file is not supported. The POSIX command cp must be used for this purpose.

Copy of UFS directories is not supported
. or .. was specified as file.

DMS FSTAT error Dxxx
The system function FSTAT for a BS2000 file returns the error code Dxxx.

element name elem too long
The element name (format 2, 3 or 4) specified as the source is longer than 64 characters.
FILE command for `filename` returned error
The FILE command for the BS2000 output file returned an error. A message of the FILE command was output immediately beforehand. (The message is connected to a bs2file or ftyp command which was issued by the user beforehand.)

file filename not found
The BS2000 input file was not found.

Invalid BS2000 filename: file
The specified file name ends with a . (period).

invalid character in elem
The specified element name contains illegal characters.

invalid element name elem
The element name elem starts or ends with . or -.

LMS error - LMSnnnn[, PLAmmmnn][, DMS or macro RC xxxx]
(fct=fct, lib=lib, elem=elem, type=type, ver=vers, file=filename)
When a library element is accessed, LMSUP reports an error with the error code LMSnnnn, possibly supplemented by the PLAM error code and/or the DMS or macro error code (the meaning of the error codes can be inquired using bs2cmd help LMSnnnn etc.).
The second line is output only if the -l or -x option was specified. fct is ADD, SEL or TOC.
When fct=TOC, the filename specification remains empty.

LMS end errorcode xx
LMSUP, which is needed to copy library elements, could not be terminated properly.

LMS init errorcode xx
LMSUP, which is needed to copy library elements, could not be initialized.

LMS toc errorcode xx
The library's directory could not be read any further.

LMS tocprim errorcode xx
The library's directory could not be read.

No POSIX file involved - use utility LMS
A library element and a BS2000 file were specified as the source and the destination or vice versa.

No write of file. OV is set to value
The specified BS2000 file was not overwritten because the -f option was not specified and the environment variable OV is set and not equal to “Y”.

No write of lib(elem, type, vers). OV is set to value
The specified element was not overwritten because the -f option was not specified and the environment variable OV is set and not equal to “Y”.

U22794-J-Z125-6-76
opening bracket missing
The opening bracket is missing in the element specification. (A closing bracket was found.)

PLAM element elem, type type, version vers not accessible
The input element which is specified as the source cannot be accessed. You either have no read authorization for the library or the element, or lib is not a PLAM library.

PLAM element elem, type type, version vers not found
The input element specified as the source was not found.

PLAM element name longer than 64
The specified element name is longer than 64 characters.

PLAM element name missing
The element name is missing in the element specification for format 1.

PLAM element version longer than 24
The specified element version is longer than 24 characters.

PLAM element version vers invalid
The version specification vers contains illegal or wildcard characters.

PLAM element type not supported
The specified type is not S, M, J, P, D, X or L.

PLAM error - PLAnnnnn[, DMS or macro RC xxxx]
(fct=fct, lib=lib, elem=elem, type=type, ver=vers, file=filename)
When a library element is accessed, PLAM reports an error with the error code PLAnnnnn, possibly supplemented by the DMS or macro error code (the meaning of the error code can be inquired using bs2cmd help PLAnnnnn etc.).
The second line is output only if the -l or -x option was specified. fct is ADD or SEL.

PLAM library name missing
The library name is missing in the element specification.

PLAM library name longer than 54
The library name is longer than 54 characters.

PLAM library lib not found
The specified PLAM library does not exist.

PLAM names must end with single closing bracket
The element name ends with more than one closing bracket.

too many opening brackets
The element specification contains more than one opening bracket.

too many tokens within brackets
The element specification contains more than three elem, type, vers specifications separated by commas.
Example 1  Copy the BS2000 techdoc file into the file directory /usr/fl, using the same file name. The character set should be converted from EBCDIC to ASCII (option -k).

```
$ bs2cp -k bs2:techdoc /usr/fl/techdoc
```

Example 2  Copy the techdoc file in the POSIX file system to BS2000, using the file name flcopy. The current character set is retained (no option -k).

```
$ bs2cp techdoc bs2:flcopy
```

Example 3  The documentation file is a D-type element in the BS2000 PLAM library product. Copy the element to the file directory /usr/product.

```
$ bs2cp 'bs2:product(documentation,D)' /usr/product/documentation
```

Example 4  $ bs2cp -x -p posix. -s .sich /home/do/sich/dat* bs2:

```
bs2cp: overwrite FILE1 ? [y=yes/n=no/a=all/q=quit] a
```

```
$ 1s /home/do/sich
file1 file2 file3
```

```
$ bs2cmd fstat posix.
  12 :ABCD:$USER.POSIX.FILE1.SICH
  9 :ABCD:$USER.POSIX.FILE2.SICH
 18 :ABCD:$USER.POSIX.FILE3.SICH
```

Example 5  $ 1s

```
genpos.c  hrcv.c  hrcv.s  hrmgt.c  hrupos.c  hsdax.c
```

```
$ bs2cp -x -l -p p hr*.c 'bs2:clib(,s,300)' 
bs2cp: copy from hrcv.c to CLIB(PHRCV.C,S,300) done
bs2cp: copy from hrmgt.c to CLIB(PHRMGT.C,S,300) done
bs2cp: copy from hrupos.c to CLIB(PHRUPOS.C,S,300) done
```

Creates the PLAM library CLIB with the following content:

```
TYPE   NAME           VER (VAR#)          date
(S) PHRCV.C  300 (0001) 2008-05-27
(S) PHRMGT.C 300 (0001) 2008-05-27
(S) PHRUPOS.C 300 (0001) 2008-05-27
```
Example 6  $ bs2cmd fstat *$user2.*kvh*
    9 :ABCD:$USER2.AKVH
   12 :ABCD:$USER2.KVHMEM
   12 :ABCD:$USER2.ZKVH

$ bs2cp -x -s .c *bs2:$user2.*kvh* /home/tag/kvh

$ ls -l /home/tag/kvh
    total 12
   -rw-r--r--  1 kvh prod5  2321 May 27 13:56 akvh7.c
   -rw-r--r--  1 kvh prod5 18549 May 27 13:56 kvhmem.c
   -rw-r--r--  1 kvh prod5   971 May 27 13:56 zkvh.c

Example 7  From the BS2000 PLAM library $USER2.DOCLIB, all elements of type D, file names starting
with KVH and version 300 are to be copied to the file directory /home/usr/doc. The prefix doc.
is to precede the names of the copies.

$ bs2cp -xl -p doc. *bs2:$user2.doclib(kvh*,d,300)* /home/usr/doc
bs2cp: copy from $USER2.DOCLIB(KVHGEN,D,300) to /home/usr/doc/doc.kvhgen done
bs2cp: copy from $USER2.DOCLIB(KVHPRD,D,300) to /home/usr/doc/doc.kvhprod done
bs2cp: copy from $USER2.DOCLIB(KVHZ,D,300) to /home/usr/doc/doc.kvhz done

$ ls -l /home/usr/doc
    total 12
   -rw-r--r--  1 kvh prod5 21738 May 31 06:21 doc.kvhgen
   -rw-r--r--  1 kvh prod5  7461 May 31 06:21 doc.kvhprod
   -rw-r--r--  1 kvh prod5 11729 May 31 06:21 doc.kvhz

See also bs2file, ftyp
bs2do calling BS2000 procedures from the POSIX shell (BS2000)

The BS2000 procedure to be executed is started in an ENTER-JOB with the CALL-PROCEDURE command. bs2do waits synchronously for the ENTER-JOB to terminate.

Syntax

```
bs2do[-DV][--o outfile] procedure [[(parameter)]]
```

- **-D** Debug mode - temporary files are not deleted and remain available for diagnosis.
- **-V** Verbose - single steps are logged to stdout.
- **-o** the BS2000 system file SYSOUT opened when the ENTER-JOB is started is copied to the POSIX file `outfile`.

`outfile` name of the POSIX file into which the BS2000 system file SYSOUT is copied when the ENTER-JOB is started. `outfile` can be any POSIX path name.

`procedure` name of the file containing the BS2000 procedure to be executed. `procedure` can be any POSIX path name or a string `bs2:BS2Name`, where `BS2Name` is a BS2000 procedure name according to the SDF syntax of the operand FROM-FILE of the BS2000 /CALL-PROC command.

`parameter` parameters of the BS2000 procedure to be executed. Entries are made according to the SDF syntax of the PROCEDURE-PARAMETERS operand of the BS2000 /CALL-PROC command, the enclosing parentheses being optional.

The following applies to `procedure` and `parameter`: special characters (e.g. (, ), $) are to be masked for the shell. bs2do does not interpret the operands, e.g. among other things keywords in keyword parameters are not converted to uppercase.

File

Input files:
- `stdin`: not used
- `procedure` file `procedure`
- `options` file `optionfile`, where `optionfile` can be any POSIX path name.

By default, `/opt/BS2DO/options/bs2doopt.std` is installed.

Output files:
- `stdout`: output of messages if the option `-V` (verbose) is specified
- `stderr`: bs2do error messages, contents of the job variable defined in the options file `optionfile`
- BS2000 system file SYSOUT copied to POSIX file `outfile`, if the `-o` option is specified

Variable

`BS2DOOPT`: POSIX path name of an options file. If the environment variable `BS2DOOPT` is not defined, the implicitly installed options file `optionfile` is used.
bs2do

Extended description

*bs2do* sets up the batch job #T.pid.ENTER in BS2000, where *pid* is the POSIX process number of the process executing *bs2do*. The batch job is initiated with

```
./DO pid LOGON
```

If the procedure *procedure* is located in the POSIX file system, it is copied to the BS2000 file T.pid.PRO. The batch job is then started as follows:

```
/ENTER-JOB FROM-FILE=#T.pid.ENTER,-
PROCESSING-ADMISSION=SAME,-
HOST=*STD,-
JOB-CLASS=*STD,-
JOB-PRIORITY=*STD
```

The operands PROCESSING-ADMISSION, HOST, JOB-CLASS and JOB-PRIORITY can be set via the options file *optionfile*.

In the batch job, the system file SYSOUT is assigned to the file #T.pid.OUT, the job variable #BS2DOJV is set up and then the *procedure* is called, the name of the #BS2DOJV job variable can be set via the options file *optionfile*:

```
/CALL-PROC FROM-FILE=T.pid.PRO,-
PROC-PAR=parameter,-
LOGGING=NO
```

The LOGGING operand can be set via the options file *optionfile*. The FROM-FILE operand is only specified if *procedure* is in the POSIX file system. Otherwise, the string *procedure* is adopted unchanged. Special characters (e.g. (,),$) are to be masked for the shell.

After the procedure *procedure* terminates, the system file SYSOUT is closed and the POSIX shell is started.

```
/START-PROG SHELL
```

The *SHELL* operand can be set via the options file *optionfile*.

If no error display was set in *procedure* (/EXIT-PROCEDURE ERROR=NO), the signal SIGUSR1 is sent to the parent process of the batch job as a terminating message.

If the error display was set in the procedure *procedure* (/EXIT-PROCEDURE ERROR=YES), the value of the job variable #BS2DOJV and the signal SIGUSR2 are sent to the parent process of the batch job as a terminating message.

In both cases, if the -o *outfile* option is specified, the file #T.pid.OUT is first copied to the POSIX file *outfile*. When the shell terminates, all T.pid files are deleted and the batch job is terminated.

All temporary #T.pid files are also deleted. In debug mode (-D option), no temporary files are created, but only T.pid files that are not deleted when the batch job terminates.
The options file

The structure of the options file is as follows:

keyword keyword = value

Lines that do not begin with a valid keyword are ignored. Continuation lines are exceptions to this rule. value can be extended beyond the end of a line with the - character.

Table of supported keywords:

<table>
<thead>
<tr>
<th>keyword</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS2DOJV</td>
<td>Job variable name (optional). If job variables are not offered by the product, no value may be specified for this keyword</td>
</tr>
<tr>
<td>SHELL</td>
<td>Operand of START-PROG for loading the POSIX shell</td>
</tr>
<tr>
<td>PROCESSING-ADMISSION</td>
<td>Corresponds to the operand of ENTER-JOB with the same name</td>
</tr>
<tr>
<td>HOST</td>
<td>Corresponds to the operand of CALL-PROC with the same name</td>
</tr>
<tr>
<td>JOB-CLASS</td>
<td>Corresponds to the operand of CALL-PROC with the same name</td>
</tr>
<tr>
<td>JOB-PRIORITY</td>
<td>Corresponds to the operand of CALL-PROC with the same name</td>
</tr>
</tbody>
</table>

The standard options file /opt/BS2DO/options/bs2doopt.std:

```
# bs2do controlling job variable
# (temporarily in order to allow parallel calls within one user ID)
# BS2DOJV = #BS2DOJV
#
# BS2000 enter-job operands
#
# PROCESSING-ADMISSION = SAME
# HOST = *STD
# JOB-CLASS = *STD
# JOB-PRIORITY = *STD
#
# BS2000 call-proc operands
#
# LOGGING = NO
#
# BS2000 shell configuration (POSIX-BC shell with
# correction level >= A12
# POSIX-SH shell with correction level
# level >= A52)
#
# SHELL = *M($TSOS.SINLIB.POSIX-BC.030.SHELL, SH, -
# RUN-MODE=ADVANCED, PROG-MODE=ANY)
```
**Restrictions**

Since `bs2do` cannot bypass the BS2000 security mechanisms, restrictions must be observed if `PROCESSING-ADMISSION <> SAME` is set in the options file:

- The user ID in which the ENTER-JOB is to be executed must have POSIX user permissions.
- If the `-o` option is specified, the user ID in which the ENTER-JOB is to be executed must have search and write permissions in the current directory in which `bs2do` was called. Otherwise, the ENTER-JOB SYSOUT file is not copied to the UFS, even if the `-o` option is set.
- The procedure to be executed is not copied from the UFS into another user ID, i.e. executing a procedure under a different user ID is only supported with `procedure` being equal to `bs2:BS2Name`.

**Example**

In the following examples, a prefixed DO: represents the shell prompt which is followed by entries to the shell. Lines between the shell prompts are the outputs from the commands concerned. The procedure `proc` is included as an example:

```
/BEGIN-PROCEDURE PAR=YES(PROC-PAR=(&CMD='STA',&CMDPAR='L'))
  
  WRITE-TEXT T='-----> ''&CMD &CMDPAR''
  &CMD &CMDPAR
  SKIP-COMMAND TO-LABEL=OK
  
  SET-JOB-STEP
  MOD-JV JV-CONTENTS=#BS2DOJV,-
      SET-VALUE=C'Command ''&CMD &CMDPAR'' failed'
  SKIP-COMMAND TO-LABEL=ERR
  
  .ERR REMARK
  WRITE-TEXT T='-----> DOING EXIT-PROCEDURE ERROR=YES'
  EXIT-PROCEDURE ERROR=YES
  
  .OK REMARK
  WRITE-TEXT T='-----> DOING EXIT-PROCEDURE ERROR=NO'
  EXIT-PROCEDURE ERROR=NO
  
END-PROCEDURE
```
Example 1  Procedure `proc` in POSIX, simple evaluation of the termination status

```plaintext
DO: bs2do proc
%  JMS0066 JOB ODO492C ACCEPTED ON 04-04-23 AT 11:28, TSN=57EE
DO: echo $? 
0
```

Example 2  Procedure `proc` in BS2000 file, SYSOUT is output in POSIX

```plaintext
DO: bs2do -o example2.out bs2:proc
%  JMS0066 JOB ODO494C ACCEPTED ON 04-04-23 AT 11:29, TSN=57EG
DO: cat example2.out
/CREATE-JV JV=#BS2DOJV
/SET-JOB-STEP
/DEFINE-JV JV=#BS2DOJV,SET-VALUE=C'
/SET-JOB-STEP
/CALL-PROC FROM-FILE=proc,LOGGING=NO
------- 'STA L'
```

```
NAME  TSN  TYPE PRI CPU-USED CPU-MAX ACCOUNT#
DO494  57EG 2 BATCH  9  220  0.6044  32000 1
RLOGIN  57C7 2 DIALOG  0  210 14.3916 9999 1
%  SPS0171 NO LOCAL SPOOLOUT JOB PRESENT
%  SPS0420 RSO WARNING: SOME RSO PRINT-JOBS CANNOT BE DISPLAYED
%  SCP1095 DPRINTSV WARNING: SOME DPRINT PRINT-JOBS CANNOT BE ..
------- DOING EXIT-PROCEDURE ERROR=NO
/Skip-commands TO-LABEL=NOERR
/.NOERR  REMARK
/SYSFILE SYSOUT=*DUMMY
```
Example 3  Procedure `proc` in BS2000 PLAM library, erroneous procedure parameter

```bash
DO: bs2cp proc 'bs2:BS2LIB(PROC,J)'  # copy proc in PLAM library
DO: bs2do -o example3.out bs2:/(BS2LIB,PROC) "CMDPAR='x x'"
%  JMS0066 JOB 'DO501' ACCEPTED ON 04-04-23 AT 12:51, TSN = 57EN
BS2DOJV: Command 'STA x x' failed
DO:echo $?  
1
DO: cat example3.out
/CREATE-JV JV=#BS2DOJV
/SET-JOB-STEP
/MODIFY-JV JV=#BS2DOJV,SET-VALUE=C' '
/SET-JOB-STEP
/CALL-PROC FROM-FILE=(BS2LIB,PROC),PROC-PAR=(CMDPAR='x x'),LOGGING=..
  -----> 'STA x x'
%  EXC0898 INVALID OPERAND IN COMMAND. COMMAND REJECTED
%  CMD0205 ERROR IN PRECEDING COMMAND OR PROGRAM AND PROCEDURE ..
  -----> DOING EXIT-PROCEDURE ERROR=YES
%  CMD0205 ERROR IN PRECEDING COMMAND OR PROGRAM AND PROCEDURE..
/SET-JOB-STEP
/SKIP-COMMANDS TO-LABEL=ERROR
/.ERROR  REMARK
/SYSFILE SYSOUT=*DUMMY
```

Example 4  Procedure `proc` in POSIX, erroneous procedure parameter, redirection of `stderr`

```bash
DO: bs2do proc "CMD='STA P',CMDPAR=',TYPE=x " 2>example4.err
DO: echo $?  
1
DO: cat example4.err
%  JMS0066 JOB 'DO620' ACCEPTED ON 04-04-23 AT 15:43, TSN = 57H1
BS2DOJV: Command 'STA P ,TYPE=x' failed
```

**bs2file**  set BS2000 file attributes *(BS2000)*

*bs2file* defines the BS2000 file attributes of the file which is to be transferred.  
*bs2file* is mapped to the BS2000 FILE command.

**Syntax**

```
bs2file[...h]..file,operand_list
```

- **-h**  Prints out the command syntax with an explanation of the options.

**file**

Fully qualified file name in BS2000. The file attributes of this file are defined and used in the associated *bs2cp* command.

Specifying "\*" ("\" is used for quoting) instead of a file name indicates that the settings in the *bs2file* command are valid for all files of the next *bs2cp* command. The file attributes then apply to the file(s) specified in the *bs2cp* command.

If a fully qualified file name is specified, the *bs2file* entry only loses its validity if the file attributes are successfully analyzed in a *bs2cp* call when the file names are identical. If an asterisk (*) is specified, the *bs2file* entry loses its validity when the *bs2cp* command terminates.

**operand_list**

List of file attributes. The format of the *operand_list* must correspond to the format of the FILE command (BS2000).

You may not use the operand "LINK=" in the FILE command.

**Hint**

If you issue multiple *bs2file* commands for a file then only the last attributes to be specified are valid.

In the case of ISAM files, only the values KEYPOS=5 and KEYLEN=8 are supported for the key position and key length respectively.

**Error**

Error messages of the FILE command

**Example**

The BS2000 file *doccopy* is to be set up as a PAM file.

```
$ bs2file doccopy,fcbtype=pam
```

**See also**

*bs2cp, ftyp*
bs2lp

send files to a printer (BS2000)

Print jobs are sent to the BS2000 spool via the PRNT macro. No ID is allocated for the print job. Management of the print job is only possible using the BS2000 spool.

Syntax

```
bs2lp[...option][...file] ...
```

option

- `-c` This option is always set implicitly. The specified file is copied and the copies are printed.

- `-d` Output medium. You can specify a printer name or a pool name (printer class name). The environment variables `LPDEST` and `PRINTER` are not supported.

- `-ncopies` Use this option to specify the number of times the file is to be printed. The largest supported value for `copies` is 255, the smallest is 1.

  `-ncopies` not specified:
  A value of 1 is assumed for the number of copies.

- `-title`

  `(t - title) title` will be printed in the header of the printout.

- `-m`, `-o`, `-s`, `-w`

  Ignored.

file

Name of the file which is to be printed. You can also enter more than one file. These files are then printed in the order in which they are specified when the command is called.

You must specify any options which you only wish to apply to one file before entering the name of the file in question.

`file` not specified:

`bs2lp` reads from the standard input.

Mode of operation

The POSIX file is copied as a temporary SAM file, i.e. the ERASE operand is always implicitly set when SPOOL is called.
Specifying additional operands for print jobs

All keyword operands of the PRNT macro can be implicitly specified through the .lprc file which is searched for in the current HOME directory. At least one PRNT operand must be fully defined for each line. A line may contain more than one operand if these are separated by commas. The entries in the .lprc file are transferred to PRNT without checking.

Example .lprc file:

```
FORM=STD
LINES=80
ROT=90, CHARS=R01
```

You cannot simultaneously specify an operand and an option which is mapped to this operand.

Option mapping:

- the option `-d` is mapped to the BS2000 operand DEST
- the option `-n` is mapped to the BS2000 operand COPIES
- the option `-t` is mapped to the BS2000 operand TEXT.

Error

`bs2lp: ERROR: No (or empty) input files`

Exit status

0, if spool jobs were started successfully.

Example

The command `bs2lp -n2 file` and the entries in the .lprc file described above result in the following spool call for BS2000:

```
PRNT file, COPIES=2, FORM=STD, LINES=80, ROT=90, CHARs=R01
```
You can set the P keys $\text{P3}$ and $\text{P4}$ as follows by calling the command $\text{bs2pkey}$

- $\text{P3}$ with $\text{@c}$ ($\text{CTRL}[C]$)
- $\text{P4}$ with $\text{@d}$ ($\text{CTRL}[D]$)

This command has no options.

**Syntax**

```bash
bs2pkey
```

The program is called either in the POSIX shell (with no options) or can be included in the file `/etc/profile`. The program is then activated every time the shell is called.
**cal**

**print calendar**

`cal` writes a calendar on the standard output.

**Syntax**

```
[ [ month ] [ year ] ]
```

**No argument specified:**

`cal` prints the calendar for the current month.

**month**

`cal` prints the calendar for the specified month.

The possible values for `month` are 1 to 12.

If you specify a value for `month`, you must also specify a value for `year`.

**year**

The calendar for the specified year is printed.

The possible values for `year` are 1 to 9999.

Please note that `cal 10`, for example, refers to 10 A.D., not 2010.

If you call `cal` using the argument `1752` or the arguments `9 1752`, the output for September is shorter than usual. This is because `cal` allows for an adjustment of 11 days which took place in that month in 1752.

**Exit status**

The exit status is non-zero if the values specified for `year` or `month` lie outside the permissible range.

**Error**

Bad argument [ for month | for year ]

The values you have specified for `year` or `month` are not within the permissible range.

**Variable**

`TZ`

Determine the timezone used to calculate the value for the current month.

**Locale**

The following environment variables affect the execution of `cal`:

`LANG`

Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

`LC_ALL`

If set to a non-empty string value, override the values of all the other internationalization variables.
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments). \textit{LC\_CTYPE} governs character classes, character conversion (shifting) and the behavior of character classes in regular expressions.

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

Determine the format and contents of the calendar.

Determine the location of message catalogs for the processing of \textit{LC\_MESSAGES}.

Example

Print the calendar for January 2009 in the Locale C:

```
$ cal 1 2009
January 2009
 Su Mo Tu We Th Fr Sa
 1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```
cancel  cancel line printer requests

You can use cancel to abort or cancel any of the print jobs that you submitted with the lp command.

Nonprivileged users can abort or cancel only print jobs submitted under their own login names.

Information on print jobs is displayed by the lpstat command.

Syntax

Format 1: cancel[jobno...]printer...
Format 2: cancel jobno...[printer...]

You must specify a value for at least one of the operands.

jobno

In jobno you specify the job number of a print job. The job number is the number that is reported on standard output when you submit a print job with the lp command. You may specify more than one job number, using blanks to separate them. If necessary, you can obtain a list of all pending print jobs with the lpstat command. Nonprivileged users can request information only about the status of print jobs submitted under their own login names. cancel issues a message on standard output to tell you which print jobs have been cancelled or aborted.

printer

In printer you specify the name of a RSO printer or a printer group. You may specify more than one name. Any print job currently running on printer is aborted, and the printer is freed for the next job. cancel issues a message on standard output to tell you which print jobs have been aborted. You can use lpstat to find out which printers currently have jobs of the own userid running on them.

Error

cancel: printer 'G005' was not busy

None of the printers in the printer group that you specified, G005 in this case, had a print job running on it. When you specify a printer group name, you can only terminate a job that is currently printing on a printer in the named group. You cannot use this method to cancel print jobs that have been submitted but are still waiting to be executed. To cancel queued print jobs, you must specify the job number.

cancel: request 'TSN-AB35' non existent

The job number you specified in the cancel call, TSN-AB35 in this case, is not the number of an existing print job. Use lpstat to obtain a list of all currently executing and pending print jobs.
cancel

UX:cancel: WARNING: "XY" is not a request id or a printer
To fix: Cancel requests by id or by name of printer where printing.
When you called cancel, you specified an argument XY which is neither a job number nor
the name of a printer group.

Locale

The following environment variables affect the execution of cancel:

**LANG**
Provide a default value for the internationalization variables that are unset
or null. If LANG is unset or null, the corresponding value from the default
locale will be used. If any of the variables contains an invalid setting, the
utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other inter-
nationalization variables.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data
as characters (for example, single- as opposed to multi-byte characters in
arguments).

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents
of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of
LC_MESSAGES.

Example 1
You inadvertently submit the same print job twice, so you want to cancel the second one.

```
$ lp test test
request id is TSN-6J1V (test)
request id is TSN-6J1W (test)
$ cancel tsn-6j1w
request "TSN-6J1W" cancelled
```

Example 2
Your system has two printer groups, G001 and G002, each of which consists of one printer.
The job currently executing on printer group G001 is to be aborted:

```
$ cancel G001
request "TSN-1234" cancelled
```
The job number of the terminated print job was TSN-1234.
The print jobs with job numbers TSN-9WJ7 and TSN-9WAS are to be cancelled:

```
$ cancel TSN-9WJ7 TSN-9WAS
request "TSN-9WJ7" cancelled
request "TSN-9WAS" cancelled
```

See also  *lp, lpstat [11]*
cat  concatenate and print files

The cat command reads files in sequence and writes them to standard output. cat has no effect on the sequence and format of the characters in the files.
If you name more than one file when calling cat, these files are output sequentially in the specified order. If you do not name a file, cat reads from standard input.

Syntax  cat[[-s][-u][...file]...]

No option specified:
Output is buffered in BUFSIZ-byte blocks. The value of BUFSIZ is governed by the machine you are working on. It is defined in the file /usr/include/stdio.h and may be 8192 bytes. If the files named on the command line do not exist, cat tells you that it cannot open them.

-s  Messages reporting that files do not exist are suppressed.
-u  Output without buffering, one byte at a time.

file  Name of the file that is to be printed. You may specify more than one file. If you use a dash as the name for file, cat reads from standard input.

file not specified:
cat reads from standard input.

Caution!
Redirecting the output of cat to one of the files being read will result in the loss of that file's original contents. In the following command, for example, the contents of file1 are lost:
cat file1 file2 file3 > file1

Error  cat >out_file out_file: cannot create
You have no write permission for the output file out_file or for the directory which contains out_file.

cat in_file cat: cannot open in_file: Permission denied
You have no read permission for the input file in_file.

Locale  The following environment variables affect the execution of cat:

LANG  Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL  If set to a non-empty string value, override the values of all the other internationalization variables.
**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments). **LC_CTYPE** governs character classes, character conversion (shifting) and the behavior of character classes in regular expressions.

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1**
Concatenate and redirect the output of two files:
```
$ echo Monday Tuesday Wednesday > file1
$ echo Thursday Friday Saturday > file2
$ cat file1 file2 > file3
$ cat file3
Monday Tuesday Wednesday
Thursday Friday Saturday
```

**Example 2**
Display the contents of **file1**
```
$ cat file1
In Xanadu did Kubla Khan
A stately pleasure dome decree:
Where Alph, the sacred river, ran

Now write two lines of text into **file2**. Terminate input with [END] or @@d depending on the terminal type used.
```
```
$ cat > file2
Through caverns measureless to man
Down to a sunless sea.

Now move the contents of **file1** and **file2** to **file3**, add two lines from standard input, and then print the contents of **file3**. Terminate input with [END] or @@d depending on the terminal type used.
```
```
$ cat file1 file2 - > file3
For he on honey dew hath fed
And drunk the milk of paradise.
```
```
$ cat file3
In Xanadu did Kubla Khan
A stately pleasure dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.
For he on honey dew hath fed
And drunk the milk of paradise.
```

*See also*  
**cp, pr**
cd  change working directory

The built-in `cd` command in the POSIX shell `sh` makes the specified directory your current working directory.
The `cd` command is rejected in a restricted shell.

### Syntax

**Format 1:** `cd[...directory]`

**Format 2:** `cd.-`

**Format 3:** `cd..old..new`

### Format 1

**Change directory using CPATH**

`cd[..directory]`

*directory*

Name of the directory that is to become your current working directory. You must have execute permission for this directory. If you specify a relative or absolute path name for *directory*, you must have execute permission for all the directories which make up this path name.

If the name of the specified directory begins with one of the following characters, the command looks for the directory without reference to the `CDPATH` environment variable (see POSIX environment variables):

- `/`  means that the search begins in the root directory.
- `./`  means that the search begins in the current directory.
- `../` means that the search begins in the parent directory.

If the name of the specified directory does not begin with any of the above characters, `cd` evaluates the `CDPATH` environment variable:

- If the `CDPATH` variable has not been defined or is null, `cd` looks for the specified directory relative to the current working directory.
- If the `CDPATH` variable has been assigned a value, `cd` looks for the specified directory sequentially in the directories whose paths are defined in the `CDPATH` variable. On finding the directory, `cd` writes the absolute path name of this directory on standard output before switching to it.

*directory* not specified:
The `cd` command puts you in your home directory. The home directory is identical to the login directory unless there is a different path name assigned to the shell variable `HOME`.

### Format 2

`cd.-`

- Specifying the `-` as an operand has the same effect as the command

  ```
  cd "$OLDPWD" && pwd
  ```

  which changes to the last active directory and writes its name.
Format 3  

Change directory with text substitution

```
cd ..old..new
```

cd substitutes the string new for the string old in the current directory name (PWD) and tries to change to this new directory.

Error

```
sh: file: not found
The specified directory does not exist. You can verify this with ls -l.

sh: file: not a directory
Your argument is not a directory. This can also be verified with ls -l.

file: permission denied
You do not have execute permission for the specified directory.

If you have specified a relative or absolute path name for directory, you do not have execute permission for one of the directories which make up this path name.

rsh: cd: restricted

cd has been rejected because you are working in a restricted shell.
```

Variable

```
HOME
contains the absolute path name of your home directory.

CDPATH
You can assign to CDPATH the absolute path names of directories that cd is to search. By default this variable is undefined.

OLDPWD
Path name of the previous directory used by cd -.

PWD
Path name of the current directory. This name is set by cd following the change to this directory.
```

Locale

The following environment variables affect the execution of cd:

```
LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
```
Built-in sh command

\texttt{cd}

\begin{itemize}
\item \textit{LC\_MESSAGES}\hfill \textit{NLSPATH}
\end{itemize}

\texttt{LC\_MESSAGES} 
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

\texttt{NLSPATH} 
Determine the location of message catalogs for the processing of \textit{LC\_MESSAGES}.

\textbf{Example 1} 
The following entry makes the subdirectory \texttt{dates} the current directory:

$\$
cd\ dates
$\$
pwd
\hspace{1cm}/home/rose\/dates

\textbf{Example 2} 
User \texttt{rose} has redefined the \textit{CDPATH} environment variable. She now wishes to change to her subdirectory \texttt{usr}, but with the following commands ends up in the directory \texttt{/usr} instead:

$\$
echo\ \$\texttt{CDPATH}
\hspace{1cm}/:/home/rose\/dates:.\$
$\$
pwd
\hspace{1cm}/home/rose\$
$\$
l\ -l
\hspace{1cm}drwxrwxr-x 2 ROSE 144 Feb 28 12:32 \texttt{usr}
\hspace{1cm}drwxrwxr-x 2 ROSE 192 Feb 28 11:51 \texttt{dates}
\hspace{1cm}-rwx------ 1 ROSE 11734 Mar 7 16:22 \texttt{tests}
\hspace{1cm}.
\hspace{1cm}.
\hspace{1cm}.
$\$
cd\ \texttt{usr}
$\$
pwd
\hspace{1cm}/usr

The \texttt{usr} directory is first looked for in the directories whose path names are assigned to the \textit{CDPATH} variable. In this case, \textit{CDPATH} contains a / for the root directory as the first path name. The current directory is the last to be searched by \texttt{cd}.

User \texttt{rose} can prevent \texttt{cd} from evaluating the \textit{CDPATH} environment variable by formulating the command in the following way:

$\$
cd\ .\texttt{/usr}
$\$
pwd
\hspace{1cm}/home/rose\texttt{/usr}

\textbf{See also} \texttt{pwd} \hspace{1cm} \texttt{chdir()} [4]
**chgrp**

**change file group ownership**

`chgrp` changes the user group for a file or a directory. You can only use it if you are the file/directory owner or the POSIX administrator.

Only the POSIX administrator is permitted to change the user group for each file as he or she wishes.

There is an operating system configuration option `_POSIX_CHOWN_RESTRICTED` which can be used to place a restriction on the groups to which users without POSIX administrator privileges can reassign their files. If this option is in effect, as an ordinary user you can assign your files to another group only if

- you are listed in the `/etc/group` file as a member of the new group (see page 203), and
- you currently belong to the new group, which means that before calling `chgrp` you must change to the new user group with the `newgrp` command (see section “newgrp change to a new group” on page 568 and the example on page 203).

If `chgrp` is called by a user who does not posses POSIX administrator permissions then all the s-bits (set-user-ID and set-group-ID, see section “chmod change file modes” on page 204) set for the specified files are reset.

**Syntax**

```
chgrp[-h][-R] gid file...
```

- `-h` If `file` is a symbolic link, `chgrp` changes the group ID of the symbolic link itself. Without this option, the group ID of the file referenced by the symbolic link is changed.

- `-R` (recursive) `chgrp` recursively descends through the specified directories, changing the group ID as it proceeds and traversing any symbolic links that it encounters.

- `gid` (group id). New group name or new group ID. `gid` must appear in `/etc/group`.

- `file` Name of the file or directory for which the user group is to be redefined. You can also list any number of files and/or directories.

**Error**

`file`: Not super-user

You are not permitted to change the user group of the specified file, since you are not the owner of the file or have not been entered as a member of the specified group or do not currently belong to the group. Only the POSIX administrator is authorized to redefine the group for all files.

`chgrp`: unknown group: `gid`

The group name you have specified for `gid` is not in the `/etc/group` file.
File

The group file /etc/group contains a list of all existing user groups. Each line of this file consists of four colon-separated fields: `groupname:groupid:user,user...`

Only the POSIX administrator is permitted to create new user groups and to enter new group members.

Locale

The following environment variables affect the execution of `chgrp`:

- **LANG**: Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**: If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**: Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example

You are currently working under the login name `cindy`; this name is entered in the /etc/group file as a member of the user groups `ag` and `prog`. At present, you belong to the user group `ag` as is evident from the fact that the name `ag` is entered for "group" when you create new files:

```
$ >file
$ ls -l file
-rw------- 1 CINDY ag 0 Feb 17 15:48 file
```

You now wish to change the user group for `file`; the new group is to be `prog`. To do this, you first use the `newgrp` command to switch to the `prog` group and then change the group for `file` with `chgrp`.

```
$ newgrp prog
$ chgrp prog file
$ ls -l file
-rw------- 1 CINDY prog 0 Feb 17 15:48 file
```

See also `chmod`, `chown`, `id`, `newgrp`
**chmod**  change file modes

`chmod` is used to change the permissions of a file (its protection mode).

Only the owner or the POSIX administrator is authorized to change the permissions of a file. A file’s set-group-id bit can be set only by a user whose current group ID is the same as the file’s group ID (see section “chgrp change file group ownership” on page 202 and section “newgrp change to a new group” on page 568).

**Syntax**

```
chmod [-R]...mode...file....
```

- `-R` (recursive) `chmod` recursively descends through the specified directories, changing the mode for each file it encounters.
- `mode`  In `mode` you specify how the permissions for one or more named files are to be changed. There are two forms of `mode` specification:
  - symbolic
  - absolute
- `file`  Name of the file for which you wish to define or alter the permissions. `file` may also be a directory. Multiple names are also allowed.

**Symbolic form**

```
[who]op[permission][,[who]op[permission]]...
```

**who**  In `who` you say who the file permissions apply to. The choices for `who` are:

- `u`  for the owner (user)
- `g`  for the group
- `o`  for others
- `a`  for `ugo`, i.e. all users.

or any combination of the letters `u`, `g`, `o`.

**who not specified:**

`who` defaults to `ugo`, i.e. all users. The permissions for `ugo` are set with allowance for the bits in the file-creation mode mask (see `umask`).

op

In op you specify whether permissions are to be granted, left unchanged, or revoked. The choices for op are:

+ to add (grant) permissions,
- to take away (revoke) permissions,
= to assign permissions absolutely, i.e. only the listed permissions are granted, all others are revoked.

permission

In permission you specify which permission(s) you wish to grant or revoke. The choices for permission are:

r for read permission
w for write permission
x (x - execute) for execute permission or for permission to browse files and directories.
X for execute permission or for permission to search in a file, if at least one x bit is set, or in directories. If the file is not a directory, or if no x bit has been set for the file, this option is ignored.
s for the set-user-ID or set-group-ID bit. Entering s in a chmod command is only useful in conjunction with u, g, or ug (if who is not specified, it defaults to ug). Set-user-ID and set-group-ID bits only apply to executable binary files (not to shell scripts) (see The s bits).
t for sticky bit (t bit). Only the POSIX administrator is able to set the sticky bit. Attempts by nonprivileged users to set the sticky bit are ignored. Entering i in a chmod command is useful only in combination with u or a or if who is not defined. A set sticky bit only applies to executable files (see The sticky bit). If you change the mode of a file which has the sticky bit set, the sticky bit is automatically cleared.
l (lock) for mandatory locking of files, directories or records, referring to a file’s ability to have its reading or writing permissions locked while a program is accessing it. file’s l bit can be set only if its group execute permission is not set and its set-group-id bit is set.

Thus the following examples are not correct and would result in error messages:

chmod g+x,+l file
chmod g+s,+l file
chmod

Built-in sh command

- `u` Use the permissions of the current owner.
- `g` Use the permissions of the current group.
- `o` Use the permissions of the current others mode.

Permission not specified:
This is only useful in combination with the `=` operator; all permissions are then revoked for the `who` in question.

As indicated above, you can specify several "who-op-permission" arguments in a row, provided they are separated by commas as follows:

```
chmod g-w,o-rw file
```

The string you specify for `mode` is processed by `chmod` from left to right. For instance, `a-w,u+w` grants write permission to the owner, revoking it for all others.

Absolute form

An absolute `mode` is a three or four digit octal number. The admissible octal values are obtained by logically ORing (in binary) the octal modes shown below. The effect is the same as adding the modes in octal or decimal. A leading zero (neither the s bit nor the sticky bit set) may be omitted.

The specified permissions are granted; all other permissions are revoked.

- 4000 Set user ID on execution
- 20#0 Set group ID on execution if # is 7, 5, 3 or 1. Set mandatory locking if # is 6, 4, 2 or 0. The value of # is ignored if `file` is a directory. In this case you can only use the symbolic form for `mode`.
- 1000 sticky bit (t bit)
- 0400 read permission for owner
- 0200 write permission for owner
- 0100 execute permission (or directory search permission) for owner
- 0040 read permission for group
- 0020 write permission for group
- 0010 execute permission (or directory search permission) for group
- 0004 read permission for others
- 0002 write permission for others
- 0001 execute permission (or directory search permission) for others
Example

To grant read, write, and execute permission to the owner and read and execute permission to the group, you would enter a value of 750 for mode:

\[400 + 200 + 100 + 40 + 10 = 750\]

The file then has the permissions `rwxr-x---`

The s bit

When an executable program which has the set-user-ID bit set is called, the effective user ID of the associated process is the same as the user ID of the owner of the file (and not that of the caller). In other words, the process runs under the user ID of the program owner and can therefore also access files for which the caller of the program does not have explicit access permission.

The real user ID of the process remains that of the program caller.

If the set-group-ID bit is set, the effective group ID of the process is the same as the group ID of the program owner. This means that the process runs under the group ID of the program owner.

The real group ID of the process remains that of the program caller.

The s bits are only useful for executable binary files (executable programs) and not for shell scripts. Although `chmod` can be used to set the s bits for files that contain a shell script, the setting will essentially have no effect.

When changes are made to a file, the s bits are reset for security reasons.

Example of an s bit application

One example of an s bit application is the `mailx` command.

Messages sent with `mailx` to user USER1 are written to the file `/var/mail/USER1`. This file belongs to the group MAIL, and its owner is USER1. Both group and owner have read and write permission; no other users have any permissions for the file:

```
-rw-rw---- USER1 MAIL /var/mail/USER1
```

Thus ordinarily a user from a different group (USER2, for example) would be unable to write messages to this file.

However, since the `mailx` command has the s bit set for the MAIL group, after calling `mailx` USER2 temporarily becomes an effective member of the MAIL group and thus has write permission for `/var/mail/USER1`. 
The sticky bit (t bit)

Only the POSIX administrator is able to set the sticky bit (t bit). Attempts by non-privileged users to set the sticky bit are ignored.

The sticky bit only has an effect on directories and executable files. It is possible to use `chmod` to set it on other files, but it will then have no effect.

If the sticky bit is set on an executable file, it is possible to some extent to save on the overhead usually involved in reading a program in from the disk file again every time the program is started.

If a directory is writable and has the sticky bit set, files within the directory cannot be removed, renamed or linked to unless one or more of the following conditions apply:

- the file belongs to the user
- the directory belongs to the user
- the user has write permission for the file
- the user is a privileged user

In the output of the `ls -l` command, the sticky bit if set appears in the last position of the listed permissions. If the x bit has simultaneously been set for "other users", a t appears; if not, a T.

The l bit

The `lockf()` function allows a program to place a lock on a file which it is accessing. If this file has its l bit set, the function call results in mandatory locking of the file (see `lockf()` [4]).

Error

`chmod: ERROR: Invalid mode`

You have defined an illegal set of permissions for `chmod`.

`chmod: WARNING: Locking not permitted on file, a group executable file`

Files with group execute permission cannot also have the l bit set.

`chmod: WARNING: Execute permission required for set-ID on execution for file`

In order to turn on a file’s set-user-ID bit you need to have execute permission for the file.

Locale

The following environment variables affect the execution of `chmod`:

- `LANG`: Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `LC_ALL`: If set to a non-empty string value, override the values of all the other internationalization variables.
Built-in sh command

**chmod**

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example**
The following examples all refer to a file with the permissions `rw-------`. The first two columns of the table contain possible *mode* arguments; the last column shows the result of a `chmod` call using these arguments.

<table>
<thead>
<tr>
<th>Symbolic form</th>
<th>Absolute form</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-w</td>
<td>400</td>
<td>r----------</td>
</tr>
<tr>
<td>-w</td>
<td>400</td>
<td>r----------</td>
</tr>
<tr>
<td>go+r</td>
<td>644</td>
<td>rw-r--r--</td>
</tr>
<tr>
<td>go=r</td>
<td>644</td>
<td>rw-r--r--</td>
</tr>
<tr>
<td>go+rw</td>
<td>666</td>
<td>rw-rw-r--</td>
</tr>
<tr>
<td>=rw</td>
<td>666</td>
<td>rw-rw-r--</td>
</tr>
<tr>
<td>+rx</td>
<td>755</td>
<td>rwxr-x-x</td>
</tr>
<tr>
<td>=r</td>
<td>444</td>
<td>r----------</td>
</tr>
<tr>
<td>ug=rw,o=r</td>
<td>664</td>
<td>rw-rw-r--</td>
</tr>
<tr>
<td>u=rwx,g=rx,o=</td>
<td>750</td>
<td>rwxr-x-x----</td>
</tr>
<tr>
<td>+x,u+s</td>
<td>4711</td>
<td>rw+s-x-x-x</td>
</tr>
<tr>
<td>+xt</td>
<td>1711</td>
<td>rw+x-x-x----</td>
</tr>
</tbody>
</table>

The sticky bit (last example) can only be set by the POSIX administrator. Attempts by non-privileged users to set the sticky bit are ignored.

**See also**
- `chgrp`, `ls`, `newgrp`, `umask`
- `chmod()`, `chown()` [4]
chown  change file ownership

chown assigns a new owner to a file or a directory.

Only the POSIX administrator may change the owners of a file as he or she wishes. In contrast, users who do not possess POSIX administrator permissions may only change the owners of their own files. If the operating system option _POSIX_CHOWN_RESTRICTED is in effect, even the owner of a file is prevented from changing the ownership of that file. If chown is called by a user without POSIX administrator permissions, the s-bit for owners, 4000, is reset.

Syntax

chown[[-h][-R]..uid[:gid]..file.....]

-h  If file is a symbolic link, chown changes the owner of the symbolic link. Without this option, the owner of the file referenced by the symbolic link is changed.

-R  (recursive) chown recursively descends through the specified directories, changing the file owner as it proceeds and traversing any symbolic links that it encounters.

uid[:gid]
    Login name or user ID of the new owner. Optionally, the group ID may also be specified.

debug  Name of the file that is to have a new owner. Directories and multiple file names are also allowed.

Error

chown: file: Not super-user
You may not change the owner of the file file since you are not the owner of file or the _POSIX_CHOWN_RESTRICTED variable is set on your system.

chown: Unknown user: newowner
You have entered a user ID for newowner which is not entered in the user table.

Locale

The following environment variables affect the execution of chown:

LANG      Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL    If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
Built-in sh command

chown

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example**
You are working as the POSIX administrator and want to change the owners of a file. You wish to assign the new owner `MARK` to the file `text1` whose owner is `CATHY`. To do this, enter:

```
# ls -l text1
-rw------- 1 CATHY  ag  2426  Feb 17 15:48 text1
```

```
# chown mark text1
```

```
# ls -l text1
-rw------- 1 MARK  ag  2426  Feb 17 15:48 text1
```

**See also**
chgrp, chmod
chown() [4]
**cksum**

write file checksums and sizes

The cksum utility calculates and writes to standard output a cyclic redundancy check (CRC) for each input file, and also writes to standard output the number of octets in each file. The CRC used is based on the polynomial used for CRC error checking in the referenced Ethernet standard.

**Syntax**

`cksum[-C][...file...]`

**options**

-C  
Calculates the CRC checksum in the same way as in previous versions.

**file**

A pathname of a file to be checked. The files may be of any type.

file not specified:

The standard input is used. You must use `END` or `@`d to terminate input from standard input.

**Encoding**

The encoding for the CRC checksum is defined by the generating polynomial:

\[ G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \]

Mathematically, the CRC value corresponding to a given file is defined by the following procedure:

1. The \( n \) bits to be evaluated are considered to be the coefficients of a mod 2 polynomial \( M(x) \) of degree \( n - 1 \). These \( n \) bits are the bits from the file, with the most significant bit being the most significant bit of the first octet of the file and the last bit being the least significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral number of octets, followed by one or more octets representing the length of the file as a binary value, least significant octet first. The smallest number of octets capable of representing this integer is used.

2. \( M(x) \) is multiplied by \( x^{32} \) (that is, shifted left 32 bits) and divided by \( G(x) \) using mod 2 division, producing a remainder \( R(x) \) of degree \( \leq 31 \).

3. The coefficients of \( R(x) \) are considered to be a 32-bit sequence.

4. The bit sequence is complemented and the result is the CRC.
Locale

The following environment variables affect the execution of `cksum`:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Stdout

For each file processed successfully, the `cksum` utility will write in the following format:

`%u %d %s
`, <ckecksum>, <# of octets>, <pathname>

If no file operand was specified, the pathname and its leading space will be omitted.

Exit status

0   All files were processed successfully.
>0   An error occurred.

Hint

`cksum` utility is typically used to quickly compare a suspect file against a trusted version of the same, such as to ensure that files transmitted over noisy media arrive intact. However, this comparison cannot be considered cryptographically secure. The chances of a damaged file producing the same CRC as the original are astronomically small; deliberate deception is difficult, but probably not impossible.

Although input files to `cksum` can be any type, the results need not be what would be expected on character special device files or on file types not described by the XSH specification. Since this document does not specify the block size used when doing input, checksums of character special files need not process all of the data in those files.

The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted between two systems and undergoes any data transformation (such as moving 8-bit characters into 9-bit bytes), identical CRC values cannot be expected. Implementations performing such transformations may extend `cksum` to handle such situations.
cksum

Example  Calculating the checksum for *test*, a file containing a list of the days of the week.

```
$ cat test
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
$ cksum test
1222782406 57 test
```

See also *sum*
**cmp**

**cmp** **compare two files**

The `cmp` command does a comparison of two files byte by byte (character by character). If the files differ, `cmp` reports the differences on standard output. If the files are identical, `cmp` remains silent.

**Syntax**

```bash
cmp[-l][-s] file1 file2
```

No option specified

If the files are identical, `cmp` remains silent.

If the files differ, `cmp` indicates the byte (character) and line number of the first difference that it detects between `file1` and `file2` as shown below:

`file1` `file2` differ: char bytenumber, line linenumber

- **-l**
  
  All differences are reported in the following form:

  ```
  bytenumber   byte(file1)   byte(file2)
  ```

  `bytenumber` represents the displacement of the difference from the beginning of the file. The first byte of the file is assigned number 1, and blanks are counted. `bytenumber` is given in decimal notation.

  The `byte` columns show the bytes which differ between `file1` and `file2` and are in octal notation. An ASCII table of octal-coded values is provided in `Tables and directories`. If the files are identical, nothing is output.

- **-s**
  
  `cmp` remains silent. The exit status value is returned but not automatically displayed on the screen. You can enter the command `echo $?` to query the exit status.

**file1**, **file2**

The names of the files that you wish to compare.

- If you use a dash - as the name for `file1`, `cmp` reads from standard input and compares your input with `file2`.

- If one of the two files ends before `cmp` can detect a difference, `cmp` reports that the end of the file has been reached in the shorter file by issuing the following message:

  ```
  cmp: EOF on file
  ```

- If one character is missing in one of two otherwise identical files, `cmp -l` reports all following bytes as differences. This is due to the shift in character positions.

- The first character of a file is assigned `bytenumber` 1, not 0.

- Blanks and newline characters are included in the `bytenumber` count.

**Exit status**

- 0 files identical
- 1 files differ
- 2 inaccessible file or missing argument
Error

cmp: cannot open file
You do not have read permission for one of the files, or one of the files does not exist.

Locale

The following environment variables affect the execution of cmp:

LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH
Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1

Compare two files and print the differing bytes (octal) and their bytenumber (decimal).

$ echo 1 2 3 4 5 7 8 a > file1
$ echo 1 2 3 4 5 6 9 a > file2
$ cmp -l file1 file2
11 367 366
13 370 371
$

Example 2

The shell script delete.eq compares two files and deletes one of them if they are identical.

if cmp -s $1 $2
then
  rm $2
fi

When you call the script with

$ delete.eq file1 file2

you pass file1 and file2 to it as positional operands. The -s option causes cmp to return the exit status. If the value of the exit status is 0 (=true), file2 is deleted; otherwise, it is retained.

See also  

comm, diff
**comm**

**select or reject lines common to two files**

`comm` compares two files in which the lines are sorted on the basis of the currently valid collating sequence. Sorting can be performed with the `sort` command (see page 688).

**Syntax**

```
comm[[-123]..file1..file2]
```

No option specified

`comm` produces three columns with the following meanings:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>lines which occur in file1 only</td>
<td>lines which occur in file2 only</td>
<td>lines which occur in both files</td>
</tr>
</tbody>
</table>

**option**

- `-1` Column 1 is not output.
- `-2` Column 2 is not output.
- `-3` Column 3 is not output.

Combinations of options 1, 2 and 3 are also permitted, e.g.:

- `-12` `comm` outputs all lines common to both files.
- `-23` `comm` outputs all lines which only occur in file1.
- `-13` `comm` outputs all lines which only occur in file2.
- `-123` `comm` generates no output.

**file1..file2**

Names of the two sorted files which you want to compare.

The `comm` command will not function properly unless both files have been sorted. If you use a dash as one of the names, `comm` reads from standard input.
Locale  The following environment variables affect the execution of comm:

**LANG**  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**  If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_COLLATE**  Determine the locale for the collating sequence comm expects to have been used when the input files were sorted.

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  Determine the location of message catalogs for the processing of LC_MESSAGES.
Example

The file *books* contains the titles of books and their authors. Each line contains the title of one book and the name of its author, with a space between them. You would now like to search the file *books* for a number of authors whose names you have listed in the file *authors1*. The contents of *books* and *authors1* are as follows:

books  
'Gormenghast' Peake  
'Buddenbrooks' Mann  
'Noddy' Blyton  
'Ulysses' Joyce  

authors1  
Blyton  
Gogol  
Joyce  
Kafka  
Mann  
Tolstoy

You can now proceed as follows:

- Use `awk` to extract the authors from *books*.
- Sort the authors in *books* using `sort`.
- Redirect the output of `sort` to the new file *authors2*.
- Compare the files *authors1* and *authors2* using `comm -2`.

```
$ awk '{printf"%s\n",$2}' books | sort > authors2
```

The file *authors2* contains the following:

Blyton  
Joyce  
Mann  
Peake

```
$ comm -2 authors1 authors2
Blyton  
Gogol  
Joyce  
Kafka  
Mann  
Tolstoy
```

All authors which are only in *authors1* are output in column 1. Then come the contents of column 3, which lists all authors present in both files.

See also  
`cmp`, `diff`, `sort`, `uniq`
command  execute a simple command

`command` informs the shell that the arguments are to be executed as simple commands. The shell functions are not evaluated.

`command` also provides information about how a command name is interpreted by the shell (see Format 2).

**Syntax**

| Format 1: command [-p]_command_name [..argument ...] |
| Format 2: command [-v |..-V]_command_name |

**Format 1**

- **command**: The name of a command or a special built-in command.
- **argument**: One of the strings as an argument for `command_name`.

**Option**

- **-p**: The command search is performed using a default value for `PATH` which ensures that all standard commands can be found.

**Format 2**

- **command**: The name of a command or a special built-in command.

**Option**

- **-v**: A string is written to the standard output. This string specifies the path name or command which the shell uses to call `command_name` in the current shell environment.
  
  - Commands, regular built-in commands, `command_names` containing a slash as well as implementation-specific functions which are specified by the `PATH` variable are written as absolute path names.
  
  - Shell functions, special built-in commands and regular built-in commands which cannot be accessed via the definition in the `PATH` variable as well as reserved shell terms are specified as simple names.
  
  - An alias is written as a call line which represents the alias definition.
  
  - In other cases, no string is written to the output. The exit status specifies that the command could not be found.

The standard output is formatted as follows:

```
"%s\n", <path name or command>
```
-V  A string is written to the standard output. This string specifies how the shell interprets the name in the command_name operand in the current shell environment. Although the format of the string is not specified, it nevertheless defines which of the following categories command_name belongs to. The associated information is also defined:

- Commands, regular built-in commands and implementation-specific functions which are found via the PATH variable are identified as such. In this case the string contains the absolute path name.
- Other shell functions are identified as functions.
- Aliases are identified as aliases. The definitions are specified in the string.
- Special built-in commands are identified as special built-in commands.
- Regular built-in commands which cannot be accessed via the definition in the PATH variable are identified as regular built-in commands. (The term "regular" is optional.)
- Reserved shell terms are identified as reserved shell terms.

The standard output is formatted as follows:

"%s\n", <undefined>

command_name
The name of a command or of a special built-in command.

Field of application

The command search sequence can be used to disable functions, regular built-in commands and the path search sequence. This is necessary so that functions which share the same name for a command are able to call this command (instead of a recursive function call).

The default system path is available via getconf. Since under certain circumstances PATH must have been set before getconf is called, the following options are available:

command -p getconf _CS_PATH

Occasionally it can be desirable to suppress the properties of special built-in commands. For example:

command exec > non_writable_file

This input does not cause a non-interactive procedure to abort. The output status can be queried by the procedure.

Return code 127 is used for command, env, nohup, time and xargs if an error occurs or if command_name could not be found. This enables applications to determine whether it was impossible to find a command or whether the command was terminated with an error. The value 127 was selected because it is not generally used for any other meaning. Low values are generally used for "normal error conditions". Values higher than 128 may be mistaken...
for termination resulting from signal reception. For similar reasons, the value 126 was
selected to indicate that although the command was found, it could not be called. Many
procedures generate informative error messages which differentiate between 126 and 127.
This distinction between return codes 126 and 127 is based on experience of the Korn shell
which handles these codes as follows: 127 is used if all attempts to execute the command
exec fail with [ENOENT]; 126 is used if an attempt to execute exec fails for any other reason.

Since with the options -v and -V the output for command is generated in accordance with the
current shell environment, command is generally provided as a regular built-in shell
command. If the call is issued in a subshell or a special run environment such as:

(PATH=foo command -v)
nohup command -v

this does not always yield correct results. Thus, for example, in certain run environments
most implementations are unable to recognize alias names, functions or special built-in
commands if the call is made via the nohup or exec functions.

Two types of regular built-in commands may be present in a system. Both are described
separately under command. The description of a command search allows for the implemen-
tation of a default command as a regular built-in command provided that this is read at the
correct position in a PATH search. Thus, for example, the result of command -v true may be
the path name /bin/true or a similar path name. Other, implementation-specific commands
which are not defined in this documentation may also occur as built-in commands. No path
name is allocated to such commands. The output is identified as that of (regular) built-in
commands. In no case can applications execute the command exec; use nohup, disable the
output by means of another PATH etc.

Locale

The following environment variables affect the execution of command:

LANG Provides a default value for the internationalization variables that are unset
or null. If LANG is unset or null, the corresponding default value from the
internationalized environment is used. If one of the internationalization
variables contains an invalid setting, the command behaves as if none of the
variables have been defined.

LC_ALL If this variable has been assigned a value, i.e. it is not a null string, this value
overrides the values of all the other internationalization variables.

LC_TYPE Determines the internationalized environment for the interpretation of byte
sequences as characters (e.g. single-byte characters as opposed to multi-
byte characters in arguments).

LC_MESSAGES Determines the format and contents of error messages. The internation-
alized environment specified here is also valid for informative messages
written to standard output.
**Built-in sh command**

**command**

**NLSPATH**

Determines the position of message catalogs for the processing of
**LC_MESSAGES**.

**PATH**

Determines the search path used during the command search, except
where a default value has been set with **-p**.

**Example 1**

A version of `cd` is generated which outputs the current directory precisely once:

```bash
cd() {
    command cd "$$" >/dev/null
    pwd
}
```

**Example 2**

A "safe shell procedure" is started in which the procedure cannot be influenced by the parent:

```
IFS=''
# The preceding value should be <space><tab><newline>.
# IFS is set to the default value.
\unalias -a
# All possible aliases are reset.
# unalias is quoted to prevent the use of
# any alias.
unset -f command
# The command may not be a user function.
PATH="$(command -p getconf _CS_PATH):$PATH"
# Introduce PATH prefix.
# ...
```

Provided that the access permissions for the directories called by **PATH** are set correctly,
then at this point the procedure can make sure that the called commands actually corre-
spond to the commands which are to be addressed. Great care is taken here since it is
assumed that implementation-specific extensions may be present which permit user
functions when called. Although this possibility is not described in this documentation, such
an extension is perfectly possible. For example the variable **ENV** precedes the call to a
procedure with a user-defined start procedure. Such a procedure might, for example, define
functions which affect the application.

**See also**  
*sh, type*
compress

compress  compress files

compress reduces the size of files using adaptive Lempel-Ziv coding: recurrent strings in the
text are reduced to unique codes ranging from 9 to a maximum of 16 bits.

If the calling process has the appropriate privileges, the owner, access permissions, and
the dates of last access and modification of the specified files remain the same.

Each specified file is replaced by a file with the same basename plus a .Z extension.

The amount of compression obtained depends on the size of the input file, the value of bits
(see option -b below), and the distribution of recurrent substrings. Typically, files containing
only text or source code are reduced by 50-60%. Compression achieved with the Lempel-
Ziv approach is generally much better than that achieved with Huffman coding, and takes
less time to compute.

No compression is performed if
– the input file is not a regular file (if it is a directory, for example)
– there are other links to the input file
– the input file already has a .Z extension
– the .Z file to be created already exists, and compress is being executed in the background
– no savings can be achieved by compression

Files compressed with compress can be decompressed with uncompress.

zcat produces uncompressed output on the standard output, but leaves the compressed file
intact.

Syntax

Format 1: compress [-fcv][-b..bits][..file ...]
Format 2: compress [-fcv][-b..bits][..file]

No option specified
The specified files are compressed if storage space can be saved as a result.

option
-c Simply writes the output of compress to standard output without modifying or creating
any files. Only one file is allowed with -c.

-f (f - force) Forces compression even if no space savings will be achieved or the .Z file
that will be created already exists. If the .Z file already exists it will be overwritten.

-f not specified:
compress prompts to verify whether an existing .Z file should be overwritten. No prompt
is issued if compress is running in the background.

-v (verbose) Displays the percentage reduction for each file compressed:

date Compression: xx.xx% -- replaced with file.Z
compress

-b..bits
Sets the upper limit (in bits) for recurrent substring codes to bits. The value for bits must be between 9 and 16. Lowering the number of bits will result in larger, less compressed files.

The bits parameter specified during compression is encoded within the compressed file, together with a magic number to ensure that it is impossible to further compress a file which is already compressed.

-b not specified:
The value of maxbits defaults to 16.

Caution!
Although compressed files are compatible between machines with large memory, the -b..12 option should be used for file transfer to architectures with a small process data space (64 Kbyte or less).

file
Name of the file to be compressed. You may specify more than one file. If you specify a hyphen (-) for one of the files, compress will read that file from standard input.

file must not be a directory, and there must be no other links to it.

The compressed file is assigned the name file.Z, and the named file is deleted after successful compression. file.Z has the same access permissions, access time and modification time as file. The maximum length of the name of file depends on the file system being used. The maximum allowable length is equal to the maximum file name length supported by the file system, minus 2 characters. This allows for the extension to file.Z. If this length is exceeded, file will not be compressed.

file not specified or specified as a hyphen (-):
Data from standard input is written in compressed form to standard output.

Exit status
0  Compression successful
1  Error
2  Compression of one or more files not performed, as it would have increased the size of the file.
>2  Error
**compress**

**Error**

*file*: filename too long to tack on .Z

The name of the compressed file is too long. No compression takes place.

*file* -- not a regular file: unchanged

If the input file is not a regular file it is left uncompressed.

*file*: -- has xx other links: unchanged

The input file has xx other links; it is left uncompressed.

*file* unchanged

No savings can be achieved by compression. The input file remains uncompressed. The

-\textit{f} option can be used to force compression.

**Locale**

The following environment variables affect the execution of \texttt{compress}:

\textbf{LANG} Provides a default value for the internationalization variables that are unset or null. If \texttt{LANG} is unset or null, the corresponding default value from the internationalized environment is used. If one of the internationalization variables contains an invalid setting, the command behaves as if none of the variables have been defined.

\textbf{LC\textunderscore ALL} If this variable has been assigned a value, i.e. it is not a null string, this value overrides the values of all the other internationalization variables.

\textbf{LC\textunderscore CTYPE} Determines the internationalized environment for the interpretation of byte sequences as characters (e.g. single-byte characters as opposed to multi-byte characters in arguments).

\textbf{LC\textunderscore MESSAGES} Determines the format and contents of error messages.

\textbf{NLSPATH} Determines the position of message catalogs for the processing of \texttt{LC\textunderscore MESSAGES}.

**Example**

The file \texttt{films}, which occupies 4862 bytes in uncompressed form, is to be compressed.

```
$ ls -l
total 10
-rw------- 1 FELIX  group1  4862 Aug 19 09:27 films
$ compress -v films
compress: filme: 50.78\% Compression -- replaced with films.Z
$ ls -l
total 6
-rw------- 1 FELIX  group1  2393 Aug 19 09:27 films.Z
```

See also \texttt{uncompress, zcat}
**cp**

**copy files**

`cp` copies files. Copying means: the file is afterwards physically present in two locations.

`cp` has four formats. The command copies

- one file to another with a different name (Format 1)
- one or more files to a different directory, with the copy retaining the same basename as the corresponding original (Format 2).
- each file file in the file hierarchy to one of the target paths specified below (formats 3 and 4)

**Syntax**

| Format 1: | cp[-fip]..file..copyfile |
| Format 2: | cp[-fip]..file.......directory |
| Format 3: | cp.-R[-fip]..file.......directory |
| Format 4: | cp.-r[-fip]..file......directory |

**Format 1**  
**Copy one file:** `cp[-fip]..file..copyfile`  
- **-f** If it is impossible to obtain a file descriptor for `copyfile` (step 3.a.ii.), then an attempt is made to call `unlink()` for `copyfile`. Processing is continued.  
- **-i** (interactive) If the named `copyfile` already exists, `cp` will ask you to confirm whether this file may be overwritten. A **y** answer means that the copy operation should proceed. Any other answer prevents `cp` from overwriting `copyfile`.  
  If the standard input is not a terminal, this option is ignored and no copying is performed.  
- **-p** The following properties of all files are duplicated in the corresponding `copyfile`:
  1. The last time the data was changed and the time of the last access. If this procedure should fail for any reason then `cp` writes a message to the standard error output.  
  2. The user ID and group ID. If this procedure should fail for any reason then `cp` writes a message to the standard error output.  
  3. The bits for permissions as well as the bits S_ISUID and S_ISGID. If this procedure should fail for any reason then `cp` writes a message to the standard error output.  

If it is impossible to duplicate user ID or group ID then the bits for S_ISUID and S_ISGID are duplicated.

The sequence for the duplication of the properties listed above is not defined. `copyfile` is not deleted if these properties cannot be retained.

**file**

Name of the original file.
copyfile
  Name of the copy file.
  If copyfile does not yet exist, a new file is created.

  Unless the -p option is used, the copy will have the same mode as the original file, and
  the user and group ID of the copy will be those of the user who called cp, but the time
  of last modification of the copy will be set to the time the copy was made.

  Caution!
  If a file named copyfile already exists and the -i option is not used, the existing
  file will be overwritten without confirmation, but its mode, owner, and group will
  be preserved.

  If copyfile is a link to a file, all links will be retained. The contents of copyfile will be
  overwritten with the contents of file.

Format 2  Copy files to another directory
  cp[[-fip]]...file......directory
  -fip see Format 1

  file
  Name of the original file. You can give a list of names and thus copy several files at once.
  Each of the copies is assigned the same basic file name (basename) as the corre-
  sponding original.

  Caution!
  If there is a file in directory with the same basename as any original, and the -i
  option has not been set, the existing file will be overwritten without confirmation.

  directory
  Name of the directory in which the copies are to be placed. This must not be the
directory in which the original files are located.

  Unless the -p option is set, the copies will have the same modes as the originals, and
  the user and group ID of the copies will be those of the user who called cp, but the time
  of last modification of each copy will be set to the time the copy was made.
Built-in sh command

**cp**

Format 3  \[ cp[-R[[-fip]][..file].....directory \]

-\[ fip \] see Format 1

-\[ R \] copies data hierarchies.

If the option -\[ R \] is set then the following steps are performed:

i. \[ copyfile \] is created with the same file type as \[ file \].

ii. The permissions for \[ copyfile \] are set to match those of \[ file \]. Modification is subsequently performed via the user’s file creation mask provided that the option -\[ p \] has not been specified.

If this procedure should fail for any reason then \[ cp \] writes a message to the standard error output. Processing of \[ file \] is halted. However, other files may be processed as appropriate.

\[ file \]..\[ directory \]

see Format 2

Format 4  \[ cp[-r[[-fip]][..file].....directory \]

-\[ fip \] see Format 1

-\[ r \] Copies file hierarchies. The way in which special files are handled is implementation-dependent.

If option -\[ r \] is set then the way in which special files are handled is implementation-dependent.

\[ file \]..\[ directory \]

see Format 2

**Procedure**

If \[ directory \] already exists and is actually a directory, then the name of the target path for the individual files in the file hierarchy consists of a string formed by \[ directory \], a slash and the relative path name of the file corresponding to the directory which contains \[ file \].

If \[ directory \] does not yet exist and two operands are specified, then the name of the corresponding target path for \[ file \] is the directory \[ directory \]. The name of the corresponding target path for all other files in the file hierarchy consists of \[ directory \], a slash and the relative path name of the file corresponding to \[ file \].

An error occurs if \[ directory \] does not exist and more than two operands are specified. An error also occurs if \[ directory \] exists and is a file type as defined in the XSH specification but is not a directory.
In the following description \textit{file} is the file to be copied irrespective of whether it is specified via an operand or whether it is specified in a \textit{file} operand as a file in a file hierarchy. The term \textit{copyfile} designates the file which is specified by the target path.

The following steps are performed for each \textit{file}:

1. If \textit{file} and \textit{copyfile} designate the same file, \texttt{cp} writes a message to the standard error output where required. Processing of \textit{file} is halted. However, other files may be processed as appropriate.

2. If \textit{file} is a directory then the following steps are performed:
   a) If neither the option \texttt{-R} nor the option \texttt{-r} is specified, \texttt{cp} writes a message to the standard error output. Processing of \textit{file} is halted. However, other files may be processed as appropriate.
   b) If \textit{file} is not specified as an operand, and if \textit{file} is either "." or "..", then processing of \textit{file} is halted. However, other files may be processed as appropriate.
   c) If \textit{copyfile/directory} is present and is a file type which is not specified in the XSH specification then the system reaction will depend on the specific implementation.
   d) If \textit{copyfile/directory} is present and is not a directory, then \texttt{cp} writes a message to the standard error output. Processing of \textit{file} as well as of any other files which are subordinate to \textit{file} in the file hierarchy is halted. However, other files may be processed as appropriate.
   e) If the directory \textit{copyfile/directory} does not exist then it is created and assigned the access permissions which apply to \textit{file}. Modification is subsequently performed via the user's file creation mask, if the option \texttt{-p} has not been set, together with a bitwise OR operation with S_IRWXU. If \textit{copyfile} cannot be created, \texttt{cp} writes a message to the standard error output. Processing of \textit{file} is halted. However, other files may be processed as appropriate. It is not possible to predict whether or not \texttt{cp} will attempt to copy files to the file hierarchy corresponding to \textit{file}.
   f) The files in the directory \textit{file} are copied to the directory \textit{copyfile/directory}. Steps 1 to 3 are followed for the files.
   g) If \textit{copyfile/directory} have already been created then the access permissions are modified (if necessary) so that they are compatible with the access permissions for \textit{file}. Modification is subsequently performed via the user's file creation mask provided that the option \texttt{-p} has not been specified.
   h) \texttt{cp} does not process \textit{file} any further. However, other files may be processed as appropriate.
3. If file is a regular file then the following steps are performed:
   a) If copyfile is present then the following steps are performed:
      i. If the option -i has been set then cp writes a prompt to the default standard output and reads a line from the standard input. If the input is not a confirmation then cp discontinues processing of file. However, other files may be processed as appropriate.
      ii. A file descriptor for copyfile is obtained by performing the steps specified in the function open() in accordance with the XSH specification. copyfile is used as the path argument and the bitwise, inclusive OR operation of O_WRONLY and O_TRUNC is used as the oflag argument.
      iii. If the attempt to obtain a file descriptor fails and if the option -f has been set then cp attempts to remove the file by performing the steps specified in the function unlink() in accordance with the XSH specification. copyfile is used as the path argument. If this attempt is successful then cp continues processing with step 3b.
   b) If copyfile is not present then file descriptor is obtained by performing the steps specified in the function open() in accordance with the XSH specification. copyfile is used as the path argument and the bitwise, inclusive OR operation of O_WRONLY and O_TRUNC is used as the oflag argument. The permissions for file are taken from the mode argument.
   c) If the attempt to obtain a file descriptor fails then cp writes a message to the standard error output. Processing of file is halted. However, other files may be processed as appropriate.
   d) The contents of file are written to the file descriptor. If write errors occur then cp writes a message to the standard error output and continues processing with step 3e.
   e) The file descriptor is terminated.
   f) cp halts processing of file. If a write error has occurred at step 3d then it is impossible to predict whether or not cp will continue to process further files. If no write error occurred in step 3d then cp will continue to process files.

Error

   cp: Cannot access file
   The named file does not exist.

   cp: Cannot open file: Permission denied
   You have no read permission for file.

   cp: cannot create file
   You do not have write permission for the directory in which file is to be created, or the named directory does not exist.
cp: dir directory

dir is a directory and cannot be copied (Format 1), or you have not set the -r option (Format 4).

Locale

The following environment variables affect the execution of cp:

LANG Provides a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding default value from the internationalized environment is used. If one of the internationalization variables contains an invalid setting, the command behaves as if none of the variables have been defined.

LC_ALL If this variable has been assigned a value, i.e. it is not a null string, this value overrides the values of all the other internationalization variables.

LC_COLLATE Determines the internationalized environment for the behavior of ranges, equivalence classes and multicharacter collating elements used in the extended regular expressions defined for the affirmative response.

LC_CTYPE Determines the internationalized environment for the interpretation of byte sequences as characters (e.g. single-byte characters as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expressions defined for the affirmative response.

LC_MESSAGES Determines the format and contents of error messages.

NLSPATH Determines the position of message catalogs for the processing of LC_MESSAGES.

Example 1 The file techlit is to be copied before it is changed. The copy is to be called tl and to be located in the same directory as techlit.

```
$ cp techlit tl
```

Example 2 All files from the current directory with names beginning with fil are to be copied into the directory /home/do/save, and their times of last modification are to be duplicated.

```
$ cp -p fil* /home/do/save
$ ls -l /home/do/save
```

See also chmod, ln, mv, rm
The `cpio` command has three functions:
- it copies one or more files into an archive file (Format 1)
- it retrieves files from a previously created archive (Format 2)
- it copies files into a directory (Format 3)

The `cpio` command has been withdrawn from the XG4 standard and replaced by the command `pax`.

Consequently, only the `pax` command should be used in the future, since `cpio` is now supported only for compatibility reasons.

**Syntax**

Format 1: `cpio -o[Bacv][..D..archive]`

Format 2: `cpio -i[Bcdmruvf][..D..archive][..pattern...]`

Format 3: `cpio -p[adlmruv]..dir`

Only the three main options `-o`, `-i` and `-p` are detailed in the descriptions of the various formats shown above. The remaining options are listed in section “Modifiers and single options” on page 235. The various modifiers that can be used with each format are indicated in the format descriptions.

**Format 1**

**Copy files out**

`cpio -o[Bacv][..D..archive]`

**Modifiers**

`Bacv`

**Single option**

`-D..archive`

`-o (out) cpio` reads a list of pathnames of plain files from standard input and copies these files along with some status information in a special archive format to standard output or to the archive specified with the single option `-D`. The archive format is the same as the one created with the command `pax -x cpio`, where output is padded to a 512-byte boundary. The number of 512-byte blocks copied is written by `cpio` to standard error.
Format 2  
**Copy files in**

```
cpio -i [Bcdmrtuvf] [..D..archive] [..pattern...]
```

Modifiers

**Bcdmrtuvf**

**Single option**

- **-D**..archive

- **-I** (in)  
  `cpio` reads from standard input or the archive specified with the single option `-D`, which is assumed to be the product of a previous `cpio -o`, and extracts from it only those files with names that match `pattern` (see below).

  The extracted files are copied into the current directory tree in accordance with the specified options. The extracted files are given the same permissions as the files copied out with `cpio -o`. The user ID and group ID are those of the user calling `cpio -i`. It is only when the system administrator calls `cpio -i` that the extracted files retain the same user or group ID as the files copied out with `cpio -o`.

  Only the system administrator can extract special files.

**pattern**

`pattern` specifies the files to be selected from the archive.

All shell metacharacters for file name generation can be used in defining `pattern` (see section "File name generation" on page 54).

If `pattern` is not specified, all files are extracted from the archive.

Format 3  
**Copy files to a directory**

```
cpio -p [adlmruv] ..dir
```

**Additional option**

**adlmruv**

- **-p** (pass)  
  `cpio` reads a list of pathnames of plain files and copies these files to the named directory `dir` in accordance with the specified options.

  The number of 512-byte blocks copied by `cpio` is written to standard error.

**dir**

Name of the directory to which the files are to be copied. This directory must exist even if the `-d` modifier is specified.
Modifiers and single options

The following modifiers may be specified in any order. Apart from the single option `-D`, they must all be directly appended to the main option `-o`, `-i` or `-p` without intervening blanks.

- **a** (access time) Resets access times of input files after they have been copied. If the modifier `l` is also specified, the access times of files to which links exist are not reset.
- **B** (block) Sets an I/O block size of 512 bytes. This option is only meaningful for special files with direct access (raw devices).
  This modifier must not be used in combination with the main option `-p`.
- **c** (compatible) `cpio` writes or reads header information in the archive in character format for compatibility reasons.
- **d** (directory) Subdirectories are automatically created as needed.
- **f** Only the files not matching pattern are copied.
- **l** (link) This option must only be used in combination with option `-p`.
  It results in the creation of links to the files, instead of the files being copied.
- **m** (modification time) The original file modification times are retained.
  This modifier has no effect on directories.
  The modifiers `m` and `a` must not be specified together, since they are mutually exclusive.
- **r** (rename) Allows files to be renamed interactively on extraction.
  `cpio` prompts for a new name for each file.
  If you respond with a blank line (by simply pressing the Enter key), the file will not be renamed.
  If you respond with a dot (.), the file is copied without changing its name.
  Otherwise, the name of the file is changed to the text entered in response to the prompt.
- **t** (table of contents) `cpio` produces a table of contents of the archive.
  No files are copied.
- **u** (unconditional) Replaces an existing file with an extracted file even if the existing file has a more recent access date than the one being extracted.
  If the `u` modifier is not specified, the existing file is not overwritten.
- **v** (verbose) Causes `cpio` to display the names of the processed files.
  If `v` is used in conjunction with the modifier `t`, additional information on the processed files is displayed.
cpio

-D archive
archive names the archive file from which files are extracted or to which they are copied.
This archive file is used instead of the standard input or standard output.

Error
Old format cannot support expanded types on file
You have not specified the -c option. Due to compatibility reasons, cpio maintains the
i-nodes of files to be copied in the old format, so i-nodes exceeding 64 Kbytes can only be
processed if you have specified the -c option.

directories are not being created
You did not specify the -d option in Format 2.

Locale
The following environment variables affect the execution of cpio:

LANG
Provide a default value for the internationalization variables that are unset
or null. If LANG is unset or null, the corresponding value from the implemen-
tation-specific default locale will be used. If any of the internationalization
variables contains an invalid setting, the utility will behave as if none of the
variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other inter-
nationalization variables.

LC_COLLATE
Determine the internationalized environment for the behavior of ranges,
equivalence classes and multicharacter collating elements in extended
regular expressions for yes/no queries.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data
as characters (for example, single- as opposed to multi-byte characters in
arguments and input files) and the behavior of character classes in
extended regular expressions for yes/no queries.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents
of diagnostic messages written to standard error.

LC_TIME
Determine the format and contents of the calendar.

NLSPATH
Determine the location of message catalogs for the processing of
LC_MESSAGES.
Example 1  All files in the current directory with names that begin with `mb` are to be copied out to an archive file named `mailarchive`:

```
$ find . -name 'mb*' -print | cpio -o -D mailarchive
15776 blocks
```

The `find` command searches the current directory for file names that begin with `mb` and writes all such file names to standard output, which is piped (|) to the standard input of `cpio`. `cpio -o` then reads the file names and copies the appropriate files along with status information to the archive file `mailarchive` (option `-D`). When it has finished, `cpio` writes the number of copied blocks to standard error.

Example 2  List the contents of the archive file named `mailarchive`.

```
$ cpio -it -D mailarchive
mbox
mbox.new
mbox.old
15776 blocks
```

Example 3  The directory named `dir.old` is to be copied together with all its subdirectories. The duplicate directory is to be named `dir.new`:

```
$ cd dir.old
$ find . -print | cpio -pdl ../dir.new
```

The `cd` command changes the current directory to `dir.old`. The command `find` then searches this directory and all other directories below it and writes the names of all the files that it finds to standard output. Since the standard output of `find` is piped (|) to the standard input of `cpio`, `cpio` reads the file names and copies the files to the directory `dir.new` (which, in this example, is at the same level in the file tree as `dir.old`). Subdirectories are created as needed (-d), i.e. all subdirectories of `dir.old` are copied recursively. Since the `-l` option also specified here, instead of copying the files, links are created to the files where possible.

See also  `ar`, `cat`, `echo`, `find`, `ls`, `pax`, `tar`
crontab

**crontab schedule periodic background work**

`crontab` is used to have commands, shell scripts or executable programs executed regularly at specific times.

You can use `crontab` to:
- schedule command jobs (Format 1)
- have scheduled jobs listed (Format 2)
- delete scheduled jobs (Format 3)
- edit scheduled jobs (Format 4)

Jobs scheduled with `crontab` are retained even if the user who scheduled them closes the POSIX shell with `exit` or if the POSIX subsystem is shut down. There is no need to reschedule the jobs.

**Before the call**

The user ID must have a standard account number for `rlogin` access. This standard account number can be assigned using the ADD-USER, MODIFY-USER-ATTRIBUTES or ADD-POSIX-USER command.

If the file `/usr/lib/cron.d/cron.allow` exists, you can only use `crontab` if your login name appears in it.

If the file `/usr/lib/cron.d/cron.allow` does not exist, you can only use `crontab` if your login name does not appear in the file `/usr/lib/cron.d/cron.deny`.

If neither `/usr/lib/cron.d/cron.allow` nor `/usr/lib/cron.d/cron.deny` exists, only the POSIX administrator is allowed to use `crontab`.

If only an empty deny file exists, for example, everyone is allowed to use `crontab`.

Only the POSIX administrator is allowed to create and modify the allow and deny files. Each line in these files contains precisely one login name.

Before the `crontab` command is called, the POSIX administrator must have started up the cron daemon (`/etc/init.d/cron...start`).

⚠️ **Caution!**

If you inadvertently enter the `crontab` command with no arguments, do not attempt to get out with `CTRL D`! This will cause all entries in your `crontab` file to be removed. Instead, use `DEL` to exit.

**Syntax**

<table>
<thead>
<tr>
<th>Format 1: <code>crontab[...file]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2: <code>crontab...-l[...login_name]</code></td>
</tr>
<tr>
<td>Format 3: <code>crontab...-r[...login_name]</code></td>
</tr>
<tr>
<td>Format 4: <code>crontab...-e[...login_name]</code></td>
</tr>
</tbody>
</table>
Format 1  Scheduling command jobs

crontab[...file]

To schedule a command job with `crontab` you have to specify:
- the command (or shell script or program) that is to be executed, and
- the times at which it is to be executed (e.g. every Friday or every January 15).

When you call `crontab` in the above format, `crontab` copies these specifications into your `crontab` file `/var/spool/cron/crontabs/login_name`. The `/usr/sbin/cron` process checks every minute whether there are commands in the `crontab` file which need to be executed, and if there are any, it has them executed.

To execute the specified `command`, `crontab` invokes a new shell (`sh`) from your home directory. `crontab` supplies a default environment for every shell, defining `HOME`, `LOGNAME`, `SHELL` (`/bin/sh`), and `PATH` (`/bin:/usr/bin:/usr/lbin`). If you wish to have your `.profile` executed, you must specify this explicitly in the `crontab` file.

If neither the standard output nor the standard error output have been redirected for the commands in the `crontab` file, both the standard output and the standard error output will be sent to you by `mailx`.

There can be at most one `crontab` file per login name. If the `crontab` file `/var/spool/cron/crontabs/login_name` does not exist, it is created when `crontab` is called. If it already exists, its contents are overwritten. In other words, if you wish to schedule additional command jobs with `crontab`, you should use the `-e` option.

`file`
Name of the file containing the commands and the times at which they are to be executed.
`file` must be in the format described below (see “Format of a crontab file” on page 240).

`file` not specified:
`crontab` reads the text entered from the standard input. The text that you enter in this case must be in the format described below (see “Format of a crontab file” on page 240).

Format 2  Listing scheduled jobs

`crontab...[-l][...login_name]`

-`l` (list) `crontab` lists the contents of your `crontab` file `/var/spool/cron/crontabs/login_name`. If the file is not available, an error message is issued (see page 242).

`login_name`
If a `login_name` is specified, the `crontab` file of the corresponding user is listed. However, only the POSIX administrator is permitted to specify a `login_name`. 
Deleting scheduled jobs

```
crontab -r [login_name]
```

- `r` (remove) `crontab` deletes your `crontab` file `/var/spool/cron/crontabs/login_name`.

`login_name`

If a `login_name` is specified, the `crontab` file of the corresponding user is deleted. However, only the POSIX administrator is permitted to specify a `login_name`.

Editing scheduled jobs

```
crontab -e [login_name]
```

- `e` (edit) The `-e` option allows you to edit a copy of your `crontab` file or to create an empty file to edit (if `crontab` does not exist) and save this file as the current `crontab` file. The `VISUAL` environment variable determines which editor is called. If `VISUAL` is not defined, the `EDITOR` environment variable is checked, and if `EDITOR` is not defined, `ed` is invoked.

`login_name`

If a `login_name` is specified, the `crontab` file of the corresponding user is edited. However, only the POSIX administrator is permitted to specify a `login_name`.

Format of a `crontab` file

Your `crontab` file, or the text that you enter from standard input, (see Format 1) must be in the following format:

Text comprising:

- lines to define a command job, and
- comment lines, if required.

Blank lines are not permitted.

Command job definition lines

A line defining a command job consists of six fields separated by blanks or tabs. The fields contain the following information:

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>minute</td>
<td>hour</td>
<td>day of month</td>
<td>month</td>
<td>day of week</td>
<td>command</td>
</tr>
</tbody>
</table>
```
Fields 1 to 5 specify the times at which the command in field 6 is to be executed.

Minute | Range of values: 0-59 or pattern
Hour | Range of values: 0-23 or pattern
Day of month | Range of values: 1-31 or pattern
Month | Range of values: 1-12 or pattern
Day of week | Range of values: 0-6 (0=Sunday) or pattern
Command | Name of the command that is to be executed at the specified time.

If your command needs to read data from standard input, you have to use the percent character %. Each % character in the command field that has not been escaped with a backslash (\) is interpreted as a newline character. The shell executes the content of the command field only up to the first unescaped percent or newline character; the remainder of the field is passed to the command as standard input (see example 3 on page 244).

pattern may be:
- an asterisk * (standing for all legal values)
- a range described as number-number
- a comma-separated list of legal numbers or ranges, e.g.: 1,3,5 or 1-3,5

Specifying the day

Two fields are available for specifying the day; field 3 (day of month) and field 5 (day of week).

If you enter a number, a range or a list in the third field as well in the the fifth field, both specifications are valid, independently of each other.

For example, if your crontab file contains the line:

```
0 0 1,15 3 1 command
```

then command will be executed on March 1 and March 15 as well as on every Monday in March.

If you wish to specify the day in one field only, you must enter an asterisk in the second field. Thus if your crontab file contains the line:

```
0 0 * 3 1 command
```

then command will be executed on every Monday in March.
Comment lines

A comment line has the hash character # in column one and may be followed by any arbitrary text. You can use comment lines in the crontab file to explain the function of the command job or to arrange the job into sections for the sake of clarity. Please note that the input text that you pass to crontab can contain any number of comment lines but must not include any blank lines.

Exit status

  0: Successful execution of crontab.
  ≠0: An error occurred during crontab execution.

Error

  crontab: you are not authorized to use cron. Sorry.
  See Before the call above.

Format 1

  crontab: can't open your crontab file.
  You have called crontab with the name of a file which does not exist or cannot be accessed.

  crontab: error in previous line: ...
  If any line of the input text that you pass to crontab is not in the correct format, crontab displays the relevant line followed by the above error message. The ellipses (...) shown here are replaced by a more detailed description of the error, e.g.:

  unexpected character found in line
  An illegal character was found in the line.

  number out of bounds
  A specified number is not in the permitted range.

Format 2

  crontab: can't open your crontab file.
  Your crontab file /var/spool/cron/crontabs/login_name does not exist and therefore cannot be read with crontab -l.

File

  /usr/lib/cron.d/cron.allow
  List of login names with permission to use crontab. One login name is entered per line.

  /usr/lib/cron.d/cron.deny
  List of login names explicitly denied permission to use crontab. One login name is entered per line.

  /var/spool/cron/crontabs/login_name
  crontab file of the user identified by login_name.
Variable

The following environment variables are used by `crontab`.

**VISUAL**

The `VISUAL` environment variable determines which editor is used when the `-e` option is specified.

**EDITOR**

If `VISUAL` is not set, the `EDITOR` environment variable determines which editor is used when the `-e` option is specified.

Locale

The following environment variables affect the execution of `crontab`:

**LANG**

Provides a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding default value from the internationalized environment is used. If one of the internationalization variables contains an invalid setting, the command behaves as if none of the variables have been defined.

**LC_ALL**

If this variable has been assigned a value, i.e. it is not a null string, this value overrides the values of all the other internationalization variables.

**LC_CTYPE**

Determines the internationalized environment for the interpretation of bytes of text data as characters (e.g. single-byte characters as opposed to multi-byte characters in arguments and input files).

**LC_MESSAGES**

Determines the format and contents of error messages.

**NLSPATH**

Determines the position of message catalogs for the processing of `LC_MESSAGES`.

Example 1

You want the system to remind you on May 15 every year that it is Aunt Jemima’s birthday. To do this, you set up a file called `birthday` with the following contents:

```
0 0 15 5 * echo Today is Aunt Jemima's birthday !!!
```

Please note that since you only wish to make an entry for the day of the month and not for the day of the week, you must enter an asterisk in the “day of week” field.

You now pass the file named `birthday` to your `crontab` file:

```
$ crontab birthday
```

The system will then send you the appropriate message via `mailx` at 00.00 hours on May 15 of each year. This will obviously only function if the system has not been turned off during that night.
Example 2  Let us assume that on every Monday, at 1:30 p.m., you want to remind yourself and a co-worker of your aerobics class that evening. Your reminder is to be displayed on the screen. You are at terminal tty007 and she is at terminal tty014. You can also include comment lines in the input text so that your text is easier to read with `crontab -l` at a later stage. Your `crontab` file could then be set up as follows:

```plaintext
# *   = all legal values
# n-n = range
# n,n = list
# Day of week: 0 = Sunday
#
# Min  Hrs  Day(Mon)  Mon  Day(Wk)  Command
#
30   13      *        *    1      echo Aerobics this evening ! >/dev/tty007
30   13      *        *    1      echo Aerobics this evening ! >/dev/tty014
```

Every Monday at 1:30 p.m., you and your colleague will now receive the message "Aerobics this evening!" at your respective terminals (and not in your mailboxes).

Example 3  Every Monday, you want to `mailx` the message below to user `melodie`, who works on a system called `jukebox`:

```
It may be blue
But it will pass too!
```

You do this by putting the following line in your `crontab` file:

```
0 0 * * 1 mailx melodie@jukebox %It may be blue %But it will pass too!
```

See also  `at`, `mailx`, `sh`, `vi`
**csplit**  

**split files based on context**

`csplit` splits the contents of a file or the text it reads from standard input into smaller sections and writes all or some of these sections to separate output files. The original file is left unaltered.  
The way in which `csplit` divides a file and the sections for which output files are created are specified in the command-line arguments.  
`csplit` creates a maximum of 100 output files per call.

**Syntax**

```plaintext

csplit[\[\-ks\[\-f\_name\[\-n\_number\]\]file\_arg1...argn
```

No option specified  
The output files are named `xx00`, `xx01`, and so on.

For each output file that it creates, `csplit` writes a character count on standard output.  
Any files that have already been created are removed if an error occurs.

**option**

- `-f\_name`
  
The output files are called `name00`, `name01`, etc.

- `-f` not specified:
  
The output files are named `xx00`, `xx01`, and so on.

- `-k`
  
Files that have already been created are retained if an error occurs.

- `-n\_number`
  
The current number of the output files comprises `number` digits, whereby `1 ≤ number ≤ 9`.

*Example*

For `-n\_4`, the output files are called `xx0000`, `xx0001` etc.

- `-n` not specified:
  
The current number consists of 2 digits.

- `-s`
  
The output of a character count is suppressed.

**file**

Name of the input file.

If you use a dash (`-`) as the name for `file`, `csplit` reads from standard input.

**arg1...argn**

You can specify several arguments, each of which references a particular line in the input file.  
These lines represent the points at which `csplit` is to split the file into sections.  
Each dividing line becomes the first line of a new section. If you specify `n` arguments, `csplit` divides the file into `n+1` sections.
These sections contain the following lines:

<table>
<thead>
<tr>
<th>Section</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All lines from the start of the input file up to but not including the line referenced by the first argument.</td>
</tr>
<tr>
<td>1</td>
<td>All lines from the first dividing line up to but not including the line referenced by the second argument.</td>
</tr>
<tr>
<td>n</td>
<td>All lines from the line referenced by the n-th argument to the end of the input file.</td>
</tr>
</tbody>
</table>

csplit usually writes each section to a separate output file.

This does not apply when the argument \%regular_expression\[+number][\-number] is used (see below). The last section (section n) is always written to an output file.

The arguments you specify are processed by csplit in the order in which you list them. To begin with, the first line of the input file is the current line. After an argument has been processed, the line referenced by this argument becomes the current line. The line referenced by the next argument must lie in the range between but not including the current line and the end of the input file. Thus the line referenced by the second argument must come after the line referenced by the first argument.

argument can be specified as follows:

/regexp/[+number][\-number]  
An argument in the form /regexp/ references the next line after the current line that matches the specified regular expression. The section from the current line up to but not including the line that matches the regular expression is written to an output file. The line matching the regular expression now becomes the current line.

The +number or -number offset shifts the dividing line number lines after (+) or before (-) the line that matches the regular expression. The line that is number lines after (+) or before (-) the line matching the regular expression thus becomes the current line.

Simple regular expressions (see Tables and directories, Regular POSIX shell expressions) are recognized. If the argument contains blanks or shell metacharacters (see Tables and directories, POSIX shell metanotation), you must either escape every such character with a backslash \ or enclose the whole argument in single quotes ''. The regular expression must not contain any newline characters.
%regexp%[+number][-number]

An argument in the form %regexp% references the next line after the current line that matches the specified regular expression. The line that matches the regular expression becomes the current line. \texttt{csplit} in this case does not create an output file for the relevant section.

If the \texttt{+number} or \texttt{-number} offset is also specified, the current line will be the line that is \texttt{number} lines after (+) or before (-) the line containing the regular expression.

Simple regular expressions (see section "Regular POSIX shell expressions" on page 877) are recognized. If the argument contains blanks or shell metacharacters (see section "Metacharacters for the POSIX shell" on page 884), you must either escape every such character with a backslash \ or enclose the whole argument in single quotes '...'. The regular expression must not contain any newline characters.

\texttt{num}

This argument references the line with line number \texttt{num}. \texttt{csplit} writes the section from the current line up to but not including the \texttt{num}th line to an output file. The \texttt{num}th line then becomes the current line.

\texttt{\{n\}}

This argument is an abbreviation for \texttt{n} arguments of the previous type (see above) and means: "repeat the preceding argument \texttt{n} times", where \texttt{n} is an integer greater than 1.

The \texttt{\{n\}} argument can be entered after any of the above-mentioned arguments, with a blank to separate them.

Thus if it follows an argument in the form \texttt{/regexp/+number][-number]} or \texttt{%regexp%[+number][-number]}, this argument will be repeated \texttt{n} times.

\textbf{Example}

\texttt{'/regexp/' \{2\}} is an abbreviation for
\texttt{'/regexp/' '/regexp/' '/regexp/'

If \texttt{\{n\}} follows an argument of the \texttt{num} type, the file will be split \texttt{n} times, from the \texttt{num}th line onward, into sections of \texttt{num} lines each.

\textbf{Example}

\texttt{100 \{2\}} is an abbreviation for \texttt{100 200 300}

\textbf{Error}

\texttt{argument – out of range}

The line referenced by the specified argument lies outside the permissible range. The legal range is from, but not including, the current line to the end of the file.

\texttt{100 file limit reached at arg ...}

You have specified so many arguments that \texttt{csplit} would need to create more than 100 output files.
Locale

The following environment variables affect the execution of `csplit`:

* `LANG` Provides a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding default value from the internationalized environment is used. If one of the internationalization variables contains an invalid setting, the command behaves as if none of the variables have been defined.

* `LC_ALL` If this variable has been assigned a value, i.e. it is not a null string, this value overrides the values of all the other internationalization variables.

* `LC_COLLATE` Determines the internationalized environment for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.

* `LC_CTYPE` Determines the internationalized environment for the interpretation of byte sequences as characters (e.g. single-byte characters as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

* `LC_MESSAGES` Determines the format and contents of error messages.

* `NLSPATH` Determines the position of message catalogs for the processing of `LC_MESSAGES`.

Example 1

The file `book` contains a text that is subdivided into three chapters. The first chapter is preceded by a preface; an appendix follows the last chapter. Each chapter begins with the title "CHAPTER ..."; the title of the appendix is "APPENDIX".

You now wish to put the preface, the individual chapters, and the appendix into separate files. The output files are to be named `chap00`, `chap01`, etc.

```bash
$ csplit -f chap book '/CHAPTER/' '/CHAPTER/' '/CHAPTER/' '/APPENDIX/
```

```
1636
15124
32743
20344
2576
```

```bash
$ ls
book
chap00
chap01
chap02
chap03
chap04
```
csplit

The file *chap00* contains the preface and consists of 1636 characters. The appendix is located in the file *chap04*.

The same results could also have been obtained by abbreviating the *csplit* call as follows:

```
$ csplit -f chap book '/CHAPTER/' {2} '/APPENDIX/'
```

You can now edit the sections separately, and later you can join them again using *cat*:

```
$ cat chap0[0-4] > book
```

**Example 2** The input file *file* is to be split into sections every hundred lines. To do this, you enter:

```
$ csplit file 100 (98)
```

The argument (98) stands for 98 arguments: 200 300 ... 9900.

If *file* contains 9900 or more lines, *csplit* creates 100 output files. The first output file *xx00* includes line 1 to 99 (inclusive); the last output file, *xx99*, contains the rest of *file* from line 9900 onward.

If *file* contains fewer than 9900 lines, *csplit* issues the error message "(98) - out of range" and terminates. If you include option -k in the call, the files already created are retained.

```
$ csplit -k file 100 (98)
```

If *file* contains only 9830 lines, for example, then *xx98* is the last output file created and includes lines 9800 to 9830.

**Example 3** The file *prog.c* contains a C source program. The program includes a *main* function and a maximum of 20 further functions. In accordance with C conventions, each function ends with a right brace at the beginning of a line (in column 1). Right braces within a function are not located in the first column of a line.

Each function is now to be written to a separate file. To do this, you enter:

```
$ csplit -k prog.c '%main{%' '/^}$/+1' {19}
```

If the program contains exactly 20 functions in addition to the *main* function, *csplit* splits the file into 22 sections.

Section 0 contains all lines from the beginning of the file up to but not including the start of the *main* function. This section will not be written to an output file (argument %main(%).

Section 1 contains the *main* function and is written to the output file *xx00* (argument /%/+1).

Functions 1 to 19 are similarly written to separate output files in succession (argument {19}). The final section, i.e. section 22, contains the rest of the input file (which in this case is function 20) and is written to the output file *xx20*.

If the program contains fewer than 20 functions, *csplit* will terminate at the last function and issue the error message "(19) - out of range". Since the -k option has been set, the created files will, however, be retained.

**See also**  
*ed, sh, split*
**cut**

**cut out selected fields of each line of a file**

`cut` reads the input text line by line from files or from the standard input and cuts out of each line selected bytes (Format 1), characters (Format 2) or fields (Format 3). `cut` outputs the removed components at the standard output.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 1: <code>cut...-b..list..[-n]1[...file...]</code></td>
<td><strong>Cutting out bytes</strong></td>
</tr>
<tr>
<td>Format 2: <code>cut...-c..list[...file]...</code></td>
<td></td>
</tr>
<tr>
<td>Format 3: <code>cut...-f..list[...-d..char[...-s][...file]...</code></td>
<td></td>
</tr>
</tbody>
</table>

**Format 1** **Cutting out bytes**

`cut...-b..list..[-n]1[...file...]`

- `-b..list`  
  (bytes) `cut` cuts out of each input line the bytes which are in the position specified in `list` and sends them to the standard output.

  `list` is a list of numbers or numerical ranges. The elements in the list must be separated by commas and arranged in ascending order. The range `n1-n2` refers to all the numbers between `n1` and `n2` inclusive.

  The following abbreviations can be used to specify ranges:

  - `-n` for `1-n`
  - `n-` for `n-"last column"

**Example**

- `1,3,5` `cut` cuts the 1st, 3rd and 5th columns.
- `1-3,5` `cut` cuts the 1st, 2nd, 3rd and 5th columns.
- `-3,5` is an abbreviation for `1-3,5`
- `3-` is an abbreviation for `3-"last column"

- `-n` A character which may consist of multiple bytes is not cut out by byte by byte. Every `list` character which is specified in the range `n1-n2` is treated as follows:

  - If `n1` is not the first byte of the character, then `n1` is ignored in order to cut the first byte of the character.
  - If `n2` is not the last byte of the character, then `n2` is ignored. The last character to precede `n2` is selected. `n2` is reset to zero if there is no preceding character.

**file**

Name of the input file. You may input multiple files.

`file` not specified: `cut` reads from the standard input.
Format 2  Cutting out columns

\texttt{cut\-[c]\-list[\-file]\...}

\texttt{-c\-list}

(column) The columns specified in \textit{list} are cut out from each input line and written to standard output. A column is exactly one character wide.

\textit{list} possesses the format described under Format 1.

\texttt{file}

Name of the input file.
You may name more than one file.

\textit{file} not specified: \textit{cut} reads from standard input.

Format 3  Cutting out fields

\texttt{cut\-[f]\-list[\-d\-char][\-s][\-file]\...}

\texttt{-f\-list}

(field) \textit{cut} cuts out the fields specified in \textit{list} from each input line and writes them on standard output.

A field comprises all characters that are located between two field delimiters. Two successive field delimiters mark an empty field. The field delimiter defaults to the tab character, but you can redefine it in the \texttt{-d} option.
The output fields are separated from one another by one field delimiter each. Input lines with no field delimiters are output in full (unless the \texttt{-s} option is set). This is generally useful for table subheadings.

\textit{list} is specified as described under Format 1.

\texttt{-d\-char}

The character given as \textit{char} serves as the field delimiter.
If \textit{char} is a blank or a shell metacharacter (see section \textit{Metacharacters for the POSIX shell} on page 884) it must be enclosed in single quotes: \texttt{-d'char'}.

\texttt{-d} not specified:
The field delimiter defaults to the tab character.

\texttt{-s}

Lines with no field delimiters are suppressed.

\texttt{-s} not specified: Lines with no field delimiters are output in full.

\texttt{file}

Name of the input file.
You may name more than one file.

\textit{file} not specified: \textit{cut} reads from standard input.
Error  ERROR: line too long
A line can have no more than 1023 characters or fields. Alternatively, the newline character may be missing.

cut: Bad list for b/c/f option
Incorrect list specification or missing option -b, -c or -f. No error is reported if the line possesses fewer fields than are required by list.

cut: No delimiter
The delimiter character missing from the -d option.

ERROR: cannot handle multiple adjacent backspaces
Adjacent backspaces cannot be processed correctly.

cut: Cannot open file
The named file cannot be read or does not exist. If multiple files have been specified, processing will continue on the other files.

Locale  The following environment variables affect the execution of cut:

LANG  Provides a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding default value from the internationalized environment is used. If one of the internationalization variables contains an invalid setting, the command behaves as if none of the variables have been defined.

LC_ALL  If this variable has been assigned a value, i.e. it is not a null string, this value overrides the values of all the other internationalization variables.

LC_CTYPE  Determines the internationalized environment for the interpretation of byte sequences as characters (e.g. single-byte characters as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES  Determines the format and contents of error messages.

NLSPATH  Determines the position of message catalogs for the processing of LC_MESSAGES.
Example 1  Print the first 72 characters of each input line:

$ cut -c1-72 file

Example 2  The first and third fields of the /etc/group file (the group name and group number) are cut out and output. The individual fields in this file are separated by means of a colon.

$ cut -f1,3 -d: /etc/group

Example 3  The names of customers whose bills are due on the first day of any month are to be filtered out of the invoice file of a mail order house along with the amount due from each of them. The file is structured as follows:

Homewood        Milwaukee         10,000        06.01.05
Mackenzie       Detroit            7,000        07.06.05
Macnamara       Boston             8,000        05.01.05
Tinniswood      Atlanta              450        06.20.05

The fields in the table are separated by exactly one tab and padded with blanks.

$ grep '.01.05' invoice  | cut -f1,3 > names
$ cat names
Homewood        10,000
Macnamara        8,000

Explanation: grep finds all lines containing the string .01.05 and writes them to standard output. cut receives these lines as input, selects the first and third fields, and writes its output to the file names.

If you would then like each customer name in the names file to be preceded by the date and then sorted accordingly, you could enter:

$ grep '.01.05' invoice  | cut -f4 > date
$ paste date names • sort
05.01.05        Macnamara        8,000
06.01.05        Homewood        10,000

Explanation: The first command line writes the date associated with the selected customer names to the file named date. The paste command pastes the lines in the date and names files horizontally, separating them with the default tab character. The sort command then sorts the output lines in ascending order of date.

See also awk, grep, paste
**date**

**write the date and time**

`date` writes the current date and time on standard output.

The form in which `date` outputs dates and times depends on the value of the environment variable `LC_TIME` or, if that is undefined or null, on the value of `LANG`. If both are undefined or null, or if the required database is not available, or if one of the NLS variables has an invalid value, `date` acts as if in a non-internationalized system, printing dates and times in American format.

### Mode of operation

The system works with UTC (Universal Time Coordinated; this is the same as GMT - Greenwich Mean Time). `date` converts UTC to local time and vice versa. If the environment variable `TZ` is defined and option `-u` is not set then this variable is used to determine the time zone and convert UTC to local time.

### Syntax

**Format 1:** `date[-u][-+format]`

**Format 2:** `date.-a[-]-.sss.fff`

#### Format 1 `date[-u][-+format]`

No argument specified

`date` prints the current date and time in a format governed by the current locale.

- **-u** Displays the current date and time in Greenwich Mean Time (GMT).
  
  If the `-u` option is set, the `+format` specification is ignored.

- **+format**
  
  The `format` argument defines the output format for `date`. If `format` contains blanks or tabs or other shell metacharacters which you do not want the shell to interpret, you should enclose `format` in single quotes:

  `+format`.

  `format` is essentially similar in format to the first argument of the `printf()` function in C or `awk` (see `awk`, `printf` [page 140] and the C function `printf()` [4]).

  The output of `date` can be formatted by means of field descriptors in the `format` argument. Field descriptors take the form of a percentage sign followed by a letter and are replaced in the output by their values (see page 255). All characters that are not part of a field descriptor are copied to the output unmodified. The output is always terminated with a newline character.
Built-in sh command  

Format 2  `date`.-a[-].-ss.fff

- `a[-].-ss.fff`

  Adjusts the system clock time by `ss.fff` seconds, where `fff` represents fractions of a second. This adjustment can be positive or negative (-). The system’s clock will be sped up or slowed down until it has changed by the specified amount.

Field descriptors

The following overview lists all legal field descriptors. Although the field descriptors `%h` and `%b` are essentially identical, both descriptors have been retained for compatibility reasons.

- `%n` newline character
- `%t` tab character
- `%c` date and time in the default format of the current locale
- `%C` century (first two digits of the year, 00-99)
- `%D` date in the format `%m/%d/%y`
- `%x` date in the format of the current locale
- `%y` year (last two digits of the year, 00-99)
- `%Y` year (last two digits of the year, 00-99)
- `%m` month
- `%h` month (in letters, abbreviated) in the format of the current locale
- `%b` identical to `%h`
- `%B` month (in letters, in full) in the format of the current locale
- `%W` week number of year (00 to 53, with Monday as the first day of the week)
- `%V` week number of the year (01 to 53, with Monday as the first day of the week). The first week in January is counted as week 1 if it contains at least 4 days. Otherwise this week is counted as week 53 of the previous year
- `%U` week number of the year (00 to 53, with Sunday as the first day of the week)
- `%j` day of year (001 to 366)
- `%d` day of month (01 to 31)
- `%e` day of month (1 to 31, single-digit numbers preceded by a blank)
- `%a` abbreviated weekday name in the format of the current locale
- `%A` full weekday name in the format of the current locale
- `%w` day of week (0 to 6, Sunday = 0)
- `%u` day of week (1 to 7, Monday = 1)
- `%R` time in the format `%H:%M`
%T  time in the format %H:%M:%S
%X  time in the format of the current locale
%r  time in a.m./p.m. notation, as %I:%M:%S %p
%H  hour (00 to 23)
%I  hour (01 to 12)
%p  string containing ante-meridiem or post-meridiem indicator(a.m./p.m. affix) in the format of the current locale.
%M  minute (00 to 59)
%S  second (00 to 61)
%Z  timezone name, or no output if no timezone exists (governed by the value of the environment variable TZ, see POSIX shell variables).

**Modified field descriptors**

If an alternative representation is defined in your local environment (e.g. before and after Christ) you can call it using modified field descriptors. Modified field descriptors are in the form %Eletter or %Oletter. The modified field descriptors that can be used are listed in the overview below.

If no alternative representation has been defined, all modified field descriptors output the value of the current unmodified field descriptors.

%Ec  date and time in the alternative format
%EC  name of the time period (e.g. "year") in the alternative representation
%Ex  date in the alternative format
%EX  time in the alternative format
%Ey  year in which the time period in the alternative representation begins
%EY  year in the alternative representation
%Od  day of month in alternative numeric representation
%Oe  day of month in alternative numeric representation
%OH  hour (24-hour clock) in alternative numeric representation
%OI  hour (12-hour clock) in alternative numeric representation
%Om  month in alternative numeric representation
%OM  minutes in alternative numeric representation
%OS  seconds in alternative numeric representation
%Ou  weekday in the alternative format (Monday = 1)
%OU  week of the year in alternative numeric representation (same as for %U)
**Built-in sh command**

**date**

%OV week of the year in alternative numeric representation (same as for %V)

%OW weekday in the alternative format (Sunday = 0)

%OW week of the year in alternative numeric representation (same as for %W)

%Oy year in the alternative format

**Variable TZ**

If defined, the environment variable TZ contains information on timezones. *date* uses TZ to determine the timezone and to convert from UTC (Universal Time Coordinated) to local time and vice versa.

The default value "MEZ-1,MSZ-2,M3.5.0/02:00:00,M10.5.0/03:00:00" of the TZ variable must be interpreted as following:

- **Standard time zone**
  - Name: MEZ
  - Time difference: Zone - 01:00:00 = UTC

- **Alternative time zone**
  - Name: MSZ
  - Time difference: Zone - 02:00:00 = UTC

- **Switchover time to the alternative time zone**
  - Month: 3 = March
  - Week: 5 (or 4)
  - Weekday: 0 = Sunday
  - Time: 02:00:00

- **Switchover time to the standard time zone**
  - Month: 10 = October
  - Week: 5 (or 4)
  - Weekday: 0 = Sunday
  - Time: 03:00:00

**Locale**

The following environment variables affect the execution of *date*:

**LANG**

Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**

If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
\textbf{LC_MESSAGES}\hfill
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

\textbf{LC_TIME}\hfill
Determine the format and contents of date and time strings written by \texttt{date}.

\textbf{NLSPATH}\hfill
Determine the location of message catalogs for the processing of \texttt{LC_MESSAGES}. 
Example  Printing date and time

If you call `date` without arguments at 17:00 hours EST on June 19, 2009, and the system clock is set to the correct time, the following will be printed:

`Fri Jun 19 17:00:00 MSZ 2009`

The command:

$ `date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'`

generates the following output:

`DATE: 06/19/09
TIME: 17:00:00`

You have defined the environment variable `CFTIME` and used `echo` to output the value of `CFTIME`:

$ `echo $CFTIME`

`%d %B %y`

If the system clock is set correctly then, on 19 June 2005 at 17.00 CET, a call to `date` with no argument results in the output:

`19 June 05`

See also  `cal`
`ctime()`, `printf()` [4]
dd convert and copy a file

The dd command can be used to copy files while making any necessary conversions, e.g. from EBCDIC to ASCII or vice versa.

dd is well-suited for input/output via raw devices (raw physical I/O), because it can read and write in blocks of any size. Note, however, that any given raw device can only handle the block sizes for which it is specifically designed (512-byte blocks for a floppy disk drive, for example. If the size of the file you want to convert is not a multiple of the appropriate block size, you must use the conv=sync option.

After completion, dd reports the number of whole and partial input and output blocks on standard error.

Caution! Never use dd to copy files between file systems which have different block sizes. Using a block device to copy a file may result in extra nulls being added to the file to pad the final block to the block boundary.

Syntax

```
[...option...]
```

No option specified

dd reads from standard input and writes to standard output. No conversions are performed. The input and output block sizes default to 512 bytes.

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>if=input_file</td>
<td>Name of input file</td>
</tr>
<tr>
<td>of=output_file</td>
<td>Name of output file</td>
</tr>
<tr>
<td>ibs=n</td>
<td>Input block size (in bytes)</td>
</tr>
<tr>
<td>obs=n</td>
<td>Output block size (in bytes)</td>
</tr>
<tr>
<td>bs=n</td>
<td>Input and output block size (in bytes)</td>
</tr>
<tr>
<td>cbs=n</td>
<td>Conversion buffer size (in bytes)</td>
</tr>
<tr>
<td>skip=n</td>
<td>Skip first n blocks of input file</td>
</tr>
<tr>
<td>iseq=n</td>
<td>Seek n blocks from beginning of input file before copying</td>
</tr>
<tr>
<td>oseek=n</td>
<td>Seek n blocks from beginning of output file before copying</td>
</tr>
<tr>
<td>seek=n</td>
<td>Seek n blocks from beginning of output file before copying</td>
</tr>
<tr>
<td>count=n</td>
<td>Copy/convert only first n blocks of input file</td>
</tr>
<tr>
<td>conv=value[,value...]</td>
<td>Several values for conv</td>
</tr>
<tr>
<td>conv=ascii</td>
<td>Convert from EBCDIC to ASCII</td>
</tr>
</tbody>
</table>
If size arguments are needed for an option, you can specify them as follows:

\[
\begin{align*}
n & \quad \text{means} \quad n \times 1 \\
\text{n \(b\)} & \quad \text{means} \quad n \times 512 \\
\text{n \(k\)} & \quad \text{means} \quad n \times 1024 \\
\text{n \(w\)} & \quad \text{means} \quad n \times 2 \\
\text{n \(x\)} & \quad \text{means} \quad n \times m
\end{align*}
\]

- **if=input_file**
  
  In `input_file` you give the name of the input file from which `dd` is to read.
  
  *if=*input_file not specified:
  
  `dd` reads from standard input.

- **of=output_file**
  
  `output_file` designates the name of the output file to which `dd` is to write.
  
  *of=*output_file not specified:
  
  `dd` writes to standard output.

- **ibs=n**
  
  (input block size) In `n` you specify the input block size in bytes.
  
  *ibs=*n not specified:
  
  The input block size defaults to 512 bytes.

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv=ebcdic</td>
<td>Convert from ASCII to EBCDIC</td>
</tr>
<tr>
<td>conv=siemens</td>
<td>Convert from ASCII to Siemens-specific EBCDIC</td>
</tr>
<tr>
<td>conv=ibm</td>
<td>Convert from ASCII to IBM-specific EBCDIC</td>
</tr>
<tr>
<td>conv=block</td>
<td>Convert newline-terminated records to fixed-length records</td>
</tr>
<tr>
<td>conv=unblock</td>
<td>Convert fixed-length records to newline-terminated records</td>
</tr>
<tr>
<td>conv=lcase</td>
<td>Convert lowercase letters to lowercase</td>
</tr>
<tr>
<td>conv=ucase</td>
<td>Convert lowercase letters to uppercase</td>
</tr>
<tr>
<td>conv=swab</td>
<td>Swap every pair of bytes</td>
</tr>
<tr>
<td>conv=noerror</td>
<td>Do not stop processing on error</td>
</tr>
<tr>
<td>conv=sync</td>
<td>Pad every input block to the number of characters defined in <code>ibs=n</code></td>
</tr>
<tr>
<td>conv=notrunc</td>
<td>Do not truncate the output file</td>
</tr>
</tbody>
</table>
obs=n
(output block size) In n you specify the output block size in bytes.

obs=n not specified:
The output block size defaults to 512 bytes.

bs=n
(block size) In n you specify a value in bytes for both the input and output block size. The value specified for bs overrides the values of ibs and obs.

cbs=n
(conversion buffer size) In n you specify a value in bytes for the conversion buffer size. This specification is only effective in the following cases:
– for conversions from EBCDIC to ASCII and vice versa
– for conversions from and to fixed-length records.

(see conv=ascii, conv=ebcdic, conv=ibm and conv=block, conv=unblock, page 263).

skip=n
The first n blocks of the input file are skipped, and copying begins at block number n+1. This option is only appropriate for magnetic tape, where iseen is undefined.

iseek=n
The first n blocks of the input file are skipped, and copying begins at block number n+1. This option is specially designed for disk files, where skip can be extremely slow. It is only accepted with devices which support the iseen system call (such as disk units).

oseek=n
The copy of the input file (after conversion, if specified) begins n blocks into the output file (i.e. at block n+1). The first n blocks of the output file remain unchanged. This option is only accepted with devices which support the iseen system call (such as disk units).

seek=n
seek performs the same function as oseen.

count=n
Only the first n blocks of the input file are copied (and converted, if specified).

conv=value[,value...]
You can specify several comma-separated values for conv and thus have a number of conversions performed.

The values ascii, ebcdic and ibm cannot be used in combination. The same applies to the pairs block and unblock, and lcase and ucase.
**conv=ascii**

Conversion from EBCDIC to ASCII.

The number of characters defined in \( cbs=n \) is copied into the conversion buffer and any specified character mapping is done. Trailing blanks are removed, and a newline character is added. The output block size of the file is thus adjusted to the value defined in \( cbs=n \). If \( cbs \) is unspecified or set to 0, input file character mapping is done, but trailing blanks are not removed.

**conv=ebcdic**

Conversion from ASCII to EBCDIC.

The ASCII characters are read into the conversion buffer and converted to EBCDIC. Trailing blanks are added in order to produce the output block size defined in \( cbs=n \). If \( cbs \) is unspecified or set to 0, input file character mapping is done, but the file’s block structure is not changed.

**conv=siemens**

Conversion from ASCII to EBCDIC as described above, but by using a Siemens-specific EBCDIC table. The Newline character is converted as follows:: \( X'0A' \) <-> \( X'05' \).

**conv=ibm**

Conversion from ASCII to EBCDIC as described above, but using an IBM-specific EBCDIC table.

**conv=block**

Conversion of newline-terminated ASCII records to fixed-length records. The ASCII characters are read into the conversion buffer, and blanks are added to adjust the record length or output block size to the value defined in \( cbs=n \). If \( cbs \) is unspecified or set to 0, this option produces a simple file copy.

**conv=unblock**

Conversion of fixed-length ASCII records to newline-terminated records. The record length is determined by the value defined in \( cbs=n \). This number of characters is repeatedly read into the conversion buffer, where excess trailing blanks are removed and a newline character is added before the characters are sent to the output. If \( cbs \) is unspecified or set to 0, this option produces a simple file copy.

**conv=lcase**

Uppercase letters are converted to the corresponding lowercase letters.

**conv=ucase**

Lowercase letters are converted to the corresponding uppercase letters.

**conv=swab**

The bytes in each pair of bytes are swapped. If an input block contains an uneven number of bytes, the last byte is ignored.
Conv=NoError
Processing is continued even if an error occurs.

Conv=Sync
Every input block is padded to the value specified in ibs=n. When block or unblock is specified, blanks are used, otherwise null-bytes.

Conv=Notrunc
The output file is not regenerated. All blocks that are not overwritten explicitly by dd remain unchanged.

Locale
The following environment variables affect the execution of dd:

LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files), the classification of characters as upper- to lower-case, and the mapping of characters from one case to the other.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH
Determine the location of message catalogs for the processing of LC_MESSAGES.

Example
The contents of an EBCDIC tape with a block structure of 10 times 80 bytes is to be written to the ASCII file xy:

$ dd if=/dev/rmt32 of=xy ibs=800 cbs=80 conv=ascii

See also cp
debug  program debugging in forked tasks

The debug command enables POSIX programs which can be started from within a shell to be debugged.

Syntax

| Format 1: debug[\-e]...program[\-arguments]... |
| Format 2: debug[\-p]...pid |

Format 1  debug[\-e]...program[\-arguments]...

The program to be debugged is loaded in the task generated from within the shell by means of a fork (fork task). Once the program has been loaded, but before the first command is executed, control is passed to the user on BS2000 command level. The process ID of the fork task is displayed as a prompt (see also Example 1 below).

\-e  program is loaded without its external symbols dictionary.

  program

  Name of the program to be debugged. The program must be executable.

  arguments

  program arguments.

Format 2  debug[\-p]...pid

The program running in the fork task with process ID pid is stopped. The user is given control on BS2000 command level. The process ID of the interrupted fork task is displayed as a prompt (see also Example 2 on page 266).

\-p  Interrupts the program in the task with process ID pid.

  pid

  Process ID of the task that is to be stopped.

Example 1  The example below illustrates what happens up to the point where the program to be debugged is loaded. When the prompt pid/ appears, the program is ready to start as it would be after a /LOAD-PROGRAM command. AID commands can be used to influence how debugging is to proceed. The program starts again after the /RESUME-PROGRAM command in the fork task.

/START-POSIX-SHELL

\$ cd myprogdir
\$ ls -l myprog
-rwxr-xr-x  1  MYUSER  MYGROUP 1150976  Wed May 11 13:10:50 MSZ 2005 myprog
\$ debug myprog
% AID0348 Program stopped due to EXEC event (PID=0000000055) 0000000055/

U22794-J-Z125-6-76
Example 2  This example illustrates what happens when a program executing in a fork task is interrupted. When the *pid/* prompt appears, the program has been interrupted as when [K2] is pressed. AID commands can be used to influence how debugging is to proceed. The program starts again after the RESUME-PROGRAM command in the interrupted fork task.

START-POSIX-SHELL

$ ps -e
  UID  PID  PPID  C    STIME TTY        TIME  CMD
SYSROOT  243   1  0  16:32:38 term/001 0:01  [sh]
SYSROOT  245   243  0  16:33:44 term/001 0:00  [loop]

$ debug -p 245
% AID0492 %STOP was send for fork task (PID=000000245)
$
% AID0348 Program stopped due to STOP event (PID=000000245)
000000245/

You can use `df` to determine the number of free disk blocks and inodes which are still available at mounted or unmounted local systems and to obtain information on the file system coding (ASCII or EBCDIC).

Syntax

Format 1: `df[[-F..FSType].. -P[[-kIv][..file]]]

Format 2: `df[[-F..FSType][[-begklnTVv][[-o..ufs_options]...][..file]]]

Format 3: `df[[-c][..file]]`

Format 1 `df[[-F..FSType].. -P[[-kIv][..file]]]`

**option**

- **-F..FSType**  
  The file system with which you want to work is of the type `FSType`. You only need to specify this option if the file system is unmounted.

  `FSType` may be:

  - `ufs`  
    Berkeley file system

  - `bs2fs`  
    bs2fs file system

  - `proc`  
    Pseudo file system for accessing process data

  - `fdfs`  
    Pseudo file system for accessing file descriptors

- **-P**  
  Prints the following information in columns for all mounted file systems or for the specified file systems:

  - `filesystem`  
    Special file for the file system

  - `bytes`  
    Total disk space in 512-byte blocks

  - `used`  
    Used disk space in 512-byte blocks

  - `available`  
    Available disk space in 512-byte blocks

  - `capacity`  
    Percentage of disk space used

  - `mounted on`  
    Name of the directory on which the file system is mounted

- **-k**  
  Prints the following information in columns for all mounted file systems or for the specified file systems (like option `-P`). Storage space is listed in 1024-byte blocks (instead of 512-byte blocks).

  The storage space specification takes account of the reserved storage space which can only be used by the POSIX administrator.
df

- I  
   df prints the number of free blocks and inodes for all locally mounted file systems or for
   the specified file systems. The -I option is set by default. It cannot be combined with
   -e or -o.

- V  
   df expands the df command line and writes the completed command to standard output,
   but does not execute the command. df completes the command line, adding information
   derived from /etc/mnttab and /etc/vfstab to input provided by the user.

   The added information relates to the option -F FSType and/or the associated special
   file, depending on which of the two components, if either, was specified on the
   command line.

   If you enter df -V without other options and arguments, you will be shown a list of
   df command lines indicating all mounted file system types and the associated special
   file names.

file

file is either the name of a directory or the name of a special file which contains a file
system. If you name a directory, df examines the file system mounted on that directory.
You may name more than one file.

file not specified:

   df prints out information on all mounted file systems or all file systems of type FSType.

Format 2  

   df[-F FSType][-begklntVv][-o ufs_options][file]...

No option specified

   df prints the number of free blocks and inodes in all mounted or all specified file
   systems.

option

- F FSType

   The file system with which you want to work is of the type FSType. You only need to
   specify this option if the file system is unmounted.

   FSType may be:

   ufs Berkeley file system
   bs2fs bs2fs file system
   proc Pseudo file system for accessing process data
   fdfs Pseudo file system for accessing file descriptors
   nfs File system mounted on the remote host (network file system, Berkeley)

- b  Prints the free disk space (in Kbytes). This option cannot be combined with -o.

- e  Prints the number of free inodes. This option cannot be combined with -o, and it
     overrides the -I option.
Prints the entire *statvfs* structure for all mounted file systems or for *FSType* or *file*. The output consists of 4 lines containing the following information:

- **dir**: Name of the directory on which the file system is mounted (the string *dir* does not appear in the output)
- **device**: Associated special file (the string *device* does not appear in the output)
- **block size**: Block size of the file system
- **frag size**: Fragment size of the file system
- **total blocks**: Total number of blocks in frag size units
- **free blocks**: Total number of free blocks
- **available**: Number of free blocks for non-POSIX administrators
- **total files**: Total number of inodes
- **free files**: Number of free inodes
- **filesystem id**: File system identification number
- **name**: File system designation (the string *name* does not appear in the output)
- **fstype**: File system type
- **flag**: File system flags
- **filename length**: Maximum length of file names

This option cannot be combined with *-o*, and it overrides options *-b*, *-e*, *-k*, *-l*, *-n*, *-r* and *-v*.

Prints the following information in columns for all mounted file systems or for the specified file systems:

- **filesystem**: Special file associated with the file system
- **kbytes**: Total disk space in 1024-byte blocks
- **used**: Used disk space in 1024-byte blocks
- **avail**: Available disk space in 1024-byte blocks
- **capacity**: Percentage of disk space used
- **mounted on**: Name of the directory on which the file system is mounted

The storage space specification takes account of the reserved storage space which can only be used by the POSIX administrator.

This option cannot be combined with *-o*, and it overrides options *-b*, *-e*, *-k*, *-l*, *-n*, *-r* and *-v*. 
-l  `df` prints the number of free blocks and inodes for all locally mounted file systems or for the specified file systems. The `-l` option is set by default. It cannot be combined with `-e` or `-o`.

-n  `df` outputs the file system type. If you combine `-n` with `-o`, then an error message is displayed.

-t  `df` prints the number of free blocks and files as well as the total number of available blocks and files for each mounted or specified file system.
-`t` overrides the `-b` option. Combining `-t` with `-o i` produces a totals line for all the columns in the `-o i` output.

-V  `df` expands the `df` command line and writes the completed command to standard output, but does not execute the command. `df` completes the command line, adding information derived from `/etc/mnttab` and `/etc/vfstab` to input provided by the user.

The added information relates to the option `-F FSType` and/or the associated special file, depending on which of the two components, if either, was specified on the command line.
If you enter `df -V` without other options and arguments, you will be shown a list of `df` command lines indicating all mounted file system types and the associated special file names.

-v  `df` prints the following information in columns for all mounted file systems or for the specified file systems:

<table>
<thead>
<tr>
<th>Mount Dir</th>
<th>Name of the directory on which the file system is mounted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filesystem</td>
<td>Special file associated with the file system</td>
</tr>
<tr>
<td>blocks</td>
<td>Total disk space, in 512-byte blocks</td>
</tr>
<tr>
<td>used</td>
<td>Used disk space, in 512-byte blocks</td>
</tr>
<tr>
<td>free</td>
<td>Free disk space, in 512-byte blocks</td>
</tr>
<tr>
<td>%used</td>
<td>Percentage of disk space used</td>
</tr>
</tbody>
</table>

This option cannot be combined with `-o`, and it overrides options `-b`, `-e`, `-k`, `-l`, `-n`, `-r` and `-v`. 
-o \_ufs\_option

If the file system type is *ufs*, you can use a *ufs*-specific option:

\[ df \]

\[ prints \] the following information in columns:

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Name of the special file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filesystem</td>
<td>Name of the associated special file</td>
</tr>
<tr>
<td>iused</td>
<td>Number of used inodes</td>
</tr>
<tr>
<td>ifree</td>
<td>Number of free inodes</td>
</tr>
<tr>
<td>%iused</td>
<td>Percentage of inodes used</td>
</tr>
<tr>
<td>Mounted on</td>
<td>Name of the directory on which the file system is mounted</td>
</tr>
</tbody>
</table>

The only options with which \(-o\_i\) can be combined meaningfully are \(-F\) and \(-t\).

*file*

*file* is either the name of a directory or the name of a special file associated with a file system. If you name a directory, \(df\) examines the file system mounted on that directory. You may name more than one *file*.

*file* not specified:

\(df\) examines all locally and remotely mounted file systems or all file systems of type *FSType*.

**Format 3**

\[ df\-c[...file]... \]

**option**

\[-c\] The command outputs the type of coding (ASCII or EBCDIC) for all locally mounted file systems or those specified with *file*. This option cannot be combined with other options.

**Hint**

In UFS file systems the available disk space is as a rule less than the free disk space. The reason for this is that some of the disk space (10%) is reserved for privileged applications and is consequently not available for normal applications.

**Error**

Some of the following error messages also indicate the correct syntax, but use some syntax elements that differ from the ones in the above description. The table below shows which syntax elements are equivalent:

<table>
<thead>
<tr>
<th>Syntax element in the error message</th>
<th>Equivalent in the above description</th>
</tr>
</thead>
<tbody>
<tr>
<td>specific_options</td>
<td>\ufs_option</td>
</tr>
<tr>
<td>directory</td>
<td>special</td>
</tr>
<tr>
<td>generic_options</td>
<td>-begktlvV</td>
</tr>
</tbody>
</table>
df: Cannot access file
An invalid or incomplete special file or directory name was specified with `df -V`.

df ufs: Usage: df [-F ufs] [generic options] [-o i] [directory|special]
An invalid `ufs_option` argument was specified with the `-o` option. `ufs_option` can only be `i`.

df: operation not applicable for FSType `FStype`
An invalid file system type (xxx) was specified for `FStype` in the command
`df -F `FStype`-o-i`. The only type that can be specified in combination with the `-o` option is `ufs`.

File
/dev/dsk/*
Special files associated with the file systems

/etc/mnttab
Table of mounted file systems

/etc/vfstab
List of default parameters for each file system

/etc/fs/FStyp/*
Commands for specific file system types

Locale
The following environment variables affect the execution of `df`:

`LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

`LC_ALL` If set to a non-empty string value, override the values of all the other internationalization variables.

`LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

`LC_MESSAGES` Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

`NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`. 


Example 1  Printing the associated file system types for all user directories:

```bash
$ df -n /h*
/home   : ufs
/home2  : ufs
/home3  : ufs
```

Example 2  Printing the entire `statvfs` structure for the file system `/stand`:

```bash
$ df -g /stand
/stand (/dev/dsk/c0d0s10): 512 block size  512 frag size
10710 total blocks  7577 free blocks  7577 available  104 total file
100 free files    10 filsys id      /stand
bfs ftype  0x00000000 flag      14 filename length
```

Example 3  The file system coding is to be output for the `/home` and `/var` directories:

```bash
$ df -c /home /var
/home : EBCDIC
/var  : ASCII
```

See also  `du`
**diff**

**compare two files**

The `diff` utility will compare the contents of `file1` and `file2` and write to standard output a list of changes (with `ed`-like commands) necessary to convert `file1` into `file2`. This list should be minimal. No output will be produced if the files are identical.

**Syntax**

```
diff[-option]..file1..file2
```

No option specified

If the compared files are identical, `diff` produces no output. When the files differ, the output is as follows:

1. `line[range] from ed command line[range] from`
   `file1 file2`
2. lines that are only in `file1`
3. lines that are only in `file2`

The lines are displayed in the following format:

```
a11. n1[,n2]  d n1[,n2]  c
2. < text of line from file1
   .
   .
   _ _ _
3. > text of line from file2
   .
   .
```

`a`, `d`, and `c` are `ed`-like commands meaning:

- `a` append
- `d` delete
- `c` change

The output is to be interpreted as follows:

The `ed` commands `a`, `d`, and `c` with their preceding line (or range) specifications show how `file1` can be converted into `file2`.

If you replace `a` with `d` and `d` with `a` and use the line (or range) specifications to the right, the commands indicate how `file2` can be converted into `file1`. Lines from `file1` are marked `<`; those from `file2` are marked `>`. 
option

-a  `diff` produces a diff with all lines of context from both files. Those lines only in `file1` are prepended with `-`; those only in `file2` are prepended with `+`. Lines which are identical in both files are prepended with `--`

-b  `diff` ignores trailing blanks or tabs at the end of lines as well as differing numbers of blanks at corresponding positions within text lines. Leading blanks and blank lines are reported as differences.

-i  `-i` must not be combined with `-h`.
   `diff` ignores the case of letters; for example, it considers 'A' to be the same as 'a'.

-t  `diff` expands tab characters in output lines. In normal or `-c` output, `diff` adds character(s) to the start of each line that may adversely affect the indentation of the original source lines and make the output lines difficult to interpret. The `-t` option preserves the original source's indentation.

-w  `diff` ignores all blanks (space and tab characters) and treats strings of blanks of any length as equivalent; for example, it considers 'if (a == b)' to be the same as 'if (a==b)'.

The following options are mutually exclusive:

-c  `diff` produces a listing in three parts with a slightly modified output format. The output begins with the names and creation dates of `file1` and `file2`. Then come the lines that differ, with the lines not present in `file2` marked with a minus sign (`-`) and the lines that differ in `file1` and `file2` marked with an exclamation point (`!`). There are also three lines of context before and after the differing lines.

-C number
   Produces a listing of differences identical to that produced by `-c`, but with `number` lines of context before and after each difference.

-e  The `-e` option must not be combined with `-h`, `-l` or `-s`.
   `diff` produces output suitable for use as an `ed` script. The `ed` script contains `a`, `d`, and `c` commands and the related text lines as input for the editor `ed` to convert `file1` to `file2`. Before you pass the script to the editor, you should add the statements `w` and `q` at the end of the script and insert the command `e file1` at the start (see section "ed interactive line editor" on page 289).

-f  The `-f` option must not be combined with option `-h`.
   `diff` generates a similar script to the one created with option `-e`, but converting in the opposite direction. This script, however, is not usable with `ed`.

Editing scripts produced under the `-e` and `-f` options may be incorrect when dealing with lines comprising nothing but a single period.
The following options modify the way in which `diff` works:

- **-h** The `-h` option must not be combined with `-e, -f, -i, -C, -n or -D.
  `diff` operates faster, and you can process files of any length. However, the result
  produced by option `-h` is not reliable!

- **-n** Produces a script similar to `-e`, but with the `ed` commands listed in reverse order. In
  addition, a count of the lines to be changed is printed after each `insert` or `delete`
  command.

- **-D** _string_
  In this case `file1` and `file2` must be C source programs or contain C source fragments.
  `diff` creates a merged version of `file1` and `file2` with C preprocessor controls included
  so that compilation of the result is equivalent to compiling `file1` if `string` is not defined, and
  is equivalent to compiling `file2` if `string` is defined.

The following options are used to compare directories:

- **-l** `diff` produces output in long format. Before checking for differences, `diff`
  pipes each text
  file through `pr` to paginate it. Differences other than those in text files are remembered
  and are summarized after all text file differences have been reported.

- **-r** Applies `diff` recursively to all common subdirectories encountered.

- **-s** Reports files that are identical; these would not otherwise be mentioned.

- **-S** _name_
  Starts a directory `diff` in the middle, beginning with the file `_name_`.

`file1`...`file2`

Names of the files `diff` is to compare. If `file1` is a directory, then a file in that directory with
the same name as `file2` is compared against `file2`. If `file2` is a directory, then the
comparison is made with a file named `file1` from that directory.

If both `file1` and `file2` are directories, `diff` looks in both directories for files with the same
name to compare. In this case `diff` will not compare block special files, character special
files or FIFO special files to any files as well as regular files to directories.

If either the `file1` or `file2` operand is a `-`, the standard input will be used in its place. In this
case the input files must be text files.

**Exit status**

0 Files are the same
1 Files are different
>1 Input error
diff

Error
diff: two filename arguments required
You have specified an incorrect number of files. Only two files can be compared at a time.
diff: No such file or directory
One of the specified files does not exist.
diff: files too big, try -h
You need to use the -h option, as the files to be compared are too large.
diff: Permission denied
You do not have read permission for one of the specified files.

VariableTZ
Determine the locale for affecting the timezone used for calculation of file timestamps written
with the -C and -c options

Locale
The following environment variables affect the execution of diff:
LANG Provide a default value for the internationalization variables that are unset
or null. If LANG is unset of null, the corresponding value from the implemen-
tation-specific default locale will be used. If any of the internationalization
variables contains an invalid setting, the utility will behave as if none of the
variables had been defined.
LC_ALL If set to a non-empty string value, override the values of all the other inter-
nationalization variables.
LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data
as characters (for example, single- as opposed to multi-byte characters in
arguments and input files).
LC_MESSAGES Determine the locale that should be used to affect the format and contents
of diagnostic messages written to standard error and informative messages
written to standard output.
LC_TIME Determine the locale for affecting the format of timestamps written with the
-C and -c options.
NLSPATH Determine the location of message catalogs for the processing of
LC_MESSAGES.
Example 1  The files `file1` and `file2` have the following contents:

<table>
<thead>
<tr>
<th>file1</th>
<th>file2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack Jill</td>
<td>Jack and Jill</td>
</tr>
<tr>
<td>went up the hill</td>
<td>went uphill</td>
</tr>
<tr>
<td>to fetch a pail</td>
<td>of water</td>
</tr>
</tbody>
</table>

By calling `diff`, you can identify the lines in which the two files differ.

```bash
$ diff file1 file2
1,3c1,2
<Jack Jill
<went up the hill
<to fetch a pail
---
>Jack and Jill
>went uphill
```

Explanation: In order to convert `file1` into `file2`, lines 1 to 3 inclusive from `file1` (1,3) must be replaced (c) by lines 1 to 2 inclusive (1,2) of `file2`. The contents of each of these lines are shown in the output lines beginning with the < or > characters. Lines in `file1` are marked <; lines in `file2` are marked >.

Example 2  Compare files and produce an `ed` script:

Contents of `file1`:    Contents of `file2`:

<table>
<thead>
<tr>
<th>today is monday</th>
<th>today is tuesday</th>
</tr>
</thead>
<tbody>
<tr>
<td>it is cold</td>
<td>it is autumn</td>
</tr>
<tr>
<td></td>
<td>it is cold</td>
</tr>
</tbody>
</table>

After the following command, `diff` outputs the `ed` commands with which `ed` could convert `file1` into `file2`. To be able to use this output as input for `ed`, you need to add the statements `w` and `q` to the end of the `ed` script.

```bash
$ diff -e file1 file2
1c
today is tuesday
it is autumn
w
```

See also  `cmp`, `comm`, `ed`, `pr`
dirname  

return directory portion of pathname

dirname can be used to strip the file name (basename) from the full access path. dirname takes a string as its argument, removes the final slash and all characters to the right of it, and writes the rest of the string on standard output. dirname is useful in shell scripts.

Syntax

dirname[string]

string

string can be any string of characters.

dirname deletes the final slash and all characters to the right of it and writes the remaining portion to standard output. If string does not contain a slash, a dot is written to standard output.

string not specified:

a dot is written to standard output.

Locale

The following environment variables affect the execution of dirname:

LANG

Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL

If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH

Determine the location of message catalogs for the processing of LC_MESSAGES.

Example

In the following example, the path prefix /usr/src/cmd is assigned as a value to the NAME variable:

NAME=`dirname /usr/src/cmd/xyz.c`

See also

basename,
du estimate file space usage

`du` outputs the amount of disk space used by directories, subdirectories, and ordinary files in blocks of 512 bytes.

Syntax

```
du[.-a][.-s][.-kx][.-r][.-s][.-file...]
```

No option specified

If `name` is a directory, `du` lists the space occupied by the specified directory and all its subdirectories. The disk space occupied by ordinary files in the specified directory is included in the count, but not listed individually.

If `name` is not a directory, `du` does not output anything for it.

Option

- `-a` If `file` is a directory then `du` reads the storage space allocated to each of the files in this directory individually. If `file` is not a directory then `du` reads the storage space allocated to `file`. The option `-a` can be combined with the option `-s`.

- `-k` `du` outputs the occupied storage space, specifying the number of occupied 1024-byte blocks.

- `-r` `du` issues an error message if `file` is a directory for which you possess no read permission or if it is a file which cannot be opened. The option `-r` is always set.

- `-s` `du` only outputs the total amount of storage space occupied by the section of the file tree or the file. The option `-s` cannot be combined with the option `-a`.

- `-x` When file sizes are calculated, only those files which use the same special file as `file` are considered.

File

Name of the file or directory for which disk usage is to be displayed. A file with two or more links is only counted once. A file that has a hole in it (e.g. if only block 1 and block 100 are used) will result in an incorrect block count. If `name` refers to an ordinary file, `du` remains silent if invoked without options.

`file` not specified:

The disk space occupied by the current directory and all its subdirectories is displayed.

Locale

The following environment variables affect the execution of `du`:

`LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
**du**

**LC_ALL**  If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1**  List the disk usage, in 512-byte blocks, of all subdirectories of the current directory whose names begin with `DIR`. The space occupied by ordinary files is included, but not listed individually.

```
$ du DIR*
6   DIR-1
136  DIR-2/SUB-1
140  DIR-2
5    DIR-3
54   DIR-4
```

**Example 2**  List the disk usage, in 512-byte blocks, of all subdirectories of the current directory whose names begin with `DIR`. The disk usage of ordinary files is listed individually in this case because the `-a` option is used.

```
$ du -a DIR*
1    DIR-1/file1
1    DIR-1/file2
1    DIR-1/file3
6     DIR-1
0     DIR-2/file4
1    DIR-2/file5
34    DIR-2/SUB-1/file6
99    DIR-2/SUB-1/file7
136   DIR-2/SUB-1
140   DIR-2
1    DIR-3/file8
1    DIR-3/file9
5     DIR-3
50    DIR-4/file10
1     DIR-4/file10.bak
54    DIR-4
```

**See also**  `df`
dumpfs dump file system

The command `dumpfs` outputs the super block and cylinder group information for the specified file system. This command can, for example, be used to determine the block and fragment sizes which are specified for the file system as well as the minimum required storage space in percent.

**Syntax**

```
dumpfs file_system
```

`file_system`  
Name of the file system for which the information is to be output.

**Hint**  
This command may only be entered by the POSIX administrator. It is only of use to system maintenance staff for diagnostic purposes.

**Example**  
Printing out information on the `/` directory:

```
# dumpfs /
magic   11954   time    Thu Jun 8 10:04:06 1995
sblkno  4       cblkno  6       lblkno  8       dblkno  72
sbsize  4096    cgszie  4096    cgoffset 8      cgmask  0xffffffff8
ncg     2       size    2048    blocks  1907
bsize   8192    shift   13      mask    0xfffffe000
                    fsize   4096    shift   12      mask    0xfffff000
                    frag    2       shift   1       fsbtodb  1
                    minfree 10%  maxbpg  1024
                    maxcontig 1  rotdelay  0ms  rps    60
                    csaddr  72      cssize  4096    shift   9      mask    0xfffffe000
                    ntrak   8      nsect   16      spc    128      ncy  32
                    cpg    16      bpg    512      fpg    1024      lpg    2048
                    nindir  2048   inopb   64      nspf    2
                    nbfree  400    ndir    87      nifree  2157    nffree  5
crgotor  0       fmod    0       ronly   0
                    cs[.cs_(nbfree,nidir,nifree,nffree):
        (66,29,918,3) (334,58,1239,2)
                    cg  0:
                    magic  90255   tell   6000    time    Thu Jun 8 08:35:05 1995
cgx     0       ncy   16      nblk    2048    ndblk   1024
                    nbfree  66    ndir    29      nifree  918    nffree  3
                    rotor   870   t rotor    833    rotor    864
                    frsum   3
        sum of frsum: 3
        fused:  0-6, 8-832, 835-846, 848-1126, 1129-1134, 1136
```
dumpfs

free:  575, 844-845, 864, 871, 876-881, 894-901, 904-913, 916-917, 920-1023

b:
  c0: (0)  0 0 0 0 0 0 0 0 0
  c1: (0)  0 0 0 0 0 0 0 0 0
  c2: (0)  0 0 0 0 0 0 0 0 0
  c3: (0)  0 0 0 0 0 0 0 0 0
  c4: (0)  0 0 0 0 0 0 0 0 0
  c5: (0)  0 0 0 0 0 0 0 0 0
  c6: (0)  0 0 0 0 0 0 0 0 0
  c7: (0)  0 0 0 0 0 0 0 0 0
  c8: (0)  0 0 0 0 0 0 0 0 0
  c9: (0)  0 0 0 0 0 0 0 0 0
  c10: (0) 0 0 0 0 0 0 0 0 0
  c11: (0) 0 0 0 0 0 0 0 0 0
  c12: (0) 0 0 0 0 0 0 0 0 0
  c13: (5) 1 0 0 0 2 0 2 0
  c14: (29) 8 0 7 0 8 0 6 0
  c15: (32) 8 0 8 0 8 0 8 0

cg 1:
  magic  90255 tell  40e000 time  Thu Jun  8 10:04:51 1995
  cgx  1 ncyld  0 niblk  2048 ndblk  1024
  nbfree  334 ndir  58 nifree  1239 nffree  2
  rotor  234 irotor  716 frotor  256
  frsum  2
sum of frsum: 2
iused:  0-805, 807, 809, 811
free:  353, 355-1023

b:
  c0: (0)  0 0 0 0 0 0 0 0 0
  c1: (0)  0 0 0 0 0 0 0 0 0
  c2: (0)  0 0 0 0 0 0 0 0 0
  c3: (0)  0 0 0 0 0 0 0 0 0
  c4: (0)  0 0 0 0 0 0 0 0 0
  c5: (14) 3 0 3 0 4 0 4 0
  c6: (32) 8 0 8 0 8 0 8 0
  c7: (32) 8 0 8 0 8 0 8 0
  c8: (32) 8 0 8 0 8 0 8 0
  c9: (32) 8 0 8 0 8 0 8 0
  c10: (32) 8 0 8 0 8 0 8 0
  c11: (32) 8 0 8 0 8 0 8 0
  c12: (32) 8 0 8 0 8 0 8 0
  c13: (32) 8 0 8 0 8 0 8 0
  c14: (32) 8 0 8 0 8 0 8 0
  c15: (32) 8 0 8 0 8 0 8 0

U22794-J-Z125-6-76
The POSIX shell built-in `echo` writes its arguments to standard output and terminates. The following takes place before the output is generated:

1. As with every other command, the shell interprets the command line arguments and then passes the edited arguments to `echo`. The argument separators recognized by the shell are blanks and tabs.

2. `echo` interprets any remaining special characters that control output (see description of `argument` below).

3. `echo` displays processed arguments as follows:
   - Each argument is separated from the next by a blank even if you have entered more than one argument separator between individual arguments in the call.
   - A newline character is output at the end of the last argument.

You can use `echo` to
- examine values of shell variables,
- generate messages in shell scripts and thus test how they function,
- test how the shell interprets a command call without the command being executed,
- redirect data to a pipe and test how the pipe processes this input.

Besides the shell built-in `echo`, there is also a command called `/usr/bin/echo`. The shell generates a new process to execute `/usr/bin/echo`. Otherwise, `/usr/bin/echo` operates in the same way as `echo`.

**Syntax**

```
<arg>[<arg>]
```

**argument**

Any string delimited by blanks or tabs. The last argument is terminated by means of a command separator.

You may specify any number of arguments, provided they are separated by at least one tab or blank character.

As with any other command, this string is also initially interpreted by the shell:

- If the string contains an asterisk, question mark, or `[]`, the shell replaces this string by all suitable file names in the current directory. If no suitable file name is found, the shell passes the string on to `echo` unaltered.

- `'string'`
  All shell metacharacters are escaped by the single quotes. The shell passes the string as an argument to `echo` without the quotes. All blanks, tabs and newline characters entered within single quotes are retained.
The shell executes `string` as a POSIX command and passes the output of this command to `echo`. Characters assigned to the environment variable IFS are interpreted by the shell in the output as argument delimiters. The default characters assigned to IFS are the blank, tab or newline characters.

The double quotes escape all shell metacharacters except for the backslash `\`, backquotes `...` and the dollar sign `$`. The shell passes the processed string to `echo` as an argument. All blanks, tabs, and newline characters within the double quotes are retained.

The shell built-in `echo` also interprets the control characters described below. Since the backslash has a special meaning for the shell, it must be escaped:

- by preceding it with another backslash `\`
  This also applies if the argument containing this control character is enclosed within double quotes.
- by single quotes `...'`
  If the argument containing this control character is enclosed within single quotes, the backslash need not be escaped again.

**Example**

```
$ echo hello\tuser
hello   user
$ echo hello'\'tuser
hello   user
$ echo 'hello\tuser'
hello   user
$ echo "hello\tuser"
hello   user
```

**Control characters**

The following control characters influence the output of `echo`, provided the backslash has been appropriately escaped:

Please note the terminal-specific characteristics.

- by preceding it with another backslash `\`
  `echo` prints the arguments entered up to this control character, omitting the newline and ignoring the remaining arguments.
if
(form feed) This control character is not converted if the output is written to the terminal. If the output from `echo` is sent to a printer, for example, all arguments entered after this control character will be written on the next page.
`\v` corresponds to `CTRL L`

\n
(nnewline) All arguments that follow the newline character are written in the next line. `\v` corresponds to `J` or `CTRL J`

\t
Control character for tab; `echo` moves the cursor to the next tab stop. Characters entered after `\v` are written by `echo` from this column onward. The setting of tab stops depends on the data terminal that is used. On a Siemens Nixdorf 97801 terminal, tab stops are set at the following columns:
1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 79
`\v` corresponds to `H` or `CTRL F`

\0n
`n` must be a one, two or three figure octal number. The `echo` command outputs the corresponding character (see Tables and directories, EDF04 character set). In this way you can, for example, generate characters with an internal code greater than FF (EDF04) even if you do not possess an 8-bit terminal. A dummy character is output to represent non-displayable characters (device-dependent).

Example
```bash
$ echo \0337|od -xc
0000000  df15
          337 \n
$ echo \00337|od -xc
0000000  1bf7    1500
          033    7 \n
$ echo \000337|od -xc
0000000
```

If a character in octal notation is to be followed by an additional digit that is not part of the octal number, you must use the full three-digit form of the octal number.
If control characters of the above type are to be interpreted by neither the shell nor `echo`, they must be entered as follows:

**Example**

```bash
$ echo \\
\n$ echo \\
\t
$ echo ‘\\t’
\t
$ echo "\\t"
\t
```

The above example applies to the remaining strings as well.

- **argument not specified:**
  - `echo` produces only a blank line.

**Locale**

The following environment variables affect the execution of `echo`:

- **`LANG`**
  - Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **`LC_ALL`**
  - If set to a non-empty string value, override the values of all the other internationalization variables.

- **`LC_CTYPE`**
  - Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **`LC_MESSAGES`**
  - Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **`NLSPATH`**
  - Determine the location of message catalogs for the processing of `LC_MESSAGES`.


**Example 1** The following interactive session will demonstrate when blanks, tabs and newline characters are retained:

```
$ echo Today is     Monday.
Today is Monday.
```

In this case `echo` receives three arguments and returns them with one blank between each.

```
$ echo "Today is     Monday."
Today is     Monday.
```

All blanks entered within the double quotes are retained.

**Example 2** User *anna* wants to know what value is assigned to the HOME variable:

```
$ echo $HOME
/home/anna
```

**Example 3** The following line in a shell script will cause an error message to be written on standard error:

```
echo File $1 not found >&2
```

**Example 4** A string constant is to be placed before the output of the `date` command:

```
$ echo "Today's date and time: \c"; date
Today's date and time: Mon Mar  9 17:22:05 MEZ 2009
```

The control character `\c` prevents `echo` from interpreting the newline. The backslash must be escaped despite the double quotes.

See also  *print*
**ed** interactive line editor

`ed` is an interactive line editor. Furthermore, with the help of `ed` scripts (see “Working with `ed` scripts” on page 303), you can easily process several files with the same sequence of commands. `ed` can handle the output of the `diff -e` command (see page 274f).

### Syntax

```
ed[-|][-s][-p_string][..file]
```

- **-s**
  The `-s` option corresponds to the old `-` option, which is still supported.

  The `-s` option suppresses the following default outputs:

  - number of bytes processed by the `ed` commands:
    - `e` (edit)
    - `r` (read)
    - `w` (write)
  - the question mark character, which warns against inadvertent deletion of the buffer contents during execution of the `ed` commands:
    - `e` (edit)
    - `q` (quit)
  - the exclamation mark `!` used as an `ed` prompt after a `!` command.

- **-p_string**
  In `string` you can define the prompt that `ed` displays in command mode. `string` can be one or more characters.

  `-p_string` not specified:
  `ed` does not display a prompt string.

- **file**
  The name of the file that you wish to process. `ed` copies the file into its internal buffer and saves `file` as the current file name.

  `file` not specified:
  You start by working on an empty buffer and only decide upon a file name when using the `w..file` command to write the buffer contents into a file.
ed, buffer and operating modes

ed buffer

When ed is invoked, a buffer is opened.

If you have not entered a file name, the buffer will be empty. You can then fill it with text during your editor session.

If you have named a file, a copy of the file is read into the buffer. During your editor session, you essentially process the contents of the buffer.

Before you exit the editor again, you must decide whether you want to save the newly created or modified buffer contents by writing them to a file.

If you wish to save the buffer contents, you use the \texttt{w[\_file]} (write) command to write the contents of the buffer back to the specified file (by default the one named when ed was invoked) and then exit the editor with the \texttt{q} (quit) or \texttt{Q} (Quit) commands or with the \texttt{END} key.

If you do not wish to save the buffer contents, you can exit the editor without writing back the contents of the buffer with \texttt{w}. You do this by pressing \texttt{Q} or \texttt{q} twice. When you press \texttt{q} the first time, ed will issue a ? as a warning to prevent you from inadvertently deleting the buffer contents. The buffer will not be deleted unless and until you press \texttt{q} a second time. If you prefer, you can also enter \texttt{Q} or \texttt{END} instead of the second \texttt{q}.

Operating modes

ed provides you with two operating modes: command mode and input mode. ed enters command mode when it is called with \texttt{ed[\_file]/[\]}]. In command mode you specify a command in a line and confirm your input by pressing \texttt{\}}. 

Input mode is activated by means of one of the following commands:

\texttt{a} (append)
\texttt{i} (insert)
\texttt{c} (change)
(see “ed commands” on page 293).

In input mode, all the input characters which follow, including various non-printing characters (e.g. key codes of cursor keys), are written to the working copy in the buffer. ed does not accept any commands in input mode. If you wish, you can define a prompt for command mode (see option \texttt{-p} and ed command \texttt{P}) so that you can instantly detect the mode in which you are currently working. You leave input mode either by pressing \texttt{DEL} or by entering a period (.) in the first column and hitting \texttt{\}}. When you press \texttt{DEL}, ed normally ignores all input since the last \texttt{\}} and displays a ? as a warning.
Command structure

For ed, there is a current line at all times. As a rule, the line last processed by a command represents the current line. If you do not specify another address in front of the commands, they will always refer to the current line.

Most ed commands have the following structure:

```
[range]ed-command[parameter...]  
```

range

The range you enter identifies the lines in the buffer to which the ed command is to be applied. One or two addresses can be specified in range:

- `range = address`
  The line identified by address is selected.

- `range = address1,address2`
  `address1,address2` identifies the range between the specified limits (inclusive). The search for both addresses begins at the current line, which is not changed until commands are executed.
  * address2 must refer to a line that follows the line referenced by address1 in the buffer. Otherwise, ed reports an error.

- `range = address1;address2`
  `address1;address2` identifies the range between the specified limits (inclusive). The new current line is set to the line identified by address1, and only then is address2 calculated. You can use this feature to define the starting line for forward and backward searches (see the section “Addresses” on page 292, // and /RE/ on the following page).
  * address2 must refer to a line that follows the line referenced by address1 in the buffer. Otherwise, ed reports an error.

- `range not specified:`
  ed assumes the default address for each command; this address is described for each of the ed commands.

If ed requires no address but you have nevertheless specified one, ed reports an error.

If you have specified more addresses than necessary, ed uses the last address(es) specified.
ed, addresses

### Addresses

Addresses are constructed as follows:

<table>
<thead>
<tr>
<th>Address</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Current line</td>
</tr>
<tr>
<td>$</td>
<td>Last line</td>
</tr>
<tr>
<td>n</td>
<td>nth line</td>
</tr>
<tr>
<td>'x</td>
<td>The line marked with the letter x. x must be a letter in lowercase (see k command).</td>
</tr>
</tbody>
</table>

/RE/ A simple regular expression RE enclosed in /.../ (see section “Regular POSIX shell expressions” on page 877) addresses the first line containing a character string that matches the regular expression, searching forward from the current line. If ed does not find a match in any line, the search wraps around and continues from the beginning of the file until a match is found or the current line is reached again. If the regular expression RE contains the delimiter / or ?, this must be escaped using \. The characters \n in a regular expression do not match newline characters in the searched text!

A regular expression may only appear once in range. Thus, /RE1/,/RE2/, for example, will only address the first line that matches /RE2/.

// A null regular expression // addresses the line matching the regular expression last specified.

?RE? Like /RE/, except that the search begins at the current line and proceeds toward the beginning of the file. If the regular expression RE contains the delimiter / or ?, this must be escaped using \.

addr[+]n nth line after the line identified by addr.

addr[-]n nth line before the line identified by addr.

+n n lines forward from the current line.

-n n lines backward from the current line.

[addr]+... One line forward (+) or backward (-) from the line identified by addr. Each occurrence of + or - respectively increases or decreases the address specification by 1. Thus, ++ addresses the second line after the current line (2 lines forward).

, A comma stands for the address pair 1,$ if followed by a command; if not, the last line is output.
A semicolon stands for the address pair .,$ if followed by a command; if not, the last line is output.

**ed commands**

The following list includes a systematic overview of all *ed* commands that you can enter in the command mode. The detailed command description that follows is arranged in alphabetical order.

<table>
<thead>
<tr>
<th>Activate input mode</th>
<th>Modify text</th>
</tr>
</thead>
<tbody>
<tr>
<td>a append</td>
<td>a append</td>
</tr>
<tr>
<td>c change</td>
<td>c change</td>
</tr>
<tr>
<td>i insert</td>
<td>i insert</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output prompt in command mode</th>
<th>Output lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>P prompt</td>
<td>p print</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Undo commands</th>
<th>Explain errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>u undo</td>
<td>h help</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abort commands</th>
<th>Modify text</th>
</tr>
</thead>
<tbody>
<tr>
<td>[DEL] --- abort execution of command</td>
<td>a append</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Explain errors</th>
<th>Modify text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help toggle help mode on and off. If help mode is on, error messages are printed for all subsequent ? diagnostics (see page 303)</td>
<td>a append</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modify text</th>
<th>Output lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>d delete</td>
<td>l list</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Output lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>- - output addressed lines</td>
<td>p print</td>
</tr>
<tr>
<td>address/+number</td>
<td>address - - output addressed lines</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Output lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>n print indicated lines with line numbering</td>
<td>p print</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Single character command</th>
</tr>
</thead>
<tbody>
<tr>
<td>u undo most recent command</td>
</tr>
</tbody>
</table>
**ed, command overview**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output line numbers</strong></td>
<td></td>
</tr>
<tr>
<td>address=</td>
<td>output addressed line number</td>
</tr>
<tr>
<td>n</td>
<td>output lines with line numbering</td>
</tr>
<tr>
<td><strong>Move specified line ranges</strong></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>append a copy of addressed lines</td>
</tr>
<tr>
<td>m</td>
<td>move lines to after addressed line</td>
</tr>
<tr>
<td><strong>Search and replace</strong></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>search and replace</td>
</tr>
<tr>
<td><strong>Join lines</strong></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>join contiguous lines</td>
</tr>
<tr>
<td><strong>Mark lines</strong></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>mark addressed lines</td>
</tr>
<tr>
<td><strong>Process selected lines with commands</strong></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>apply command list globally to all lines that match the given /RE/</td>
</tr>
<tr>
<td>G</td>
<td>apply interactive command list globally to all lines that match the given /RE/</td>
</tr>
<tr>
<td>v</td>
<td>like g, but for all lines that do not match /RE/.</td>
</tr>
<tr>
<td>V</td>
<td>like G, but for all lines that do not match /RE/.</td>
</tr>
<tr>
<td><strong>Change current file name</strong></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>change/display current file name</td>
</tr>
<tr>
<td><strong>Execute shell commands</strong></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>send command to shell for interpretation</td>
</tr>
<tr>
<td><strong>Read files into buffer</strong></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>delete buffer and read named file into it</td>
</tr>
<tr>
<td>E</td>
<td>clear buffer without warning and reload original</td>
</tr>
<tr>
<td>r</td>
<td>read file into buffer</td>
</tr>
<tr>
<td><strong>Save buffer contents</strong></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>write buffer contents into file</td>
</tr>
<tr>
<td>W</td>
<td>append buffer contents to file</td>
</tr>
<tr>
<td><strong>Quit the editor</strong></td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>quit ed</td>
</tr>
<tr>
<td>Q</td>
<td>quit ed without warning</td>
</tr>
<tr>
<td>[END]</td>
<td>quit ed</td>
</tr>
</tbody>
</table>
Description of the ed commands

The square brackets [] are not to be entered. They merely indicate that the entry enclosed within them is optional.

As a rule, only one command may be entered per line. However, you can append the suffixes \l, \n, or \p to the commands (with the exception of \e, \f, \r, and \w) if the functions described under \l, \n, and \p are to be executed.

\[address\]\l
\text

\l (append) reads the text input and appends it to the line addressed in address. The current line is now either the last line of the inserted text or, if you have not entered any text, the addressed line. Address 0 is legal for this command: it causes the "appended" text to be inserted at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 2048 per line (including the newline character).

address not specified:
address = .

\[range\]\c
\text

\c (change) deletes the specified range and replaces these lines with the text input. The current line is now either the last line of the entered text or, if you have not entered any, the line preceding the deleted lines.

range not specified:
range = ..

\[range\]\d
\text

\d (delete) deletes the specified range. The line after the last line deleted becomes the current line. If the deleted lines were at the end of the buffer, the new last line becomes the current line.

range not specified:
range = ..

\e[\_file]
\text

\e (edit) deletes the entire buffer and reads in a copy of the contents of the named file. If the contents of the buffer have been modified but not saved with \w, \ed prevents inadvertent deletion of the buffer by first issuing a ? as a warning. If you now enter \e again, the old buffer contents are deleted without further comment. The number of bytes read is output provided you did not call \ed with the -s option. The current line is the last line of the buffer. The specified file name file is remembered for possible use as a current file name in subsequent \e, \r, and \w commands. If file is replaced by an exclamation point !, the rest of the line is interpreted as a shell command and executed. The output of the shell command is read into the buffer. A shell command that is preceded by an ! is not stored as the file name.
ed, command description

file not specified:
file = current file name

E[...file]
(Em) behaves like edit, except that it overwrites the buffer without issuing a ? as a warning even if the buffer contents have been modified but not saved.
file not specified:
file = current file name

f[...file]
(file) sets the current file name to file. The current file name is used by the commands e, E, r and w.
file not specified:
ed outputs the current file name.

[range].g/RE/commandlist
(global) first marks all lines containing a character string which matches the regular expression RE. RE is a simple regular expression (see section “Regular POSIX shell expressions” on page 877). Then commandlist is executed for each line marked, with the current line being set to the next marked line in each case.

A single command or the first in a commandlist must be entered in the same line as the g command. All lines in commandlist except the last one must end with \; the last command itself with \.

The commands a, i, and c with their associated input text are allowed. All lines in text must also be terminated with \. The period "." usually used to terminate input can be omitted from the last line of commandlist.

An empty commandlist is equivalent to the p command.

The g, G, v, and V commands are not permitted in the commandlist.

The global command must not be combined with the ! command.

range not specified:
range = 1,.$

[range]G/RE/
(Global) is the interactive variant of the g command. First, every line that matches RE is marked. RE is a simple regular expression (see section “Regular POSIX shell expressions” on page 877). The first of the marked lines is printed. At the same time, this line becomes the current line.

You now have the option of specifying a command to be executed (other than a, c, i, g, G, v, or V). After the execution of that command, the next marked line is printed, and so on.
A newline acts as a null command; an ampersand & causes the re-execution of the most recent command executed within the current invocation of $G$.

Note that the commands input as part of the execution of the $G$ command may address and affect any lines in the buffer. The $G$ command can be terminated by pressing the [DEL] key.

\[\text{range not specified:}\]
\[\text{range} = 1,$\]

\[\text{h (help)}\] issues a short error message that explains the reason for the most recent ? symbol displayed on the screen. See Error for a list of possible error messages.

\[\text{H (Help)}\] causes ed to enter a mode in which error messages are printed instead of the ? symbol for all errors that follow. It will also explain the previous ?, if any. You can deactivate the help feature by calling the $H$ command again.

The $H$ mode is normally off. See Error for a list of possible error messages.

\[\text{[address]}i\]
\[\text{text}\]
\[\text{. (insert) inserts the given text before the line referenced by address. The last inserted line of text, if any, becomes the current line; otherwise, the addressed line does. This command differs from the append command only in the placement of the entered text. Address 0 is not legal. The maximum number of characters that may be entered from a terminal is 2048 per line (including the newline character).}\]

\[\text{address not specified:}\]
\[\text{address} = .\]

\[\text{[range]}j\]
\[\text{. (join) joins all lines in the specified range into one line by removing the appropriate newline characters. If you have only specified one address, nothing happens. The new line becomes the current line.}\]

\[\text{range not specified:}\]
\[\text{range} = .,.\]

\[\text{[address]}kx\]
\[\text{. (mark) marks the line referenced by address with the letter specified in x, where x must be in lowercase. The marker itself is not output. The marked line can then be addressed by using 'x (single quote x). The current line is unaffected.}\]

\[\text{address not specified:}\]
\[\text{address} = .\]
ed, command description

[range]
(list), in contrast to p, outputs the specified range as follows: some non-printing characters are output in alternative representation (e.g. tab characters), the remaining non-printing characters are output as octal numbers. Overlength lines are folded into several lines, each terminated by the line continuation character \l. Each line must end with \$. An l command can be appended to any command other than e, f, r, or w.

The following alternative representations are used:

\l Backslash (for distinguishing octal characters)
\a Warning, Bell 1)
\b Backspace 1)
\f Form Feed
\n Newline
\r Carriage Return
\t Tab
\v Vertical tab 1)

1) These non-printing characters are supported on character terminals only (i.e. when the POSIX shell is accessed via rlogin)

range not specified:
range = ..

[range]maddress
(move) re-positions the lines in the specified range after the line addressed by address. The last of the shifted lines becomes the current line. If you specify a value of 0 for address, the range is moved to the beginning of the file. If address falls within the range of moved lines, ed issues an error message.

range not specified:
range = ..

[range]n
(number) prints the range of addressed lines, preceding each line by its line number and a tab character. The last line to be printed becomes the new line. The n command can be appended to any command other than e, f, r, or w.

range not specified:
range = ..
[range]p
(print) prints the range of addressed lines. Non-printing characters are not output unchanged. Overlength lines are continued in the next line, i.e. are not identifiable as such. The current line is the last line printed. The p command can be appended to any command other than e, f, r and w. For example, dp deletes the current line and prints the new current line.

range not specified:
range = ..

P (Prompt) causes ed to use an asterisk * or the defined prompt string as a prompt in command mode (see option -p above). You can deactivate this mode again by calling P a second time. This mode is normally deactivated.

q (quit) terminates ed. If you have changed the buffer contents since the last time the buffer was saved or overwritten, but have not yet written these changes to a file with w, ed outputs a ? as a warning (to prevent inadvertent deletions) and waits for further input. By entering [END], Q, or the q command a second time, you can now exit ed without further warnings and without saving the buffer.

Caution!
If you continue after the first q with actions that do not change the buffer contents, no new warning will be issued when you press q again. ed will be terminate without saving the buffer.

Q (Quit) terminates ed immediately without warning even if you have changed the buffer contents since last saving or overwriting and have not yet written these changes to a file with w.

[address][r_file]
(read) reads the named file and inserts its contents after the line identified by address. The address 0 is legal for this command and causes the file contents to be written at the start of the buffer. If the read was successful, the number of bytes read is output unless you called ed with the -s option. The current line is the last line read in. The currently remembered file name is not changed to file unless you invoked ed without a file name and file is the very first file name mentioned since ed was invoked. If file is replaced by the ! character, the rest of the line is interpreted as a shell command and executed. The output of this command is then read. Such a command is not remembered as the current file name.

address not specified:
address = $

file not specified:
file = current file name
ed, command description

[rangeres/RE/replacement_string/[g][l][n][p][count]]
(substitute) searches each line in range for strings which match RE. RE is a simple
regular expression (see section “Regular POSIX shell expressions” on page 877). On
each line in which a match is found, strings that match RE are replaced by
replacement_string.
If ed does not find a matching string, it returns a ? to indicate an error. To delimit the
regular expression RE from the s command and the replacement_string, any other
character except the blank or newline can be used instead of /. The character selected
is recognized as a delimiter by virtue of coming immediately after s. Afterwards, the
current line is the line in which the last substitution took place.

count Substitute for the countth occurrence only of the RE found on each addressed
line.
g Globally substitute for all non-overlapping instances of the RE rather than
just the first one. If both g and count are specified, the results are unspec-
ified.
l Write to standard output the final line in which a substitution was made. The
line will be written in the format specified for the l (list) command.
n Write to standard output the final line in which a substitution was made. The
line will be written in the format specified for the n (number) command.
p Write to standard output the final line in which a substitution was made. The
line will be written in the format specified for the p (print) command.

Metacharacters in the replacement string

Metacharacters Meaning
& is replaced by the string which matches regular expression RE in a
successful search.
\n is replaced by the character string matching the nth regular
subexpression, delimited by(...), of RE where n is a decimal digit. If
nested parenthesized subexpressions are present, n is determined
by counting occurrences of \ from left to right.
% is replaced by the replacement string from the most recent s
command if replacement_string consists solely of the % character.

The special meaning of these characters can be suppressed by preceding each of them
with a backslash \.
ed, command description

Splitting a line in the replacement string

To split a line you can include an escaped newline character, i.e. \n, in replacement_string. This type of substitution command cannot be used in a commandlist with a g or v command.

range not specified:
range = ..

[ranger]a
copies the addressed range after the specified line a. 0 is allowed for a. The current line is the last of the copied lines.

range not specified:
range = ..

u (undo) nullifies the effect of the most recent command that modified anything in the buffer. The following commands can be undone: a, c, d, g, i, j, m, r, s, t, v, G, and V.

[ranger]v/RE/commandlist
(vice versa) causes commandlist to be executed on all lines containing no string that matches the regular expression RE. RE is a simple regular expression (see section “Regular POSIX shell expressions” on page 877). v functions identically to the global command g with the selection criterion reversed.

v should not be combined with the ! command.

range not specified:
range = 1,$

[ranger]w[...file]
(write) writes the specified range into the named file. The currently remembered file name is not changed if previously set. If the file name has not been set, file is used as the new current file name. The old contents of file are overwritten in the process. If the named file does not exist, it is created. The current line remains unchanged. After a successful write operation the number of characters written is displayed, provided you did not invoke ed with the -s option.

If file is replaced by !, the rest of the line is interpreted as a shell command and executed. The standard input for the shell command is the range of addressed lines. Such a command is not remembered as the current file name.
**ed, command description**

- **range** not specified:
  - range = 1,$

- **file** not specified:
  - file = current file name

- `[range]W[..file]`
  (Write) appends the specified range to the end of the named file. This command is the same as the `w` command above, except that `w` overwrites existing files. If file does not exist, it is created.

- **address**
  The line identified by address is output.

- `[address]+num`
  This command outputs the line that is num lines after the line identified by address.

  - address not specified:
    - address = .

- `[address]=`
  The line number of the line identified by address is output; the current line is not changed by this command.

  - address not specified:
    - address = $

- `!command`
  The remainder of the line after the ! is interpreted and executed as a shell command. If an unescaped % character appears within the text of command, it is replaced by the remembered file name. !! causes the last command to be repeated. In both cases, the expanded command line is echoed. On completion of the command, ed is reactivated. The current line remains unaffected. This command must not be combined with `g` and `v`!

- `J`
  Entering `J` alone in the command mode causes the line after the current line to be printed. This is the same as specifying `.+1p`. You can use this feature to step through the buffer.

  The input of a command in command mode or a text line in input mode must be terminated by pressing the `J` key.

- [DEL]
  You can use the [DEL] key to interrupt a currently executing ed command or to cancel the entry of a line. ed will then respond by displaying a ‘?’.

- [END] or [CTRL-D] or @@d
  The [END] key (or any of the key combinations) has the same effect as the `q` command.
If the closing delimiter of a regular expression or of a replacement string (e.g. a `/`) is the last character before a newline, you may omit that delimiter. In such a case, the addressed line is printed. The following pairs of commands are equivalent:

\[ s/s1/s2 \quad s/s1/s2/p \]
\[ g/s1 \quad g/s1/p \]
\[ ?s1 \quad ?s1? \]

**Working with ed scripts**

Since `ed` reads commands and text that is to be inserted from standard input, you can redirect the input to a file and have `ed` read the file instead. Thus

\`
$ ed - file < ed_scriptfile > output
``

edits the named `file` and processes it with the `ed` commands stored in the specified `ed_scriptfile`. The `-s` option suppresses default output of message texts to the screen.

The advantage of using `ed scripts` is that they allow you to reproduce specific command sequences at any time and use them as often as required. Furthermore, this feature enables you to perform these tasks in a background process while you continue working at the terminal without interruptions:

\`
$ ed file < ed_scriptfile&
``

If your `ed` script contains errors, `ed` will exit on encountering the first error.

**Exit status**

- 0 if successful
- >0 if `ed` is called incorrectly or if a script is aborted due to incorrect use of `ed` commands.

**Error**

If you do not enter a blank between option `p` and `string` in the `ed` call or if you forget to enter the `string`:

\`
ed: -p arg missing
``

If you make a mistake when entering `ed` commands:

? This means that the command contains syntax errors.

?file

The indicated `file` is not available or cannot be read.
More detailed information can be obtained by using the \textit{h} and \textit{H} commands. The commonest error messages are listed below. They are all self-explanatory:

- line out of range
- warning: expecting 'w'
- no space after command
- unknown command
- bad range
- cannot open input file
- illegal or missing delimiter
- illegal suffix
- illegal or missing filename
- no match

File

\textit{ed.hup}

Data is saved in this file if \textit{ed} receives the SIGHUP signal (see section \textit{``kill terminate or signal processes''} on page 444).

/\texttt{var/tmp}

If this directory exists, it is used as the directory for storing the temporary work file.

/\texttt{tmp}

/\texttt{tmp} is the directory used to store the temporary work file if the \textit{TMPDIR} variable is not assigned the name of an existing directory and \texttt{var/tmp} does not exist.

Variable \textit{TMPDIR}

If this variable is set and is not null, it is used instead of /\texttt{var/tmp} as the directory for storing the temporary work file.

\textit{HOME}

Determine the pathname of the user's home directory.

Locale

The following environment variables affect the execution of \textit{ed}:

\textit{LANG}

Provide a default value for the internationalization variables that are unset or null. If \textit{LANG} is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

\textit{LC_ALL}

If set to a non-empty string value, override the values of all the other internationalization variables.

\textit{LC_COLLATE}

Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.
**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions. **LC_CTYPE** also governs which characters the *ed* command *i* treats as non-printing.

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1** Example of an *ed* Script:
The first three lines of a file are to be replaced by one line with the text "Addresses", and all occurrences of the word "Street" are to be replaced by "Plaza".

Contents of the *edscript*:
```
1,3c Addresses
1,$s/Street/Plaza/g
w
```

Processing of a file with commands from *edscript*:
```
$ ed file < edscript
```

When *ed* reads commands from a file instead of the keyboard, the editor is exited as soon as *ed* encounters the first incomprehensible command.

**Example 2** Example of a *here* Script (see *sh)*:
In a set of files that are to be passed as command-line arguments to the script *xy*, the first three lines are to be replaced by one line with the text "Addresses", and all occurrences of the word "Street" are to be replaced by "Plaza".

Contents of the script file *xy*:
```
for i in $*
do
   ed $i << scrend
   1,3c Addresses
   1,$s/Street/Plaza/g
   w
scrend
done
```

Processing of the files \texttt{text1}, \texttt{text2} and \texttt{text3} with script \texttt{xy}:

\begin{verbatim}
$ sh xy text1 text2 text3
\end{verbatim}

The string \texttt{<< scrend} after the \texttt{ed} call causes the shell to transfer the text up to the string \texttt{scrend} as input to \texttt{ed}. The second string \texttt{scrend} must exist as a single word in a line, without leading blanks. The special meaning of the $ symbol must be escaped with a \texttt{\backslash} in the address specification \texttt{1,$}. This is because the shell would otherwise interpret the following \texttt{s} as the name of a shell variable.

Example 3 The following example demonstrates and explains a number of \texttt{ed} commands:

\begin{verbatim}
$ ed
P
*a
line1
line2
line3
.
*1,$p
line1
line2
line3
*p
line3
*n
3 line3
*1,$n
1 line1
2 line2
3 line3
*2c
line2 \rightarrow line2 \rightarrow line2
another line2
.
*p
another line2
*1,$n
1 line1
2 line2 line2 line2
3 another line2
4 line3
*4s/3/4
line4
*n
4 line4
*1,$s/l/L/g
\end{verbatim}

- Call \texttt{ed}
- Output prompt * in command mode
- Append to current line, in this case beginning of file
- Input text
- Terminate input mode
- Output line 1 through to last line
- Output current line (= last line processed)
- Output current line with line number
- Output line 1 through to last line with line numbers
- Replace line 2 with following input including tabs
- Output current line
- Output current line
- Output line 1 through to last line with line numbers
- In line 4, search for the character 3 and replace it with 4
- Output current line with line number
- Search from line 1 through to last line for all l characters and replace them with L
*1.$p - Output line 1 through to last line
Line1
Line2  Line2  Line2
another Line2
Line4

*2l - Output line 2 with non-printing characters as mnemonic
overstrikes.
Line2>Line2>Line2

*2r file - Read the contents of file into the buffer after line 2
and output the number of characters read in. file must
already exist!

46
*1.$n - Output line 1 through to last line with line numbers
1  Line1
2  Line2  Line2  Line2 - This line and the two which follow are
3  line1 of file out of file
4  line2 of file
5  last line of file
6  another Line2
7  Line4
*6,7c - Replace lines 6 to 7 by the following input
very last line

*1.$n - Output line 1 through to last line with line numbers
1  Line1
2  Line2  Line2  Line2
3  line1 of file
4  line2 of file
5  last line of file
6  very last line

*q - Attempt to quit editor with q
? - Warning
*h - Explain question mark

warning:expecting 'w'
- Explanation: w command expected by ed (to save contents
of the buffer) was not entered

*w exp - Write the contents of the buffer to the file exp
85 - Number of characters written is output

See also ex, grep, sed, sh, stty, umask, vi
Tables and directories, Regular expressions
### edt

**screen-oriented editor EDT (BS2000)**

`edt` calls the BS2000 file editor, see the EDT (BS2000/OSD) manual [7]. EDT V17.0 and higher is called in compatibility mode. `edi` is the editor for block-mode terminals. If you are working with character-mode terminals, you can use `ex` or `vi` for example.

You can use this command and the EDT functions to copy files from the POSIX file system to the BS2000 and vice versa.

**Syntax**

```
edt[-k][-index]..file
```

- **-k** Prior to editing, the file contents are converted from ASCII to EBCDIC code. Before being written back to the file, the contents are recoded to ASCII. You can use this option if you wish to use `edt` to process an ASCII-coded file.

- **-index**
  This option is required if files containing more than 9999 lines have to be read. The product `<index> * 0.0001` gives the number of the first line and the step size used between line numbers. If there is no `<index>` specification then `<index>`=10000, i.e. the number of the first line and the step size value are both 1. Within EDT itself, the EDT statement `renumber` can be used to perform renumbering.

**file**

File name of the POSIX file. You may specify one file only. If the specified file does not yet exist, the empty work file 0 is opened.

**Hint**

EDT can create, read in, copy, write back and close POSIX files. The `@XOPEN`, `@XCOPY`, `@WRITE` and `@CLOSE` statements are available for this purpose (see the manual “EDT (BS2000/OSD)” [7]).

To write the opened POSIX file back to the BS2000 file system, use the EDT statement `@write'<filename>' when in EDT level 0.

You can terminate EDT with `@HALT` (write the opened POSIX file back to the POSIX file system in interactive mode) or `@RETURN` (unconditionally write the opened POSIX file back to the POSIX file system in interactive mode). Use `@WRITE` at EDT level 0 to write an opened UFS file back (to UFS).

Blank lines in the file are displayed as lines with a length of 1 (the character X'0D'). They are subsequently converted back to blank lines on saving the file.

The tab character (‘t’) is not expanded to the corresponding number of spaces.

The `edt` command cannot be used in a pipe.
Built-in sh command

**Error**

edt cannot be used within a pipe
The `edt` command was incorrectly used in a pipe.
edt cannot be used within a forked process
The `edt` command was entered in a subshell. This is not allowed.
edt: file `file` open for read failed
You do not possess read permission for `file`.

*** read only *** in the edt message line
The file which you wish to edit is write-protected.
edt: file `file` open for write failed, permission denied
You do not have write permission. The following messages are then displayed:
edt: edited file(s) not saved!
edt: terminate edt? reply (y=yes; n=no)?
edt: file `file` open for write failed, no such file or directory
The path does not exist. The following messages are then displayed:
edt: edited file(s) not saved!
edt: terminate edt? reply (y=yes; n=no)?
edt: file `file` open for write failed, UFS file system failed, no space
Not enough space in the POSIX file system to write the file. The following messages are then displayed:
edt: edited file(s) not saved!
edt: terminate edt? reply (y=yes; n=no)?

If, for the reasons above, the file cannot be written, the user may stay in EDT and take the necessary action to save the file, for example change the write permission or use `@WRITE 'file'` to save the file.

**Example**
The POSIX file `/usr/home/file.pos` is to be saved in the BS2000 file system as the BS2000 file `file.bs2`.

$ edt /usr/home/file.pos

......
@WRITE `file.bs2` (EDT statement)
......
egrep  search a file with an ERE pattern

`egrep` reads lines from one or more files or from standard input and compares these lines with the specified patterns. Unless told otherwise (by options), `egrep` copies every line that matches one of the patterns to standard output.

`egrep` permits the use of extended regular expressions in the specified pattern (see section “Regular POSIX shell expressions” on page 877).

If you specify more than one input file, the relevant file name will be displayed before each output line.

Syntax

<table>
<thead>
<tr>
<th>Format 1:</th>
<th><code>egrep[...-bchilnvy...]-e...patternlist[...file...]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2:</td>
<td><code>egrep[...-bchilnvy...]-f...patternfile[...file...]</code></td>
</tr>
<tr>
<td>Format 3:</td>
<td><code>egrep[...-bchilnvy...]...patternlist[...file...]</code></td>
</tr>
</tbody>
</table>

The formats are described together since the patterns which `egrep` uses to compare the input lines are specified either via `patternlist` or using the option `-e patternlist` or `-f patternlist`. You must specify one of these arguments. Two of them or all three together are not permitted.

No option specified

`egrep` outputs all lines that match at least one of the patterns specified in `patternlist`. If you specify more than one input file, each output line will be preceded by the name of the file in which the line was found.

Option

- `-b` (block) Each output line is preceded by the number of the block in which it was found. Each file is made up of 512-byte blocks which are numbered consecutively from 0. The `-b` option is sometimes useful in locating disk block numbers by context (see the `offset` argument for the `od` command, for example).

- `-c` (count) `egrep` outputs only the number of lines found (i.e. the lines that `egrep` would have displayed without the `-c` option, see Example 4); the lines themselves are not displayed.

- `-h` (hidden) When searching multiple files, `egrep` does not write the file name before each output line.

- `-l` or `-y` (ignore) `egrep` does not distinguish between uppercase and lowercase

- `-l` (list) `egrep` simply outputs the names of files that contain at least one of the matching lines. (These are the lines that `egrep` would output if the `-l` option were omitted, see Example 5.) Each file name is printed just once. The lines themselves are not displayed.

- `-n` (number lines) Each output line is preceded by its line number in the relevant input file. Line numbering starts at 1. If `egrep` is reading from standard input, the line number refers to the standard input.
egrep

-\(v\) (v - vice versa) \textit{egrep} outputs all the lines which correspond to \textit{none} of the specified patterns.

Together with option -c:
\textit{egrep} only outputs the number of such lines.

Together with option -l:
\textit{egrep} only outputs the names of the files containing such lines.

-\(e\).patternlist
(expression) You will need this option if the first expression in \textit{patternlist} begins with a minus sign. When used in conjunction with the -e option, a \textit{patternlist} of this type is not interpreted as an option itself but as a list of patterns which \textit{egrep} is to use in searching for matching input lines.

-\(f\).patternfile
(file) \textit{egrep} reads the pattern list from the file named \textit{patternfile}. Each line in \textit{patternfile} is interpreted as an extended regular expression.

patternlist
A list of extended regular expressions that \textit{egrep} is to use in searching for matching input lines (see section "Regular POSIX shell expressions" on page 877). Individual regular expressions must be separated by the newline character. Any newline character within \textit{patternlist} is interpreted like an OR separator (\(|\)) in an extended regular expression. Regular expressions of the type (rls) can also be specified without the parentheses: rls (see Example 1 on page 313).

If \textit{patternlist} contains newline characters or other characters that have a special meaning for the shell, you must enclose the specified \textit{patternlist} in single quotes: 'patternlist'.

If the first expression in \textit{patternlist} begins with a minus sign, you must specify \textit{patternlist} along with the -e option or terminate the option list with -- to prevent the \textit{patternlist} from being interpreted as an option itself.

file
Name of the file that \textit{egrep} is to scan. You may name any number of files.

\textit{file} not specified:
\textit{egrep} reads its input from standard input.

grep, fgrep and egrep
The \textit{grep}, \textit{fgrep} and \textit{egrep} commands perform similar functions and are largely identical in terms of usage. The following section lists the most important differences between these commands.

\textit{grep} processes simple regular expressions. Only one regular expression may be specified in each call.
*fgrep* processes strings only. However, you may specify several strings in one call. The strings can either be entered directly in the command line, separated by newline characters, or passed to *fgrep* from within a file.

*fgrep* is fast and compact and can search for a large number of strings. All specified strings are searched for in each individual line.

*egrep* processes extended regular expressions. Among other things, these include all simple regular expressions with one exception: the \(...\) construct used in simple regular expressions does not have a special meaning in extended regular expression syntax and is hence not processed by *egrep*.

Several regular expressions can be specified together, separated by newline characters. *egrep* interprets these newline characters as an OR operator (the vertical bar character; see *Tables and directories, Regular POSIX shell expressions*). The regular expressions can either be specified directly in the command line or passed to *egrep* via a file.

### Exit status

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Matching lines found</td>
</tr>
<tr>
<td>1</td>
<td>No matching lines found</td>
</tr>
<tr>
<td>&gt;1</td>
<td>Syntax error or &quot;File cannot be opened&quot;. This exit status remains valid even if lines have been found in other input files.</td>
</tr>
</tbody>
</table>

### Locale

The following environment variables affect the execution of *egrep*:

- **LANG**: Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**: If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_COLLATE**: Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.

- **LC_CTYPE**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

- **LC_MESSAGES**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**: Determine the location of message catalogs for the processing of *LC_MESSAGES*. 

Example  The files *customer1* and *customer2* will be used as a basis for all following examples. Their contents are given below:

**customer1:**

```
080685    999.98  20 Units  Item 038   Nicolson Ltd.
120387   1240.25   3 Units  Item 023   Robinson Ltd.
180588    330.87   1 Units  Item 332   Robinson Ltd.
```

**customer2:**

```
morrow lance, 86 sherwood street, london w1
robinson peter, 16 garden hill, london ec3
```

Example 1  Output lines that match a pattern (without an option and with option `-i`):

```bash
$ egrep Robinson customer1 customer2
```

```
customer1:120387   1240.25   3 Units  Item 023   Robinson Ltd.
customer1:180588    330.87   1 Units  Item 332   Robinson Ltd.
```

If you also wish to find lines containing the word *robinson* in lowercase you enter:

```bash
$ egrep -i robinson customer1 customer2
```

```
customer1:120387   1240.25   3 Units  Item 023   Robinson Ltd.
customer1:180588    330.87   1 Units  Item 332   Robinson Ltd.
customer2:robinson peter, 16 garden hill, london ec3
```

More complicated patterns can be set up with the help of regular expressions, e.g.:

Display lines which contain the string *Nicolson* or *Robinson*:

```bash
$ egrep '(Nicol|Robin)son' customer1 customer2
```

```
customer1:080685    999.98  20 Units  Item 038   Nicolson Ltd.
customer1:120387   1240.25   3 Units  Item 023   Robinson Ltd.
customer1:180588    330.87   1 Units  Item 332   Robinson Ltd.
```

Instead of the regular expression (NicolRobin)son you could also use the following regular expression:

```
(Nicolson|Robinson)
```

In this case you can leave out the parentheses;

```
Nicolson|Robinson
```

The OR operator (`|`) in the last expression Nicolson|Robinson could also be replaced by a newline character (see Example 2).
Example 2  Using several patterns (without an option and with option -f):

```
$ egrep '^1 > 1$' customer1 customer2
customer1:120387 1240.25 3 Units  Item 023  Robinson Ltd.
customer1:180588 330.87 1 Units  Item 332  Robinson Ltd.
customer2:morrow lance, 86 sherwood street, london w1
```

Alternatively, you could write both patterns into a file called names (each pattern in a separate line) and then call egrep as follows:

```
$ egrep -f names customer1 customer2
```

The same result is obtained when the newline character that separates the patterns ^1 and 1$ is replaced by the OR operator:

```
$ egrep '^1|1$' customer1 customer2
```

Example 3  Output lines that match none of the specified patterns (option -v):

```
$ egrep -v '^1 > 1$' customer1 customer2
customer1:080685 999.98 20 Units  Item 038  Nicolson Ltd.
customer2:robinson peter, 16 garden hill, london ec3
```

The above call thus yields all lines that neither begin nor end with 1. The same result can also be obtained with the following call (see Example 2):

```
$ egrep -v '^1|1$' customer1 customer2
```

Example 4  Display the number of lines found (option -c):

To begin with, the number of lines that start with 1 are to be output for each input file.

```
$ egrep -c '^1' customer1 customer2
customer1:2
customer2:0
```

The number of lines that do not begin with 1 are now to be displayed.

```
$ egrep -c -v '^1' customer1 customer2
customer1:1
customer2:2
```
Example 5  Display file names only (option -l):

Display file names only (option -l):

The names of files containing lines that begin with 1 are to be output first.

$ egrep -l '^1' customer1 customer2
   customer1

The names of files containing lines that do not start with 1 are displayed next.

$ egrep -l -v '^1' customer1 customer2
   customer1
   customer2

Example 6  Display found lines with relevant line numbers (option -n):

$ egrep -n -i robinson customer1 customer2
   customer1:2:120387   1240.25   3 Units Item 023 Robinson Ltd.
   customer1:3:180588    330.87   1 Units Item 332 Robinson Ltd.
   customer2:2:robinson peter, 16 garden hill, london ec3

See also  ed, fgrep, grep, sed
**env**

**set environment for command execution**

The `env` command can be used to display current environment variables and their values or to set them for the execution of a specific command. `env` inspects the current environment, modifies it according to the assignment in `name=value`, and then executes the command in the modified environment. The existing specifications for `name` and `value` are overwritten by the new ones, and the new arguments are merged into the inherited environment before the command is executed. The valid environment for the execution of `command` thus consists of the new specifications together with any unmodified environment variables.

If no command is specified, `env` prints the resulting environment.

**Syntax**

```bash
env[[-il..-][name=value]...command[...arg...]]
```

- `-i` - The original environment is ignored; `command` is then executed with exactly the environment specified by the arguments.
- `option` corresponds to the old `-` option, which will continue to be supported.

**name=value**

- `name` specifies the name of a variable that is to be valid for `command`.
- `value` is the value of `name` which is to apply to `command`.

**command**

- Name of the command or shell script which you would like to have executed in the defined environment.

**arg**

- Argument, e.g. positional parameters or user-defined variables which can be passed to `command`.

**Exit status**

- **0** The `env` utility was completed successfully
- **1-125** Error
- **126** The utility specified by `utility` was found but could not be invoked.
- **127** The utility specified by `utility` could not be found.

**Variable**

**PATH**

Determine the location of the `utility`. If `PATH` is specified as a `name=value` operand to `env`, the value given will be used in the search for `utility`. 
Locale  The following environment variables affect the execution of `env`:

**LANG**  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**  If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example 1**  Output the current values of environment variables:

```bash
$ env
__=/usr/bin/env
TTY=/dev/term/004
IO_CONVERSION=NO
PATH=/usr/bin:/opt/bin::/etc:/usr/sbin
TERM=BLOCK
HZ=100
LOGNAME=TESTUSER
PROGRAM_ENVIRONMENT=SHELL
HOME=/TESTUSER
TZ=MEZ-1MSZ-2,M3.5.0/02:00:00,M9.5.0/03:00:00
MAIL=/var/mail/TESTUSER
SHELL=/sbin/sh
USER=TESTUSER
LANG=
```
Example 2  Output the modified values of environment variables:

```
$ env PATH=$HOME/proz
_=/usr/bin/env
TTY=/dev/term/004
IO_CONVERSION=NO
PATH=/TESTUSER/proz
TERM=BLOCK
HZ=100
LOGNAME=TESTUSER
PROGRAM_ENVIRONMENT=SHELL
HOME=/TESTUSER
TZ=MEZ-1MSZ-2,M3.5.0/02:00:00,M9.5.0/03:00:00
MAIL=/var/mail/TESTUSER
SHELL=/sbin/sh
USER=TESTUSER
LANG=
```

The `PATH` environment variable has been modified.

Example 3  Output modified values of environment variables using the - option:

```
$ env - PATH=$HOME/proz
```

PATH=/TESTUSER/proz

The original environment is ignored.

Example 4  Invocation of the file `ardor`, located in `/TESTUSER/SAYINGS`, i.e. in a subdirectory of the HOME directory.

Contents of file `ardor`:

```
echo "$1 $2 $3 $4 $2 $3 $1 love $4!"
```

Now call `ardor` from any location in your file tree with the arguments `I know that you`:

```
$ env PATH=$HOME/SAYINGS ardor I know that you
```

I know that you know that I love you!

With the new definition of the `PATH` variable, you effectively define the location where the entered command is to be sought (the file `ardor` in our case), i.e. in a subdirectory of the HOME directory, which you specify with the value of the variable HOME ($HOME). The character strings I, know, that, and you are passed as arguments to the positional parameters $1, $2, $3 and $4.

The contents of `ardor` are correctly executed here because `echo` is a built-in command of the `sh` shell. Due to the change in the `PATH` variable, all POSIX commands in `/usr/bin`, `/usr/sbin` or `/opt/bin` can no longer be found. To enable the execution of POSIX commands again, the `PATH` variable must be modified as shown in the example below.
Example 5  Call the file *delcopy*, which is located in the directory /TESTUSER/proc. This file contains a script that compares two files and deletes one of them if both files are identical.

Contents of the file *delcopy*:

```bash
if cmp -s $1 $2
then
rm $2
fi
```

Call the *delcopy* file from any location in your file tree with the arguments *file1* and *file2*:

```bash
$ env PATH=$PATH:$HOME/proc delcopy file1 file2
```

In this case, the new path is appended to the original one. This ensures that the *delcopy* script can be executed along with all SINIX commands contained in it. If only the path for *delcopy* were specified, the following error message would be issued.

```
/TESTUSER/proc/delcopy: cmp: not found
```

See also  *sh*, *set*, *exec*,
*profile*, *environ* [12]
eval construct command by concatenating arguments

The POSIX shell built-in `eval` can be used to pass command-line arguments to the `sh` shell for execution. The shell evaluates these arguments and then executes them as commands.

The specified command-line arguments are thus evaluated twice by the shell:

- First, when the shell processes the `eval` command line.
- Second, when the processed command-line arguments are executed by the shell as commands. Each command-line is processed by the shell before execution.

When do you need eval?

The shell processes each command line in a number of steps (see section “Execution” on page 64), interpreting specific metacharacters in each processing step. The original command line may be modified by the individual processing steps.

If, for example, the shell replaces a variable with its value or a command with its output, metacharacters may occur in the command line which may not be interpreted by the shell in subsequent processing steps. The shell built-in `eval` makes sure that metacharacters left by the previous step are interpreted in the following step (see also Examples on page 321).

Syntax

```bash
eval [argument....]
```

argument

Any string delimited by blanks or tabs. The last argument must be terminated by a command separator.

You may specify any number of arguments with at least one blank or tab between them. The specified arguments are executed by the shell as a command.

Locale

The following environment variables affect the execution of `eval`:

- `LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `LC_ALL` If set to a non-empty string value, override the values of all the other internationalization variables.

- `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
Built-in sh command

**eval**

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

Example 1  The following command line cannot be executed as intended unless **eval** is used:

```bash
$ cmd='date;echo name'
$ cmd
sh: date; echo: not found
$ eval $cmd
Mon Mar 09 15:46:02 MEZ 2009
name
```

To begin with, the shell replaces the `cmd` variable with its value. After the value has been substituted for the variable, the shell no longer recognizes the colon as a command separator. Calling **eval** causes the shell to interpret the semicolon as intended, since the `$cmd` argument is processed twice, i.e. rescanned by the shell.

Example 2  The shell script `evaltest` will be used to demonstrate how the **eval** command is processed by the shell. The script file has the following contents:

```bash
set -x
loginname=max
for i in 1 2 3 4
do
  eval group$i=$loginname$i
  eval echo \$group$i
done
```

`set -x` causes the shell running the shell script to write the commands processed to standard error output before executing them.

On executing this shell script, you will receive the following output on the screen (without the line numbers):

```bash
$ sh evaltest
1 + loginname=max
2 + eval group1=max1
3 + group1=max1
4 + eval echo $group1
5 + echo max1
6 max1
```

This example shows only what is output the first time the loop is run.
Line 2 shows how the first `eval` command line is processed: here the shell has substituted the value 1 for $i and the value max for $loginname. `eval` must be invoked as the shell does not recognize the equals sign as an assignment operator the first time the loop is run. The shell executes an assignment only if the variable name (i.e. the string in front of the equals sign =) begins with a letter or the underscore character _ and contains letters, numbers and underscores only. `group$i` is an illegal variable name as it contains the dollar character.

Line 3 shows that the assignment is made. When the command-line arguments are interpreted by the shell a second time, the equals sign is interpreted as intended, since the argument preceding the = sign represents a legal name for a shell variable.

Line 4 demonstrates how the second `eval` command line is processed: the shell has substituted the value 1 for the positional parameter $i and removed the escape character \ preceding the $. `eval` is required in this case to ensure that the shell replaces the `group1` variable by its value.

Line 5 shows how the `echo` command is processed. Here, the shell has substituted the value max1 for $group1. Line 6 contains the output of the `echo` command.

See also `set`
**ex** command and display editor

*ex* is a line-based text editor.

Only users who have accessed the POSIX shell via *rlogin* can use *ex* (except in line mode).

*ex* offers you various operating modes:

- In the *ex* input mode you enter text directly.
- In the *ex* command mode you enter commands. Among other things, commands enable you to:
  - position the cursor
  - search for (and replace) text patterns with the aid of regular expressions
  - switch to another file
  - invoke a shell

You also have the option of switching from the line editor *ex* to the screen editor *vi*.

**Layout of this description**

The description of how to call *ex* on POSIX shell level is followed by the sections:

- Functionality of *ex*
- Operating modes of *ex*
- Saving the editor buffer and quitting *ex*
- Presetting *ex*
- Current and alternate files
- Regular expressions
- Replacement strings
- Buffers
- Error and signal handling

- *ex* commands
- Addressing
- Parameters
- Commands

- *ex* options
option

-s (silent) All interactive output of the editor is suppressed. The -s option must be set when
\textit{ex} is to read its commands from a command script.

The -s option has replaced the old -option.

-t..tag
\textit{tag} \textit{ex} loads the file containing the \textit{tag} for editing and positions the tag definition under
the cursor at the center of the screen. The tags file containing the search strings for the
definitions must be in the same directory. This option is typically used by
C programmers to quickly locate the definition of a function or macro on calling the
editor. The tags file needed for this purpose must first be built with the \textit{ctags} command.

-r..file
(recover) Restores the \textit{ex} session in which you were editing \textit{file} if \textit{ex} or the system
\textit{crashed} during your session. Changes that are only in the editor buffer are not copied to the edited file after a system
or \textit{ex} editor crash. POSIX does, however, try to save the buffer contents by creating a
copy of it, if possible, \textit{file} is recovered, including any changes made prior to the crash,
and is retrieved into the \textit{vi} buffer. You can now continue editing the file or save the
changes in another file.

-L Lists the names of all files saved following an editor or system crash.

-w..n
Sets the value of the \textit{ex} window option (see page 349) to \textit{n}.

-R (readonly) The named \textit{file} is opened in readonly mode. This mode prevents the original
\textit{file} from being accidentally overwritten, but allows you to write the buffer contents to a
file with another name.

\textbf{Caution!}
The file can still be overwritten with \textit{w!} in the \textit{ex} command mode.

-v (vi) Invokes the \textit{vi} editor. A detailed description of \textit{vi} is provided in this manual (see
page 803).

-c..command
Positions the editor at a specific line of the file to be edited or executes an \textit{ex} command
when \textit{ex} is invoked. If you position the editor at a line, the selected line will be displayed
in the center of the screen. By contrast, if you have an \textit{ex} command executed, the editor
will be positioned at the last line of the file after executing the command, unless the \textit{ex}
command you specify (e.g. search) positions the editor itself.

\textit{command} not specified:
The \textit{ex} editor is positioned at the end of the file.
command specified:

command may be specified as a line number (n), a search command (/pattern), or any other ex command (see “Commands” on page 334). The specified command is executed on invocation of ex:

n  n must be an integer. ex is positioned to the nth line of the file.

/pattern  ex is positioned to the first line containing pattern. If the pattern you specify includes metacharacters, you must escape them to protect them from being interpreted by the shell.

'ex-command' or "ex-command"

ex-command can be any ex command. The ex command must be quoted (using single or double quotes) in order to prevent it from being interpreted by the shell. If the editor is not already positioned as a result of an ex command, ex will be positioned to the last line of the file.

Example

When ex is called with:

ex -c /tuesday appointments

the file appointments is opened, and the cursor is positioned at the first line that contains the word tuesday.

file

Name of the file that you wish to edit. If you specify a number of files, these files will be processed in the order in which they are listed. The ex command n can be used to switch to the next file.

file not specified:

ex opens an empty buffer into which you can write your text. The contents of this buffer are not assigned a name until you write them back (with w) to a file or assign a name explicitly using the f command.

Caution!

If your file contains null characters, ex will discard them when reading in the file. No null characters appear in files written by ex.
Functionality of `ex`

`ex` always works on a buffer, not on files. When you start an `ex` session, a working copy of the file you are editing is read into the editor buffer. All the editing changes you make are initially performed in this buffer and are not saved until you use the `w` (write) command to copy the buffer contents back into the original file. Once you have done this, you can quit `ex` using the `q` command. If you wish to exit the `ex` editor without writing back the changes to the original file, you must enter `q!`.

Null ASCII characters (see section “ASCII character set (ISO 646)” on page 889) are discarded in input files and thus cannot appear in output files.

Operating modes of `ex`

`ex` offers you two file editing modes:

- the `ex` command mode, and
- the `ex` input mode.

You can also switch from `ex` to the `vi` editor, which provides you with additional operating modes (for more details see “`vi modes`” on page 808).

When `ex` is invoked, it comes up in command mode, as indicated by the prompt `:` displayed on the screen. The commands `a` (append), `i` (insert), and `c` (change) can be used to switch to the input mode, which enables you to add or modify text in the buffer (see “Commands” on page 334).

In input mode, all subsequent input characters, including various non-printing characters, are written into the buffer. Commands are not interpreted as such. You can exit from input mode by entering a period (.) in the first column and then pressing `]`.

Saving the editor buffer and quitting `ex`

`ex` must be in command mode to save the editor buffer or to quit the `ex` editor.

Use the following command to save the contents of your editor buffer to the file edited or that specified:

```
:w[.file]
```

(write) The contents of the editor buffer are saved to the file specified.

`file` not specified:

The contents of the editor buffer are written back to the edited file.
You can quit `ex` using one of the following options:

- **q** (quit) `ex`. This works only if the editor buffer has not been modified at all or if the modified editor buffer has been saved to a file.

- **q!** (quit) `ex`. Unsaved modifications made to the editor buffer are not saved.

- **x** Quit `ex` and write the modified editor buffer to the edited file.

- **wq[[_file]]** (write and quit) Quit `ex` and write the modified editor buffer to the `file` specified.

  **file** not specified:
  The contents of the editor buffer are written to the edited file.

**Caution!**

`ex` does not issue a warning if you place text in named buffers and do not save it before exiting the editor. The text is irretrievably discarded.

**Presetting `ex`**

The `ex` editor can, to some extent, be adapted to suit your requirements and working habits. This entails using the `ex` command `set`, which allows you to set or change specific options (see "`ex options` on page 345). The changes you make remain in effect for the current session. Details on how to change default settings permanently can be found in the `vi` command description (see page 803).

**Current and alternate file**

The file (or its copy) being edited by `ex` at any given moment is referred to as the current file. The alternate file is the file most recently mentioned in an editing command. If you switch to the alternate file, the file you switch from becomes the new alternate file. When specifying files, you can use the `%` character for the current file and the `#` character for the alternate file. These characters are replaced in file names by the names of the current file and the alternate file, respectively.

**Example**

```
w %#.bak
```

backs up the current file to a file with the same name but with the extension `.bak`.
Regular expressions

The interpretation of metacharacters in regular expressions depends on whether or not the magic option has been set for the ex editor (see “ex options” on page 345).

When magic is set:

character
An ordinary character stands for itself. The following are metacharacters, not ordinary characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>(caret) at the beginning of a pattern</td>
</tr>
<tr>
<td>$</td>
<td>(dollar sign) at the end of a pattern</td>
</tr>
<tr>
<td>*</td>
<td>(asterisk) anywhere except at the beginning of a pattern</td>
</tr>
<tr>
<td>.</td>
<td>(period) at any position in a pattern</td>
</tr>
<tr>
<td>[</td>
<td>(left square bracket) at any position in a pattern</td>
</tr>
<tr>
<td>~</td>
<td>(tilde) at any position in a pattern</td>
</tr>
</tbody>
</table>

Metacharacters have special meanings and must be escaped with a backslash \ if they are to lose their special meanings.

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>^ at the beginning of a pattern stands for the beginning of the line.</td>
</tr>
<tr>
<td>$</td>
<td>$ at the end of a pattern stands for the end of the line.</td>
</tr>
<tr>
<td>.</td>
<td>. stands for any one character.</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt; matches the beginning of a word. The word must begin with a letter, digit, or underscore and be preceded by the beginning of the line or a character other than the above.</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt; matches the end of a word.</td>
</tr>
</tbody>
</table>
| [string]  | Any character contained in string, where string is not a sequence of blank characters. The following characters have special meanings within string:

- a dash - between a pair of characters defines a range, e.g. [a-z] matches a, z, and all the characters which come between them in the ASCII collating sequence.
- if a caret ^ is the first character in string, it causes the construct to match any characters not included in string. |

If these characters are escaped by a backslash, they lose their special meanings.

* Zero, one or more occurrences of the regular expression preceding the * character.

~ Matches the replacement string used in the last s command (substitute, see “Commands” on page 334).
\( \text{\textbackslash (pattern)} \)

A regular expression \textit{pattern} can be enclosed in parentheses escaped by backslashes. This serves only to identify them for substitution actions (see Replacement strings below).

\textbf{rs} A sequence of two regular expressions \textit{r} and \textit{s} is a regular expression in itself. \textit{rs} matches all strings which consist of a string matching \textit{r} followed by a string matching \textit{s}.

When \textit{nomagic} is set:

If \textit{nomagic} has been set, the only characters that have special meanings are:

- caret \textbackslash{} at the beginning of a pattern
- dollar \$ at the end of a pattern, and
- backslash \textbackslash{}.

The following characters only have a special meaning if they have been escaped by a preceding backslash \textbackslash{}:

- period \.
- asterisk *
- left square bracket [
- tilde ~

\textbf{Replacement strings}

The ampersand character \& (with a preceding backslash, i.e. \textbackslash\&, if \textit{nomagic} is set) in the replacement string stands for the text matched by the pattern, i.e. the text to be replaced.

The tilde \textbackslash{} (\textbackslash{} if \textit{nomagic} is set) is replaced by the replacement string used in the previous \textit{s} (substitute) command.

The sequence \textbackslash{}\textit{n}, where \textit{n} is an integer, is replaced by the text that is matched by the pattern enclosed in the \textit{n}th set of parentheses \textbackslash{}\textit{(...)}.

If the replacement string includes the sequence \textbackslash{}\textit{u} or \textbackslash{}\textit{l}, the character that immediately follows \textbackslash{}\textit{u} (upper) or \textbackslash{}\textit{l} (lower) in the replacement string will be converted to uppercase (for \textit{u}) or lowercase (for \textit{l}) if this character is a letter. The sequence \textbackslash{}\textit{U} or \textbackslash{}\textit{L} converts into uppercase or lowercase all letters until the end of the replacement string or until the sequence \textbackslash{}\textbackslash{}\textit{E} or \textbackslash{}\textbackslash{}\textit{e} is encountered.
Buffers

*ex* uses one unnamed and 26 named buffers, named a through z.

You can use these buffers to copy text with *ya* (yank), to save deleted text with *d* (delete), and to subsequently retrieve this text with *pu* (put).

The *d*, *pu*, and *ya* commands use the unnamed buffer unless you explicitly name a buffer. In other words, if you do not specify a buffer for a delete operation, for example, the material you delete will be saved in the unnamed buffer.

You can save blocks of text in up to 26 buffers. The buffers are designated in either lowercase or uppercase letters from a-z (or A-Z). If you use lowercase letters, the old contents of the buffer will be overwritten by the new text, i.e. the old contents are deleted. If you use uppercase letters, the old buffer contents are not overwritten; the new text is appended to the old one instead.

Error and signal handling

When an error occurs during an *ex* session, *ex* sends the BEL character (acoustic signal; see section “ASCII character set (ISO 646)” on page 889) to the terminal and prints an error message.

If an interrupt signal is received, *ex* returns to command mode, allowing you to enter a new *ex* command.
If the editor input is from a file, *ex* exits at the interrupt signal.
Entering commands

Unlike most \texttt{vi} commands, which are interpreted and executed by \texttt{vi} immediately, \texttt{ex} commands must be "dispatched" by pressing the \texttt{Esc} key.

Command lines which start with double quotes " are ignored. This means that you can use lines of this kind to insert comment lines in a command script.

Addressing

An address is used to specify a particular line. Some \texttt{ex} commands expect one or more addresses as input. The specified line or line range (an inclusive range of all lines from the first address to the second) is then processed by the \texttt{ex} command.

For \texttt{ex}, there is at all times a current line. This is explicitly addressed by a period (.). As a rule, the current line is the line last processed by a command or the line to which the editor has been explicitly positioned (e.g. by a search command).

Addresses must be separated from one another by a comma or a semicolon. Such address lists are evaluated by \texttt{ex} from left to right.

\begin{itemize}
\item When a semicolon (;) is used as a separator, the current line is set to the value of the first address before the second address is interpreted.
\item When a comma (,) is the separator, all addresses are calculated relative to the current line. The current line is not changed until the command is executed.
\end{itemize}

If more addresses are given than the command requires, then all but the last address (when a line is expected) or the last two addresses (if a range is expected) are ignored. When a command requires two addresses, the first line that is addressed must precede the second one in the buffer. If you do not specify any address (null address), \texttt{ex} will use the current line by default.

\begin{itemize}
\item \texttt{address}
\item . Current line of the buffer.
\item $ Last line of the buffer.
\item n n\textsuperscript{th} line in the buffer. The lines are numbered sequentially from 1.
\item 'x The line marked with the letter \texttt{x}, where \texttt{x} must be a lowercase letter (see "Commands" on page 334, \texttt{ma} or \texttt{k}). \texttt{'}x\texttt{'} is the address of the marked line. Before \texttt{ex} executes a movement command, the current line is marked. You can return to it with \texttt{''} (2 single quotes).
\end{itemize}
/RE/
A simple regular expression enclosed in /.../ (see section “Regular POSIX shell expressions” on page 877) addresses the first line which contains a character string which matches the regular expression, searching forward from the current line. If necessary, `ex` wraps around from the end of the buffer to its start to continue the search (see “ex options” on page 345, `wrapscan`). If the line matching `RE` is only to be printed, the second `/` may be omitted.

`RE` not specified:
The previous specified pattern is used.

`?RE?`
As `RE`, but searching backward.

`+n`
`-n`
If an address starts with a plus or minus sign followed by a decimal number `n` (optional), the line that lies `n` lines after (+) or before (-) the current line will be addressed. Thus, `.+3`, `+3` and `+++` have the same effect.

`adr+`
`adr-`
If an address ends with a plus or minus sign, the line that is located one line after (+) or before (-) the line identified by the address is referenced. A single `-` refers to the line before the current line. The + or - signs at the end of an address have a cumulative effect; the address `++` thus addresses the second line after the current line, `3++` addresses line 5, and `++` and `.+2` are identical.

`%`
The `%` (percent) sign stands for the address pair 1, $, i.e. the entire buffer.
Parameters

The following parameters are used with `ex` commands:

- **line** = *address*
  A single line address, which you can specify in any of the forms described earlier in the Addressing section.

  - *line* not specified:
    The default value (the period .) applies. The period stands for the current line.

- **range** = *address,address*  
  A pair of line addresses that defines an inclusive range of lines. The addresses can be separated by a comma or a semicolon. A range should not be specified together with a number *n*, since the last address of the specified range will then be interpreted as the first address of a range of *n* lines that begins with the last address. In other words, you would end up addressing a range that comes after the one intended. This has been taken into account in the `ex` command syntax.

  - *range* not specified:
    The default value .. applies.
    .. represents only the current line.

- **n**  
  A positive integer. In *n* you specify the number of lines to be processed.

  - *n* not specified:
    The default value is 1.

- **flag**  
  One or a combination of the `ex` commands `#` (see `nu`), `p`, and `l`. These commands display a line in a specific format and are executed after the preceding `ex` command completes.

  .. The blanks need not always be specified. However, in the interest of clarity, they have not been indicated as optional here.

Any number of plus or minus characters may be combined with these parameters.
## Commands

**Overview of ex commands and their abbreviations**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTRL-D</code></td>
<td>display $n$ lines, where $n=$scroll` (see options)</td>
</tr>
<tr>
<td><code>ab</code></td>
<td>abbrev</td>
</tr>
<tr>
<td><code>a</code></td>
<td>append</td>
</tr>
<tr>
<td><code>ar</code></td>
<td>args</td>
</tr>
<tr>
<td><code>c</code></td>
<td>change</td>
</tr>
<tr>
<td><code>co</code></td>
<td>copy</td>
</tr>
<tr>
<td><code>d</code></td>
<td>delete</td>
</tr>
<tr>
<td><code>e</code></td>
<td>edit</td>
</tr>
<tr>
<td><code>f</code></td>
<td>file</td>
</tr>
<tr>
<td><code>g</code></td>
<td>global</td>
</tr>
<tr>
<td><code>i</code></td>
<td>insert</td>
</tr>
<tr>
<td><code>j</code></td>
<td>join</td>
</tr>
<tr>
<td><code>l</code></td>
<td>list</td>
</tr>
<tr>
<td><code>map</code></td>
<td>---</td>
</tr>
<tr>
<td><code>k</code></td>
<td>mark</td>
</tr>
<tr>
<td><code>m</code></td>
<td>move</td>
</tr>
<tr>
<td><code>ma</code></td>
<td>mark</td>
</tr>
<tr>
<td><code>n</code></td>
<td>next</td>
</tr>
<tr>
<td><code>nu</code></td>
<td>number</td>
</tr>
<tr>
<td><code>#</code></td>
<td>number</td>
</tr>
<tr>
<td><code>pre</code></td>
<td>preserve</td>
</tr>
<tr>
<td><code>p</code></td>
<td>print</td>
</tr>
<tr>
<td><code>pu</code></td>
<td>put</td>
</tr>
<tr>
<td><code>q</code></td>
<td>quit</td>
</tr>
<tr>
<td><code>r</code></td>
<td>read</td>
</tr>
<tr>
<td><code>rec</code></td>
<td>recover</td>
</tr>
<tr>
<td><code>rew</code></td>
<td>rewind</td>
</tr>
<tr>
<td><code>se</code></td>
<td>set</td>
</tr>
<tr>
<td><code>sh</code></td>
<td>shell</td>
</tr>
<tr>
<td><code>so</code></td>
<td>source</td>
</tr>
<tr>
<td><code>s</code></td>
<td>substitute</td>
</tr>
<tr>
<td><code>unab</code></td>
<td>unabbrev</td>
</tr>
</tbody>
</table>
No command specified
If you specify only a line or a range, the command p (print) is automatically assumed and executed. If a null line is entered, the next line is printed (equivalent to .+1p)

CTRL-D
(ASCII EOT) prints the next n lines, where n is the value of the ex option scroll.

ab[brev]..word..text
(abbreviate) This command is only effective in vi mode!

If you enter word in the vi input mode as a complete word (i.e. not as a word fragment), it is replaced by the string text. If text is to include special characters, e.g. ñ, these characters must be preceded by [CTRL] V.

[line]..append][!] (append) Causes ex to enter input mode, placing the input text after the specified line.
Use a line value of 0 to put a line at the beginning of the buffer. The last line of new input becomes the new current line. If there is no new input, the target line becomes the current line.
Input is terminated by a period (.) entered in column 1.

args
(arguments) Prints the argument list from the command line of the ex call, enclosed in square brackets [...].
ex, commands

```
line..c[range][!].n
range..c
 (change) ex switches to input mode. n lines or the lines in range are replaced by the lines you enter next.
The last input line becomes the current line. If no text lines are entered, the specified range is deleted, i.e. c has the same effect as the d (delete) command. Input is terminated by a period (.) entered in column 1.

chd[ir][!][directory]
chd[ir][!][directory]
 (chd/cd - change directory) ex changes the current working directory to directory.

directory not specified:
The directory name in the HOME environment variable will become the new working directory.

[range]..co[py].line[.flags]
[range]..t..line[.flag]
 (copy) Copies the lines specified by range to after line. If line is assigned a value of 0, the range of lines (i.e. text) will be copied to the beginning of the buffer.

[line]..d[elete][.buffer][.n]
[range]..d[elete][.buffer]
 (delete) The lines in range, or n lines from the given line, are deleted from the buffer. If a named buffer is specified, the deleted text is saved in it. You can use the ex command pu to retrieve the contents of this buffer if you wish. The first line following the deleted material becomes the current line, unless the deleted lines were at the end of the buffer. In that case, the last line in the buffer becomes the current line.

 e[dit][!][+line][.file]
ex[!][+line][.file]
 (edit) The named file is edited. If the current buffer has been modified since the last w (write) command, ex warns you and aborts the command. You can, however, force the execution of the e command by appending an exclamation mark to the e: e! file. The current line is the last line of the buffer; however, if this command is called from vi, the current line becomes the first line of the buffer.

+line
 The specified line becomes the current line. line can be given as:
- a line number n
- the dollar symbol $ (for end of file)
- a regular expression in the form /RE or ?RE.
```
ex, commands

\texttt{f\[file\]..name]}

(file) Prints the current file name and other information, e.g. the number of lines and the position of the current line.

\texttt{name} specified:
\texttt{name} becomes the current file name

\texttt{range..g\[lobal\]/RE/cmds}

(global) First marks all lines within the given range that match the pattern specified in \texttt{RE}; then applies the given \texttt{cmds} to the each of marked lines in turn, with each marked line becoming the current line while it is being edited.

The commands given in \texttt{cmds} may be specified on separate lines, provided newline characters are escaped with a backslash. If \texttt{cmds} are omitted, all lines will be printed. The \texttt{append}, \texttt{change}, and \texttt{insert} commands are allowed; the terminating period at the end of \texttt{cmds} may be omitted. The \texttt{visual} command is also permitted, and takes input from the display terminal.

The following commands are not permitted in \texttt{cmds}: \texttt{global} and \texttt{undo}.

The following edit options are not allowed in \texttt{cmds}: \texttt{autoprint}, \texttt{autoindent} and \texttt{report}.

Example

\begin{verbatim}
1,$g/^\*/s//+/
\end{verbatim}

All occurrences of * in the first column ^ of line 1 and of all lines through to the end of the file ($) are replaced by a plus character (+) by the substitute command (s).

\texttt{[line]..[insert][!]}

(insert) ex enters input mode; the input text is placed before the specified line. The last line inserted becomes the current line, but if no text is inserted, the line before the target line becomes the current line. Input is terminated by a period (.) entered in column 1.

\texttt{[line]..[join][!]..[n]..[flag]}

(join) Joins the lines in \texttt{range}, or \texttt{n} lines from \texttt{line}, into one line. The command inserts at least one blank between two previously split lines; two blanks are provided if a line ends with a period. If a following line begins with a right parenthesis, no blanks are inserted. Extra white space at the start of a line is discarded.

Appending an exclamation mark to the \texttt{j} command (\texttt{j!}) produces a simpler join with no white space processing.

\texttt{[line]..[list][..n]..[flag]}

(list) Displays the specified lines. Tab characters are replaced by ^I, and the end of each line is marked with a trailing $. The only useful \texttt{flag} is #, when the lines are to be preceded by their line numbers. The last line printed becomes the current line.
map\[.x...commands]\] Format 1
map!\[.x...text]\] Format 2

Only works with vi!

Format 1: Macro definitions for vi command mode

This format of the map command is used to define macros for use in the vi command mode. The first argument \(x\) is a single character, or the sequence \#n\(, \)where \(n\) is a digit that refers to function key \(n\). When this character or function key is typed in vi command mode, vi responds as if the corresponding commands were entered. The specified commands are interpreted as a sequence of vi commands. If special characters, white space, or newline characters are to be used in the commands, they must be escaped with \[CTRL\]V.

Example 1

You want to define a macro which replaces an * occurring at the beginning of a line (column 1) by a blank \(.) throughout the entire file. The name of this macro is to be *. Enter the following in your .exrc file (see section Presetting ex) so that you can call the macro when required:

:\(\text{map ~ *} :1,\$,s/^\="/ \)

Example 2

You now want to define a macro and bind it to function key [F1]. This macro is to search for the next line that begins with a particular regular expression \(RE\) and then automatically delete this line.

To do this, you enter (from vi command mode):

:\(\text{map} \{\text{CTRL}\}V \{\text{F1}\} /\text{"rA}{\text{CTRL}\} V \{\text{J}\} \text{dd} \{\text{J}\} \)

Format 2: Macro definitions for vi input mode

This format of the map command defines macros for use in vi input mode (see Format 1 above). Every \(x\) entered in vi input mode is replaced by text.
(In the ab command, by contrast, \(x\) is replaced only when alone).

If \(x\) is not to be replaced by text, it must be escaped bypreceding it with \[CTRL\]V.

Formats 1 and 2:

Macro definitions can be cancelled with:

unmap \(x\) or unmap! \(x\)

[line]map\[rk]\_x
[line]k\_x

(mark) Marks the specified line with the character \(x\), which must be a single lowercase letter. The \(x\) must be preceded by a blank or tab. The current line position is not affected.
[range].m[ove].line
(move) Moves the specified lines (range) to after the target line. The first of the moved lines becomes the current line.

n[ext][!][...file]...
file not specified:
(next) The next file from the command line argument list is edited (cf. f and ar). If the current buffer has been modified since the last w (write) command, ex warns you and aborts the command. You can, however, force the execution of the n command by entering an exclamation mark after the n: n!.

file specified:
The previous argument list is replaced by the specified file(s), and the first specified file is edited.

[l]ine[...num][b]er[...n][...flag]
[range].num[ber][...flag]
[l]ine[#...n][...flag]
[range].#...flag
(number) Prints the lines, each preceded by its line number. The only useful flag in this case is l. The last line printed becomes the current line.

pre[serve]
(preserve) The current editor buffer is saved as though the system had just crashed. The pre command is for use in emergencies, e.g. when a w (write) does not work because the disk is full, and the buffer contents cannot be saved in any other way.

[l]ine[...pr]int[...n]
[range].print[...]
(print) Prints the specified lines, with non-printing characters printed as control characters in the form \^x; DEL is represented as ^?. The last line printed becomes the current line.

[l]ine[...pu]t[...buffer]
(put) Lines deleted with d (delete) or copied to a buffer with y (yank) are put back in after the specified line. buffer represents a buffer name from a to z.

buffer not specified:
The text in the unnamed buffer is retrieved.

q[uit][!]
(quuit) Quits the editor. If the buffer has been modified since the last w (write) command, a warning is printed and the q command fails. You can, however, force the execution of the q command by appending an exclamation mark to the q: q!. All changes not saved with w will then be discarded.
ex, commands

[line].r[ead][!][..file]
(read) Places a copy of the specified file in the buffer after the target line. Use a value of 0 to place text at the beginning of the buffer. If there is no current file at this point (e.g. when ex is called without a file name), then file becomes the current file. The last line read becomes the current line; in vi mode, the first line read becomes the current line.

file not specified:
The current file is read.

If file is given as \cmd, then \cmd is interpreted as a POSIX command and passed to the shell. The output of the command is then read in to the buffer.

rec[over].file
Recover file from the save area after an editor or system crash.

rew[ind][!]
(rewind) The argument list (see ar) is rewound, and the first file in the list is edited. If the buffer has been modified since the last w (write) command, a warning is printed and the rew command is not executed. You can, however, force the execution of the rew command by appending an exclamation mark to the rew command: rew!. All changes not saved with w are then discarded.

set[!][..parameter]
(set) The set command can be used to display or set options. There are two types of options: those that have "Boolean" (i.e. on or off) values and those that do not. One example of a Boolean parameter is the number option. If this option is set, line numbers are displayed; when nonumber is set, no line numbers are displayed. By contrast, the report option is non-Boolean. The value of this option (default = 5) defines the number of lines that must be changed by a command before ex and vi issue a message.

parameter

- all: The values of all options are displayed.
- option?: The current value of the specified option is displayed.
- option: Only for options with Boolean values: The option is set.
- nooption: Only for options with Boolean values: The option is not set.
- option=value: Only for options with non-Boolean values: The option is assigned the specified value.

parameter not specified:
set displays user-defined ex options, i.e. options that deviate from default settings.

More detailed information on ex options is provided in the corresponding section.
**ex, commands**

**sh[ell]**
If the `SHELL` variable is set, the shell defined in that variable is invoked; otherwise, the Bourne shell `sh`. When you exit the shell, you will be returned to the editing session at the point you left off.

**so[urce]...file**
Reads and executes commands from the specified `file`. Such `so` commands may be nested.

```plaintext
[line]...s[substitute][/RE/repl[options][n]...flags]]
```

**range...s[substitute][/RE/repl[options][n]...flags]]**
(substitute) The first occurrence of a pattern that matches `RE` (regular expression) in each line of the given range is replaced by the string `repl` (see "Regular expressions" on page 328 and "Replacement strings" on page 329). The range is either given in `range` or specified as `n` lines from the specified `line`.

/RE/repl not specified:
The `/RE/repl` from the previous `substitute` command will be used.

**options**

```plaintext
g    (global) All strings that match `RE` in the line are substituted.
c    (confirm) Before each substitution, the line is output with the pattern to be replaced shown in the line below it, marked by a ^ character. Entering y causes the substitution to be made; any other input causes the command to abort. The last line that contains a substitution becomes the current line.
```

**una[bbrev].word**
(unabbreviate) `word` is deleted from the list of abbreviations set up by `ab`.

**u[ndo]**
(undo) Reverses the changes made by the previous editing command. g (global) and vi (visual) are considered single commands in this case. Commands which affect the external environment, such as w (write), e (edit), and n (next), cannot be undone.

An `undo` can itself be reversed with a follow-up `u`.

All markers for lines changed and subsequently restored are lost with `u`.

**un[m]ap[|!]..x**
(unmap) Removes the macro definition created by `map` for `x`.

```plaintext
[range].v[ice]/RE/cmds
```

(vice versa) This is the same as the `global` command, except that `cmds` are applied to all lines that do not match the pattern `RE`.

**ve[rsion]**
Prints the current version of the editor.
ex, commands

[line].vi[sual][.type][.n]
Enters vi mode at the specified line. The type is optional and may be given as a minus sign (-) or a period (.), as in the z command, to specify the position of the specified line on the screen window. The default is to place the line at the top of the screen window. n specifies an initial window size; the default is the value of the ex option window. The command Q exits vi mode and returns you to ex. For further information, see section “vi screen oriented (visual) display editor” on page 803.

[range..]write![!][..file] Format 1
[range..]wq![!][..file] Format 2

Format 1: Write to a file
(write) The specified range is written to the named file, and the number of lines and characters written is printed.

If an alternate file is specified, and the file exists, the command will fail in order to prevent the alternate file from being accidentally overwritten. The execution of w can, however, be forced by appending an exclamation mark to the command: w!.

To append to the file, use the form w>>. If the file does not exist, an error message is issued.

If !cmd is specified instead of file, then cmd is interpreted as a POSIX command. The command interpreter is invoked, and the specified range is passed as standard input to the command.

range not specified:
The entire current file is written to file.

file not specified:
The current file is used by default. The w command cannot be executed if there is no current file and no file is specified.

Format 2: Write to a file and quit ex
(write and quit) The command wq is equivalent to a w followed by a q; wq! is equivalent to w! followed by q.

x[it][!]
(exit) Writes out the buffer if changes have been made since the last w and then (in any case) quits.

line..ya[nk][..buffer][..n]
range..ya[nk][..buffer]
Copies (yanks) the specified range, or n lines from the specified line, to the named buffer.

buffer not specified:
If no buffer is specified, the unnamed buffer is used.
ex, commands

[line].z[.type[.n]]
  (window) Displays n lines from the area in which line is located.

  n not specified:
  n defaults to the value of the window variable.

  type

  type is an optional parameter and can be specified as follows:

  - A - causes the line to be placed at the bottom of the screen.
  + A + causes the line to be placed at the top of the screen.
  . A . causes the line to be placed at the bottom of the screen.
  ^ A ^ writes out n lines starting n*2 lines before the addressed line. The net effect of this will be that a z^ command following another z command writes the previous page.
  = A =centres the addressed line on the screen with a line of hyphens written immediately before and after it. The number of preceding and following lines of text written will be reduced to account for these lines of hyphens.

  The last line printed becomes the current line.

  type not specified:
  line is placed at the top of the displayed area.

  Caution:
  ex prints the number of logical rather than physical lines. More than a screen full of output may result if there are lines which are longer than the screen width.

!...command
  (escape) The remainder of the line after the exclamation mark is passed to the system command interpreter for execution. A warning is issued if the buffer has been changed since the last w (write) command. The specified command is then executed. If the command output has not been redirected, it is displayed on the screen, but does not change the file. The command itself may, however, modify the file, e.g.

  !cat /etc/profile>> %.

  A single ! is printed when the command completes. The current line position is not affected.

  Within the text of command, the percent sign % and the hash character # are expanded as file names (see “Current and alternate file” on page 327).

  An exclamation mark in command is replaced with the text of the previous ! command. Thus, !! means "repeat the preceding command". If any such expansion is done, the expanded line will be echoed.
ex, commands

[range].!..command
  (escape) In this form of the ! command, the lines specified in range are passed to the
cmd as standard input and replaced by the output of the cmd.
  range not specified:
  range is not replaced by the current line (..); the other form of the ! command applies.

" Command lines that start with double quotes " are ignored. This means that you can
use lines of this kind to insert comment lines in a command script.

line..&options..n..flags
range&options..flags
line..s[substitute]options..n..flags
ranges[substitute]options..flags
  (resubst) Repeats the previous s (substitute) command, as if & were replaced by the
previous s/RE/repl/. The same effect is obtained by omitting the /RE/repl/ string in an
s command.

line..<..n
range..<
  (lshift) Shifts the defined line range (or the n lines that follow line) to the left by the
number of spaces specified by the shiftwidth option. White space (blanks and tabs) is
lost in the shift process; other characters are not affected. The last line changed
becomes the current line.

line..->..n
range..->
  (rshift) Shifts the defined line range (or the n lines that follow line) to the right by the
number of spaces specified by the shiftwidth option. White space (blanks and tabs) is
not lost, but inserted as required.

line=
  (line number) Prints the line number of the specified line. The current line position is not
affected.

line not specified:
The line number of the last line is displayed.
**ex, options**

**ex options**

`ex` provides a number of options with which you can govern the behavior of both the `ex` and (more importantly) the `vi` editors (see “Customizing the `vi` environment” on page 839).

All options have default settings; these settings are valid at the time `ex` or `vi` is invoked. Use the `ex` command `se` (set) if you wish to have all options displayed:

```
set all
```

The above command can also be used to change one or more options:

```
set showmode scroll=15
```

This command shows the `vi` input mode in the status line and sets the number of lines scrolled by the `CTRL`D key to 15.

The command:

```
set
```

displays all options set to non-default values.

There are two types of options: those that have “Boolean” (i.e. on or off) values and those that do not. One example of a Boolean parameter is the `showmode` option. When this option is not set, it is named `noshowmode`. The `scroll` option, by contrast, is an example of a non-Boolean parameter.

**autoindent, ai**

**noautoindent, noai** *(default)*

If `autoindent` is set, each line in insert mode is aligned with the beginning of the previous line; tabs and blanks are inserted to produce this alignment. The starting indentation of a line is determined by the line it is appended after (`a`), the line it is inserted before (`i`), or the first line to be changed (`c`). Extra indentation can be provided as usual; succeeding lines will automatically be indented to the new alignment.

To reduce the level of indentation, you can press `CTRL`D one or more times to move the start of the line to positions which are multiples of `shiftwidth`. Thus if the current cursor position is 25 and `shiftwidth` is set to 10, pressing `CTRL`D once will move the cursor to the left to position 20, while pressing it twice will take the cursor to position 10.

A caret ^ followed by a `CTRL` D removes all indentation temporarily for the current line; a `0` (zero) followed by a `CTRL` D removes all indentation.

**autoprint, ap**

**noautoprint, noap** *(default)*

Prints the current line after each command that changes buffer text. `autoprint` is suppressed during global search and replace operations (see `g`, `s` and `v`).
ex, options

**autowrite, aw**
**noautowrite, noaw** (default)
Writes the buffer to the current file if the buffer has been modified, and an \( e \) (edit), \( n \) (next), \( rew \) (rewind) or \( ! \) (escape) command has been given.

**beautify, bf**
**nobeautify, nobf** (default)
Causes all control characters other than tab, newline and form feed to be discarded from the input text.

**directory=dir**
**dir=dir**
The value of \( dir \) specifies the directory which is to hold the editor buffer. If the user does not have write permission for this directory, the editor quits.

**edcompatible, ed**
**noedcompatible, noed**
Causes the presence of \( g \) and \( c \) suffixes on \( s \) (substitute) commands to be remembered for use as toggles.

**errorbells, eb**
**noerrorbells, noeb**
Rings the terminal bell before displaying error messages in the last line (usually in reverse video).

**exrc, ex**
This option has been deactivated for security reasons. It normally causes .exrc files in the current directory to be evaluated.

**flash, fl** (default)
**noflash, nofl**
If the terminfo description contains an entry for flashing the screen to indicate an error, this option can be used to inform users when they have made an invalid entry.

**hardtabs=number**
**ht=number**
If the terminal has hardware tabs, this option can be used to set the number of spaces to which the tab stops are set (to optimize screen output).

**ignorecase, ic**
**noignorecase, noic** (default)
Treats all uppercase characters in the text as lowercase when a search is made. Uppercase characters in regular expressions are also mapped to lowercase, except in character class expressions.
lisp

nolisp (default)

Activates the *autoindent* option and modifies the *vi* commands left parenthesis (, right parenthesis ), left brace {, right brace }, double left brackets [[, and double right brackets ]] suitably.

list

nolist (default)

All printed lines will be displayed with tabs shown as ^I, and the end of line marked by a $.

magic (default)

nomagic

Changes interpretation of characters in regular expressions and substitution replacement strings (see the relevant sections).

mesg (default)

nomesg

Grants/denies other users permission to write to the terminal (e.g. with the *write* command).

modelines (default)

nomodelines

The first and last five lines of a file that is read in are interpreted as editor commands and executed if they are in the form:

\texttt{ex:command:}

novice

nonovice (default)

Defines the level of detail for error messages.

number, nu

nonumber, nonu (default)

Causes lines to be printed with line numbers.

optimize, opt

nooptimize, noopt (default)

Allows you to set the terminal with or without automatic carriage returns (to optimize output).

paragraphs=string

para=string

The value of this option is a string in which each successive pair of characters specifies the name of an *noff* macro that begins a paragraph. A macro appears in the text in the form .XX, where the . is the first character in the line.
prompt (default)
noprompt
When prompt is set, the : prompt is displayed in command mode; otherwise, there is no command-mode prompt.

readonly
noreadonly (default)
If readonly is set, the editor can only be used to read files, not to make changes to the text. If noreadonly is set, you can make changes to files.

redraw (default)
noredraw
The editor simulates an intelligent terminal on a dumb terminal. Since this is likely to require a large amount of output to the terminal, it is useful only at high transmission speeds.

remap (default)
noremap
If remap is set, macro translation allows for macros defined in terms of other macros; translation continues until the final product is obtained. Only one step of translation is performed if noremap is set.

report=n
When the number of lines changed, deleted, or copied by the last command exceeds n, a message is issued in the status line.

scroll=n
The value n specified for scroll determines the number of lines that the screen will scroll when a \texttt{CTRL}D command is entered. It also controls the number of lines displayed by a z command (twice the value of scroll).

sections=string
sect=string
The value assigned to this option is a string in which each successive pair of characters specifies the name of an \texttt{nroff} macro which begins a section. A macro appears in the text in the form .XX, where the . is the first character in the line.

shiftwidth=n
sw=n
The value n specified for sw is used for the spacing of tab stops used by the tab key, the autoindent option, and by the < and > (shift) commands.

showmatch, sm
noshowmatch, nosm (default)
When a right parenthesis ) or right brace } is entered in vi, the matching left parenthesis ( or left brace { is shown, provided the matching character is still on the screen.
showmode, smd
noshowmode, nosmd (default)
    When smd is set, an indication is provided in the status line as to whether vi is in vi input mode.

slowopen, slow
noslwopen, noslow (default)
    In vi, prevents screen updates during input to improve throughput on unintelligent terminals.

tabstop=n
ts=n
    n specifies the software tab spacing to be used by the editor when expanding tabs in the input file.

taglength=number
tl=number
    Defines how many characters are to be considered significant in searches with the :tag command. A value of 0 means that all characters are to be considered.

tags=files
    Names of tag files, separated by blanks.

term=string
t=tell vi what type of terminal is being used. (The default is the value of the environment variable TERM.)
terse
noterse (default)
    Setting terse causes shorter error messages to be issued.

timeout
notimeout
    Sets/cancels a timeout period.

ttytype=string
ty=string tells vi what type of terminal is being used. (The default is the value of the environment variable TERM.)

warn (default)
nowarn
    When warn is set, the warning No write since last change is issued if a :! command is given before the most recent changes to the editor buffer have been saved with w.

window=n
    The number of lines in a vi text window.
wrapmargin=n
wm=n

Only effective in vi. If n>0, a newline is automatically added to an input line at a word boundary such that lines end at least n spaces from the right margin of the terminal screen.

The setting of automatic line breaks can be deactivated by setting the value of n to 0 (default).

wrapscan, ws (default)
nowrapscan, nows

Setting wrapscan causes searches begun by the /.../ or ?...? commands to wrap around from the end of a buffer to the beginning (or vice-versa). When nows is set, the search runs through to the end or beginning of the buffer as appropriate and then terminates.

writeany, wa
nowriteany, nowa (default)

When nowa is set, a check is performed before a w command to determine whether the file exists. This is to prevent an existing file from being accidentally overwritten. You can override the overwrite protection with w!. Setting wa inhibits the normally provided overwrite protection.

Error

When an error occurs, ex sends the acoustic signal BEL (see section “ASCII character set (ISO 646)” on page 889) to the terminal and prints a message. If an interrupt signal is received, ex returns to command level, allowing you to enter a new ex command. If the editor input is from a file, ex exits at the interrupt signal.

If ex detects an internal error, it attempts to preserve the buffer if the most recent changes have not been saved. If you wish to access the backed up data, you must call ex with the -r option.

File

$HOME/.exrc

File containing default values for ex and vi. These defaults are overridden if conflicting settings have been assigned to EXINIT.

Variable

TERM
The type of data terminal used must be defined in the environment variable TERM.

EXINIT
Environment variable with default values for ex and vi. These defaults always apply.

PATH
Determine the search path for the shell command specified in the editor commands shell, read and write.
HOME
Determine a pathname of a directory that will be searched for an editor startup file named \texttt{.exrc}.

SHELL
Determine the preferred command-line interpreter for use in \texttt{!}, \texttt{shell}, \texttt{read} and other commands with an operand of the form \texttt{String}. For the shell command, the program will be invoked with the single argument \texttt{-i}, for all others it will be invoked with the two arguments \texttt{-c} and \texttt{string}. If no \texttt{SHELL} variable is set, or it is set to a null string, the \texttt{sh} utility will be used.

COLUMNS
Override the system-selected horizontal screen size.

LINES
Override the system-selected vertical screen size, used as the number of lines in a screenful and the vertical size in visual mode.

Locale
The following environment variables affect the execution of \texttt{ex}:

\texttt{LANG}  
Provide a default value for the internationalization variables that are unset or null. If \texttt{LANG} is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

\texttt{LC_ALL}  
If set to a non-empty string value, override the values of all the other internationalization variables.

\texttt{LC_COLLATE}  
Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.

\texttt{LC_CTYPE}  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the classification as upper- or lower-case letters, the case conversion of letters, and the detection of word boundaries.

\texttt{LC_MESSAGES}  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

\texttt{NLSPATH}  
Determine the location of message catalogs for the processing of \texttt{LC_MESSAGES}. 
Example  The following example demonstrates and explains a number of ex commands:

```
$ ex figures  - Call ex (file name "figures")
'figures" 4 lines, 19 characters
:set number  - Output line numbers
:1,$p  - Output line 1 through to last line
    1 one  - Original contents of figures
    2 two
    3 three
    4 four
:1 c 1  - Modify line 1
    1 ONE  - New line 1
    2 .  - Quit input mode
:2,3 co 4  - Copy lines 2 to 3 after line 4
    4 three
:1,$ g/two/s//TWO/  - Replace "two" by "TWO" throughout the file
:2 a  - Insert new line after line 2
    3 THREE  - New line 3
    4 .  - Quit input mode
:4 i  - Insert new line before line 4
    4 FOUR  - New line 4
    5 .  - Quit input mode
:5,$ d a  - Delete line 5 and all lines to end of file
    4 FOUR
:1,$p  - Output line 1 through to last line
    1 ONE
    2 TWO
    3 THREE
    4 FOUR
:wq  - Save the contents of the buffer and quit ex
'figures" 4 lines, 19 characters
```

See also  ed, vi
exec **execute commands and open, close or copy file descriptors**

The POSIX shell built-in *exec* performs two functions:

- It overlays the current shell with another program (Format 1). When this program starts, the current shell is terminated. No new process is created, as is evident from the fact that the process ID does not change (see Example 2 on page 356).

  If you enter *exec* interactively, when the specified program exits you return to the parent shell of the previous shell; if your previous shell was a login shell, your session terminates.

  If *exec* is called from a shell script, the script terminates. Commands that follow an *exec* call in the script will never be executed.

- It can be used to redirect the standard input or the standard output of the shell to a file (Format 2). All commands entered after *exec* has been executed will read from or write to this file until you terminate the current shell.

Syntax

<table>
<thead>
<tr>
<th>Format 1: exec..program[...redirection]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2: exec..redirection</td>
</tr>
</tbody>
</table>

Format 1 **Replacing the shell with another program**

```
exec..program[...redirection]
```

**program**

Any command, program or shell script, but not another *sh* command. You will need execute permission for the associated file.

The current shell terminates, and *program* is executed instead. On completion of the specified *program*, control reverts to the old shell’s parent, or the welcome screen is displayed.

**redirection**

If the specified program reads from standard input or writes to standard output, a file can be assigned for the input or output instead.

`redirection` can be specified as follows:

```
>file  The standard output of the specified program is redirected to *file*. Any data previously in *file* will be deleted.

>>file The standard output of the program specified is redirected to *file*. Data already contained in *file* is not deleted; program output is appended to it instead.
```


exec

2>file The standard error output of the specified program is redirected to file.

<file The standard input of the specified program is redirected to file, i.e. the program reads its input from this file.

Format 2 Redirecting the shell's standard input/standard output

exec redirection

Commands that read from standard input or write to standard output are assigned a file for their input/output. The redirection applies to all commands that the current shell executes after exec.

redirection can be specified as follows:

>file All commands executed by the current shell write their output to the specified file sequentially. Data previously in file is deleted. The output from all commands can be collected in this way.

If you have entered exec interactively, the redirection can only be cancelled by:

– pressing the END or @ @d. This terminates the current shell.

– entering exec >/dev/tty. This redirects the standard output back to the screen.

If exec is called from a shell script, the redirection will only apply to commands that follow the exec call in the script. Redirection can be cancelled within the script by including the command exec >/dev/tty at the appropriate position in the script. This will redirect the standard output of all subsequent commands back to the screen.

>>file All commands executed by the current shell write their output to the specified file sequentially. Data already contained in file is not deleted; program output is appended to it instead.

2>file Standard error is redirected to file for all commands executed by the current shell.
Built-in sh command

exec

<file

If you specify exec interactively, the current shell will read the commands to be executed from the specified file. The shell exits after the last command.

If this command appears in a shell script, all commands that follow the exec call in the script read their input from the specified file. Each read operation modifies the position of the read pointer in this file. If the read pointer is set to the end of the file (EOF), all following commands will receive no input.

The command exec 〈/dev/tty can be used to redirect the standard input back to the keyboard for all subsequent commands.

file

If you have entered exec interactively, file must be the name of a shell script. You will only need read permission for this shell script.

When exec is called from a shell script, file designates the name of the file from which all subsequent commands are to obtain their input.

Locale

The following environment variables affect the execution of exec:

LANG

Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL

If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH

Determine the location of message catalogs for the processing of LC_MESSAGES.
Example 1  What happens when you make the following input?

$ exec date

First, the `exec` command terminates the shell and then causes the `date` command to be executed. The current date is displayed on the screen.

If you have entered the command in your POSIX shell the BS2000 prompt is output. If you want to recommence work with the POSIX shell you must re-enter the `START-POSIX-SHELL` command.

If you entered the command from a subshell, control will be returned to the parent shell.

Example 2  As indicated earlier, the process ID does not change when the current shell is replaced by another program. This concept is demonstrated with reference to the following files:

- **Contents of the file `proc1`:**
  
  ```bash
  echo The process ID of proc1 is: $$
  sh proc2
  ```

- **Contents of the file `proc2`:**
  
  ```bash
  echo The process ID of proc2 is: $$
  exec proc3
  ```

- **The file `proc3`, which must be executable, contains the following:**
  
  ```bash
  echo The process ID of proc3 is: $$
  ```

The shell script `proc1` is now initiated:

$ sh proc1
The process ID of proc1 is: 2755
The process ID of proc2 is: 2760
The process ID of proc3 is: 2760

Since `proc3` is called from `proc2` with `exec`, both shell scripts run under the same process ID. To be exact, the shell that executes `proc2` is replaced by the shell that executes `proc3`. 
Example 3 The `exec` command is used in the `exctest` shell script to redirect the standard input to the file `/etc/group` for all following commands. The contents of the `exctest` file is given below:

```bash
: Invocation with sh exctest
exec \</etc/group
read lines
echo $lines
echo
cat
```

The shell script `exctest` is now initiated:

```bash
$ sh exctest
root::0:root
daemon::1:daemon
sys::2:sys:
bin::3:bin,admin
uucp::4:
.
```

The shell built-in `read` assigns the first line of the `/etc/group` file to the variable `line`. The `echo` `$line` command outputs the content of this variable.

The `cat` command also reads its input from the `/etc/group` file. Since the read pointer is now set to the second line of this file, `cat` displays the file's contents from the second line onward.

The read pointer is now set to EOF. Any further commands after the `cat` call would consequently receive the null string as input.

See also `exec()` [4]
**exit**

**cause the shell to exit**

The POSIX shell built-in `exit` is used to terminate shell scripts. If you wish, you can define an exit status for the terminated script by following `exit` with a number.

Without `exit`, shell scripts terminate:
- when the last command has been executed, i.e. when the executing shell detects an end-of-file (EOF). The exit status is then the status of the last command executed.
- when the executing shell cannot find a command or detects a syntax error; the exit status is then non-zero.

If you call `exit` interactively, either the shell (or subshell) in which you are currently working or the POSIX shell terminates.

**Syntax**

```
exit[n]
```

- **n** Any number between 0 and 256. This number defines the Exit status with which the shell script terminates.
  - If you specify a larger number, the exit status will not be set to the specified number but to the integer remainder of dividing the specified number by 256.
  - As described in other sections, certain exit status values are reserved for specific purposes. Applications may only use these values for these purposes:
    - 126: An executable command was found but the file is not an executable file.
    - 127: No executable file was found.
    - >128: A command was interrupted by a signal.

  `trap` is executed for EXIT before the shell terminates. This fails to occur only if `exit` itself is called in this `trap`. In this latter case, the shell terminates immediately.

  You can use the command `echo $?` to query the exit status.

  - **n not specified:** The exit value of the command executed before the `exit` call is used.

**Exit status**

The exit status is `n` if it is set. Otherwise the exit status is the exit status of the last command executed or zero if no command has been executed. If `exit` is executed as part of a `trap` then the last command executed is taken to be the command which was executed immediately before the `trap`. 

Built-in sh command

Example 1  The `false` command can be implemented by means of the following shell script:

```bash
: Shell version of the false command
: Exit status always 1
exit 1
```

Example 2  The two shell scripts named `end` and `check` will be used to demonstrate how the exit status can be evaluated:

- The `end` script contains the following:
  ```bash
  : Invocation with sh end file
  if [ -f "$1" ]
  then exit 2
  elif [ -d "$1" ]
  then exit 3
  else exit 4
  fi
  ```

- The `check` file is scripted as follows:
  ```bash
  : Invocation with sh check file
  sh end "$1"
  case $? in
  2) echo Contents of file $1:; cat $1;;
  3) echo Contents of $1:; ls -l $1;pg;;
  4) echo Entering $1 is meaningless for this script!
  esac
  ```

The shell script `check` is now initiated:

```bash
$ sh check .profile
Contents of file .profile:
HOME=/usr1/rose/src
export HOME
...
```

The shell script `check` calls the script named `end`. This script returns an exit status of 2, since `.profile` is a normal file. The commands specified in the case statement under the matching pattern 2 are then executed.

The contents of the above two shell scripts cannot be simply combined into a single file, since the `exit` command terminates the script.
This shell script *endaction* shows how the exit status can be interpreted within a script:

```sh
: Invocation with sh endaction file
(if [ -f "$1" ]
    then exit 2
  elif [ -d "$1" ]
    then exit 3
    else exit 4
fi)
```

```sh
case $? in
    2) echo Contents the file $1:; cat $1:;
       3) echo Content of $1:; ls -l $1•pg::
       4) echo Entering $1 is meaningless for this script!
      esac
```

The parentheses around the if statement cause a subshell to be invoked. This subshell terminates with the exit status that is specified in the relevant *exit* command. Control is subsequently returned to the parent shell, i.e. the shell that processes the *endaction* script. This shell executes the remaining commands.

See also  
*false*  
*exit()* [4]
**expand**

`expand` convert tabs to spaces

The `expand` command writes files or the standard input to the standard output with tab characters replaced by one or more blanks needed to pad the line to the next tab stop.

All backspace characters are copied to the output and cause the column position count for tab stop calculations to be decremented. The column position count will not be decremented below zero.

**Syntax**

```plaintext
Format 1: expand[-t...tablist][...file...]
Format 2: expand[-tabstop l...-tab1,tab2,...,tabn][...file...]
```

**Format 1**

`expand[-t...tablist][...file...]`

- `-t...tablist`
  Specifies the tab stops. The argument `tablist` must consist of one or more numbers, separated by blanks or commas, in ascending order. A list separated by blanks must be enclosed in quotes.

  If only one number is specified, the tab stops will be set to `tablist` column positions instead of the default 8 column positions. If multiple numbers are given, the tabs will be set at the specified column positions.

  Each tab stop position N must be an integer value greater than zero, and the specifications must be in ascending order. This means that tabbing from the start of the line of output to position N causes the next character output to be in the (N +1)th column position in that line.

  If the `expand` command has to process a tab character at a position beyond the last position specified in a multiple tab stop list, the tab character is replaced by a blank in the output.

- `...file`
  The `file` whose tab characters are to be replaced by blanks.
Format 2  **expand**[-tabstop -tab1,tab2,...,tabn][..file...]

- tabstop -tab1,tab2,...,tabn
  Specifies the tab stops. A single number is specified as **tabstop** with a leading minus; multiple tabstops are specified after a leading minus as **tab1**, **tab2**, ..., **tabn** and so forth.

file  The file whose tab characters are to be replaced by blanks.

Standard Output (stdout)

The standard output is equivalent to the input files with tab characters converted into the appropriate number of blanks.

Locale  The following environment variables affect the execution of **expand**:

**LANG**  Specifies a default value for the internationalization variables that are unset or null. If **LANG** is unset or null, the corresponding default value from the internationalized environment is used. If one of the internationalization variables contains an invalid setting, the command behaves as if none of the variables have been defined.

**LC_ALL**  If this variable has been assigned a value, i.e. it is not a null string, this value overrides the values of all the other internationalization variables.

**LC_CTYPE**  Determines the internationalized environment for the interpretation of byte sequences (e.g. single-byte characters as opposed to multi-byte characters in arguments and input files). The internationalized environment for the processing of tab characters and blanks and for the specification of the width in column positions each character would occupy on a constant-width output device.

**LC_MESSAGES**  Determines the format and contents of error messages.

**NLSPATH**  Determines the position of message catalogs for the processing of **LC_MESSAGES**.

See also  **tabs**, **unexpand**
**export**  set export attribute for variables

The POSIX shell built-in *export* marks the specified shell variable for export. This means that the name of this variable and its value will subsequently be known and accessible to all commands.

Exported variables are deleted when the shell in which they were defined and exported is terminated. Frequently used variables should therefore be defined and exported in the $HOME/profile file.

Positional parameters and shell functions cannot be exported. Some standard variables of the shell are available in all subshells without needing to be exported. These include `HOME`, `IFS`, `PATH`, `PS1` and `PS2`. If you wish to assign some other value to these variables, and if the modified value is to be valid in every subshell, you will need to *export* these variables.

Otherwise, the default value is valid in each subshell.

The built-in *sh* command *set* outputs all variables and associated values which are defined in the current shell. This therefore includes variables which you have not exported. The command *export* outputs the names and values of all shell variables which are passed to each called command and each subshell.

**Syntax**

Format 1: `export[...name[=value]]...`

Format 2: `export...-p`

**Format 1**

`export[...name[=value]]...`

`name[=value]`

Name of the shell variable which you wish to export. You may also assign a value `value` to this variable after calling `export`. You may specify as many shell variables as you wish, each separated by a space.

*name* not specified:

`export` writes the names of all exported shell variables in the current shell to the standard output. The output takes the following form:

`name=value`

...
### Format 2: `export -p`

- `-p` If `-p` is set, `export` outputs the names and values of all the variables which have been exported. The output has the following format and is sent to the standard output:

  ```text
  "export %s=%s
", name, value
  ```

  The option `-p` provides transferrable access to the values which can be saved and then subsequently restored (e.g. by means of a dot procedure).

  The shell formats the output and ensures that the quotation marks are used correctly. This means that output is suitable for re-entry in the shell in the case of commands which have the same export results.

### Locale

The following environment variables affect the execution of `export`:

- **`LANG`**: Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **`LC_ALL`**: If set to a non-empty string value, override the values of all the other internationalization variables.

- **`LC_CTYPE`**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **`LC_MESSAGES`**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **`NLSPATH`**: Determine the location of message catalogs for the processing of `LC_MESSAGES`. 
Example  The shell script *ps1* is to display the value of the *PS1* variable. The script file contains the following:

```bash
$ echo $PS1
```

The following interactive session demonstrates why the shell variable *PS1* has to be exported:

```bash
$ export HOME
$ export PATH
$ export PS1
$ sh ps1
PS1: $PS1
```

When you define a shell variable, it is only known in the current shell. Since a shell script is always run by a subshell, variables for shell scripts have to be exported.

See also  *env, set*
`expr` evaluate arguments as an expression

`expr` interprets command-line arguments as expressions and evaluates them in succession. The result of the evaluation is displayed on standard output. You can use `expr` to compare strings with one another, for example, or to perform calculations with integers.

**Syntax**

```
expr..expression....
```

`expression` may either consist of one operand only, or two operands linked by an operator. Any arbitrary string can be specified as an operand. Strings that consist of digits only (0 to 9) are interpreted as integers. Integers preceded by a minus sign are interpreted as negative numbers.

Operands and operators are separated from one another by a separator. The value specified for the `IFS` variable is used as the separator. If you wish to enter a string which contains spaces or tabs as an operand you must therefore enclose the entire string in apostrophes `"..."` or quotes `"..."` to prevent these characters being interpreted as separators. You must also enclose special shell characters in apostrophes `"..."` or quotes `"..."` or quote them by means of the backslash `\` (see `IFS` in the section “POSIX shell variables and parameter substitution” on page 52).

If `expression` consists of only one operand, the result of the evaluation will be the operand itself.

The following section describes how two operands can be linked. The operators are arranged in the order of increasing precedence; operators with equal precedence are enclosed in braces `{...}`.

A result of 0 stands for the value 0, not a null string.

**Linking operators**

```
op1...1..op2
```

If `op1` is neither the null string nor 0, the expression evaluates to `op1`; otherwise, `op2`.

```
op1...&..op2
```

If neither `op1` nor `op2` is equal to the null string or 0, the result returned is `op1`; otherwise, 0.
Built-in sh command

\texttt{expr}

\begin{itemize}
  \item \texttt{op1} \texttt{rel} \texttt{op2}
    \begin{itemize}
      \item \texttt{rel} may be one of the following relational operators:
        \begin{itemize}
          \item \texttt{<} \quad \text{less than}
          \item \texttt{<=} \quad \text{less than or equal to}
          \item \texttt{=} \quad \text{equal to}
          \item \texttt{>=} \quad \text{greater than or equal to}
          \item \texttt{>} \quad \text{greater than}
          \item \texttt{!=} \quad \text{not equal to}
        \end{itemize}
    \end{itemize}
  \end{itemize}

  If the condition is fulfilled, the comparison returns a result of 1. If the condition is not satisfied, the result of the comparison is 0. If both \texttt{op1} and \texttt{op2} are integers, the comparison is numeric; otherwise, they are interpreted as strings and compared alphanumerically.

  \begin{itemize}
    \item \texttt{op1} \texttt{[+ -]} \texttt{op2}
      \begin{itemize}
        \item If \texttt{op1} and \texttt{op2} are both integers, the result returned is the sum of, or difference between, the two numbers. If either of the arguments is not an integer, \texttt{expr} issues an error message (see page 368).
      \end{itemize}
    \end{itemize}

  \begin{itemize}
    \item \texttt{op1} \texttt{[*/%]} \texttt{op2}
      \begin{itemize}
        \item When \texttt{op1} and \texttt{op2} are integers, the result is equal to the value obtained from the specified arithmetic operation:
          \begin{itemize}
            \item \texttt{*} \quad \text{Multiplication}
            \item \texttt{/} \quad \text{Division (integers only)}
            \item \texttt{%} \quad \text{Remaindering}
          \end{itemize}
      \end{itemize}
  \end{itemize}

  \begin{itemize}
    \item \texttt{op1} \texttt{:} \texttt{op2}
      \begin{itemize}
        \item The strings \texttt{op1} and \texttt{op2} are compared with one another, starting with the first character in each string and ending with the last character in \texttt{op2}. \texttt{op2} may be specified in the form of a simple regular expression (see section “Regular POSIX shell expressions” on page 877). If \texttt{op1} and \texttt{op2} match each other (i.e. from the first character in both strings to the last character in \texttt{op2}), \texttt{expr} usually returns the number of matching characters. However, if you enter the pattern \texttt{\textbackslash{...}} for \texttt{op2}, the part of \texttt{op1} that matches this pattern will be displayed (see Example 6 on page 369 and Example 7 on page 369).
        \end{itemize}
  \end{itemize}

  \begin{itemize}
    \item \texttt{(...)}
      \begin{itemize}
        \item The use of parentheses \texttt{(...)} enables you to influence the sequence of evaluation. Parenthesized expressions are evaluated first even if they contain operators with lower precedence.
      \end{itemize}
  \end{itemize}
**expr**

**Built-in sh command**

**Exit status**
- 0 if the expression is neither invalid nor null nor 0
- 1 if the expression is null or 0
- 2 for invalid expressions or division by 0

**Error**

- **expr: Non-numeric argument**
  You are only allowed to specify integers as operands when using the arithmetic operators +, -, *, /, %.

- **expr: Division by zero**
  You have tried to divide by 0.

- **expr: Syntax error**
  You have made a syntax error when calling `expr`, e.g. you have not delimited operands and operators by blanks or tab.

- **expr: RE error**
  When using the relational operator : in an expression, you have not specified the second operand as a simple regular expression in the correct format.

**Locale**
The following environment variables affect the execution of `expr`:

- **LANG**
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_COLLATE**
  Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions. `LC_COLLATE` also governs the behavior of relational operators in string comparisons. Thus if the letters a and ä form part of an equivalence class, the expression `expr $var:^[[:a=]]` returns a value of 1 if the value of the variable is a or ä.

- **LC_CTYPE**
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  Determine the location of message catalogs for the processing of `LC_MESSAGES`. 
Built-in sh command

Example 1  Simple calculation:

$ expr 21 + 9 '* 2 / 6
24

Example 2  A value of 1 is added to variable a in the following example.

$ echo $a
3
$a=`expr $a + 1`
$ echo $a
4

Example 3  This example compares two environment variables:

$ echo $op1 $op2
text1 text2
$ expr $op1 = $op2
0
If the value of a variable is itself an operator used by expr, the comparison will not work. This
problem can be solved by combining the variable with a "safe" character:

$ echo $op1 $op2
= =
$ expr $op1 = $op2
expr: syntax error
$ expr X$op1 = X$op2
1

Example 4  Displaying the number of characters in VAR:

$ echo $VAR
Hallo
$ expr $VAR : '.*'
5

Example 5  Comparing two strings:

$ expr boycott : boy
4

Example 6  The absolute path name of a file, e.g. /usr/latino/parnassum/infinitum is stored in variable a.
To obtain the basic file name, i.e. infinitum, you enter:

$ expr abc : [\^d-f]
1

Example 7  The absolute path name of a file, e.g. /usr/latino/parnassum/infinitum is stored in variable a.
To obtain the basic file name, i.e. infinitum, you enter:

$ expr $a : '.*/(.*\')'
infini

U22794-JZ125-6-76
expr

Example 8 If variable \( a \) contains either the absolute path name or the basename of a file, you can extract the file basename as follows:

\[
expr \ a : '.*\/(.*\)' | \ a
\]
false  return false value

The built-in POSIX shell `sh` command `false` returns an exit status which is not equal to zero. This enables you to generate the condition `false` in shell procedures. Similarly, you can use the command `true` to generate the condition `true` (exit status equal to 0).

**Syntax**

```
false
```

**Exit status**

Always non-zero

**Example**

The exit status of `false`:

```
$ false
$ echo $?
1
```

**See also**

`true`
The commands issued from an interactive POSIX shell are stored in a *history* file which can be accessed using the built-in *fc* command. *fc* allows you to:
- edit individual command lines from the *history* file before they are executed (Format 1)
- list command lines on standard output (Format 2)
- change commands before they re-execute, without editing the *history* file (Format 3)

**Syntax**

**Format 1:**
```
fcc[-r][-e...editor][...first][...last]
```

- `-r` The commands are made available in reverse order.
- `-e...editor`  
  *editor* is the name of the editor to be used. If the editor name is not supplied, the value of the variable *FCEDIT* is used. If *FCEDIT* is not set, the default editor */usr/bin/ed* is invoked.

- `...first`  
  The range of commands from *first* to *last* in the last *HISTSIZE* commands typed at the terminal is selected for editing and subsequent execution. The *first* and *last* arguments may be specified as numbers or as strings.

- `[+]-number` A positive number representing a command number; command numbers can be displayed with the `-i` option.
- `-number` A negative decimal number representing the command that was executed by this many commands previously. For example, `-1` is the immediately preceding command.
- `string` A string is used to select the most recently entered command that begins with this string.

*first* and *last* specified  
All commands entered between *first* and *last* are selected for editing. If *first* is a more recent command than *last*, the commands are listed for editing in reverse order (as with option `-r`).

*last* not specified  
The value for *first* is used.

*first* and *last* not specified  
The last (most recent) command is edited.
Format 2  \texttt{fc..-l[\ldots-nr]\ldots[\ldots[\ldots[last]]]}

- \texttt{l}  The commands and command numbers are merely written to standard output and can neither be edited nor executed.
- \texttt{n}  Command numbers are suppressed when listed with -\texttt{l}.
- \texttt{r}  Commands are listed in reverse order.

\texttt{first[\ldots last]}

The range of commands from \texttt{first} to \texttt{last} in the last \texttt{HISTSIZE} commands typed at the terminal is selected for editing and subsequent execution. The \texttt{first} and \texttt{last} arguments may be specified as numbers or as strings.

\texttt{[+\ldots number}}  A positive number representing a command number; command numbers can be displayed with the -\texttt{l} option.

\texttt{{\ldots -number}}  A negative decimal number representing the command that was executed by this many commands previously. For example, -1 is the immediately preceding command.

\texttt{string}  A string is used to select the most recently entered command that begins with this string.

\texttt{first} and \texttt{last} specified

All commands entered between \texttt{first} and \texttt{last} are listed. If \texttt{first} is a more recent command than \texttt{last}, the commands are listed for editing in reverse order (as with option -\texttt{r}).

\texttt{last} not specified

The previous command is listed.

\texttt{first} and \texttt{last} not specified

The previous 16 commands are listed.

Format 3  \texttt{fc..-s[\ldots old=\ldots new]\ldots[\ldots first]}

- \texttt{s}  The commands are re-executed. No editor is invoked.

\texttt{old=new}

The command is modified before being executed. The command is re-executed after \texttt{new} has been substituted for \texttt{old}.

\texttt{first}

All commands since \texttt{first} are executed. See Format 2 for a detailed description.

\texttt{first} not specified

The previous command is used.
Error

sh: /bin/ed: not found
The shell variable `FCEDIT` is not set to `ed`.

Variable

`HISTFILE`
If this variable is set when the POSIX shell is invoked, its value is the path name of the file that will be used to store the command history (see the section “Command re-entry” on page 65).

`HISTSIZE`
If this variable is set when the POSIX shell is invoked, the shell will remember the commands you enter (command history). The number of previously entered commands that are accessible by this shell will be greater than or equal to the given number. The default is 128.

`FCEDIT`
The name of the standard editor for the built-in `fc` command.

Locale

The following environment variables affect the execution of `fc`:

`LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

`LC_ALL` If set to a non-empty string value, override the values of all the other internationalization variables.

`LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

`LC_MESSAGES` Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

`NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example 1

Listing the previous 16 commands:

```
$ ls -l
$ fc -s ls=fc
:
```

is identical to

```
$ fc -l
```
Example 2  Re-executing the previous command:

```
$ fc -s
```

is identical to

```
$ fc -s -- -1
```
You can use the built-in POSIX shell `sh` command `fg` to run jobs in the foreground. You should only use this command if you used `rlogin` to access the POSIX shell.

**Syntax**

```
fg[...job-id...]
```

*job-id*  
Each of the entered jobs is moved to the foreground. The section *Jobs* in the chapter "Entering commands from the POSIX shell" contains a description of the format of *job-id*.

*job-id* not specified:  
The current job is moved to the foreground.

**Exit status**

0 if execution is successful
>0 if an error occurs

**Locale**

The following environment variables affect the execution of `fg`:

- **LANG**  
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**  
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**  
  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example**

Job `%1` is to be moved to the foreground:
```
$ fg %1
```

**See also**

`bg, jobs, kill, wait`
fgrep

search a file for a fixed-string pattern

*fgrep* reads lines from one or more files or from standard input and searches the lines for strings. Unless told otherwise (by options), *fgrep* copies every line containing one of the specified strings to standard output.

If you specify more than one input file, the relevant file name will be displayed before each output line.

Syntax

Format 1: `fgrep[[-bchlnrvxy]]-e..patternlist[..file...]`

Format 2: `fgrep[[-bchlnrvxy]]-f..patternfile[..file...]`

Format 3: `fgrep[[-bchlnrvxy]]..patternlist[..file...]`

The formats are described together since the strings which *fgrep* uses to compare the input lines are specified either via *patternlist* or using the option `-e patternlist` or `-f patternfile`. You must specify one of these arguments. Two of them or all three together are not permitted.

No option specified

*fgrep* outputs all lines that match at least one of the strings specified in *list*. If you specify more than one input file, each output line will be preceded by the name of the file in which the line was found.

**option**

- **-b** (block) Each output line is preceded by the number of the block in which it was found. Each file is made up of 512-byte blocks which are numbered consecutively from 0. Option `-b` is sometimes useful in locating disk block numbers by context (see the *offset* argument for the *od* command on page 584, for example).

- **-c** (count) *fgrep* outputs only the number of lines found (i.e. the lines that *fgrep* would have displayed without the `-c` option, see Example 4); the lines themselves are not displayed.

- **-h** (hidden) When searching multiple files, *fgrep* does not write the file name before each output line.

- **-l** or **-y** (ignore) *fgrep* does not distinguish between uppercase and lowercase.

- **-l** (list) *fgrep* simply outputs the names of files that contain at least one of the matching lines. (These are the lines that *fgrep* would output if the `-l` option were omitted, see Example 5 on page 381.) Each file name is printed just once. The lines themselves are not displayed.

- **-n** (number lines) Each output line is preceded by its line number in the relevant input file. Line numbering starts at 1. If *fgrep* is reading from standard input, the line number refers to the standard input.
fgrep

- **r**  (recursive) Names that refer to directories are processed recursively; in other words, all the files and subdirectories in that directory are scanned as well.

- **v**  (vice versa) *fgrep* outputs all lines that do not match any of the specified strings.
  
  In conjunction with option **-c**:  
  *fgrep* prints only the number of lines that do not match.

  In conjunction with option **-l**:  
  *fgrep* only outputs the names of files containing such lines.

- **x**  (exact) *fgrep* outputs lines consisting solely of one of the specified strings.
  
  In conjunction with option **-c**:  
  *fgrep* only outputs the number of such lines.

  In conjunction with option **-l**:  
  *fgrep* outputs the names of files containing such lines.

- **-e**.**patternlist**  
  (expression) This option is needed if the first expression in **patternlist** begins with a `-` (dash). The **-e** option ensures that such strings are not interpreted as an option but as a list of strings to be matched with the input lines.

- **-f**.**patternfile**  
  (file) *fgrep* reads the search strings from the named **patternfile**. Each line in **patternfile** is interpreted as a string.

**patternlist**  
List of strings that *fgrep* is to use when comparing input lines. The individual strings must be separated by newline characters. If **patternlist** includes newline characters or other characters that have a special meaning for the shell, you must enclose **patternlist** in single quotes: ‘**patternlist**’.

**file**  
Name of the file that *fgrep* is to scan. You may name any number of files.

**file** not specified:  
*fgrep* reads input lines from the standard input.

**grep, fgrep and egrep**

The *grep, fgrep* and *egrep* commands perform similar functions and are largely identical in terms of usage. The following section lists the most important differences between these commands.

*grep* processes simple regular expressions. Only one regular expression may be specified in each call.
**fgrep**

*fgrep* processes strings only. However, you may specify several strings in one call. The strings can either be entered directly in the command line, separated by newline characters, or passed to *fgrep* from within a file.

*fgrep* is fast and compact and can search for a large number of strings. All specified strings are searched for in each individual line.

*egrep* processes extended regular expressions. Among other things, these include all simple regular expressions with one exception: the \(...\) construct used in simple regular expressions does not have a special meaning in extended regular expression syntax and is hence not processed by *egrep*.

Several regular expressions can be specified together, separated by newline characters. *egrep* interprets these newline characters as an OR operator (the vertical bar character; see section "Regular POSIX shell expressions" on page 877). The regular expressions can either be specified directly in the command line or passed to *egrep* via a file.

**Exit status**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Matching lines found</td>
</tr>
<tr>
<td>1</td>
<td>No matching lines found</td>
</tr>
<tr>
<td>&gt;1</td>
<td>Syntax error or “Cannot open file”. This exit status remains valid even if lines have been found in other input files.</td>
</tr>
</tbody>
</table>

**Locale**

The following environment variables affect the execution of *fgrep*:

- **LANG**
  
  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_COLLATE**
  
  Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.

- **LC_CTYPE**
  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

- **LC_MESSAGES**
  
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  
  Determine the location of message catalogs for the processing of **LC_MESSAGES**.
Example The files `customer1` and `customer2` are the basis for all the examples below. Their contents are as follows:

**customer1:**

- 080685 999.98 20 Units Item 038 Johnson Ltd.
- 120387 1240.25 3 Units Item 023 Skinner Ltd.
- 180588 330.87 1 Units Item 332 Skinner Ltd.

**customer2:**

- morrow lance, 86 sherwood street, london w1
- skinner robert, 16 garden hill, london ec3

Example 1 Output lines that match a string (without an option and with option `-i`):

```bash
$ fgrep Skinner customer1 customer2
```

```bash
customer1:120387 1240.25 3 Units Item 023 Skinner Ltd.
customer1:180588 330.87 1 Units Item 332 Skinner Ltd.
```

If you also wish to display lines containing the word `skinner` in lowercase you enter:

```bash
$ fgrep -i skinner customer1 customer2
```

```bash
customer1:120387 1240.25 3 Units Item 023 Skinner Ltd.
customer1:180588 330.87 1 Units Item 332 Skinner Ltd.
customer2:skinner robert, 16 garden hill, london ec3
```

Example 2 Search several strings (without an option and with the `-f` option):

```bash
$ fgrep 'Skinner
> Johnson' customer1 customer2
```

```bash
customer1:080685 999.98 20 Units Item 038 Johnson Ltd.
customer1:120387 1240.25 3 Units Item 023 Skinner Ltd.
customer1:180588 330.87 1 Units Item 332 Skinner Ltd.
```

Alternatively, you could write both strings into a file called `names` (each string in a separate line) and then call `fgrep` as follows:

```bash
$ fgrep -f names customer1 customer2
```

Example 3 Output lines that do not contain the given string (option `-v`):

```bash
$ fgrep -v Skinner customer1 customer2
```

```bash
customer1:080685 999.98 20 Units Item 038 Johnson Ltd.
customer2:morrow lance, 86 sherwood street, london w
```

```bash
customer2:skinner robert, 16 garden hill, london ec3
```
Example 4  Display the number of lines found (option -c):

To begin with, the number of lines containing the string *Skinner* are to be output for each input file.

```bash
$ fgrep -c Skinner customer1 customer2
  customer1:2
  customer2:0
```

Next the number of lines that do not contain the string *Skinner* is to be displayed.

```bash
$ fgrep -c -v Skinner customer1 customer2
  customer1:1
  customer2:2
```

Example 5  Display file names only (option -l):

First the names of files containing the string *Skinner* are to be output.

```bash
$ fgrep -l Skinner customer1 customer2
  customer1
```

Now the names of files with lines not containing the string *Skinner* are to be displayed.

```bash
$ fgrep -l -v Skinner customer1 customer2
  customer1
  customer2
```

Example 6  Display found lines with their line numbers (option -n):

```bash
$ fgrep -n -i skinner customer1 customer2
  customer1:2:120387 1240.25 3 Units Item 023 Skinner Ltd.
  customer1:3:180588 330.87 1 Units Item 332 Skinner Ltd.
  customer2:2:skinner robert, 16 garden hill, london ec3
```

See also  *ed, egrep, grep, sed*
The `file` command takes a list of one or more files and classifies each file on the basis of its contents. The types distinguished by `file` include directories, special files, FIFO files, archive libraries, C source programs, executable programs, shell scripts, and normal text files.

Caution! `file` looks up the magic number in the magic file to identify the target file. If it fails to identify the file by this method, it tries various plausibility checks. Consequently the results are not always reliable.

Syntax

| Format 1: file[-h][-m..magicfile][-f..ffile]..argfile..... |
| Format 2: file[-h][-m..magicfile][-f..ffile][..argfile.....] |
| Format 3: file..-c[-m..magicfile] |

Classify a file

Format 1  file[-h][-m..magicfile][-f..ffile]..argfile.....

Format 2  file[-h][-m..magicfile][-f..ffile][..argfile.....]

- **-h** (hidden) If `argfile` is a symbolic link, the link will not be followed and the following error message will be printed:

  `file1: symbolic link to file2`

- **-m..magicfile**
  (magic) Causes `file` to use `magicfile` instead of the system file `/etc/magic` to identify the magic numbers of the files being classified.

- **-f..ffile**
  `file` interprets the `ffile` argument as the name of a file which contains the names of all files to be examined. If this option is omitted, you must name at least one file to be classified.

- **argfile**
  Name of the file to be classified by the `file` command. If `argfile` is a symbolic link, `file` will follow the link and test the original file referenced by the symbolic link. You can name any number of files. If the `-f` option is omitted, you must name at least one.

Format 3  Check the magic file

| file..-c[-m..magicfile] |

- **-c** (check) The magic file, by default the system file `/etc/magic`, is checked for format errors. However, if the `-m` option is specified, the magic file `magicfile` is checked instead.
-m...magicfile

(magic file) Causes file to check the alternate magic file magicfile for format errors.

Mode of operation

file performs a series of tests on each input file and attempts to classify it on the basis of its contents. If it appears to be a text file, file examines an initial segment (the first 512 bytes) and tries to guess the language it was created in. The accuracy of the guess cannot be guaranteed, however.

If the input file is an executable program, it is identified as such, and further information is provided with respect to its contents. To do this, file searches the file for "magic numbers", i.e. for numeric constants or string constants that give an indication of the type of file. The file /etc/magic contains an explanation of these magic numbers.

Output

file writes its file classification to standard output. The following table lists the most important file types that file classifies:

<table>
<thead>
<tr>
<th>Output</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii text</td>
<td>ASCII text file</td>
</tr>
<tr>
<td>block special</td>
<td>Block special file</td>
</tr>
<tr>
<td>c program text</td>
<td>C source program</td>
</tr>
<tr>
<td>character special</td>
<td>Character special file</td>
</tr>
<tr>
<td>commands text</td>
<td>Shell script</td>
</tr>
<tr>
<td>cpio archive</td>
<td>Archive generated by cpio</td>
</tr>
<tr>
<td>current ar archive</td>
<td>Archive library (see ar)</td>
</tr>
<tr>
<td>data</td>
<td>Data file</td>
</tr>
<tr>
<td>directory</td>
<td>Directory</td>
</tr>
<tr>
<td>executable</td>
<td>Executable file (e.g. LLMs)</td>
</tr>
<tr>
<td>empty file</td>
<td>Empty file</td>
</tr>
<tr>
<td>fifo</td>
<td>FIFO file</td>
</tr>
<tr>
<td>fortran program text</td>
<td>FORTRAN source program</td>
</tr>
<tr>
<td>tar archive</td>
<td>Archive generated by tar, pax</td>
</tr>
<tr>
<td>text</td>
<td>EBCDIC text file</td>
</tr>
</tbody>
</table>

File

/etc/magic

File containing a key to the magic numbers
Locale  The following environment variables affect the execution of file:

*LANG*  Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

*LC_ALL*  If set to a non-empty string value, override the values of all the other internationalization variables.

*LC_CTYPE*  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

*LC_MESSAGES*  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

*NLSPATH*  Determine the location of message catalogs for the processing of LC_MESSAGES.

Example  The file list contains the following file names:

dir
letter
lib.a
prog.c

The following command line will yield information on the contents of each classified file type:

```
$ file -f list
dir: directory
letter: ascii text
lib.a: current ar archive
prog.c: c program text
```

Thus dir is a directory, and letter contains normal ASCII text. lib.a is an archive library; prog.c contains a C program.

The same output could also have been obtained with a command line reading:

```
$ file dir letter lib.a prog.c
```

See also  c89 [5]
**find**

**find files**

`find` scans directories and all their subdirectories for files which match a specified search criterion. Files located in this way can be listed and/or processed with a command. `find` locates not only ordinary files, but also directories, special files and FIFOs.

**Syntax**

```
find directory expression
```

- **directory**
  
  Name of the directory that is to be scanned. `find` will search through this directory and recursively through its subdirectories, provided you have read and execute permission (`r` and `x` bit) for them. More than one directory may be named.

- **expression**
  
  `expression` may be given as one or more primary expressions (primaries) of the kind listed below or as a combination of these primaries.

`find` applies `expression` to each file in the specified directories or subdirectories by evaluating primaries from left to right. Each primary is considered to have a "true" or "false" value for a file. The truth value of each expression and the next relational operator determine whether or not `find` will process the next primary.

Most primaries define conditions, i.e. they return a value of true if the current file satisfies the condition. The expressions:

- `-print`
- `-exec command \;`
- `-ok command \;`

are not conditions. They cause actions to be executed, e.g. output of the file name or deletion of the file. When `find` processes one of these expressions, the associated action is always performed, regardless of whether the expression returns a value of true or false. If none of the above action primaries are specified, `find` exits with an error message. Shell metacharacters that appear in `expression` must be escaped to ensure that they are passed to `find` and not interpreted by the shell.

**Primaries as conditions**

The following section describes primaries; the method of combining primaries is shown thereafter.

If you use options to find files with specific attributes (e.g. the files modified within a given period), the relevant option must come before the `-print` option (see below), as otherwise all files will be listed.

In the following discussion, `n` represents a decimal integer; `+n` means "more than `n"`, `-n` means "less than `n"`, and `n` means "exactly `n"."
find

-name file
True if file matches the name of the current file.
You can also use the usual shell metanotation for file name generation, but then file
must be enclosed in single quotes, e.g. find /usr/adam -name 'group.*' (see section
"Metacharacters for the POSIX shell" on page 884). Only file basenames may be
specified; pathnames such as text/tmp are not permitted.

-perm mode
True if the file permission flags exactly match mode (see section "chmod change file
modes" on page 204). mode can also be an octal number. If mode is prefixed by a minus
sign (-), only the bits that are set in mode are compared with the file permission flags,
and the expression evaluates true if they match.

-type character
True if the current file is of type character, where character can be:
f ordinary file
d directory
b block special file
c character special file
l symbolic links
p named pipe or FIFO file

-fstype name
True if the file system to which the current file belongs is of type name, where name can
be ufs or some special file systems such as proc or fdfs.

-links [+|−]n
True if there are (more/less than) n links to the current file.

-inum [+|−]n
True if the current file has an inode number greater than (+), less than (-), or equal to n.
The expression -inum +n -inum -m can be used to find all inode numbers between n
and m.

-user login_name
True if the current file belongs to the user login_name. This option is only of use for users
with uid=0 (command id).

-nouser
True if the current file belongs to a login name which is not present. This option is only
of use for users with uid=0 (command id).

-group group_name
True if the current file belongs to the group group_name.
find

-nogroup
True if the current file belongs to a group that is not listed in the /etc/group file.

-size...[+][-]n[<c>]
  c not specified:
  True if the current file is (more / less than / equal) \( n \) blocks long, where one block is 512 bytes.
  c specified:
  True if the current file is (more / less than / equal) \( n \) bytes long.

-atime...[+][-]n
  (access) True if the current file was last accessed (more / less than / equal) \( (n-1) \) to \( n \times 24 \) hours ago. The access time of directories searched by find is changed by find itself.

-mtime...[+][-]n
  (modification time) True if the current file was last modified (more / less than / equal) \( (n-1) \) to \( n \times 24 \) hours ago.

-ctime...[+][-]n
  True if the inode of the current file was last modified (more / less than / equal) \( (n-1) \) to \( n \times 24 \) hours ago.

-newer...file
  True if the current file is "newer" than the specified file, i.e. if it was created or modified at a later point in time.

-local
  True if the current file physically resides on the local system.

-prune
  Always true. If this expression is processed no search is conducted in files or directories which lie below the current level in the file hierarchy.

-xdev
  Always true. If this expression is processed no search is conducted in files or directories which lie below directories which have a different device number (\( st_{dev} \), see the stat() function [4]). If -xdev is set then this applies to the entire expression even if -xdev is not normally processed.

Action primaries

-print
  Always true. If this expression is present, it causes find to print the path name of the current file on standard output.

-exec...command...;\
  command is executed as soon as this expression is processed.
  You must terminate command with a blank and an escaped semicolon (\;).
A set of braces {} can be used to represent the name of the current file.

The expression `-exec command`\; is true if the executed command returns an exit status of 0. The truth value is significant when this expression is combined with other expressions.

`-ok`..command..\;

Like `-exec`, except that `find` asks if the command is to be executed each time the expression is processed. The command is only executed if you respond with the character or string signifying "y[es]" in the current locale (see environment variable `LANG`).

**Expressions that affect the behavior of find**

- `-depth`
  Always true. If present, this expression causes `find` to act upon entries in a directory before the directory itself.

- `-follow`
  Always true. This expression causes symbolic links to be followed. It should not be combined with the `-type l` expression.

- `-mount`
  Always true. If this expression is present, it causes `find` to restrict its search to the file system containing the directory specified.

**Combining expressions**

The expressions described above can be combined with one another as shown below. Please note the blanks used before and after the operators! The operators are listed in order of decreasing precedence.

\( (\ldots \text{expression} \ldots \ldots) \)

Parentheses group expressions together. The parentheses themselves must be escaped with a backslash, since they have a special meaning for the shell.

\(!\text{expression}\)

Negation, i.e. true if the expression evaluates to false.

`expression..expression`

Logical AND, i.e. true if both expressions are true. If the first expression is false, `find` does not process the second expression.

**Example**

If the second expression is

`-exec command`\;

the specified command will only be executed if the first expression is true.
expression...-o...expression
Logical OR, i.e. true when either of the expressions is true. If the first expression is true, find does not process the second expression.

Example
If the second expression is
-exec command \\
the command will only be executed if the first expression is false.

Error
Error messages without command termination (exit status 0)
find: cannot read dir dir: Permission denied
Since you have no read permission for the dir directory, this directory cannot be scanned. find does not terminate after this error message, but continues scanning directories for which you do have the required access permissions (r bit).

find: cannot access dir/file dir/file: permission denied
Since you have no execute permission (x bit) for dir, this directory cannot be scanned. find does not terminate after this message but searches through those directories for which you do have the appropriate permissions (r and x bits).

find: cannot open dir dir: No such file or directory
There is no directory named dir.

Error messages with command termination (non-zero exit status)
find: missing conjunction
You have either combined primaries incorrectly in the find call or specified more than one argument in a primary.

find: no action specified
You have not specified an action (print, exec, ok).

find: incomplete statement
You have forgotten the escaped semicolon \\
which is required to terminate a command in an -exec or -ok expression.

find: insufficient number of arguments
You have specified too few arguments.

find: path-list predicate-list
You have failed to specify the pathname list for directory.

File
/etc/group
File containing group names
The following environment variables affect the execution of `find`:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_COLLATE**
Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements used in pattern matching notation for the `-n` option. In file name generation patterns in square brackets (such as `find . -name '[=a=]' -print`), the `LC_COLLATE` environment variable governs the scope of character ranges, equivalence classes and collating elements.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments), the behavior of character classes in pattern matching notation for the `-n` option and the behavior of character classes within regular expressions.

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

### Example 1
A simple example to illustrate how `find` works:

Display the path names of all files that
- are located in the current directory or in a subdirectory and
- have the name `tmp`.

```
$ find . -name tmp -print
```

In this case, *expression* comprises two primaries that are linked by a logical AND operator. `find` first checks the `-name tmp` primary for each file. If this expression is true (i.e. the current file is named `tmp`), `find` will also process the expression `-print`, which outputs the path names. Otherwise, `find` does not process the `-print` expression and remains silent.

Please note the sequence. If you entered:

```
$ find . -print -name tmp
```

instead, `find` would process the `-print` expression for each file, i.e. would output the path names of all files.
Example 2  A more complex example:

Display the path names of all files that
– are located in the current directory or its subdirectories, and
– are named tmp or have names ending in .xx.

$ find . \( -name tmp -o -name '*.xx' \) -print

Please note the use of blanks between the arguments!

find begins by processing the parenthesized specifications for each file. These comprise
two primaries linked by a logical OR operator.
find first tests the -name tmp primary. If this expression is true (i.e. if the current file is named
tmp), find does not process the expression -name '* .xx' at all; this is because the result of
the expression in parentheses will be true in any case. If -name tmp evaluates to false, find
tests the second primary, i.e. -name '.* .xx'. If this expression is true (i.e. the file name ends
in .xx), the parenthesized expression will also evaluate to true; otherwise, the result of the
expression in parentheses is false.

If the parenthesized expression is true, find then processes the -print expression, i.e.
outputs the file names (AND operator; see Example 1). If false, find does not process the
-print expression, i.e. remains silent.

Example 3 Search all entries in the /usr/santaclaus directory and its subdirectories and print the files not
owned by santaclaus.

$ find /usr/santaclaus ! -user santaclaus -print

Example 4 Delete all files that,
– are named tmp or have names ending in .xx, and
– have not been accessed for more than 7 days.

Confirmation is to be requested before the removal of each file.

$ find / \( -name tmp -o -name '* .xx' \) -atime +7 -ok rm {} ;

Example 5 Recursively print all file names in the current directory and below, but skipping all SCCS
directories:

$ find . -name SCCS -prune -o -print

See also chmod, ln, test
stat() [4]
fold  filter for folding lines

fold folds the lines of text files or the standard input so that they do not exceed the preset or specified line length. The default presetting is 80 characters per line.

The lines are folded through the insertion of a newline character so that each output line (termed "segment" in the following) does not exceed the preset or specified line length (or does not exceed the maximum number of bytes). However, lines are not folded in the middle of a character. Behavior is not predictable if number is smaller than the number of columns which may be occupied by a single input character.

If a carriage return, backspace or tab is read in the input and the option -b is not set then the input is considered to be a character:

- **Backspace**
  The current line length is reduced by one. However, it cannot assume a negative value. fold does not insert a newline character either immediately before or after a backspace.

- **Carriage return**
  The current line length is set to zero. fold does not insert a newline character either immediately before or after a carriage return.

- **Tab**
  All tabs that are read move the column position pointer to the next tab stop position. Tab stop positions are located at all column positions \( n \) where: \( n \mod 8 = 1 \).

⚠ **Caution!** If the input lines contain underscores then fold may react incorrectly.

Syntax  

```bash
fold[[-bs][[-w..width]][..file...]]
```

No option specified

fold folds the input lines so that no line exceeds a maximum width of 80 characters (default).

option

- **-b** (bytes) The line length is calculated in bytes not column positions.

- **-s** (segment) If a line segment contains a blank within the first number of column positions (or bytes) then the line is folded after the last space which fulfills the number condition. If no space which fulfills this condition exists then the -s option is ineffective for this input line.
-w..width

(width) Defines the maximum width of a line, where width is the number of characters per line.

width should be a multiple of 8 if tabs are present in the input line.

file

Name of the text file to be folded. You may name several such files, in which case the specified options will be applied to each file.

file not specified:

fold reads from standard input.

Locale

The following environment variables affect the execution of fold:

LANG

Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL

If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files), and for the determination of the width in column positions each character would occupy on a constant-width font output device.

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH

Determine the location of message catalogs for the processing of LC_MESSAGES.
Example  Printing a file named notes on standard output.

The following result is obtained with the cat command:

```
$ cat
Notes:
Remind Carl to send in a bid within the next two weeks. The new deadline
for the evaluation of all bids is the end of June.
```

Using the fold command, you have the option of redefining the length of the output lines.

The following command folds the contents of the notes file so that each line has a maximum
width of 40 characters:

```
$ fold -w 40 notes
Notes:
Remind Carl to send in a bid within the next two weeks. The new deadline for the
evaluation of all bids is the end of June.
```

See also  pr
fsck

**file system check**

`fsck` checks file systems for inconsistencies and corrects these interactively with the user. The user's agreement is requested before any corrective measures are implemented. Any inconsistencies other than those mentioned above may result in data loss. Diagnostic messages provide information about the extent and seriousness of such losses.

`fsck` automatically corrects less serious inconsistencies such as unreferenced inodes, excessive reference counter values in inodes, missing blocks in the free blocks list, blocks which simultaneously appear in the free blocks list and in files or incorrect superblock counters. A message is output for each inconsistency which is corrected. This message displays the type of correction and the file system in which it was performed. After the file system has been corrected, `fsck` outputs the number of files in the file system, the number of occupied and free blocks and the corresponding percentage values.

By default, the command waits for the user's response, which must be either `yes` or `no`, before performing any correction. If the user does not possess write permission for the file system then `fsck` defaults to `-n` (no correction).

The command checks for the following inconsistencies:

1. Blocks claimed by multiple inodes or the free block list.
2. Blocks which are claimed by multiple inodes or the free block lists which lie outside the file system.
3. Incorrect reference counters.
4. Incorrect directory sizes.
5. Incorrect inode formats.
6. Blocks which are not listed anywhere.
7. Directory checks, files which point to an unassigned inode, inode numbers outside the allocated range, omission of `.' and `..' as the first directory entry.
8. Superblock checks: More blocks for inodes than are present in the file system.
9. Incorrect format for free block list.
10. Total of free blocks and/or free inodes incorrect.

If the user agrees, orphaned files or directories (i.e. allocated but unreferenced) are relinked through being written to the `lost+found` directory. The allocated name corresponds to the inode number. If the `lost+found` directory does not yet exist it is created at this point. If not enough space is available, the directory is extended.
To designate a file system you may specify the name of the block or character-oriented device on which it is located or the name of the mountpoint.

Syntax

```
fsck[-F_..ufs|UFS][-y|-n|-m][special_file ...]
```

The following options are possible:

- `-F_UFS`
  Specifies ufs as the file system type.

- `-y`
  All inconsistencies are checked. The questions asked by `fsck` are answered with "yes".

- `-n`
  All inconsistencies are checked. The questions asked by `fsck` are answered with "no".

- `-m`
  The status in the file system's superblock is checked. This option ensures that the file system is suitable for mounting.

`special_file`

`special_file` represents a block or character special device (for example, `/dev/dsk/234`). It is preferable to use a character special device. `fsck` cannot be applied to mounted block special devices.

`special_file` not specified:

`fsck` looks through the `/etc/vfstab` file and executes `fsck` for all the character special devices in the `fsckdev` field of `/etc/vfstab` for which there is a numeric entry in the `fsckpass` field.

Hint

In almost all cases it is quicker to examine character-oriented devices. For file systems which were mounted with a journal, the `-m` option is also sufficient to place the file system in a consistent status in a very short time after a system crash. The check of all the inconsistencies, on the other hand, can take a very long time depending on the size and utilization level of the file system.

The `fsck` command is not supported for bs2fs file systems.

File

`lost+found`

`/etc/vfstab`

List of default parameters for each file system

See also

`mount`, `mountall`, `umount`
fsexpand expand existing file systems

fsexpand enables file systems to be expanded by PAM pages or cylinder groups. The file systems may not be mounted.

Syntax

```
fsexpand[\[-i\]]\[-p\]{pam pages}|\[-c\]{cylinder groups}\[-i\] device
```

The following options are possible:

- `-i` Outputs file system information. Output takes place to stdout. In particular information is output on how ideal an expansion is, e.g. to prevent unused PAM pages in the expanded file system. In conjunction with the `-p` or `-c` option, information is output before the expansion (source file system) and after the expansion (target system).

- `-p` pam pages
  Expansion of a file system by PAM pages.
  The file system is expanded by the specified number of PAM pages.

- `-c` cylinder groups
  Expansion of a file system by cylinder groups.
  The file system is expanded by the specified number of cylinder groups. The size of a cylinder group in the source file system can be ascertained before expansion using `fsexpand -i`. Up to a file size of a little more than 2 GB, the size is 2048 PAM pages. From 2 GB on, the size increases. With a file size of 4 GB, for example, it is then 4096 PAM pages.

- `device`
  Device name `/dev/rdsk/...` (character-oriented device only, i.e. not `/dev/dsk`) or BS2000 file name.
  `device` must be writeable when an expansion is to take place. If the file system is located on the HOME pubset, the BS2000 file name (with or without catid) must match the notation in the `/etc/partitions` file.

Hints

The `fsexpand` command is not supported for bs2fs file systems.

A file system is expanded in two steps:

1. Physical expansion
2. Combining (administration) units of the file system (known as cylinder groups) to form larger units (compactification)
   Compactification is performed only if the expanded file system is greater than 2 GB and at least double the size of the original file system.
Compactification requires a considerable amount of the runtime for \textit{fsexpand}. If, for example, a file system which is 1 GB in size is expanded by a value which is only slightly above 1 GB (i.e. with compactification), the runtime is extended by a factor of 3 - 4 compared to expansion by a value which is slightly less than 1 GB (i.e. without compactification). However, compactification has the advantage that considerable gains in performance can be achieved when the file system is used because at runtime the requirement for memory and CPU time is lower thanks to the administration units being combined.

\textbf{Example} \ $\texttt{fsexpand} \ -i$ \ \$\texttt{BACH.BACH.TEST}$

\begin{verbatim}
device /dev/rdsk/8 in Containerfile: $BACH.BACH.TEST
size of BS2000 Containerfile (PP): 12288
last page used in Containerfile (PP): 12288
size used for POSIX filesystem (PP): 12288
unused in Containerfile (PP): 0
Cylindergroups in filesystem: 6
Cylinders in cylindergroup (PP): 16
size of a cylindergroup (PP): 2048
inodes per cylindergroup: 2048
inodes total: 12288
datablocks (4K) in filesystem: 5731
  free blocks directories free inodes 1853 4 12232
optimal values for expansion of container: 0 PP + N * 2048 PP
\end{verbatim}
ftyp  define file processing mode (BS2000)

ftyp determines whether files are interpreted as text or binary files when copied between BS2000 and the POSIX file system using bs2cp. The command is only effective for SAM files and for text-type PLAM library elements (element type other than LLM). PAM files and LLMs are always interpreted as binary files and ISAM files are always interpreted as text files. If no ftyp command is entered, all SAM files are interpreted as text files.

Syntax

ftyp[-h][...text|binary|textbin]

No option specified
The text option is used.

option

-h  Prints out the command syntax with an explanation of the options.

text

SAM files and text-type library elements are to be interpreted as text files.

Given this setting, the following conversions are performed when writing to the BS2000:

<table>
<thead>
<tr>
<th>POSIX</th>
<th>BS2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab character (x’05’)</td>
<td>Corresponding number of spaces to tab position</td>
</tr>
<tr>
<td>Newline character (x’15’)</td>
<td>Line change (Record change)</td>
</tr>
</tbody>
</table>

When a file is read from the BS2000 a newline character is appended to each record which is read (x’15’).

This option can not be used for SAM files with fixed record length.

binary

SAM files are to be interpreted as binary files.

No conversions are performed.

This option can also be used for SAM files with fixed record length. In this case the last record is padded out with binary zeros.
textbin
SAM files and text-type library elements are to be interpreted as binary text files.

Given this setting, the following conversions are performed when SAM files are written to the BS2000:

<table>
<thead>
<tr>
<th>POSIX</th>
<th>BS2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newline character (x´15´)</td>
<td>Line change (record change)</td>
</tr>
</tbody>
</table>

Tabs are not converted.

When a file is read from the BS2000 a newline character is appended to each record which is read (x´15´).

This option can not be used for SAM files with fixed record length.

Example
The input file of the POSIX file system should be transferred to the BS2000 as a binary file.

```
$ ftyp binary
$ bs2cp input bs2:output
```

See also  
bs2cp, bs2file
gencat  generate a formatted message catalog

The gencat utility merges the contents of a message text source file into a binary encoded message catalog.

The message catalog is created if it does not already exist. If it does exist, its messages are included in the new message catalog. If the set numbers and message numbers of existing and newly defined message texts match, gencat replaces the old message texts currently contained in the message catalog with the new message texts defined in the message text file.

Message catalogs produced by gencat are binary encoded, which means that their portability cannot be guaranteed between different types of machine. Thus, just as C programs need to be recompiled for each type of machine, message catalogs must be recreated with gencat.

A message text source file can contain either set and message numbers or simply message numbers. In the later case, the set NL_SETD is assumed for the set numbers.

Syntax

```
  gencat[[-lm]..catfile[消息文件]]
```

option

- l  If catfile already exists, information on messages contained in it will be written to standard output in the following format:

```
SET set_num, MESSAGE mesg_num, OFFSET offset, LENGTH msg_len, message_txt
```

**offset** indicates the distance from the beginning of the file to the start of the message text, i.e.

- first message: offset (1) = 0
- nth message: offset (n) = offset (n-1) + msg_len (n)

- m  The -m option ensures compliance with the X/Open Standard.

The -m option is supplied for compatibility with previous versions of gencat released in a number of specialized internationalization products. This option will cause gencat to build a single file catfile which is compatible with the format catalogs produced by the earlier versions. The retrieval routines detect the type of catalog they are using and act appropriately.

The -m option corresponds to the default behavior of gencat and need not be explicitly set.
No option specified

gencat behaves as described under the -m option.

catfile

Name of the binary encoded message catalog that gencat is to generate from the
message text file msgfile.

If a dash (-) is specified for catfile, gencat prints to standard output.

msgfile

Name of the message text source file. gencat generates a binary encoded message
catalog from this file.

If a dash (-) is specified for msgfile, gencat reads from standard input.
More than one message text file may be specified.

Format of a message catalog

A message catalog created by gencat with the option -m contains the following structures in
the given order:

- The catalog header, CAT_HDR, consisting of:
  - the file's magic number
  - the number of sets in the message file
  - the space (in bytes) needed to load the file, excluding the length of the file header
  - the position at which the message headers begin, excluding the length of the file
    header
  - the position at which the message texts begin, excluding the length of the file
    header

- A set header, CAT_SET_HDR, for each existing set. This header consists of:
  - the set number
  - the number of messages in the set
  - the initial offset in the table

- A message header, CAT_MSG_HDR, for each existing message. This header consists
  of:
  - the message number
  - the length of the message in bytes
  - the message offset in the table

- The individual message texts, delimited by \0.
Locale

The following environment variables affect the execution of `gencat`:

**LANG**

Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**

If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

**LC_MESSAGES**

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**

Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example

Generating a binary encoded message catalog `en_wc.cat` from the English message text file `en_wc.msf` for the `wc` command. The contents of the message text file are as follow:

```
 1 wc: cannot open %s
 2 together
 3 Syntax: wc[-c][-m][-lw][...file...]
```

The following command generates the message catalog:

```
$ gencat en_wc.cat en_wc.msf
```

See also

`iconv`, `catopen()`, `catgets()`, `catclose()`, `limits.h`, `nl_types.h` [4]
getconf get configuration values

In the first synopsis form, getconf writes the value of the variable specified by the system_var operand to the standard output.

In the second syntax form, getconf writes the value of the variable specified by the path_var operand to the standard output. This value is valid within the path specified by the pathname operand.

The value of each configuration variable is output as if it were determined by calling the function from which it was defined. The value reflects the conditions in the current execution environment.

If the specified variable is valid, but is not defined on the system, getconf writes %undefined to the standard output.

Syntax

Format 1: getconf..system_var
Format 2: getconf..path_var..pathname

Format 1 getconf..system_var

Name of a configuration variable whose value can be obtained using the confstr() or sysconf() functions [4]. All of the values listed in the following table are supported:

<table>
<thead>
<tr>
<th>ARG_MAX</th>
<th>NL_TEXTMAX</th>
<th>POSIX2_C_DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_BASE_MAX</td>
<td>OPEN_MAX</td>
<td>POSIX2_C_VERSION</td>
</tr>
<tr>
<td>BC_DIM_MAX</td>
<td>_POSIX_ARG_MAX</td>
<td>POSIX2_EXPR_NEST_MAX</td>
</tr>
<tr>
<td>BC_SCALE_MAX</td>
<td>_POSIX_CHILD_MAX</td>
<td>POSIX2_FORT_DEV</td>
</tr>
<tr>
<td>BC_STRING_MAX</td>
<td>_POSIX_JOB_CONTROL</td>
<td>POSIX2_FORT_RUN</td>
</tr>
<tr>
<td>CHAR_BIT</td>
<td>_POSIX_LINK_MAX</td>
<td>POSIX2_LINE_MAX</td>
</tr>
<tr>
<td>CHAR_MAX</td>
<td>_POSIX_MAX_CANON</td>
<td>POSIX2_LOCALEDEF</td>
</tr>
<tr>
<td>CHAR_MIN</td>
<td>_POSIX_MAX_INPUT</td>
<td>POSIX2_RE_DUP_MAX</td>
</tr>
<tr>
<td>CHILD_MAX</td>
<td>_POSIX_NAME_MAX</td>
<td>POSIX2_SW_DEV</td>
</tr>
<tr>
<td>CLK_TCK</td>
<td>_POSIX_NGROUPS_MAX</td>
<td>POSIX2_UPE</td>
</tr>
<tr>
<td>COLL_WEIGHTS_MAX</td>
<td>_POSIX_OPEN_MAX</td>
<td>POSIX2_VERSION</td>
</tr>
<tr>
<td>CS_PATH</td>
<td>_POSIX_PATH_MAX</td>
<td>RE_DUP_MAX</td>
</tr>
<tr>
<td>EXPR_NEST_MAX</td>
<td>_POSIX_PIPE_BUF</td>
<td>SCHAR_MAX</td>
</tr>
<tr>
<td>INT_MAX</td>
<td>_POSIX_SAVED_IDS</td>
<td>SCHAR_MIN</td>
</tr>
<tr>
<td>INT_MIN</td>
<td>_POSIX_SSIZE_MAX</td>
<td>SHRT_MAX</td>
</tr>
<tr>
<td>LINE_MAX</td>
<td>_POSIX_STREAM_MAX</td>
<td>SHRT_MIN</td>
</tr>
</tbody>
</table>
getconf

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONG_MAX</td>
<td>_POSIX_TZNAME_MAX</td>
</tr>
<tr>
<td>LONG_MIN</td>
<td>_POSIX_VERSION</td>
</tr>
<tr>
<td>MB_LEN_MAX</td>
<td>POSIX2_BC_BASE_MAX</td>
</tr>
<tr>
<td>NGROUPS_MAX</td>
<td>POSIX2_BC_DIM_MAX</td>
</tr>
<tr>
<td>NL_ARGMAX</td>
<td>POSIX2_BC_SCALE_MAX</td>
</tr>
<tr>
<td>NL_LANGMAX</td>
<td>POSIX2_BC_STRING_MAX</td>
</tr>
<tr>
<td>NL_MAX</td>
<td>POSIX2_CHAR_TERM</td>
</tr>
<tr>
<td>NL_MSGMAX</td>
<td>POSIX2_COLL_WEIGHTS_MAX</td>
</tr>
<tr>
<td>NL_SETMAX</td>
<td>POSIX2_C_BIND</td>
</tr>
<tr>
<td>CHARCLASS_NAME_MAX</td>
<td>NL_TEXTMAX</td>
</tr>
<tr>
<td>CHAR_BIT</td>
<td>_XOPEN_VERSION</td>
</tr>
<tr>
<td>NL_ARGMAX</td>
<td>TMP_MAX</td>
</tr>
<tr>
<td>NL_LANGMAX</td>
<td>WORD_BIT</td>
</tr>
<tr>
<td>NL_MAX</td>
<td>_XOPEN_CRYPT</td>
</tr>
<tr>
<td>NL_MSGMAX</td>
<td>_XOPEN_ENH_I18N</td>
</tr>
<tr>
<td>NL_SETMAX</td>
<td>_XOPEN_SHM</td>
</tr>
</tbody>
</table>

The symbol `PATH` is also recognized. It returns the same value as the `confstr()` value `CS_PATH`. `getconf` also recognizes the variables `LOGNAME_MAX`, `PAGE_SIZE`, `PAGESIZE` and `PASS_MAX`.

All variable names can be specified with or without a leading underscore (`_`).

405
getconf

Format 2  \texttt{getconf\_path\_var\_pathname}

\begin{itemize}
  \item \texttt{path\_var}
    \begin{itemize}
      \item Name of a configuration variable whose value can be obtained using the \texttt{pathconf()} \cite{pathconf} function. All of the values listed in the following table are supported:
    \end{itemize}
  \end{itemize}

\begin{tabular}{|l|l|l|}
  \hline
  LINK\_MAX & NAME\_MAX & \_POSIX\_CHOWN\_RESTRICTED \\
  \hline
  MAX\_CANON & PATH\_MAX & \_POSIX\_NO\_TRUNC \\
  \hline
  MAX\_INPUT & PIPE\_BUF & \_POSIX\_VDISABLE \\
  \hline
\end{tabular}

\begin{itemize}
  \item \texttt{pathname}
    \begin{itemize}
      \item Pathname for which the variable specified by \texttt{path\_var} is to be determined.
    \end{itemize}
  \end{itemize}

Locale

The following environment variables affect the execution of \texttt{getconf}:

\begin{itemize}
  \item \texttt{LANG}
    \begin{itemize}
      \item Provide a default value for the internationalization variables that are unset or null. If \texttt{LANG} is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
    \end{itemize}
  \end{itemize}

\begin{itemize}
  \item \texttt{LC\_ALL}
    \begin{itemize}
      \item If set to a non-empty string value, override the values of all the other internationalization variables.
    \end{itemize}
  \end{itemize}

\begin{itemize}
  \item \texttt{LC\_CTYPE}
    \begin{itemize}
      \item Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
    \end{itemize}
  \end{itemize}

\begin{itemize}
  \item \texttt{LC\_MESSAGES}
    \begin{itemize}
      \item Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
    \end{itemize}
  \end{itemize}

\begin{itemize}
  \item \texttt{NLSPATH}
    \begin{itemize}
      \item Determine the location of message catalogs for the processing of \texttt{LC\_MESSAGES}.
    \end{itemize}
  \end{itemize}

Exit status

The following exit values are returned:

\begin{itemize}
  \item 0  \begin{itemize}
        \item The specified variable is valid and information about its current status was written to the standard output.
      \end{itemize}
  \item >0  \begin{itemize}
        \item An error occurred.
      \end{itemize}
\end{itemize}
Example 1  In this example, the value of \{NGROUPS_MAX\} is obtained (Format 1):

\$ getconf NGROUPS_MAX
16

Example 2  In this example, the value of \{NAME_MAX\} is obtained for a specific directory (Format 2):

\$ getconf NAME_MAX /usr
255

Example 3  This example shows how to deal more carefully with results that might be unspecified:

```bash
if value=$(getconf PATH_MAX /usr); then
    if [ "$value" = "undefined" ]; then
        echo PATH_MAX in /usr is infinite.
    else
        echo PATH_MAX in /usr is $value.
    fi
else
    echo Error in getconf.
fi
```

Note that the following calls in a C program could return different results:

```c
sysconf(_SC_POSIX_C_BIND);
```

and:

```c
system("getconf POSIX2_C_BIND");
```

The `sysconf()` \[4\] call returns a value that corresponds to the conditions when the program is either compiled or executed. The `system()` \[4\] call to `getconf` always returns a value that corresponds to the conditions when the program is executed.

See also  `pathconf()`, `confstr()`, `sysconf()` \[4\]
getopts parse utility options

The POSIX shell built-in `getopts` is used by shell scripts to parse arguments and options in the command line and to check for valid options. It supports all applicable rules of the command syntax standard.

You can use `getopts` in shell procedures to analyze the arguments specified when the procedure is called. The individual options and arguments are stored in sequence in shell variables and can then be easily queried or checked. If an option in the argument list is not permitted or if `getopts` finds no corresponding argument for an option which requires an argument then `getopts` issues a corresponding error message.

The argument list should always be processed using `getopts` and examined for valid options in order to ensure that all procedures and commands process the argument list in the same way.

**Syntax**

```bash
getopts ..optstring..name[...arg]...
```

**optstring**

A string which may consist of letters and colons : . The letters specified in `optstring` are considered by `getopts` to be permitted shell procedure options. If a letter is followed by a colon then `getopts` expects an argument or a group of arguments for this option. Any arguments must be separated from the option by spaces or tabs.

**name**

Name of the shell variable in which `getopts` places the next option each time it is invoked.

**arg...**

Argument list parsed by `getopts`.

*arg not specified:

`getopts` parses the argument list of the command line with which the script was invoked.
Mode of operation

Each time it is invoked, `getopts` places the next option in the shell variable `name` and the index of the next argument to be processed in the shell variable `OPTIND`. Whenever the shell command interpreter or a shell script is invoked, `OPTIND` is initialized to 1.

In the case of options which require an argument, this argument is assigned to the shell variable `OPTARG`. Options which require an argument must be identified by a colon `:` in `optstring`. If no option is present or if the option has no argument then `OPTARG` is reset.

If an invalid option is identified then a question mark `?` is assigned to the shell variable `name`. Invalid options are those which are not present in `optstring`. In this case (if the first character in `optstring` is a colon `:`) the located option character is assigned to the shell variable `OPTARG`. However, no output is written to the standard error output. Otherwise the shell variable `OPTARG` is reset and a message is written to the default error output. This is interpreted as a detected error in the argument presentation for the calling function. It is, however, not a `getopts` error.

If an option argument is missing, then:

- If the first character of `optstring` is a colon `:`, the shell variable specified in `name` is set to the colon and the shell variable `OPTARG` is set to the detected option character.
- Otherwise the shell variable specified in `name` is set to the question mark, the shell variable `OPTARG` is reset and a message is written to the standard error output. This is interpreted as a detected error in the argument presentation for the calling function. It is, however, not a `getopts` error. A message is output as indicated but the exit status is zero.

When the end of the option list is encountered, `getopts` exits with a non-zero exit status. The special option `--` may be also used to identify the end of the options.

Changing the value of the shell variable `OPTIND` or invoking `getopts` with different sets of arguments may lead to unexpected results.
Locale

The following environment variables affect the execution of `getopts`:

**LANG**

Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**

If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

**LC_MESSAGES**

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**

Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example

The following fragment of a shell script `scrpt` shows how you might process the arguments for a script that can take the options `-a` or `-b`, as well as the option `-o`, which requires an option argument. Options `-a` and `-b` are mutually exclusive, and if both are specified, only the one listed second applies:

```bash
while getopts abo: c
  do
    case $c in
      [ab])   FLAG=$c;;
      o)      OARG=$OPTARG;;
      \?)     echo "usage: $0 [-a| -b] [-o <arg>]"
               exit 2;;
    esac
done
shift `expr $OPTIND - 1`
```

This code accepts any of the following invocations of `scrpt` as equivalent:

- `proz -a -b -o "xxx z yy" file`
- `proz -a -b -o "xxx z yy" -- file`
- `proz -ab -o "xxx z yy" file`
**grep** search a file for a pattern

`grep` reads lines from one or more text files or from standard input and compares these lines with a specified pattern. Unless told otherwise (by options), `grep` copies every line that matches the pattern to standard output. `grep` permits the use of simple regular expressions in the specified pattern (see section "Regular POSIX shell expressions" on page 877). If you specify more than one input file, the relevant file name will be displayed before each output line.

### Syntax

**Format 1:**
```
grep[ -E| -F][ -c| -l| -q][-bihnrsvxy][-e patternlist][-f patternlist][ -file...]
```

**Format 2:**
```
grep[ -E| -F][ -c| -l| -q][-bihnrsvxy][-e patternlist][-f patternfile][ -file...]
```

**Format 3:**
```
grep[ -E| -F][-c| -l| -q][-bihnrsvxy][-patternlist][ -file...]
```

The formats are described together since the patterns which `grep` uses to compare the input lines are specified either via `patternlist` or using the option `-e patternlist` or `-f patternfile`.

**No option specified**

`grep` outputs all lines that match the given pattern. If you specify more than one input file, each output line will be preceded by the name of the file in which the line was found.

**option**

- **-E (E - extended)** `grep` treats each pattern as an extended regular expression. Option `-E` is equivalent to the `egrep` command.

- **-F (F - fast grep)** `grep` searches for strings. Option `-F` is equivalent to the `fgrep` command.

- **-c (count)** `grep` outputs only the number of lines found (i.e. the lines that `grep` would have displayed without the `-c` option, see Example 3); the lines themselves are not displayed.

- **-l (list)** `grep` simply outputs the names of files that contain at least one of the matching lines. (These are the lines that `egrep` would output if the `-l` option were omitted, see Example 4 on page 415.) Each file name is printed just once. The lines themselves are not displayed.

- **-q (quiet)** `grep` writes nothing to the standard output even if matching lines are found.

- **-b (block)** Each output line is preceded by the number of the block in which it was found. Each file is made up of 512-byte blocks which are numbered consecutively from 0. The `-b` option is sometimes useful in locating disk block numbers by context (see the `offset` argument for the `od` command on page 584, for example).
-i or -y
(ignore) when performing comparisons, grep does not distinguish between uppercase and lowercase.

-h (hidden) When searching multiple files, grep does not write the file name before each output line.

-n (number lines) Each output line is preceded by its line number in the relevant input file. Line numbering starts at 1. If grep is reading from standard input, the line number refers to the standard input.

-r (recursive) Names that refer to directories are processed recursively; in other words, all the files and subdirectories in that directory are scanned as well.

-s (silent) Error messages produced for non-existent files or files which the user is not authorized to read are suppressed.

-v (vice versa) grep outputs all lines that do not match the specified pattern.

   In conjunction with option -c:
   grep prints only the number of lines that do not match.

   In conjunction with option -l:
   grep only outputs the names of files containing such lines.

-x (x - exact) grep only outputs lines which contain one of the specified strings and no other characters.

-e..patternlist
(e - expression) You need to set this option when the first expression in patternlist starts with a hyphen -. When -e is set, such a pattern list is not interpreted as an option but as a list of patterns with which egrep is to compare the input lines.

-f..patternfile
(f - file) egrep reads the list of patterns from the file patternfile. Every line in patternfile is interpreted as an extended regular expression.

patternlist
A simple regular expression to be used by grep when searching for matching input lines (see section “Regular POSIX shell expressions” on page 877). If patternlist contains characters that have a special meaning for the shell, you must enclose it in single quotes: ‘patternlist’.

file
Name of the file that grep is to scan. You may name any number of files.

   file not specified:
   grep reads its input from standard input.
**grep** can only be applied to text files. Applying **grep** to binary files (history files, for example) will produce an undefined result, because the occurrence of a null byte logically terminates an input line.

**grep, fgrep and egrep**

The **grep**, **fgrep** and **egrep** commands perform similar functions and are largely identical in terms of usage. The following section lists the most important differences between these commands.

**grep** processes simple regular expressions.

**fgrep** processes strings only. However, you may specify several strings in one call. The strings can either be entered directly in the command line, separated by newline characters, or passed to **fgrep** from within a file. **fgrep** is fast and compact and can search for a large number of strings. All specified strings are searched for in each individual line.

**egrep** processes extended regular expressions. Among other things, these include all simple regular expressions with one exception: the `\(...\)` construct used in simple regular expressions does not have a special meaning in extended regular expression syntax and is hence not processed by **egrep**.

Several regular expressions can be specified together, separated by newline characters. **egrep** interprets these newline characters as an OR operator (the vertical bar character; see section “Regular POSIX shell expressions” on page 877). The regular expressions can either be specified directly in the command line or passed to **egrep** via a file.

**Locale**

The following environment variables affect the execution of **grep**:

- **LANG**
  
  Provide a default value for the internationalization variables that are unset or null. If **LANG** is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_COLLATE**
  
  Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.

- **LC_CTYPE**
  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.
**grep**

*LC_MESSAGES*  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

*NLSPATH*  
Determine the location of message catalogs for the processing of *LC_MESSAGES*.

Exit status

0  Lines found  
1  No line found  
>1  Syntax error or “Cannot open file”. This exit status remains valid even if lines have been found in other input files.

**Example**  
The files *customer1* and *customer2* will be used as a basis for all following examples. Their contents are given below:

*customer1*:

```
080685  999.98  20 Units  Item 038   Johnson Ltd.
120387  1240.25   3 Units  Item 023   Skinner Ltd.
180588   330.87   1 Units  Item 332   Skinner Ltd.
```

*customer2*:

```
morrow lance, 86 sherwood street, london w1
skinner robert, 16 garden hill, london ec3
```

**Example 1**  
Output lines that match a specified pattern (without an option and with option `-i`):

```
$ grep Skinner customer1 customer2  
customer1:120387  1240.25   3 Units  Item 023   Skinner Ltd.
customer1:180588   330.87   1 Units  Item 332   Skinner Ltd.
```

If you also wish to display lines containing the word *skinner* in lowercase you enter:

```
$ grep -i skinner customer1 customer2  
customer1:120387  1240.25   3 Units  Item 023   Skinner Ltd.
customer1:180588   330.87   1 Units  Item 332   Skinner Ltd.
customer2:skinner robert, 16 garden hill, london ec3
```

More complicated patterns can be set up with the help of regular expressions, e.g.:

Display all 1994 entries from the file *customer1*; these are lines that contain the number 94 in columns 5 and 6:

```
$ grep '^....94' customer1  
120394 1240.25   3 Units  Item 023   Skinner Ltd.
```
Example 2  Output lines that do not match the specified pattern (option -v):

```
$ grep -v '^1' customer1 customer2
```

customer1:080685 999.98 20 Units Item 038 Johnson Ltd.
customer2:morrow lance, 86 sherwood street, london w1

customer2:skinner robert, 16 garden hill, london ec3

Example 3  Display the number of found lines (option -c):

First display the number of lines that start with 1 for each input file.

```
$ grep -c '^1' customer1 customer2
```

customer1:2
customer2:0

The number of lines that do not start with 1 are to be displayed next.

```
$ grep -c -v '^1' customer1 customer2
```

customer1:1
customer2:2

Example 4  Display file names only (option -l):

The names of files containing lines that begin with a 1 are to be output first.

```
$ grep -l '^1' customer1 customer2
```

customer1

customer2

The names of files containing lines that do not start with 1 are displayed next.

```
$ grep -l -v '^1' customer1 customer2
```

customer1

customer2

Example 5  Display found lines with their line numbers (option -n):

```
$ grep -n -l skinner customer1 customer2
```

customer1:2:120387 1240.25 3 Units Item 023 Skinner Ltd.
customer1:3:180588 330.87 1 Units Item 332 Skinner Ltd.
customer2:2:skinner robert, 16 garden hill, london ec3

See also  ed, egrep, fgrep, sed, stdio [4]
hash

**hash**  **remember or report utility locations**

The POSIX shell built-in `hash` has two functions:

- it can write the contents of the hash table on standard output or enter the specified command in the hash table (Format 1),
- it can delete the contents of the hash table (Format 2).

Each shell maintains its own hash table, in which it enters all commands that are invoked under their basic file names (basenames). Whenever you invoke a command under its basename, the shell first searches the hash table for your command. This accelerates the search process. If the command is not yet in the hash table of the current shell, it is entered there.

When you start a subshell, its hash table is empty. When you terminate the subshell, the parent shell returns with its own hash table, and the hash table of the subshell is deleted.

Syntax

```
Format 1: hash[...name]...
Format 2: hash.-r
```

**Format 1**  **Display or extend hash table**

`hash[...name]...`

**name**

Basename of a command, an executable shell script, or an executable program. The shell searches for this file by examining the contents of the PATH variable. When the file is found, the following information is passed to `hash`:

- the appropriate relative path name in the form `./name` if the current directory is assigned to the PATH variable and contains `name`, or
- the absolute path name.

The `hash` command enters this path name into the hash table. If `name` is not present in any of the directories assigned to the `PATH` variable, an error message is issued.

The following cannot be specified for `name`:

- shell scripts for which you have no execute permission.
- commands with a `name` that includes a slash. This means that you cannot enter absolute or relative path names.
- shell built-ins, since they are subroutines of `sh` and thus do not have names that represent executable files.

By the same token, executable shell scripts and commands are not entered in the hash table unless they are invoked under their basenames.

`name` not specified:
The `hash` command writes the contents of the hash table on the standard output. The output can, for example, be structured as follows:

```
ls=/usr/bin/ls
cat=/usr/bin/cat
chmod=/usr/bin/chmod
```

### Format 2 — Delete hash table

**hash**

**-r** Deletes the contents of the hash table.

If you modify the value of the `PATH` variable, the contents of the hash table are automatically deleted. When you terminate the current shell, the hash table associated with this shell is also deleted.

You should always delete the contents of the hash table in the current shell in the following circumstances:

You have created an executable file in a directory `dira` whose path name is assigned to the `PATH` variable. Some other directory `dirb`, also specified in `PATH`, already contains an identically named command file which already appears in the current hash table. In this case the shell will always execute the older command, even if its directory (`dirb`) comes after the first one (`dira`) in the `PATH` variable.

Deleting the hash table with `hash -r` will cause the shell to execute the first command it finds the next time either of these commands is invoked. In other words, the sequence of the entries in the `PATH` variable will be the determining factor. The path name of the first command found is now entered into the hash table, i.e. your command will always be executed from now on when you invoke it under its basename.

### Error

**name**: not found

This error message may be produced for any of the following reasons:

- **name** is not contained in any of the directories whose respective paths have been entered in the `PATH` variable.

- **name** is actually contained in one of these directories, but is not executable.

- **name** is a shell built-in or a shell function.

### Variable

**PATH**

Search path of the shell
hash

Built-in sh command

Locale

The following environment variables affect the execution of hash:

- **LANG**: Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**: If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**: Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1

Display the contents of the hash table:

```
$ echo $PATH
/bin:/usr/bin:
$ hash
cat=/usr/bin/cat
ls=/usr/bin/ls
```

Example 2

Add a command to the hash table:

```
$ hash cat
$ hash
cat=/usr/bin/cat
```

The command `/usr/bin/cat` is now a new entry in the hash table.
Example 3  The following excerpt from a session demonstrates when the hash table is cleared:

```
$ hash
    cat=/usr/bin/cat
    ls=/usr/bin/ls
$ sh
$ hash
$ cp <file1> <file2>
    ...
    ls
    ...
$ hash
    cp=/usr/bin/cp
    ls=/usr/bin/ls
$ exit
$ hash
    cat=/usr/bin/cat
    ls=/usr/bin/ls
```

The hash table in the new subshell is empty. When the subshell is terminated, the hash table of the parent shell becomes valid again, while that of the subshell is deleted.
**hd**  

**hex dump**

*hd* prints the contents of files in hexadecimal, octal or decimal notation or as character strings. It can also display the position of characters within a string.

**Syntax**

```
hd[-format][...-A][-t][-s offset][wlbk][...-n count][wlbk]][...file]...
```

If no format, offset, and count is specified:

The output is identical to that of *hd -abx -A*, i.e. addresses and bytes are shown in hexadecimal. In addition, all bytes that represent printable characters are shown as such, and non-printing characters are represented by a dot. Addresses are displayed at the beginning of each line, followed by the hexadecimal representation of the bytes in the next few columns and the actual characters (with dots if required) on the right.

Addresses are calculated relative to the start of the file. If no file is specified, the standard input is read; otherwise, the contents of all specified files are listed.

-**format**

Format that determines how individual byte blocks are to be interpreted and output (see "Format description" on page 421).

-**A** (ASCII) All printable characters appear unaltered; non-printing characters are represented by a dot. The characters are shown in the column to the right of the first output format.

-**t** (text) If this option is set, *hd* will ignore all format options that do not affect addresses. Each text line is printed with the address shown at the start of the line. Long lines are folded (split). The notation for control characters (values 0x00 to 0x1f) is a caret followed by the corresponding character (^@ to ^_). Bytes with the high order bit set are shown with a preceding tilde (~), but without the high order bit itself. The caret, tilde and backslash characters are preceded by a backslash in the output. Some special cases are represented by numeric values, e.g. a 7-bit DELETE (127) character as \177, and an 8-bit DELETE (255) as \377, respectively.

-**s offset**[wlbk]

Relative address at which printing of the file is to begin. If you do not specify a file or if you supply the input via a pipeline, a corresponding number of bytes will be skipped. *hd* will abort processing of the current file if you give it an invalid address. The relative address consists of a number, which can be specified in decimal, hexadecimal (preceded by 0x) or octal (preceded by 0), and optionally a flag indicating a unit size directly after the number. The possible flags are:

- **w** 2-byte units (i.e. one word)
- **l** 4-byte units (i.e. one long word)
- **b** 512-byte units (i.e. half of one Kbyte)
- **k** 1024-byte units (i.e. one Kbyte)
You can mark the difference between a hexadecimal number that includes the digit b and the b flag by putting an asterisk (*) between the hexadecimal number and the b flag.

The following examples show how the offset can be specified:
- `s 111` (111 bytes), 
- `s 124l` (496 bytes), 
- `s 0xa*b` (5120 bytes), 
- `s 011k` (9216 bytes).

- `s_offset[*][wlbk]` not specified:
The output begins at the start of the file.

- `n_count[*][wlbk]`
  Number of bytes to be dumped. The count is specified in the same format as the offset, i.e. in decimal, hexadecimal or octal, followed by an optional w, l, b or k flag (see the -s option).

file
  Name of the file to be dumped by hd. More than one file may be named.

file not specified:
  hd reads input lines from standard input.

Format description

A format consists of the following components:
- the byte block option (a, b, c, l or w) and
- the method by which a byte block is to be interpreted in the output, i.e.: hexadecimal (x), decimal (d) or octal (o).

All specified interpretation methods are applied to all specified byte blocks in a format. Format options can be combined and repeated to display addresses, characters, words, etc. in various ways. For example, you could combine `-ax -bx` into `-abx`, or specify `-cxdo` to show all characters in hexadecimal, decimal, and octal.

Byte block options

a  (address) Format option for addresses. Addresses are only interpreted by one method, i.e. in hexadecimal, octal, or decimal. The address is always shown at the start of each line to be displayed, or in the first line of an output block if multiple lines are required for the formats.

b  (byte) Format option for byte.

c  (character) Format option for characters. All printing characters are displayed. C-language escape sequences are output as defined in the language; all other characters are shown in octal, hexadecimal or decimal, depending on the interpretation method.

l  (long word) Format option for 4 bytes.

w  (word) Format option for 2 bytes.
**hd**

*Interpretation methods*

x  (hexadecimal) *hd* interprets addresses or byte blocks as hexadecimal numbers.

d (decimal) *hd* interprets addresses or byte blocks as decimal numbers.

o (octal) *hd* interprets addresses or byte blocks as octal numbers.

Without an interpretation method, but with a byte block option:
The format is interpreted as `-xdo`.

With no byte block option except for addresses:
*hd* uses `-bx` in addition to the specified address format.

Without a byte block option, but with an interpretation method:
The format is interpreted as `-acbwl`.

*format not specified:*
*hd* acts as if *hd* `-abx -A` were specified.

See also  *od*
head  

**copy the first part of files**

`head` copies the opening lines of a file to standard output. If no file is given, `head` reads from standard input.

**Syntax**

**Format 1:**  
```
head[-n_number][..file]
```

**Format 2:**  
```
head[-number][..file]
```

**Format 1**

```
head[-n_number][..file]
```

- `-n_number`
  
  Number of lines to be output. `number` must be a positive decimal number. The space `-` between `-n` and `number` is optional.
  
  `-n_number` not specified:
  
  The first 10 lines are output.

- `file`
  
  Name of the input file. If more than one file is named, the files will be processed in the order in which they are listed, and the output of each file begins with:
  
  ```
  ==>file<<=
  ```

**Format 2**

```
head[-number][..file]
```

- `-number`
  
  Number of lines to be output. `number` must be a positive decimal number.
  
  `-number` not specified:
  
  The first 10 lines are output.

- `file`
  
  Name of the input file. If more than one file is named, the files will be processed in the order in which they are listed, and the output of each file begins with:
  
  ```
  ==>file<<=
  ```

**Locale**

The following environment variables affect the execution of `head`:

- `<LANG>`
  
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `<LC_ALL>`
  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- `<LC_CTYPE>`
  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).
head

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents
of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of
**LC_MESSAGES**.

**Example**
To see the first 5 lines of three files, you enter the *head* command as shown below:

```
$ head -5 file1 file2 file3
```

The first five lines of each of the three files are written to standard output as follows:

```
==>file1===
Lines 1-5 of file 1

==>file2===
Lines 1-5 of file 2

==>file3===
Lines 1-5 of file 3
```

**See also** *cat, more, tail*
iconv  code set conversion

iconv reads input characters from a file or from standard input, converts the encoding of these characters, and writes the results on the standard output.

The conversions that can be performed with iconv are defined in conversion tables located in the directory /usr/lib/iconv.

iconv is typically used to convert characters from the ISO 8859-1 character set to an ISO 646 ASCII variant codeset for a particular language and vice versa (see Examples on page 426).

BS2000  You can use iconv for code conversions between ISO646 and EDF03, that is to say between ASCII 7-bit code and EBCDIC.

Syntax  iconv -f fromcode -t tocode [ file]

-f fromcode  
-t tocode  

iconv expects the conversion table to be in the file /usr/lib/iconv/fromcode.tocode.t. Any character that does not exist in the target codeset is converted to an underscore '_'.

file  Name of the file for which codeset conversion is to be performed.

file not specified:  
iconv reads from standard input.

Error  Not supported xx to yy  
iconv does not support the requested conversion from codeset xx to target codeset yy.

File  /usr/lib/iconv  
Directory containing the standard conversion tables for codeset conversion  
/usr/lib/iconv/iconv_data  
Auxiliary file for iconv  
/usr/lib/iconv/*/t  
Conversion tables

Locale  The following environment variables affect the execution of iconv:

LANG  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
iconv

### LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

### LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments). During translation of the file, this variable is superseded by the use of the `fromcode` option-argument.

### LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

### NLSPATH
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example 1** List all conversion tables:

```
$ ls /usr/lib/iconv
```

```
646.edf03.t 646es.8859.t 8859.646.t 8859.646es.t 8859.edf04.t
646da.8859.t 646fr.8859.t 8859.646da.t 8859.646fr.t edf03.646.t
646de.8859.t 646lt.8859.t 8859.646de.t 8859.646lt.t edf04.8859.t
646en.8859.t 646sv.8859.t 8859.646en.t 8859.646sv.t iconv_data
```

**Example 2** Convert the file named `letter` and save the result in a file named `letter.conv`. The conversion is to be made from the German variant of the ISO 646 codeset (ASCII derivative) to the target codeset ISO 8859-1:

```
$ iconv -f 646de -t 8859 letter > letter.conv
```

**Example 3** You want to convert the contents of the file `bs2000` from ASCII to EBCDIC and write the result to the file `bs2000.conv`:

```
$ iconv -f 646 -t edf03 bs2000 > bs2000.conv
```
id  return user identity

id writes the following on the standard output for the invoking process:
- the user ID (UID)
- the login name
- the group ID (GID)
- the group name.

If the effective and real IDs/names are not identical, both are printed.

Syntax

Format 1: id[-a][...user]
Format 2: id[-G][-n][...user]
Format 3: id[-g][-nr][...user]
Format 4: id[-u][-nr][...user]

Format 1  id[-a][...user]

-a (all) In addition to the ID and login name of the user, id reports all the groups to which
the invoking process belongs and all the groups to which the invoking user belongs.

user
Login name for which the information is output.
If user is specified and the process possesses the relevant access permission then the
user ID and group ID of the selected user are output. In this case it is assumed that the
effective and real IDs are identical. If the database lists more than one permitted group
assignment for the user then these assignments are also output.

user not specified
If the user operand is not specified then id outputs the user and group IDs together with
the corresponding login name and group name of the calling process at the standard
output.

Format 2  id[-G][-n][...user]

-G Only the various group IDs (effective, real and supplementary) are output in the format
"%u\n". If more than one group assignment is present then all group assignments are
output in the format "%u" before the newline character.
-n Outputs the name in the format "%s" instead of the format "%u".

user
see format 1
id

Format 3  
\texttt{id[-g][-nr][-u][-r][user]}
-\texttt{g}  Only the effective group ID is output.
-\texttt{n}  Outputs the name as a string.
-\texttt{r}  Only the real ID is output.
user  
  see format 1

Format 4  
\texttt{id[-u][-nr][-u][-r][user]}
-\texttt{u}  Only the effective user ID is output.
-\texttt{n}  Outputs the name as a string.
-\texttt{r}  Only the real ID is output.
user  
  see format 1

File  /etc/group  
Group file containing group names and the associated group IDs and login names.

Locale  
The following environment variables affect the execution of \texttt{id}:
\texttt{LANG}  
Provide a default value for the internationalization variables that are unset or null. If \texttt{LANG} is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

\texttt{LC_ALL}  
If set to a non-empty string value, override the values of all the other internationalization variables.

\texttt{LC_CTYPE}  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

\texttt{LC_MESSAGES}  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

\texttt{NLSPATH}  
Determine the location of message catalogs for the processing of \texttt{LC_MESSAGES}.  

Example To check your current user ID, group ID, and their corresponding names, you enter:

```
$ id
```

*id* might then report the following:

```
uid=227(USER1) gid=100(USROTHER) groups=100(USROTHER)
```

See also *logname, newgrp, who, getuid()* [4]
info online diagnostic tool

*info* outputs POSIX subsystem characteristics during a POSIX session. In addition to general information relating to the status of the subsystem it supplies selected items of process information relating to the POSIX users in varying degrees of detail.

Ordinary POSIX users are shown information on the general status of the POSIX subsystem (option `-s`), on the POSIX control parameters (option `-t`) and on their own processes. The POSIX administrator can also view information on any user process running on the system.

**Syntax**

```
info [-d] [-h] [-s] [-t] [-p[-f][-u][-g][-p][pid]]
```

- **-d** Produces a dump for the POSIX subsystem
- **-h** Displays help
- **-p** Concise information on all POSIX user processes
- **-s** General information on the status of the POSIX subsystem
- **-t** Lists the POSIX control parameters (see the manual “Basics for Users and System Administrators” [1])
- **-pf** Full information on all POSIX user processes
- **-u user**
  Concise information on all processes belonging to *user*. *user* may be a login name or a user ID number (UID).
- **-g gid**
  Concise information on all processes with the group ID number *gid*.
- **-pf user**
  Full information on all processes belonging to *user*. *user* may be a login name or a user ID number (UID).
- **-pf gid**
  Full information on all processes with the group ID number *gid*.
- **-fp pid**
  Detailed information on the process with the process ID number *pid*. 
Example 1  A POSIX user requesting information about the POSIX subsystem:

```
$ info -s
GLOBAL SYSTEM INFORMATION:
init-TSN: 3BYK
System status: System is activ !
Rootfsname: :POSX:$SYSROOT.FS.ROOT
Number of POSIX users (without 'init' and 'info'):
    TU users          : 6
    TPR users         : 0
    Connected tasks   : 6
```

Example 2  The POSIX administrator requesting information about POSIX user processes:

```
# info -p
PROCESS INFORMATION GENERAL:
USERNAME   PID   PPID   TU/TPR   TSN   SYSCALL
ROOT       18989  1   TU-Task  8RJZ    3
ROOT         1    0   TU-Task  3BYK    0
ROOT         16   1    ------  8RJ5    87
ROOT         23   1    ------  8RKC    3
ROOT       9059  9043    ------  5YQU    54
ROOT      18001  16    ------  9VQK    87
VSX0       7050  7039    ------  5W3F    3
```

**ipcrm**

**remove inter-process communication facilities**

The command `ipcrm` can be used to remove one or more semaphores, message queues, shared memory or inter-process communication facilities (IPC facilities) from the system. These can be specified either by their identifiers or by the key with which each such IPC facility was created.

The identifiers and keys of IPC facilities can be listed with the help of the `ipcs` command (`ID` and `KEY` output columns). For more information on how a message queue, shared memory or semaphore is removed, refer to the `msgctl()`, `shmctl()` and `semctl()` functions in the manual “C Library Functions (BS2000/OSD)” [4].

**Syntax**

```
ipcrm[...option]...
```

**option**

- `--msgqid` (queue) Removes the message queue identified as `msgqid` from the system and destroys the data structures associated with it.
  
  `msgqid`
  
  Identifier of the message queue to be removed. This identifier is displayed by the `ipcs` command in the `ID` column.

- `--msgkey` (queue) Removes the message queue identifier `msgkey` from the system and destroys the data structures associated with it.
  
  `msgkey`
  
  Key of the message queue to be removed. This key is displayed by the `ipcs` command in the `KEY` column.

- `--shmkey` (memory) Removes the shared memory identified as `shmkey` from the system and destroys the data structures associated with it.
  
  `shmkey`
  
  Key with which the shared memory to be removed was created. This key is displayed by the `ipcs` command in the `KEY` column.
**ipcrm**

- **-s** semid
  (semaphore) Removes the semaphore set identified as *semid* from the system and destroys the data structures associated with it.

  **semid**
  Identifier of the semaphore set to be removed. This identifier is displayed by the `ipcs` command in the *ID* column.

- **-S** semkey
  (semaphore) Removes the semaphore set identified as *semkey* from the system and destroys the data structures associated with it.

  **semkey**
  Key with which the semaphore set to be removed was created. This key is displayed by the `ipcs` command in the *KEY* column.

**Example**

Use `ipcs` to first print a report on the status of the inter-process communication facilities and then remove the message queue with the identifier 40 from the system:

```
$ ipcs
IPC status from /dev/kmem as of Mon Mar 9 08:40:41 2009
T ID KEY     MODE OWNER   GROUP
Message Queues:
q  40  0x0000004b -Rrw-rw-rw- user1   usrother
Shared Memory:
Semaphores:
$ ipcrm -q 40
```

**See also**

*ipcs*

`msgctl()`, `msgget()`, `semctl()`, `semget()`, `semop()`, `shmctl()`, `shmget()` [4]
**ipcs**

**inter-process communication status**

`ipcs` prints information about active inter-process communication facilities (IPC facilities).

You can specify options to control

- the type of IPC facilities for which information is to be displayed
- what information is to be shown.

Since the states of IPC facilities may change while `ipcs` is running, the displayed information is only current at the time of the request.

Syntax

```
ipcs[...option]...
```

No option specified

`ipcs` prints information in short format for

- message queues
- shared memory
- semaphores

that are currently active in the system. The meanings of individual output columns are described in the section “Output” on page 436.

**option**

There are options for defining the type of IPC facility and options for defining the type of information.

**Defining the type of IPC facility**

- `-q` (message queues) Prints information about active message queues.
- `-m` (memory) Prints information about active shared memory.
- `-s` (semaphores) Prints information about active semaphores.

If the `-q`, `-m` or `-s` option is set, only information about the corresponding IPC facility is shown. Any combination of these options is possible. If none of the options are specified, information on all types of IPC facilities will be displayed.
Defining the type of information

The following section describes the options that select the type of information to be provided for the IPC facilities specified by the 
-a, -m and -s options. See the “Output” on page 436 for the meaning of the individual output columns.

-a (all) Sets all options that print various types of information. This is a shorthand notation for -b, -c, -o, -p and -t.
-b (biggest) Prints the biggest allowable size for each respective IPC facility:
  - For message queues: the maximum number of bytes in a message to be placed on the queue.
  - For shared memory: the size of the memory segments.
  - For semaphores: the number of semaphores in each set of semaphores.
-c (creator) Prints the user ID (login name) and group name of the creator of the IPC facility.
-o (outstanding) Prints information on outstanding IPC facilities, including:
  - the number of messages on queue
  - the total number of bytes in messages on queue
  - the number of processes attached to a shared memory segment.
-p (process) Prints process number information, including:
  - the process ID of the last process that sent a message
  - the process ID of the last process that received a message
  - the process ID of the process that created a shared memory segment
  - the process ID of the last process that used a shared memory segment (attach, detach).
-t (time) Prints time information. The time of the last control operation that changed the access permissions for all IPC facilities is displayed:
  - For message queues: time when the system call msgrcv() [4] (receive message from queue) or msgsnd() [4] (send message to queue) was last used.
  - For shared memory: time when the system call shmat() [4] or shmdt() [4] was last used.
  - For semaphores: time when the system call semop() [4] was last used on a semaphore.
Output

The column headings and the meanings of the columns in an *ipcs* listing are given below. The letters in parentheses (...) indicate the options which cause the corresponding heading to appear. *all* means that the heading always appears.

**CBYTES (a,o)**
- The number of bytes in messages currently outstanding on the associated message queue.

**CGROUP (a,c)**
- The group name to which the creator of the IPC facility belongs.

**CREATOR (a,c)**
- The user ID (login name) of the creator of the IPC facility entry.

**CTIME (a,t)**
- The time when the associated entry was created or changed.

**GROUP (all)**
- The group name of the group to which the owner of the IPC facility belongs.

**ID (all)**
- The identifier for the IPC facility.

**KEY (all)**
- The key used as an argument when creating the IPC facility with *msgget()* (create message queue) or *semget()* (create semaphore set).

**LRPID (a,p)**
- The process ID of the last process to receive a message from the associated queue.

**LSPID (a,p)**
- The process ID of the last process to send a message to the associated queue.

**MODE (all)**
- The IPC facility access modes and status indicators. The mode consists of 11 characters that are interpreted as follows:
  - The first character is:
    - **S** if a process is waiting on a *msgsnd()*.
    - **D** if the shared memory segment in question was detached. The character disappears when the shared memory segment is detached by the last process that was attached to it.
    - **-** if a process is not waiting on a *msgsnd()*.
The second character is:

R if a process is waiting on a `msgrcv()`
C if the shared memory segment is cleared on executing the first attach.
- if a process is not waiting on a `msgrcv()`.

The next 9 characters are interpreted as three sets of three bits each:
- The first set refers to the access permissions of the owner of the IPC facility.
- The second set represents the access permissions of others belonging to the same user group as the owner of the IPC facility entry.
- The third group refers to the access permissions of all other users.

Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the IPC facility entry, and the last character is currently unused.

The permissions are specified as follows:

r if read permission is granted
w if write permission is granted
a if permission to alter the IPC facility entry is granted
- if the indicated permission is not granted.

NSEMS (a,b)
The number of semaphores in the associated semaphore set.

OTIME (a,t)
The time the last semaphore operation was completed on the associated semaphore set.

OWNER (all)
The user ID (login name) of the owner of the IPC facility entry.

QBYTES (a,b)
The maximum number of bytes allowed in messages outstanding on the associated message queue.

QNUM (a,o)
The number of messages currently outstanding on the associated message queue.

RTIME (a,t)
The time the last message was received from the associated message queue.

STIME (a,t)
The time the last message was sent to the associated message queue.

NATTCH (a,o)
The number of processes attached to the shared memory segment involved.

SEGSZ (a,b)
The size of the shared memory segment involved.
**ipcs**

**CPID (a,p)**
The process ID of the process that created the shared memory segment.

**LPID (a,p)**
The process ID of the last process that used a shared memory segment (attach, detach).

**ATIME (a,t)**
The time at which the last attach on the shared memory segment involved was completed.

**DTIME (a,t)**
The time at which the last detach on a shared memory segment was completed.

**T (all)**
Type of IPC facility:

- *q* stands for message queue
- *m* stands for shared memory
- *s* stands for semaphore.

**File**
*etc/group*

The `/etc/group` file contains a list of all created group names.

**Example**
The `server` program sets up a message queue. You first start this program in the background and then check the status of the inter-process communication facilities with `ipcs`:

```sh
$ server &
$ ipcs
IPC status from /dev/kmem as of Mon Mar 09 08:40:41 2009
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
q     40  0x0000004b -Rrw-rw-rw-  user1   usrother
Shared Memory:
Semaphores:
```

**See also**
`ipcrm`, `msgctl()`, `msgget()`, `semctl()`, `semget()`, `semop()`, `shmctl()` [4]
**Built-in sh command**

**jobs**

display status of jobs in the current session

`jobs` writes to the standard output. It provides information about the specified jobs or, if `job-id` is missing, all active jobs.

You should only use this command if you have used `rlogin` to access the POSIX shell.

**Syntax**

```
jobs[[-l|p][-n][job-id]...]
```

- `-l` Lists process IDs in addition to the normal information.
- `-p` Lists only the process group.
- `-n` Lists only the jobs which have already been completed.

`job-id`

Information about the specified jobs is output. The section `jobs` in the chapter "Entering commands from the POSIX shell" contains a description of the format of `job-id`.

`job-id` not specified:

Information about all active jobs is output.

**Locale**

The following environment variables affect the execution of `jobs`:

`LANG`

Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

`LC_ALL`

If set to a non-empty string value, override the values of all the other internationalization variables.

`LC_CTYPE`

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

`LC_MESSAGES`

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

`NLSPATH`

Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Exit status**

0 if execution is successful

>0 if an error occurs

**See also**

`bg, fg, kill, wait`
**join**

**relational database operator**

`join` compares two files on the basis of relations ("join fields") and joins all pairs of lines with identical join fields. The result is displayed on the standard output.

When `join` is invoked, a join field on which the files are to be compared must be specified for each of the two files. Each field is bounded by a pair of field separators. `join` compares each line in the first file with lines in the second and displays one output line on the standard output for each pair of lines with identical join fields. The output line comprises specific fields from both lines.

### Before the call:

Each input file must be sorted so that the join fields are arranged in the currently valid collating sequence (see `sort`). If the default field separator is used (`join` without the `-t` option), leading separators must be ignored (see `sort` (page 688) option `-b`) when the files are sorted. However, if you invoke `join` with option `-t`, leading field separators must be taken into account when sorting the files (see `sort` (page 688) without option `-b`).

### Syntax

**Format 1:**

```
     join[-a...][-v...][-e...][-o...][-t...][-1...][.-2...][..file1..file2]
```

**Format 2:**

```
     join[-a...][-e...][-j...][-j1...][.-j2...][..file1..file2]
```

The formats are described together since the option `-j.field` in format 2 corresponds to the options `-1.field -2.field` in format 1. `-j1.field` is equivalent to `-1.field` and `-j2.field` is equivalent to `-2.field`.

### No option specified

The first field in a line is the default join field for both files; the default separators are blanks, tabs, and newline characters. Multiple field separators count as one field separator, and leading separators are ignored.

`join` displays one output line on the standard output for each pair of lines with identical join fields. Each output line consists of the following entries in the given order:

- the common field
- the rest of the line from the first file
- the rest of the line from the second file

The default output field separator is a blank.
option

-a..n
(additional output) in addition to the normal output, join outputs all the lines of the \( n \)th input file whose comparison field does not match any comparison field in the other file. You may enter 1 or 2 for \( n \). If you require output for both files enter -a..1 -a..2.

Option -a and option -v must not both be specified.

-v..n
Instead of the standard output, a line is generated for each line in \( n \) for which no match is found. \( n \) may be 1 or 2. If you enter both -v..1 and -v..2 then all the lines for which there is no match are output.

-e..string
(empty output fields) Replaces empty output fields with the specified string.

-\([n]\)..m
The \( m \)th field is specified as the comparison field for the \( n \)th file. You may enter 1 or 2 for \( n \); \( m \) must be a whole number greater than or equal to 1.
If you do not specify the option -j for the other file then the comparison field for this other file is the 1st field.

\( n \) not specified:
The join field for both files is the \( m \)-th field.

-j not specified:
The join field for both files is the first field.

-o..list
(output format) join changes the output line format, so that each output line comprises the individual fields specified in list. The common field is not printed unless you explicitly specify it in list. The list you specify must consist of elements in the form \( n.m \), where \( n \) is either 1 or 2, and \( m \) is greater than or equal to 1. Each element in the form \( n.m \) stands for the \( m \)th field in the \( n \)th file. Individual elements must be delimited by blanks or tabs.

-t..c
Defines character \( c \) as a field separator for both input and output lines. Each occurrence of \( c \) is interpreted as a field separator, i.e.
- two consecutive \( c \) separators designate an empty field, and
- a leading \( c \) is significant and designates an empty first field.

In addition, the newline character acts as a field separator for the input lines. The default field separators (blanks and tabs) are interpreted as field separators only if you specify them as a value for \( c \).
join

-1..field
   Joins the field field from file 1. The fields are decimal whole numbers starting with 1.

-2..field
   Joins the field field from file 2. The fields are decimal whole numbers starting with 1.

file1 file2
   Names of the two files to be joined on the basis of common fields by join.
   If you use a dash (-) as the name for file1, join reads from standard input.
   If the files are not sorted on their join fields, join will not process all lines!
   Problems may arise if a numeric file name (e.g. 1.2) is specified for file1 and the -o option is used immediately before this file name is listed. To avoid such conflicts, a numeric file name should be preceded by a slash (e.g. ./1.2).

Locale
   The following environment variables affect the execution of join:

   LANG
   Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

   LC_ALL
   If set to a non-empty string value, override the values of all the other internationalization variables.

   LC_COLLATE
   Determine the locale for the collating sequence join expects to have been used when the input files were sorted.

   LC_CTYPE
   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

   LC_MESSAGES
   Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

   NLSPATH
   Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1
   In the file place, a place is assigned to a name. In the file amount, an amount and a date are assigned to the same names. Both files are sorted by name. join is to join the two files on the names:

   Contents of place:
   Albert Buffalo
   Hugh Washington
   Irene Philadelphia
Join the two files on the first join field:

```bash
$ join place amount
Albert Buffalo 287.56 20.03.94
Hugh Washington 23.15 25.06.93
Hugh Washington 167.87 16.12.93
Irene Philadelphia 1212.12 12.12.94
Irene Philadelphia 1.98 01.01.94
```

Join the two files and format in columns with `awk`:

```bash
$ join place amount | awk '{printf("%-10s %-15s %-10s %-10s\n",$1,$2,$3,$4)}'
Albert     Buffalo         287.56     20.03.94
Hugh       Washington      23.15      25.06.93
Hugh       Washington      167.87     16.12.93
Irene      Philadelphia    1212.12    12.12.94
Irene      Philadelphia    1.98       01.01.94
```

Example 2 In the file `city`, a name is assigned to a city. In the file `amount` (see Example 1), an amount and a date are assigned to a name. `city` is sorted by cities, `amount` by names. `join` is to join the two files on the names.

Contents of `city`:

```
Buffalo   Albert
Buffalo   Frank
Washington Hugh
New York  Eric
Philadelphia Irene
```

In this example, the join field for `city` is field 2, while that of `amount` is field 1. Before the files are joined, `city` must be sorted on field 2. The output is subsequently formatted into columns with `awk`:

```bash
$ sort -b +1 city | join -j 2 - amount | \     awk '{printf("%-10s %-15s %-10s %-10s\n",$1,$2,$3,$4)}'
Albert     Buffalo         287.56     20.03.94
Hugh       Washington      23.15      25.06.93
Hugh       Washington      167.87     16.12.93
Irene      Philadelphia    1212.12    12.12.94
Irene      Philadelphia    1.98       01.01.94
```

See also `awk`, `comm`, `sort`, `uniq`
kill  terminate or signal processes

The *kill* command sends the specified signal to a set of processes indicated by a process number (PID). The *ps* command can be used to find the process number of any process to which you want to send a signal.

**Syntax**

<table>
<thead>
<tr>
<th>Format 1: kill[-signal]...pid.....</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2: kill.-s_signal...pid.....</td>
</tr>
<tr>
<td>Format 3: kill.-signal.-pgid.....</td>
</tr>
<tr>
<td>Format 4: kill.-l[exit-status]</td>
</tr>
</tbody>
</table>

**Format 1**  Send signals to processes

`kill[-signal]...pid.....`

- **signal**
  
  Signal to be sent to the processes. This signal can be given in the form of a number or a symbolic name. The symbolic signal name is the name as it appears in the header file `<sys/signal.h>`, without the SIG prefix. A list of these names can be printed with the -l option.

  Any signal defined in the header file `<sys/signal.h>` may be specified. The signals below are significant on command level:

<table>
<thead>
<tr>
<th>Signal no.</th>
<th>Symb. name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIGHUP</td>
<td>Halt if the user hangs up</td>
</tr>
<tr>
<td>2</td>
<td>SIGINT</td>
<td>Interrupt via [DEL]</td>
</tr>
<tr>
<td>3</td>
<td>SIGQUIT</td>
<td>Quit</td>
</tr>
<tr>
<td>9</td>
<td>SIGKILL</td>
<td>Kill, terminate process unconditionally</td>
</tr>
<tr>
<td>15</td>
<td>SIGTERM</td>
<td>Software termination, terminate process gracefully</td>
</tr>
</tbody>
</table>

The symbolic name 0 which represents the signal value zero is also recognized.

*signal not specified:*

`kill` sends the signal SIGTERM (15) to the specified processes. This will normally kill processes that do not catch or ignore the signal.
Built-in sh command  

**kill**

**pid**
Number (ID) of the process to which a signal is to be sent.
Users can only send signals to their own processes. The POSIX administrator can send signals to all processes.

Current process IDs can be listed with the `ps` command.

A `pid` value of 0 has a special meaning: the signal is sent to all processes in your process group.

**Format 2  Send signals to processes**

```
kill -s signal pid......
```

- `s..signal`
  Signal to be sent to the processes. This signal can be given in the form of a number or a symbolic name. The symbolic signal name is the name as it appears in the header file `<sys/signal.h>`, without the SIG prefix. A list of these names can be printed with the `-l` option.

**Format 3  Send signals to process groups**

```
kill -signal -pgid......
```

- `signal`
  Signal to be sent to the processes of a process group. This signal can be given in the form of a number or a symbolic name. The symbolic signal name is the name as it appears in the header file `<sys/signal.h>`, without the SIG prefix. A list of these names can be printed with the `-l` option.

- `pgid`
  The signal is sent to all processes with process group ID `pgid`.

Users can only send signals to their own processes.
The POSIX administrator can send signals to all processes.

If you specify a value of 1 for `pgid`, the designated `signal` will be sent to every process whose real user ID matches the effective user ID of the `kill` command. As privileged user you send the signal to all processes, with the exception of a number of system processes.
Format 4  List symbolic signal names

    kill[-l][exit-status]

    -l  A list of all symbolic signal names is printed.

    exit-status
        Signal number or exit status of a process which is terminated by a signal.

Error  no such process
    You specified an invalid value for pid.

    no such process group
    You specified an invalid value for pgid.

File    <sys/signal.h>
    Header file in which the symbolic names of signals are defined.

Locale  The following environment variables affect the execution of kill:

    LANG  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

    LC_ALL  If set to a non-empty string value, override the values of all the other internationalization variables.

    LC_CTYPE  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

    LC_MESSAGES  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

    NLSPATH  Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1  The signal SIGKILL (9) is used to terminate process number 312:

    $ kill -9 312
**Example 2**  The following example outputs the status of a terminated job:

```bash
job
stat=$?
if [ $stat -eq 0 ]
then
    echo job completed successfully.
elif [ $stat -gt 128 ]
then
    echo job terminated by signal SIG$(kill -l $stat).
else
    echo job terminated with error code $stat.
fi
```

In order to prevent the possibility that either a signal number or a process group may be specified in the presence of a start argument which contains a negative number and thereby cause an ambiguity, ISO POSIX-2 DIS requires that the former case is always assumed. Therefore if the default signal is to be sent to a process group (e.g. 123), the application must use a command which resembles the following:

```bash
kill -TERM -123
kill -- -123
```

See also  *ps, trap, kill(), signal()* [4]
**let** integer arithmetic

The built-in shell command *let* provides integer arithmetic. Calculations are performed using *long* arithmetic. Constants are represented in the form `[base#]n` where *base* is an integer between 2 and 36 which specifies the base and *n* is a number accompanying this base. If *base* is not specified, calculations are performed in base ten.

**Syntax**

**Format 1:** `let`...argument....

**Format 2:** `((argument))`....

**Format 1**

`let`...argument

argument

Each argument is an arithmetic expression. The results obtained on evaluation are output.

**Format 2**

`((argument))`

Since many of the arithmetic operators must be quoted for the POSIX shell, an alternative form of the *let* command is provided. For any command which begins with a double left parenthesis `((`, all the characters until a matching double right parenthesis `))` are treated as a quoted expression. Thus `((a=a+b))` is equivalent to *let* "a=a+b".

**Arithmetic evaluation**

An arithmetic expression uses much the same syntax, precedence, and associativity as the C language. All the integral operators, other than `++`, `--`, `?:`, and the comma are supported. Variables can be referenced by name within an arithmetic expression without using the parameter substitution syntax (the `$` character). When a variable is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a variable can be specified as an attribute with the `-i` option of the built-in *typeset* command. Arithmetic evaluation is performed on the value of each assignment to a variable with the `-i` attribute. If you do not specify an arithmetic base, the first assignment to the variable determines the arithmetic base. This base is used when parameter substitution is performed.

**Exit status**

0, if the value of the last expression was not equal to 0
1 otherwise.

**Error**

sh:<expr>: bad number
Incorrect expression
Example The following example illustrates a simple arithmetic operation. Both *let* notations are used.

```
$ let var=10
$ echo $var
10
$ ((var=var-3))
$ echo $var
7
```
**lex**

**generate programs for lexical tasks**

`lex` generates a C program from a file which contains the "lex source text" which you have developed for the problem in hand. A `lex` source text consists of a maximum of three sections: Definitions, rules and user functions. The rules specify the patterns which are searched for in an input text and the action which is taken if a pattern is found. The definitions and user functions are optional.

`lex` generates a file with the name `lex.yy.c`. If `lex.yy.c` is compiled and linked with the Lex library, it copies the input to the output unless a pattern specified in the file is found. In this case the corresponding program text is executed. The pattern which has been matched is located in `yytext[]`, an external character field. Checking and matching of the input file is performed for the search patterns in sequence.

Syntax

```plaintext
lex[.-ctvnV][.-Q[yin]][...file ...]
```

**options**

- `-c` represents the use of C responses and is the default
- `-t` the program is written to the file `lex.yy.c`, not to the standard output
- `-v` provides a two line statistical summary
- `-n` prevents the printout of the summary generated by `-v`
- `-V` outputs version information at the standard error output
- `-Q[yin]` determines whether or not version information is to be output to the output file `lex.yy.c`. `yin` stands for a yes/no argument in whatever language environment is set.
  - In an English-language environment you enter `-Qy` to have version information written to `lex.yy.c` and `-Qn` to suppress the version information. In a German-language environment, for example, you would use `-Qj` or `-Qn` (for ja or nein).
  - By default, no version information is output.

**file** Input file. Multiple files are treated as a single file.

`file` not specified

- If no file is specified the standard input is used.

Some standard table sizes are too small for some users. The table sizes for the automatons which are finally generated can be set in the definition section:

- `%p n` Number of positions is $n$ (default 2500)
- `%n n` Number of statuses (default 500)
- `%e n` Number of nodes on syntax tree is $n$ (1000)
- `%a n` Number of transitions is $n$ (2000)
- `%k n` Number of packed character classes is $n$ (2500)
- `%o n` Size of output field is $n$ (3000)
The use of one or more sizes automatically entails the option `-v` if the option `-n` is not used.

The rules section of `file` starts with the delimiter `%%`. In the rules section you can define local variables for `yylex()`. In the rules section, all lines which start with a space or a tab and precede the first rule are copied to the start of the function `yylex()` directly after the first left-hand parenthesis.

Each rule consists of a regular expression which describes a pattern which is to be located and actions which are to be performed if the pattern is found. Input text which corresponds to no search pattern is passed on unchanged to the input file by `lex`. A regular expression consists of text characters with or without additional operators.

The following operators can be used with `lex`:

- `\x`  
  - `x`
- `"xy"`  
  - `xy`, even if `x` and/or `y` are lex operators (except `\`)
- `[xy]`  
  - `x` or `y`
- `[x-z]`  
  - `x`, `y`, or `z`
- `[^x]`  
  - any character except `x`
- `.`  
  - any character except newline character
- `^x`  
  - `x` at line start
- `<y>x`  
  - `x` if `lex` is in start status `y`
- `x$`  
  - `x` at line end
- `x?`  
  - `x` once or not at all
- `x*`  
  - empty string or multiple occurrences of `x`
- `x+`  
  - one or more occurrences of `x`
- `x{m,n}`  
  - `m` to `n` occurrences of `x`
- `xx|yy`  
  - `xx` or `yy`
- `x |`  
  - the action of `x` is also the action for the next rule
- `(x)`  
  - `x`
- `x/y`  
  - `x` if `y` follows
- `{xx}`  
  - substitution for `xx` from definition section

Special tasks can be performed in the action section of a rule. To this end, `lex` provides the following macros:

- `input()`  
  - another character is read from the input stream
- `unput()`  
  - a character is deferred for a later read process
- `output()`  
  - a character is written to the output stream
You can redefine these macros yourself if you want to control input/output yourself. In this case, ensure that consistency is maintained.

Apart from the storage of detected patterns in `yytext[]` there are other ways of processing detected text patterns using `lex` functions:

- `yymore()`  Newly recognized characters are appended to those which are already present in `yytext[]` (`yytext[]` is normally overwritten with the next character to be found).
- `yyless(n)`  Only the first `n` characters in `yytext[]` are considered.
- `REJECT`  Strings which overlap or which are partially contained in other strings are processed. `REJECT` jumps directly to the next rule without modifying the contents of `yytext[]`.

**Hint**  If a `lex` program is linked with `c89` [5], then `-ll` must be specified as the archive parameter.

**Locale**  The following environment variables affect the execution of `lex`:

- `LANG`  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding default value from the internationalized environment is used. If any of the internationalisation variables contains an invalid setting, the command behaves as if none of the variables have been defined.

- `LC_ALL`  If set to a non-empty value, override the values of all the other internationalisation variables.

- `LC_COLLATE`  Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions. If this variable is not set to the POSIX locale, the results are unspecified.

- `LC_CTYPE`  Determine the locale for the interpretation of sequences of bytes of text data as characters (e.g. single-byte characters as opposed to multi-byte characters in arguments and input files), and the behavior of character classes within regular expressions. If this variable is set to the POSIX locale, the results are unspecified.

- `LC_MESSAGES`  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**See also**  `yacc`
In link files

In creates links to existing files or directories. This allows you to access these files and/or directories under various file names or path names (see Functionality).

To make a link with in, you need write permission for the directory in which the link is to be entered.

There are two types of link:

- Hard link:
  When you create a hard link to a file, there are multiple directory entries for this file, either in one directory or in different directories, but the physical file is only present once. The inode for each file contains a link counter which is decremented by one when a link is removed. The file itself is not deleted until all links to it have been removed.
  A hard link cannot be used to refer to a directory or to files on different file systems.

- Symbolic link:
  A symbolic link is a file that contains a path name. When the shell encounters a file name that represents a symbolic link, it replaces this name with the specified path name. Thus what you access is not the symbolic link, but the file to which the path name points.
  You can set up symbolic links to any files or directories, even to those on different file systems.

Syntax

Format 1: ln[...option]..source..target
Format 2: ln[...option]..source.......targdir
Format 3: ln-..source..target
Format 4: ln-..source.......targdir

Format 1 Make a hard link

In[...option]..source..target

In makes a new link named target to the file named source. The file can then be accessed under both names, i.e. source and target.

No option specified

If there is already a file with the same name as the new link and you do not have write permission for it, ln will display its access permissions and ask you to confirm whether or not the link should be made. The link will not be created unless your answer begins with y (see and target (page 454) and Example 2 on page 460).

Caution!

If the standard input is not a terminal, no confirmation is requested and the link is not made.
option

- **f** (force) If a file named target already exists, `ln` creates the link without asking questions, regardless of whether or not you have write permission for the file.

- **n** If a file named target already exists, its contents are not overwritten. The -f option overrides this option.

source

Name of the file to which you want to make a link. The file must already exist when you call `ln`. You are not allowed to specify a directory as the source.

target

Name of the link that you wish to make to source. target can be a simple file name (basename) or an absolute or relative path name:

- a basename: `ln` enters the basename target in the current working directory.

- an absolute or relative path name in the form prefix/name: `ln` enters the basename name in the directory identified as prefix.

If there is already a file named target and you have write permission for it, `ln` creates the link without asking questions. In other words, the name target now references source, and not the file originally called target; and if target was the only link to (i.e. name of) the original file, the contents of the original file are deleted.

If you do not have write permission for the original target, `ln` asks you if you really want it to make the link (see option -f and Example 2 on page 460).

If the parent directory of target is writable but has the sticky bit set, one of the following conditions must be fulfilled in order to for target to be created:

- the file must be owned by the user
- the directory must be owned by the user
- the user must have write permission for the file
- the user must be a privileged user.

This format of `ln` cannot be used to create links that span different file systems. Such links can be made using the -s option (see Format 3 on page 456 and Format 4 on page 456).
Format 2  **Make a hard link with the same name in another directory**

\[ \text{ln} \ldots \text{source}......\text{targdir} \]

\textit{ln} creates a link to the file (or files) named \textit{source} in another directory named \textit{targdir}. The file can then be accessed in two different directories under the same basic file name (basename).

No option specified

If there is already a file with the same name as the new link to be created, and if you do not have write permission for it, \textit{ln} will display its access permissions and ask you to confirm whether or not the link should be made. The link will not be created unless your answer begins with \textit{y}:

\[ ! \]

\textbf{Caution!}

If the standard input is not a terminal, no confirmation is requested and no link is created.

**option**

- \textbf{-f} (force) If \textit{targdir} already contains a file named \textit{source}, \textit{ln} creates the link without asking questions, regardless of whether or not you have write permission for it.

- \textbf{-n} If a file named \textit{source} already exists in \textit{targdir}, its contents are not overwritten. The \textit{-f} option overrides this option.

**source**

Name of the file to which you want to make a link. The file must already exist when you call \textit{ln}. You are not allowed to make links to directories. You can name any number of files in one call. The specified \textit{source} can be a file basename or an absolute or relative path name:

- a basename:
  \textit{ln} enters the basename \textit{source} in the named \textit{targdir}.

- an absolute or relative path name in the form \textit{prefix/name}:
  \textit{ln} enters the basename \textit{name} in the named \textit{targdir}.

If there is already a file with the basename \textit{source} in \textit{targdir} and you have write permission for it, \textit{ln} creates the link without asking questions. In other words, the link now references \textit{source}, and not the file in \textit{targdir} originally called \textit{source}; and if \textit{source} was the only link to (i.e. name of) the original file, the contents of the original file are deleted.

If you do not have write permission for the original file in \textit{targdir}, \textit{ln} asks you if you really want it to make the link (see option \textit{-f} and Example 2 on page 460).

**targdir**

Name of the directory in which the link is to be entered. The directory must already exist.

This format of \textit{ln} cannot be used to create links that span different file systems.
Format 3  **Make a symbolic link**

`ln -s source target`

*ln* makes a symbolic link named *target* to the named *source*, where *source* can be a file or a directory. The main difference with *ln* is that it creates symbolic links, which can span different file systems. This is not possible with hard links.

**source**
Name of the file or directory to which a symbolic link is to be made. *source* may be specified as any path name and need not exist. It may also reside on a different file system from *target*.

**target**
Name of the symbolic link that you wish to make to *source*. *target* can be a file basename or an absolute or relative path name:

- a basename:
  *ln* enters the basename *target* as a symbolic link in the current working directory.

- an absolute or relative path name in the form *prefix/name*:
  *ln* enters the basename *name* in the directory identified as *prefix*.

If a file named *target* already exists, an error message is returned (see page 458). The existing file is not overwritten.

Format 4  **Make a symbolic link with the same name in another directory**

`ln -s source ...targdir`

For each file or directory that is specified as *source*, *ln* creates a symbolic link in the directory *targdir*. These links can span different file systems.

**source**
Name of the file or directory to which you want to make a symbolic link. You can name any number of sources in one call.

The specified *source* can be an absolute or relative path name:

- an absolute or relative path name in the form *prefix/name*:
  *ln* enters the basename *name* in the directory *targdir* as a symbolic link to *prefix/name*.

If *targdir* already contains a file with the same basename as *name*, an error is returned (see page 458), and the existing file is not overwritten.

**targdir**
Name of the directory in which the symbolic links are to be created. This directory must exist.
Functionality

- **Hard links**

  When `ln` makes a link to a file, the basic file name (basename) associated with the link is entered in the appropriate directory. This entry receives the same inode number as the original file name. Thus, both file names have the same inode, and consequently the same attributes (access permissions, owner, dates, etc.). The physical file referenced by the file names is only present once, but the user can now access the same file under different file or path names (see Example 1 on page 459).

  The inode number indicates whether two file names are linked to the same file (see `ls -i` on page 481); the number of links shows you how many directory entries exist for the file (see `ls -l` on page 481).

  The `rm` command can be used to remove an entry (link) from a directory. If there was more than one link to the file, it can still be accessed under the remaining names. `rm` does not delete the file itself until the last link (i.e. name) is removed.

- **Symbolic links**

  A symbolic link is a file that contains a path name. These path names can be listed with the command `ls -l`. When the shell encounters a file name that represents a symbolic link, it replaces this name with the specified path name. In other words, one path name is mapped to another. There is no link counting mechanism in this case; when a symbolic link is removed, the file containing the path name is deleted.

  The inode of the physical file that is referenced by a symbolic link does not contain any information about the link. The link counter only keeps track of hard links. This means that if you delete the target of a symbolic link, the link will continue to exist, but will now refer to a file having no contents and no inode.

  Symbolic links are not restricted by file system boundaries, and the path names contained in them may refer both to files and to directories.

  Symbolic links can be shown with the `ls` command:

  - `ls -l` indicates which files in the specified directory are symbolic links. The contents of each symbolic link, i.e. the path name of the referenced file, are shown following the file name and the symbol `->`.

  - `ls -L` provides information on the file or directory that is referenced by the symbolic link.

  Operations involving `..` (such as `cd ..`) in a directory that is symbolically linked will reference the original directory, not the target.
Error

- `ln: Cannot link <source> to <target>: Permission denied`
  - `ln` cannot create the named link, since you do not have write permission for the directory in which the link is to be entered.

- `ln: cannot access <file>: no such file or directory`
  - `ln` cannot access the named file, either because it does not exist or because you have no execute permission (x bit) for a directory that appears in the path name of file.

- `ln: <dir> is a directory`
  - You have specified the directory `dir` instead of a file. `ln` only establishes hard links to regular files, special files, or FIFO files, not to directories.

- `ln: are on different file systems`
  - You have tried to create a link to a file and enter it in a different file system. `ln` does not create hard links across file systems; try using the `-s` option.

- `ln: cannot create file`
  - You have tried to use the `-s` option to create a link named file in a directory, but there is already a file under the same name in that directory.

Locale

The following environment variables affect the execution of `ln`:

- `LANG`
  - Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `LC_ALL`
  - If set to a non-empty string value, override the values of all the other internationalization variables.

- `LC_CTYPE`
  - Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- `LC_MESSAGES`
  - Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- `NLSPATH`
  - Determine the location of message catalogs for the processing of `LC_MESSAGES`. 
Example 1  User Pat is the owner of the files frogs and snails, both of which are in the directory 
/HOME/PAT. Mary is to now work with these files as well. To continue working in her own 
directory and save herself the bother of entering long path names, Mary establishes links 
to these files in her own directory /HOME/MARY. To be able to do this, she has to have: 
– write permission for her own directory /HOME/MARY 
– execute permission for Pat's directory /HOME/PAT:

To be able to read and modify the files, Mary also needs read and write permission for the 
files themselves. In our example these conditions are met if Mary is a member of the group 
proj (see below).
The fastest way for Mary to set up the links is to call ln in Format 2:

$$
ln /HOME/PAT/frogs /HOME/PAT/snails /HOME/MARY
$$
The same result can be achieved by calling ln twice in Format 1:

$$
ln /HOME/PAT/frogs /HOME/MARY/frogs 
ln /HOME/PAT/snails /HOME/MARY/snails
$$

As is evident from the following ls calls, ln has entered the file names frogs and snails in the 
directory /HOME/MARY. The number of links to the files has been incremented by 2 (ls -l).
The ls -i calls show that the file names frogs and snails have the same inode numbers in both 
directories.

$$
ls -l /HOME/MARY
total 6
-rw-rw----  2 PAT  proj  34 Mar 09 15:08 frogs
-rw-rw----  2 PAT  proj 1217 Mar 09 18:59 snails
$$

$$
ls -i /HOME/PAT/frogs /HOME/MARY/frogs
16435 /HOME/PAT/frogs
16435 /HOME/MARY/frogs
$$

$$
ls -i /HOME/PAT/snails /HOME/MARY/snails
4766 /HOME/PAT/snails
4766 /HOME/MARY/snails
$$

The ls -l call above shows that Pat is still the owner of the files. Pat can at any time cancel 
Mary's right to use the files, e.g. by saying:

$$
chmod 600 frogs snails
$$

If Mary wants to create links with different names, she has to call ln twice in Format 1.

$$
ln /HOME/PAT/frogs /HOME/MARY/sugar
ln /HOME/PAT/snails /HOME/MARY/spice
$$
Example 2  The following examples shows you what happens if you call `ln` in Format 1 when there is already a file with the chosen link name:

In the current working directory there are three file names: `letter`, `list` and `text`. `letter` is a link to file1, `list` and `text` are links to file2. You do not have write permission for file2:

```
$ ls -l
 total 3
-rw-r--r--   1  BRIAN    other    57  Jul 17 14:29 letter
-r--r--r--   2  NEIL     other    103  Jul 16 15:30 list
-r--r--r--   2  NEIL     other    103  Jul 16 15:30 text
```

You now enter:

```
$ ln letter list
ln: list: 444 mode (y/n) ?n
```

You answered `n`, so `ln` does not make the link.

If you have write permission for file2, `ln` makes the link without asking questions:

```
$ ls -l
 total 3
-rw-r--r--   1  BRIAN    other    57  Jul 17 14:29 letter
-rw-rw-rw-   2  NEIL     other    103  Jul 16 15:30 list
-rw-rw-rw-   2  NEIL     other    103  Jul 16 15:30 text
```

```
$ ln letter list
```

```
$ ls -l
 total 3
-rw-r--r--   2  BRIAN    other    57  Jul 17 14:29 letter
-rw-r--r--   2  BRIAN    other    57  Jul 17 14:29 list
-rw-rw-rw-   1  NEIL     other    103  Jul 16 15:30 text
```

As `ln` has created a link named `list` for `letter` (file1), `list` is now a link to file1, and no longer to file2. `text` is now the only link to file2.

```
$ ln letter text
$ ls -l
 total 3
-rw-r--r--   2  BRIAN    other    57  Jul 17 14:29 letter
-rw-r--r--   2  BRIAN    other    57  Jul 17 14:29 list
-rw-rw-rw-   1  NEIL     other    103  Jul 16 15:30 text
```

Now `text` is also a link to file1; file2 no longer exists.
Example 3  User Norbert needs a quick and simple way to access the directory of his colleague Andrea (/HOME2/ANDREA) from his own directory (/HOME/NORBERT). He therefore creates the following symbolic link by calling `ln` in Format 3:

```bash
$ ln -s /HOME2/ANDREA /HOME/NORBERT/andr
```

Norbert can now use the symbolic link `andr` to directly access Andrea’s directory from his own directory, even though her directory resides on another file system:

```bash
$ ls -lL andr
```

```
total 16
drwxr-xr-x 2 ANDREA usrother 2560 Feb 27 13:20 PASCAL
drwxr-xr-x 2 ANDREA usrother 2048 Mar  5 17:32 COURSES
drwxr-xr-x 2 ANDREA usrother 512 Mar  5 12:07 LETTERS
-rw-r-xr-x 1 ANDREA usrother 148 Mar  5 17:10 test
```

Example 4  User Norbert has created the directories `letters89`, `letters90` and `letters91` in his home directory `/HOME/NORBERT`:

```bash
$ ls /HOME/NORBERT
letters 89  letters 90  letters 91
```

He now wants direct access from `/HOME/NORBERT/letters91` to the other two directories. To do this, he creates two symbolic links by calling `ln` in Format 4:

```bash
$ cd /HOME/NORBERT
$ ln -s /HOME/NORBERT/letters 89 /HOME/NORBERT/letters90 letters91
$ cd letters91
$ ls -og letters??
```

```
lrwxrwxrwx 1 25 Mar  6 12:42 letters89 -> /HOME/NORBERT/letters89
lrwxrwxrwx 1 25 Mar  6 12:42 letters90 -> /HOME/NORBERT/letters90
```

```bash
$ ls letters90
```

```
offers  letter.henry  letter.nigel  test
```

See also  `chmod`, `cp`, `ls`, `mv`, `rm`

`link()`, `readlink()`, `stat()`, `symlink()` [4]
locale  get locale-specific information

The `locale` command writes information about the current locale or other public locales to the standard output. In the following section, a public environment is an environment provided by the implementation that the application can access.

If `locale` is called without any arguments, it summarizes the current locale for each environment category as determined by the settings of the environment variables.

If the command is called with operands, it outputs values assigned to the keywords in the environment categories as follows:

- If a keyword name is specified, the keyword as well as the category containing the keyword is output.
- If a category name is specified, the category as well as all the keywords contained in it are output.

Syntax

| Format 1:(locale[...]|a|-m] | Format 2:locale[...|c|k]... |

options

- `-a` Outputs information about all available public locales. The locales available include POSIX.
- `-c` Outputs the selected environment categories (see “Standard output (stdout)” on page 463).
  The `-c` option improves legibility if several categories are selected (e.g. using several keyword names or a category name). This option is valid with or without the specifica-
  tion of the `-k` option.
- `-k` Outputs the names and values of the selected keywords.
- `-m` Outputs the character map name.

name

The name of the environment category (see section “localedef define local environment” on page 466), the name of a keyword in an environment category or the reserved name `charmap`. The specified category or the specified keyword can be specified as `name` operands in any order.

Locale

The following environment variables have an effect on the execution of `locale`:

- `LANG` Specifies a default value for the locale variable that is unset or null. If `LANG` is unset or null, the corresponding locale default value is used. If the locale variable contains an invalid setting, locale behaves as if no variables had been set.
### locale

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LC_ALL</strong></td>
<td>If set to a non-empty string value, override the values of all the other internationalization variables.</td>
</tr>
<tr>
<td><strong>LC_CTYPE</strong></td>
<td>Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).</td>
</tr>
<tr>
<td><strong>LC_MESSAGES</strong></td>
<td>Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.</td>
</tr>
<tr>
<td><strong>NLSPATH</strong></td>
<td>Determine the location of message catalogs for the processing of <strong>LC_MESSAGES</strong>.</td>
</tr>
</tbody>
</table>

#### File

**Standard output (stdout)**

If `locale` is called without options or operands, the names and values of the `LANG` and `LC_*` environment variables are written to the standard output. One line is used for every variable, `LANG` is output first. Only variables set in the environment and not overwritten by `LC_ALL` are output in the following format:

'\%s=%s\n', variable_name, value

The names of the `LC_*` variables not set in the environment or overwritten by the `LC_ALL` variable are output in the following format:

'\%s="\%s"\n', variable_name, implied_value

*implied_value* is the name of the locale selected by the implementation for this category based on the values in `LANG` and `LC_ALL`.

*value* and *implied_value* are enclosed in single quotes so they can be reused as an entry some time in the future.

*value* is not enclosed in double quotes (to differentiate it from *implied_value*, which is always enclosed in double quotes).

The `LC_ALL` variable is output last, using the first format outlined above. If this variable is not set, it is output in the following format:

'LC_ALL="\n"
The following points apply when specifying arguments:

1. If the `-a` option is set, the names of all the public locales are output in the following format:
   
   `"%s\n". locale_name`

2. If the `-c` option is specified, the names of all the selected categories are output in the following format:
   
   `"%s\n". category_name`
   
   If keywords were also selected, these are output immediately after the category to which they belong.
   
   If the `-c` option is not specified, only the keywords and not the category names are output.

3. If the `-k` option is specified, the names and values of the selected keywords are output. If the value is non-numeric, it is output in the following format:
   
   `"%s="\n", keyword_name, keyword_value`
   
   For the `charmap` keyword, the name of the character map is output if a character map was specified using the `localedef -f` option during the creation of the locale.
   
   `charmap` is used as `keyword_name` here.

   Numeric values are output in one of the following formats:
   
   `"%s=%d\n". keyword_name, keyword_value`
   
   `"%s=%c%o\n". keyword_name, escape_character, keyword_value`
   
   `"%s=%cx%x\n". keyword_name, escape_character, keyword_value`
   
   `escape_character` is the character defined by the `escape_char` keyword in the current locale (see XBD specification, Section 5.3, Locale Definition [14]).

   Keyword values that are combined (list entries) are separated by semicolons in the output. If the keyword values contain semicolons, double quotes, backslashes, and/or control characters, these characters are escaped using a preceding escape character.

4. If the `-k` option is not specified, the values of all the keywords are output in the following format:
   
   `"%s\n". keyword_value`
   
   For the `charmap` keyword, the name of the character map is output (if available).

5. If the `-n` option is specified, a list of all the available character maps is output in the following format:
   
   `"%s\n". charmap`
   
   The output can be used as an argument for the `localedef -f` option.
locale

Exit status

0 All the information requested was found and output.

>0 An error occurred.

Example

The examples are based on the following locales:

LANG=locale_x
LC_COLLATE=locale_Y

The `locale` command would have the following output:

LANG=locale_x
LC_CTYPE="locale_x"
LC_COLLATE=locale_Y
LC_TIME="locale_x"
LC_NUMERIC="locale_x"
LC_MONETARY="locale_x"
LC_MESSAGES="locale_x"
LC_ALL=

The `LC_ALL=POSIX` locale `-ck decimal_point` command would create the following output:

LC_NUMERIC
decimal_point="."

The following command shows a `locale` application that can be used to determine whether a response entered by a user is a yes-response (affirmation):

```bash
if printf "%s
" "$response" | grep -Eq "$(locale yesexpr)"
then
    affirmative processing goes here
else
    non-affirmative processing goes here
fi
```

See also

`localedef`

XBD specification, Section 5.3, Locale Definition [14]
localedef

localedef define local environment

The localedef command converts source definitions for locales into a format that can be used by the functions and commands whose operational behavior is determined by the setting of the locale variables. This format is defined in the XBD specification, Chapter 5, Locale [14].

Every POSIX system user has the facility to create new locales locally. However, only a user with the appropriate privileges (usually system administrators) can install these locales on the system so that they can be used by functions, applications, and commands. This is done either by copying the files created locally to /usr/lib/locale/name and /usr/lib/charmap/charmap.

localedef reads source definitions for one or more of the categories in the locales, which belong to the same locale, from the file specified with the -i option (if specified), or from the standard input.

The name operand indicates the target environment. The command supports the creation of locales which are "public" (i.e. can be accessed generally), as well as locales which are private (i.e. with restricted access rights). On POSIX systems, only users with the appropriate privileges can create or alter general access locales.

Every category source definition is indicated by the name of the accompanying environment variable and is terminated with an END category-name instruction. The following categories are supported:

- **LC_CTYPE** Defines character classification and case conversion.
- **LC_CCollATE** Defines collation rules.
- **LC_MONETARY** Defines the format and symbols used for the formatting of monetary information.
- **LC_NUMERIC** Defines the decimal point, the thousands separator and radix character and the grouping symbol for the editing of non-monetary numeric information.
- **LC_TIME** Defines the format and contents of date and time specifications.
- **LC_MESSAGES** Defines the format and values of yes/no responses (affirmations and negations).
localedef

Syntax

localedef[\[-c\]][\[-f\..charmap\]][\[-i\..sourcefile\]]..name

options

\-c  Creates an output file, even if warning message is output.

\-f..charmap

  Specifies the pathname of a file which contains an assignment of the symbolic
  character symbols and collating element symbols to the actual character coding
  (character map). The charmap format is described in [14]. This option must be specified
  if symbolic names are used that were not defined using the collating-symbol key word.

\-f not specified:

  The character assignment defined in the /usr/lib/charmap/posix file (ISO 8859-1) is used.

\-i..sourcefile

  The pathname of a file which contains the source definitions.

\-i not specified:

  The source definitions of the default input are read. The format of the input file is
  described in [14].

name

  Specifies the locale. The use of this name is described in [14].

  \- If name contains one or more slashes, name is interpreted as a pathname, in which
     the definitions created for the locales are saved.

  \- If name does not contain a slash, the locale is private, and is generated in the current
     directory.

Because only one name can be specified, the only categories that can be processed in
a call are those that belong to the same locale.

Locale

The following environment variables have an effect on the execution of localedef:

\textbf{LANG}

  Provide a default value for the internationalisation variables that are unset
  or null. If LANG is unset or null, the corresponding locale default value is
  used. If any of the internationalisation variables contains an invalid setting,
  the utility will behave as if none of the variables had been defined.

\textbf{LC\_ALL}

  If set to a non-empty string value, override the vaules of all the other inter-
  nationalisation variables.

\textbf{LC\_COLLATE}

  (This variable has no affect on localedef, the POSIX locale will be used for
  this category.)
localedef

**LC_CTYPE**  
Determines the locale for the interpretation of sequences of bytes of text data as characters (e.g. single-byte as opposed to multibyte characters in arguments and input files). This variable has no affect on the processing of `localedef` input data; the POSIX locale is used for this purpose, regardless of the value of this variable.

**LC_MESSAGES**  
Determines the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  
Determines the location of the message catalogs for the processing of `LC_MESSAGES`.

**File**

**Standard input**

If the `-i` option is not specified, the standard input must be a text file containing one or more source definitions for categories for locales, as described in the XBD specification, Section 5.3, Locale Definition [14]. If lines are continued using the escape mechanism, there is no length restriction for the entire input line.

**Input files**

The file containing the description of a character set (charmap argument in `-f`) is described in [14]. If a source definition of a category for an locale contains the copy instruction (as described in [14]) and this instruction specifies a valid locale, `localedef` behaves as if the source definition contained a valid source definition of the category for the locale specified.

**Output files**

The format of the created output corresponds to the format of the internationalization database on POSIX systems:

*name* (if slashes are contained) or *name* (without slashes) is the pathname of a directory containing the following newly generated files:

- `LC_COLLATE`: File with collating information
- `LC_CTYPE`: File with information on the character type (ctype)
- `LC_MONETARY`: File with monetary information
- `LC_NUMERIC`: File with numeric information
- `LC_TIME`: File with date and time information
- `LC_MESSAGES`: Directory with message information
- `LC_MESSAGES/Xopen_info`: File with yes/no information
- `loc_info`: File with information on the name and path of the character map
localedef

A copy of the character map is also created in /usr/lib/charmap, if the -f option is used.

Exit status

The following exit values are transferred:

0  No errors occurred and the locales were successfully created.
1  Warnings were output, but the locales were created successfully nonetheless.
2  The locale specification exceeded implementation limits or the coded character set or sets used were not supported by the implementation. No locale was created.
4  Warnings occurred and no output was created.

Error

If an error is detected in the input file, no output file is created.
If warnings occur, an output file is created provided that the -c option was specified. The following conditions will cause warning messages to be output:

– A symbolic name not contained in the charmap file was used for the description of the LC_CTYPE or LC_COLLATE categories (for other categories this causes an error condition).
– The number of operands for the order keyword exceeds the {COLL_WEIGHTS_MAX} limit.
– The source contains optional keywords which are not supported by the implementation.

Hint

The character map definition is optional, and is not contained in the locale definition. This allows both completely self-defined source files and generic sources (applicable to more than one coded character set). To aid portability all character map definitions must have the same symbolic name for portable character sets. As described in [14], it depends on the relevant implementation whether users or applications can use additional character set description files. Therefore, the -f option might only operate correctly if a charmap supported by the implementation is specified.

See also
locale
X/Open CAE Specification. System Interface Definitions [14]
logger  log messages

The logger utility allows logging of information in the file /var/adm/logger for later use by the POSIX administrator. The locations of the saved messages, their format and retention period are all unspecified.

This utility is useful for determining why non-interactive utilities have failed.

Syntax

```
logger_string
```

string

One of the string arguments whose contents are concatenated together, in the order specified, separated by single space characters.

string appears in the output (see Example below).

Locale

The following environment variables affect the execution of logger:

- **LANG**
  Provide a default value for the internationalisation variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation varibles contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalisation variables.

- **LC_CTYPE**
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error. (This means diagnostics from logger to the user or application, not diagnostic messages that the user is sending to the system administrator.)

- **NLSPATH**
  Determine the location of message catalogs for the processing of LC_MESSAGES.

Example

A batch application, running non-interactively, tries to read a configuration file and fails; it may attempt to notify the POSIX administrator with:

```
logger myname: unable to read file foo. [timestamp]
```

See also  mailx, write
logfile

logfile return user’s login name

logfile writes the user’s login name on the standard output.

Syntax

logfile

File

/etc/profile

File that is evaluated by each login shell. It is used for setting a shell environment, depending on the contents of the LOGNAME (login name of the user) environment variable passed by login. This file is created by the POSIX administrator.

Locale

The following environment variables affect the execution of logfile:

LANG

Provide a default value for the internationalisation variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalisation variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL

If set to a non-empty string value, override the values of all the other internationalisation variables.

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH

Determine the location of message catalogs for the processing of LC_MESSAGES.

Example

You log in under the login name POSUSER1 and then display this login name.

$ logfile

POSUSER1

See also

env, id, who
**lp**

**send files to a printer**

With *lp* you can:

- have files printed on a printer (print job)
- select the RSO printer or printer group on which a print job is to be run
- have a header page printed
- modify printer options for print jobs

*lp* associates a print job with each named file. If no file is specified, the data to be printed is expected from the standard input. The files are printed in the order in which they are listed. *lp* assigns a unique job number (request ID) to each print job and writes this number on the standard output. This number can be used later to cancel or abort a print job or to have status information on the print job displayed.

Currently printing jobs and jobs which have already been submitted, but not yet started, can be aborted or cancelled with *cancel*.

Information on the status of print jobs can be displayed with *lpstat*.

**Syntax**

```
lp[...-c][...-d..dest][...-n..copies][...-s][...-o..par] [...-t..title][...file...]
```

No option specified

The specified files are printed on any free printer in the default printer group (normally ALL).

option

- **-c** (c - copy) *lp* makes copies of the specified files when invoked and then prints these copies. The -c option is always implicitly enabled.

- **-d..dest**
  
  Chooses *dest* as the RSO printer or printer group that is to do the printing. If the printer group includes more than one printer, the job (or request) will be printed on the first available printer in the group.

  *dest* can be given as the name of any RSO printer or printer group.

  The names of legal RSO printer or printer groups can be listed with the command *lpstat* -a.

- **-d..dest** not specified:
  
  If the environment variable *LPDEST* is set, its value is used. If *LPDEST* is undefined, the print job is printed on a printer in the default printer group. This is either printer group ALL or the user/terminal-specific printer group.
-n copies
This option defines how many copies of the files are to be printed. copies can be any integer that falls within an effective range from 1 to 99.

-n copies not specified:
copies defaults to 1.

-o par
par can be used to set any or all of the print parameters line-spacing, control-char-pos and control-mode or selects the print parameters for the output of PostScript files.
If these parameters are to be used in combination, the -o par option must be specified for each parameter.
The parameters can have the following values:
line-spacing = 1 | 2 | 3 | *s[td] | *e[bcdf] | *a[sa] | *i[bi]
control-char-pos = *std | <int 1...2040>
postscript | ps

If the print parameters line-spacing, control-char-pos or control-mode are specified, the values are passed through to the corresponding operands of the BS2000 PRINT-DOCUMENT command (DOCUMENT-FORMAT), siehe Handbuch „Commands“ [10].
If there are control characters in the file being printed (such as \f for form feed), they will be interpreted only if both of the following conditions are met:
– the file is being printed on an RSO printer
– the control-mode parameter is set to *line-mode or *physical
If control-mode is not specified, any control characters in the file will be represented as non-printing characters (smudge characters).

When the postscript (or ps) parameter is specified, the BS2000 print parameters for outputting PostScript files are set. The other printer parameters are then not meaningful. The printer selected must be able to print PostScript files.

-s (s - silent) Suppresses the message "request id is <request-id>" from the lp command.

-t title
(t - title) Prints the specified title on an additional header (banner) page.
file | -
Name of the file to be printed. More than one file may be specified. If you name a number of files, they are printed in the order in which you list them.
If you use a dash (-) as the name for file, \texttt{lp} reads from standard input and creates a temporary file in the /var/spool/lp/temp directory.

For each file to be printed, \texttt{lp} issues a message in the form:

\begin{verbatim}
request id is <request-id> (<basename>)
<request-id> is output in the format “TSN-nnnn”, where nnnn is the BS2000 TSN.
<basename> is the basename of file, i.e. the part of the path name which comes after the last slash (“/”).
\end{verbatim}

When \texttt{lp} reads from standard input, the message is:

\begin{verbatim}
request id is printer group-id (standard input)
\end{verbatim}

\texttt{file} not specified:
\texttt{lp} reads from standard input.

\textbf{Error}
\begin{verbatim}
lp: can't access the file "file"
lp: Make sure file names are valid
lp: No (or empty) input files.
You do not have read permission for file, or file does not exist. The print job for file is rejected.
lp: standard input is empty
lp: request not accepted
You omitted the file name or used a dash (-) instead, but you have not entered anything from standard input. The job has consequently been rejected.
lp: Requests for destination 'XYZ' aren't being accepted.
lp: Use the 'lpstat -a' command to see why this destination is not accepting requests.
The printer group XYZ that you specified for dest when calling \texttt{lp} with the \texttt{-d} option is not a valid RSO printer or printer group.
It may also be that the environment variable \texttt{LPDEST} is assigned the value XYZ. Unless explicitly specified otherwise, this value will then be substituted for \texttt{dest} for all \texttt{lp} calls.
lp: unrecognized option "-x"
An invalid option, \texttt{-x} in this case, has been used.
\end{verbatim}

\textbf{File}
\begin{verbatim}
/var/spool/lp/temp/*
Temporary file \texttt{lp} creates when reading from standard input.
\end{verbatim}
Variable

*LPDEST*
Name of a printer group (class of printers).
-`-d...LPDEST` is the default when `-d...dest` is not specified.

Locale

The following environment variables affect the execution of `lp`:

*LANG*  
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

*LC_ALL*  
If set to a non-empty string value, override the values of all the other internationalization variables.

*LC_CTYPE*  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

*LC_MESSAGES*  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

*LC_TIME*  
Determine the format and contents of date and time strings displayed in the `lp` banner page, if any.

*NLS_PATH*  
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example 1  
Three files, `t1`, `t2` and `t3`, are to be printed. Three copies each of files `t2` and `t3` are required.

$ lp t1 -n 3 t2 t3

request id is TSN-153D (t1)
request id is TSN-153E (t2)
request id is TSN-1540 (t3)

`lp` issues a message for each file to be printed.
Example 2  Two files, \textit{t1} and \textit{t2}, are to be printed. Two printer groups, G001 and G002, are defined. The value of the environment variable \textit{LPDEST} is G001. \textit{t1} is to be printed on printer group G001; \textit{t2} on G002:
\begin{verbatim}
$ lp t1
request id is TSN-234A (t1)
$ lp -dG002 t2
request id is TSN-234C (t2)
\end{verbatim}
Since the environment variable \textit{LPDEST} is set to G001, the printer group need not be specified for \textit{t1}. If \textit{LPDEST} were undefined or assigned some value other than G001, the command would be written as follows:
\begin{verbatim}
$ lp -dG001 t1
$ lp -dG002 t2
\end{verbatim}

See also \textit{cancel}, \textit{lpstat}
lpstat

report line printer status information

lpstat outputs information about the current status of all requests submitted with lp.

Nonprivileged users are shown information relating only to print jobs submitted under their own login names.
Privileged users ($TSOS) are shown information relating to all print jobs.

Currently printing jobs and jobs which have already been submitted can be aborted or canceled with cancel.

Syntax

lpstat[-drst][-a[<list>][:...]][-c[<list>][:...]][-o[<list>][:...]][-p[<list>][:...]][-u[<list>][:...]][...]

No option specified
lpstat prints the following information of all the user's print requests made by lp on standard output:
- the job number assigned by lp when invoked,
- the login name of the job originator,
- the size of the file to be printed (in bytes),
- if a job is currently printing, the name of the RSO printer on which the job is being running.

option

list can be a list of items separated from one another by a comma, or a quoted list of items separated from one another by a comma or one or more blank characters, or combination of both.

-a[<list>]
Write the acceptance status of destinations for output request.
The list argument is a list of intermixed RSO printer names and class names.

-c[<list>]
Write the class names and their members.
The list argument is a list of class names.

-d Write the system default destination for output request in the following format:

    system default destination: *CENTRAL

-o[<list>]
Write the status of output requests.
The list argument is a list of intermixed RSO printer names, class names and request IDs.

-p[<list>]
Write the status of printers.
The list argument is a list of RSO printer names.

-r Write the status of the line printer request scheduler: scheduler is running
-s  Write a status summary, including the status of the line printer scheduler, the system
default destination, a list of class names and their members:

  scheduler is running
  system default destination: *CENTRAL

-t  Write all status information (see Example 2 on page 479).

-u[1st]
  Write the status of output requests for users.
  The list argument is a list of login names.
  As a nonprivileged user you are only allowed to specify your own login name.

ID  A request ID, as returned by lp.

Variable  TZ
  Defines the time zone within date and time specifications.

Locale  The following environment variables affect the execution of lpstat:

  LANG  Provide a default value for the internationalization variables that are unset
         or null. If LANG is unset or null, the corresponding value from the implemen-
         tation-specific default locale will be used. If any of the internationalization
         variables contains an invalid setting, the utility will behave as if none of the
         variables had been defined.

  LC_ALL  If set to a non-empty string value, override the values of all the other inter-
           nationalization variables.

  LC_CTYPE  Determine the locale for the interpretation of sequences of bytes of text data
            as characters (for example, single- as opposed to multi-byte characters in
            arguments).

  LC_MESSAGES  Determine the locale that should be used to affect the format and contents
                of diagnostic messages written to standard error and informative messages
                written to standard output.

  LC_TIME  Determine the format of date and time strings output when displaying line
           printer status information with the -a, -o, -p, -t or -u options.

  NLSPATH  Determine the location of message catalogs for the processing of
           LC_MESSAGES.
Example 1  Information of all submitted and currently running printing jobs:

```
$ lpstat
TSN-6XDZ QM212JNA 12837 *ACT LE       *CENTRAL HP
```

Example 2  Output of all status information

```
$ lpstat -t
scheduler is running
system default destination: *CENTRAL
DROS336  accepting requests
DRS03   accepting requests
DR9001  accepting requests
DR9022  accepting requests
EMDRS001 accepting requests
:
TC12G002 accepting requests
DROS336  9001RP  idle
DRS03   9022RP  idle
EMDRS003  9022RP  stopped
EMDRS016  9022RP  idle
LB      HP-PRINT printing
LC      HP-PRINT printing
LD      HP-PRINT printing
LE      HP-PRINT printing
EMDRS05  9001RP  idle
:
TSN-6XK4 QM212JNA 329853 *ACT LE       *CENTRAL HP
```

See also  `lp, cancel` [11]
Is

list directory contents

Is is used to obtain detailed information on files and directories. The type, scope and output format of the displayed information can be specified by setting various options.

Syntax

Is[...option]...[...file]...

No argument specified

Is lists the names of files and directories in the current working directory. The output is sorted in alphabetical order.

No option specified

file is a directory:
Is lists the names of the files and directories in it using a single-column output format. The output is sorted in alphabetical order. The only files not listed are those which begin with a dot, such as .profile.

file is a file:
Is displays the name of the file. You can use a partially qualified file name. If an asterisk (*) is specified for file, all files and directories (and their contents) are listed.

option

-a Lists all file names, including those that begin with a dot.

-b Non-printing characters in file names are represented in the octal notation \ddd, where ddd is the octal value of the character (see Tables and directories, EBCDIC character set).

-c Together with option -l
Lists time of last modification of the inode (file created, mode change, etc.) instead of the time at which the file was last modified.

Together with option -r:
Is uses the time of last modification of the inode as the sort criterion instead of the time of last modification of the file.

-C Multi-column output with entries sorted down the columns in alphabetical order. The default is one entry per line of output.

Is uses the environment variable COLUMNS to determine the number of character positions per line. If this variable is not set, the value of the environment variable TERM is checked in the terminfo database in order to determine the number of columns. If this information cannot be obtained, 80 characters per line are assumed.

-d If file is a directory, Is lists its name (not its contents). This option is often used together with -l in order to obtain information on the status of the directory. If file is omitted, Is outputs a period for the current directory.
-f Forces each argument to be interpreted as a directory and lists the entries of each directory in the order in which they appear. If no argument is specified, the entries in the current directory are listed. This option turns on the -a option and turns off options -l, -i, -F, -s and -r.

-F Puts a slash (/) after each file name if the file is a directory, an asterisk (*) if the file is declared as an executable, a commercial at symbol (@) if the file is a symbolic link, a vertical bar | if it is a FIFO file.

-g Has the same effect as -l, except that the owner is not printed.

-i Prints the unique identification number (inode number) for each file in the first column of the report.

-l A list of detailed information is provided in long format for each file.

This information is displayed as shown below:

```
-rwxrwxrwx n name group nnnn Mon nn:nn file
  identification
  access permission
  number of links
  login of owner
  group name
  size in bytes
  day
  month
  time
```

**Identification**

- d for a directory
- l for a symbolic link
- b for a block special file
- c for a character special file
- m for a shared data (memory) file
- p for a FIFO file (named pipe)
- s for a semaphore
- - for an ordinary file
Access permissions

3 sets of 3 characters each provide information on the permissions of
– the file owner (characters 1 to 3),
– users belonging to the same group (characters 4 to 6),
– other users (characters 7 to 9).

Each set uses the following codes:

Position 1: 
- r for read permission or - for no read permission.

Position 2: 
- w for write permission or - for no write permission.

Position 3: 
- x for execute permission
  - s for execute permission with s bit set
  - t for execute permission with t bit (sticky bit) set
  - T for execute permission with t bit (sticky bit) set
  - S for the I bit (set-user-ID bit set, execute permission not set)
  - - for no execute permission with no special bit set

If the s bit (set-user-ID mode) is set for the owner or the group and the corresponding x bit (execute permission) is also set, the x will be replaced by an s. The s bit cannot be set without the corresponding x bit.

In the case of group permissions, an S may occupy the position of the x bit if the I bit is set for the file, i.e. if mandatory locking has been enabled for the file. Neither the x bit nor the s bit can be set for the group in this case.

If the I bit is set for a file, a program using the lockf() function can lock read and write access to the file for as long as it is accessing that file (see chmod).

For others permissions, the position of the x bit may be occupied by a t or T. These refer to the state of the sticky bit (t bit): t stands for set sticky bit with x bit, T for set sticky bit without x bit (see chmod).

Number of links
Decimal number indicating the number of links to the file; at least 1.

Login of owner
Login name of the file owner.

Group name
Group name of the file owner.

Size in bytes
Decimal number that indicates the file size in bytes.
If a special file is listed, the major device number and the minor device number are displayed instead of the file size.
Built-in sh command

`ls`

*Month, day, time*
Indicates the date and time of the last modification to the file. If the last modification was made more than six months ago, the year is output instead of the time.

*File name*
Name of the file.

In the case of directories, the size of the entire directory is displayed in addition to the individual file sizes. This size is specified in 512-byte blocks (see Example 4 on page 486).

If the file or directory is a symbolic link, the file name is printed followed by an arrow `->` and the pathname of the referenced file.

Note that in a Remote File Sharing environment, you may not always have the permissions indicated by the output of the `ls -l` command.

- `-L` With symbolic links, the link name is listed rather than the name of the original file or directory.

*Example*

```
$ ls -l
-rw------- 1   HUGO     other       7593   Nov 30 10:48 file
-rw------- 1   HUGO     other          3   Dec 17 10:53 linker -> file

$ ls -lL
-rw------- 1   HUGO     other       7593   Nov 30 10:48 file
-rw------- 1   HUGO     other       7593   Nov 30 10:48 linker
```

- `-m` Lists files across the page (stream output format), separated by commas. `ls` uses an environment variable, `COLUMNS`, to determine the number of character positions per line. If this variable is not set, the value of the environment variable `TERM` is checked in the terminfo database in order to determine the number of columns. If this information cannot be obtained, 80 columns are assumed per line.

- `-n` Same effect as `-l`, except that the user ID and the group ID are displayed instead of the login and group names.

- `-o` Same effect as `-l`, except that the group is not displayed.

- `-p` Puts a slash (`/`) after each file name if the file is a directory.

- `-q` Non-printing characters in file names are represented by question marks.

- `-r` Reverses the sorting order.
-R If file is not specified:
  All files and directories in the current directory are listed, followed by a recursive listing
  of all subdirectories and the files contained in them.

  if file is specified file is the name of a directory.

  The names of all files and subdirectories of the specified directory are listed recursively.

-s Prints the size of each file in column 1 (in 512-byte blocks) in addition to the file names.

-t Sorts by time last modified instead of by file name (latest first).

-u Together with option -t
  Uses the time of last access as the sort criterion.

  Together with option -l
  The time of last access is displayed instead of the time of last modification.

-x Multi-column output with entries sorted across instead of down the page. ls uses the
  environment variable, COLUMNS, to determine the number of character positions per
  line. If this variable is not set, the value of the environment variable TERM is checked in
  the terminfo database in order to determine the number of columns. If this information
  cannot be obtained, 80 characters per line are assumed.

-1 Prints only one entry per line of output.

file
  Name of the file or directory for which you want information to be listed. You can also
  name more than one file or directory.

  file not specified:
  The contents of the current directory will be listed.

File /etc/group
  Contains all the groups that have been set up.

Variable COLUMNS
  Determine the user’s preferred column position width for writing multiple text-column output.
  If this variable contains a string representing a decimal integer, the ls utility calculates how
  many pathname text columns to write (see -C) based on the width provided. The column
  width chosen to write the names of files in any given directory will be constant. Filenames
  will not be truncated to fit into multiple text-column output.

TZ
  Determine the timezone for date and time strings written by ls.
Locale

The following environment variables affect the execution of `ls`:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_COLLATE**
Determine the collating sequence of the output from `ls`.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments). It determines which characters are defined as non-printing characters in conjunction with the `-q` option.

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**LC_TIME**
Determine the format and contents of date and time strings when the `-g`, `-l`, `-n`, and `-o` options are used.

**NLSPATH**
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example 1**
Listing the contents of the current directory in long format. User’s login name: *HUGO*.

```
$ ls -l
  total 142
  drwxr-x-x- 2 HUGO other  48 Dec  1 16:13 ADDRESSES
-rw------- 1 HUGO other  356 Dec 17 13:58 card_a
-rw------- 1 HUGO other 24802 Dec  1 12:13 card_b
-rw------- 1 HUGO other  7890 Nov 30 10:48 card_c
-rwxr-xr-x 1 HUGO other  7253 Dec 21 13:37 card_d
-rw------- 1 HUGO other  9476 Dec 21 13:37 card_e
-rwxr-xr-x 1 HUGO other   0 Dec 18 13:16 card_f
-rw------- 1 HUGO other   593 Nov 30 10:48 letter1
-rw------- 1 HUGO other   837 Dec 17 10:53 letter2
-rw------- 1 HUGO other  3247 Dec 17 13:46 letter3
-rw------- 1 HUGO other  5222 Nov 30 10:48 letter4
-rw-rw-rw- 1 HUGO other  4687 Dec 21 11:15 letter5
-rw------- 1 HUGO other   228 Nov 30 10:48 letter6
-r-------- 1 HUGO other   105 Dec 21 13:39 typescript
```
Example 2  Multi-column output of the current directory using a line width of 80 columns per line. (This output form is also the default for terminal output).

```
$ ls -C
ADDRESSES  card_c  card_f  letter3  letter6
          card_a  card_d  letter1  letter4  typescript
          card_b  card_e  letter2  letter5
```

Example 3  Multi-column output after defining the line width as 40 columns per line.

```
$ COLUMNS=40
$ export COLUMNS
$ ls -C
ADDRESSES  card_e  letter4
          card_a  card_f  letter5
          card_b  letter1  letter6
          card_c  letter2  typescript
          card_d  letter3
```

Example 4  Listing the current directory in long format (option `-l`), marking directories (option `-p`) and including all files that begin with a period (option `-a`). User's login name: sisyphus.

```
$ ls -pla
total 120
drwxr-xr-x 10 SISYPHUS other  5720 Nov 17 08:00 .
drwxr-xr-x 13 SYSROOT  root  3380 Nov 04 11:48 ../
-rw-------  1 SISYPHUS other   79 Jul 19 14:21 .profile
-rw-------  1 SISYPHUS other  14 Oct 21 08:56 .rhosts
-rwx-------  1 SISYPHUS other  125 May 25 10:29 begin
drwxr-xr-x  2 SISYPHUS other  3380 Nov 09 10:30 commands/
drwxr-xr-x  2 SISYPHUS other  3380 Nov 09 10:30 commands/
drwxr-xr-x  3 SISYPHUS other  2340 Oct 11 15:35 lingua/
-rw-------  1 SISYPHUS other  2082 Nov 08 12:29 ls.ex
-rw-------  1 SISYPHUS other 11597 Nov 17 07:59 ls.rc.1
-rw-------  1 SISYPHUS other  1351 Jul 19 15:14 plural
drwxr-xr-x  2 SISYPHUS other  3380 Oct 11 15:36 post/
drwxr-xr-x  2 SISYPHUS other  1560 Oct 11 15:36 pro/
drwxr-xr-x  2 SISYPHUS other  2080 Nov 07 10:43 proc/
drwxr-xr-x  2 SISYPHUS other  1560 Oct 11 15:36 screens/
drwxr-xr-x  2 SISYPHUS other  1040 Nov 15 08:23 sdg/
```
Example 5  List all files and directories (and their contents). Each directory name is followed by a colon.

```
$ ls -C *
Mailboxes  authors  bwp.col
lingua:
car.cdr   engl.ger  examples  terms
screens:
out       pause     sinix
```

See also  *chmod, find, ln*
mailx  interactive message processing system (mail extended)

This description is divided into the following sections:

- Introduction
- Synopsis of mailx formats
- Description of individual formats
  - Format 1: Read mode (from page 490)
    - mailx commands in read mode
    - Input format
    - Functional overview
    - Descriptions in alphabetical order
    - Functionality in read mode
  - Format 2: Send mode (from page 509)
    - mailx commands in send mode (tilde commands)
    - Input format
    - Functional overview
    - Descriptions in alphabetical order
    - Functionality in send mode
- mailx command and startup files (from page 515)
- Variables
  - mailx variables (from page 517)
  - Environment variables (from page 522)
- Note on error messages
- Files
- Examples
- See also
Introduction

`mailx` enables you to send and receive electronic mail. If MAIL of the interNet Services (formerly `interNet Value Edition`) ist installed you can exchange messages with other users on the same system.

The presence of mail, if any, is usually indicated when you log in. You are also notified if new mail arrives while you are using `mailx`.

Messages have a message header holding information needed for message forwarding.

The message header contains the following fields:
- **Message** Number of the message
- **From** Sender of the message, date and time
- **To** Recipient of the message
- **Subject** Title of the message (see option `-s`)
- **Status** Processing status of the message (see description on page 508)

The header is followed by a blank line and then the text of the message.

The `mailx` utility allows you to:
- check for the presence of mail (read mode, Format 1, option `-e`)
- read messages (read mode, Format 1)
- use SINIX commands to process incoming messages (read mode, Format 1, `mailx` commands `!`, `l` and `pipe`)
- send messages (send mode, Format 2)
- use an editor to edit messages during a `mailx` session (read and send mode, `mailx` commands `edit`, `visual`, `^e` and `^v`).

`mailx` automatically collects read messages in a user-specific mailbox (`$HOME/mbox` by default).
mailx, Format 1: Read mode

Synopsis of mailx formats

Syntax

<table>
<thead>
<tr>
<th>Format 1: mailx[...option]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2: mailx[...option]...recipient...</td>
</tr>
</tbody>
</table>

Format 1
Read mode

mailx[...option]

No option specified

mailx behaves as described in Functionality in read mode.

option

-e  mailx simply checks whether there are any messages, terminating with an exit status of 0 if the user has mail. Otherwise, an exit status of 1 is returned.

mailx does not execute any startup files (see “mailx command and startup files” on page 515).

-f [...file]

mailx reads messages from file.

file not specified:

mailx reads messages from the user-specific mailbox $HOME/mbox.

-f [...file] not specified:

mailx reads messages from the default mailbox /var/mail/$USER.

-H  (H - Header) mailx displays only the header summaries and terminates. The exit status is 0 if the user has mail, otherwise 1.

The structure of a header summary is described in Functionality in read mode).

-i  (i - ignore) mailx ignores the signal SIGINT (see the mailx variable ignore on page 519).

-n  mailx does not initialize from the global startup file /etc/mail/mailx.rc (see “mailx command and startup files” on page 515).

-N  mailx does not print the initial header summary.

-u...login_name

mailx reads messages from the default mailbox of the specified user, provided you have the required read permission.

-V  mailx displays its version number and exits.
mailx commands in read mode

Input format

mailx commands in read mode have the following format:

\[\text{command}][\text{msglist}][\text{argument}]\ldots\]

command
Name of a \textit{mailx} command. Most command names can be abbreviated. In “Descriptions in alphabetical order” on page 496 the accepted abbreviations are shown in bold print.

\textit{command} not specified:
If you simply press \[\] at the \textit{mailx} prompt, \textit{mailx} executes the \textit{print} command.

msglist
One or more messages to be processed by the command. If several messages are specified, they are separated by blanks.

\textit{msglist} can be one or more of the following:

\begin{itemize}
\item \textit{n} Message number \textit{n}
\item . The current message (marked > in the header summary)).
\item ^ The first undeleted message.
\item $ The last message.
\item * All messages.
\item + The next message.
\item – The preceding message.
\item n-m Messages \textit{n} through \textit{m} inclusive.
\item login_name All messages from the specified user.
\item /\textit{str} All messages with the string \textit{str} in the subject field (the case is ignored).
\end{itemize}
mailx, Read-mode commands, Overview

:msgtype
   All messages of type msgtype, which can be any of the following:
   
   d  deleted messages (useful only with the mailx command undelete)
   n  new messages
   o  old messages (any messages not in state new or read)
   r  read messages
   u  unread messages

   msglist not specified:
   mails defaults to the current message.

argument
   String argument as appropriate to the command (see descriptions). If argument is a file
   name, the usual shell metanotation may be used. If a string containing blanks is to be
   interpreted as a single argument, it must be enclosed in double quotes.

Functional overview

This section provides an overview of all mailx read-mode commands, grouped by function.
Some commands may appear more than once. The overview is followed by descriptions of
all the commands in alphabetical order.
Most commands can be abbreviated. In “Descriptions in alphabetical order” on page 496
the abbreviated forms are shown in bold print.

Help functions

?    Display summary of mailx commands
help Display summary of mailx commands
list List names of all mailx commands
=    Display current message number
size Display size of message
from Display header summary
z+   Scroll header display one page forward
z-   Scroll header display one page back
headers Display screen page of headers
top  Display first five lines of message header
folders List contents of directory defined by mailx variable folder
version Display mailx version number

Quitting mailx

exit Quit mailx without changing mailbox
xit  Quit mailx without changing mailbox
quit Quit mailx
mailx, Read-mode commands, Overview

Displaying the header summary

from  Display header summary
headers  Display screen page of headers
z+  Scroll header display one page forward
z-  Scroll header display one page back

Manipulating and displaying the message header

discard  Suppress message header fields
undiscard  Undo the effect of discard
ignore  Suppress message header fields
unignore  Undo the effect of ignore
top  Display first five lines of message header
retain  Display only specified fields of message header

Displaying messages

print  Display message
type  Display message
next  Skip to next matching message
Print  Display message with full message header, ignoring discard
Type  Display message with full message header, ignoring discard

Editing messages

edit  Call editor to edit message (value of mailx variable EDITOR, default: ed)
visual  Call editor to edit message (value of mailx variable VISUAL, default: vi)

Changing the mailbox

file  Close current mailbox and open named one
folder  Close current mailbox and open named one

Saving messages

hold  Hold messages in mailbox
preserve  Hold messages in mailbox
save  Write message to file
copy  Write message to file
write  Write message to file, omitting header
mbox  Write message to user's standard mailbox
touch  Write message to user's standard mailbox
Save  Write messages to file named after sender of first message
Copy  Write messages to file named after sender of first message
Deleting messages
delete  Delete message
dp     Delete message, display next
dt     Delete message, display next

Switch to send mode and send or reply to message
mail   Send message
Mail   Send message, recording copy in file
reply  Reply to message
respond Reply to message
followup Reply to message, recording reply
Reply   Reply to list of messages
Respond Reply to list of messages
Followup Reply to list of messages, recording replies

Undoing mailx commands during a mailx session
undelete Restore deleted messages
touch   Undo effect of hold
hold    Undo effect of touch
unalias Erase aliases
undiscard Undo effect of discard
unignore Undo effect of ignore
unset   Erase variables

Invoking command interpreters, running shell commands
!       Escape to shell
!!      Repeat last shell command
shell   Invoke command interpreter
pipe    Pipe messages to standard input of shell command
|        Pipe messages to standard input of shell command

Miscellaneous
#       Null command (for including comments in command files)
=       Display current message number
alias   Declare aliases for mail recipients (same as group)
alternates Declare alternate names for your login name
cd      Change directory
chdir   Change directory
echo    Echo string (like SINIX echo command)
folders List contents of directory defined by mailx variable folder
group   Declare aliases for mail recipients (same as alias)
mailx, Read-mode commands, Overview

if mode cmdlist1 else cmdlist2 endif if

construct which selects a command list to execute on the basis of the mode
(send, read)

set
Set variables

size
Display size of message

source
Read and execute command file

unset
Erase variables

mailx commands not allowed in command files

! Escape to shell

Copy
Write messages to file named after sender of first message

edit
Call editor to edit message

followup
Reply to message, recording reply

Followup
Reply to list of messages, recording replies

hold
Hold messages in mailbox

mail
Send message

Mail
Send message, recording copy in file

preserve
Hold messages in mailbox

reply
Reply to message

Reply
Reply to list of messages

respond
Reply to message

Respond
Reply to list of messages

shell
Invoke command interpreter

visual
Call editor to edit message
Descriptions in alphabetical order

Some command names have synonyms. The full description is always next to the name which comes first in alphabetical order.

Most of these commands can be used both interactively and in command files. Exceptions to this rule are mentioned at the appropriate place (see “Functional overview” on page 492).

The bold print in the command names refers to the abbreviated forms.

!shell_command

Executes shell_command. By default the command interpreter specified in the mailx variable SHELL will be invoked and the specified command line will be passed to it.

If SHELL is not set, /bin/sh will be invoked.

If the bang variable is set, the last shell command executed is saved by mailx and can be repeated with !!.

The / command is not permitted in a command file.

#command

This is the null command used to introduce comments in command files (e.g. .mailrc).

=  Displays the current message number.

?  Displays a summary of all mailx commands.

alias[...alias-name[...recipient]]...

group[...alias-name[...recipient]]...

Declares an alias for the given recipients. The defined recipients are substituted when you use the aliases as recipients.

alias-name : Any string.

alias-name not specified : mailx displays a list of defined aliases.

recipient not specified: mailx displays the definitions for alias-name.

alternates[...name]...

Declares alternate names for your login name. Once you reply to a message, mailx deletes these alternative names from the list of recipients.

name: String for the alternate name.

name not specified: mailx displays the current list of alternate names.

cd_[directory]

Changes to the indicated directory.

directory not specified: mailx changes to $HOME.
mailx, Read-mode commands, Alphabetical

**copy[...msglist]...file**
Copies the specified messages to the named file. If file already exists, it is extended. The messages are marked as read (R) and copied to the user-specific mailbox when the default mailbox is closed.

No argument specified:
mailx appends the current message to $HOME/mbox.

If you do not subsequently delete the message, it will be saved again the next time the default mailbox is closed.

**Copy[...msglist]**
Copies the specified messages to a file in the current directory. The name of this file is derived from the name of the author of the first message in the message list (From entry). If this file already exists, it is extended. The messages are marked as read (R) and copied to the user-specific mailbox when the default mailbox is closed. The Copy command is not permitted in a command file.

**delete[...msglist]**
Deletes the specified messages from the current mailbox. If the mailx variable autoprint is set, the message following the last deleted message is printed.

A deleted message can be restored with undelete during a mailx session.

**discard[...field]...**
**ignore[...field]...**
Suppresses printing of the specified header fields in the output if they appear at the start of a line and come before a colon, e.g.: Cc:, Date:, Status:, Subject:, To:. You do not need to include the colon, and mailx ignores the case of the letters in field.

discard has an effect on the mailx commands next, pipe (or |), print, type, "f and "m, but not on Print, Type, "F and "M.

No fields are suppressed when messages are saved.

The effect of discard can be undone with undiscard or unignore. retain suppresses the effect of discard: all fields are ignored except those explicitly specified.

field not specified:
discard displays the current list of fields being ignored, if any.

**dp[...msglist]**
**dt [...msglist]**
(delete and print) Deletes messages from the mailbox and displays the message that follows the last one deleted.

A deleted message can be restored with undelete during a mailx session.
**mailx, Read-mode commands, Alphabetical**

**echo...string....**
Echoes string on standard output (like the SINIX echo command).
The value of an environment variable can be accessed as $name. echo always displays
the value of the environment variable, even if a mailx variable of the same name has
been defined.

**edit[...msglist]**
Invokes the editor specified with the mailx variable EDITOR (default editor: ed) and loads
the specified messages.

The edited message will be available in the mailbox again at the end of the editing
session.

The text is edited in a temporary file named /tmp/Rz$$, where $$ is the process ID of the
mailx process (see Files).

The edit command is not permitted in a command file.

**exit**

**exit**
Exits from mailx without changing the current mailbox, i.e.
– deleted messages are restored,
– read messages are not saved in $HOME/mbox, and
– edited messages retain their original status.

Also refer to the mailx command quit.

**file[...file]**

**folder[...file]**
Quits the current mailbox (like quit) and reads in the specified file as another mailbox.
The appropriate header lines are displayed.

**file**
Name of the mailbox to be processed, or the following metanotation:
% the current mailbox
%login_name
the default mailbox of the named user (/var/mail/$USER)
+file the file file in the directory folder (see mailx variable folder)
# the previous mailbox
& the user-specific mailbox ($HOME/mbox or the mailbox defined by the mailx
variable $MBOX)

file not specified:
mailx remains in the current mailbox and simply reports the number of messages in
it.

fi % closes and reopens the current mailbox. This allows you to read any new messages
that have arrived during your mailx session.
folders

Lists the names of all files in the directory defined by the mailx variable folder (messages are saved and recorded in this directory by mailx).

followup[.message]

Replies to the specified message like the mailx command reply.

mailx switches to send mode and considers the recipients to be

- the author of the specified message, i.e. the entry in the "From" field is taken over to the "To" field,
- the other recipients of the message, i.e. the entries in the "To" field are transferred to the "To" list, while those in the Cc field are copied to the Cc list.

When you have completed your input, mailx sends the message.

In contrast to reply, followup records the message in a file named after the recipient (with the network path removed). The storage location of this file depends on whether the mailx variables folder and outfolder are set. If both are set, the file is stored in the directory defined by the folder variable. Otherwise it is stored in the current directory. The file is extended if it already exists.

The followup command is not permitted in a command file.

Followup[.msglist]

Replies to the first of the messages specified in the message list in the same way as the mailx command Reply.

mailx switches to send mode and sends the reply to each sender of a message in msglist.

When you have completed your input, mailx sends the message.

In contrast to reply, Followup records the message in a file named after the sender of the first message (with the network path removed). The storage location of this file depends on whether the mailx variables folder and outfolder are set. If both are set, the file is stored in the directory defined by the folder variable. Otherwise it is stored in the current directory. The file is extended if it already exists.

The Followup command is not permitted in a command file.

from[.msglist]

Prints the header summary for all specified messages on the standard output.

group...alias-name recipient...

Declares aliases for the given recipients (see alias).
mailx, Read-mode commands, Alphabetical

headers[.message]
Displays the screen page of headers which includes the specified message. A screen page contains 20 lines or the number of lines defined by the screen variable.

message not specified:
mailx displays the current message.

help
Displays a summary of the mailx commands (see also ?).

hold[.msglist]
Holds (i.e. preserves) the specified messages in the default mailbox.

The messages are marked H in the header and remain in the mailbox, even if they have been read or saved.

The effect of hold can be undone with touch (and vice versa).

The hold and preserve commands are not permitted in a command file.

if mode
command_list1
else
command_list2
endif

if construct which selects a command list to execute on the basis of the specified mode.

mode

mode is the mode (send or read) in which you invoked mailx. It can be:

s  (send) command_list1 is executed if you invoked mailx in send mode; otherwise, command_list2 is executed.

r  (read) command_list1 is executed if you invoked mailx in read mode; otherwise, command_list2 is executed.

collection

Lists of mailx commands. Commands which are not permitted in command files are also not permitted here, i.e.: !, edit, followup, Followup, mail, Mail, reply, Reply, respond, Respond, shell and visual.

if, else, endif and all the commands in the command lists must each be on a separate line.

ignore[.field]...
Suppresses printing of the specified header fields (see discard on page 497).
mailx, Read-mode commands, Alphabetical

list
Lists the names of all available mailx commands on standard output (also refer to help and ?).

mail...recipient...
Sends a message to recipient.
mailx switches to send mode and mails the message as soon as you have finished entering the text.
If the mailx variable record is set, the message will be written to the file defined there. If the file already exists, it is extended.
The mail command is not permitted in a command file.

Mail...recipient...
Sends a message to recipient.
mailx switches to send mode and mails the message as soon as you have finished entering the text.
mailx records your message in the current directory in a file named after the recipient. If the file already exists, it is extended.
The Mail command is not permitted in a command file.

mbox[...msglist]
Writes the listed messages to the user-specific mailbox when the current mailbox is closed and then deletes them from the current mailbox, even if they have not been read. All such messages are marked M in the header summary.
The user-specific mailbox is $HOME/mbox or the file defined by the mailx variable MBOX.

next[...message]
Goes to the next message containing message in the header summary. For message you can enter characteristics as for a message list.
If, for example, the next message you want to read is that of a particular sender, specify next sender.
The sender can be an e-mail address or a local user ID.
message not specified: mailx displays the message following the current one.
next otherwise works in the same way as print.
mailx, Read-mode commands, Alphabetical

pipe[[..msglist]..shell_command]
I [[..msglist]..shell_command]
    Pipes the specified messages to the standard input of the given shell_command.
The messages are marked as read (R) in the header summary. If the mailx variable page
is set, a form feed character is inserted after each message (FF = CTRL L = X'0C').

No argument specified:
The default message is the current message, and the default command is the one
specified by the mailx variable cmd. If cmd is not set, the pipe command is ignored.

preserve[[..msglist]]
    Preserves the specified messages in the default mailbox (see hold).

print[[..msglist]]
type[[..msglist]]
    Prints the specified messages on the standard output.
The messages are marked as read (R) in the header summary. They are copied to the
user-specific mailbox upon termination of the default mailbox and then deleted from the
current one. The user-specific mailbox is $HOME/mbox or the file defined by the mailx
variable MBOX.
If the mailx variable crt is set, messages longer than the number of lines specified by
the crt variable are paged through the POSIX command more. You can use the mailx
variable PAGER to specify a POSIX command other than more.

Print[[..msglist]]
Type[[..msglist]]
    Like print, except that the whole header is always displayed, i.e. Print overrides the
effect of discard and ignore.

quit
    Exits from mailx, closing the currently processed mailbox.
If the default mailbox was being processed, the following applies:
- Read messages (0) and messages processed with mbox (M) are copied to the user-
specific mailbox and then deleted. The user-specific mailbox is $HOME/mbox or the file defined by the mailx variable MBOX.
- Unread messages (U) and messages processed with hold or preserve (P) are
  retained in the default mailbox.
- Explicitly saved messages (S) are deleted from the mailbox, if the keepsave variable
  is not set.

Also refer to the mailx commands exit and xit.
**mailx, Read-mode commands, Alphabetic**

`reply[…]message`
Replies to the indicated message.

`respond[…]message`
`mailx` switches to send mode and considers the recipients to be

- the author of the specified message, i.e. the entry in the “From” field is taken over to the “To” field,
- the other recipients of the message, i.e. the entries in the "To" field are transferred to the "To" list, while those in the Cc field are copied to the Cc list.

When you have completed your input, `mailx` sends the message.

Unlike `followup`, `reply` does not automatically create a file to record your reply. It only does so if the `mailx` variable `record` is set, in which case the message will be written to the file defined there. If the file already exists, it is extended.

The `reply` and `respond` commands are not permitted in a command file.

`Reply[…]msglist`
`Respond[…]msglist`
Replies to the first message in the message list.

`mailx` switches to send mode and sends the response to the sender of each message in `msglist`.

When you have completed your input, `mailx` sends the message.

Unlike `Followup`, `Reply` does not automatically create a file to record your reply. It only does so if the `mailx` variable `record` is set, in which case the message will be written to the file defined there. If the file already exists, it is extended.

The `Reply` and `Respond` commands are not permitted in a command file.

`retain[…]field`
Displays only the specified fields of the message header. The other fields are suppressed. `retain` displays the specified fields as well, if they are contained in the list of fields to suppress, i.e. `retain` overrides the effects of `discard` or `ignore`.

*field* not specified:
`retain` displays the current list of fields to display if available.

`save[…]msglist[…]file`
Saves the indicated messages in the named `file`, extending it if it already exists.

The messages are marked as saved (`S`), which means that they are deleted from the default mailbox as soon as you quit `mailx`, unless you have set the `keepsave` variable.

No argument specified:
`mailx` appends the current message to the end of `$HOME/mbox`. 
mailx, Read-mode commands, Alphabetical

Save[msglist]
Saves the specified messages in a file in the current directory. The file name is derived from the name of the sender of the first message in msglist ("From" entry; network addresses are removed). If the file already exists, it is extended.

The messages are marked as saved (S), which means that they are deleted from the default mailbox as soon as you quit mailx, unless you have set the keepsave variable.

set[name=value]
Sets the variable name.

name Name of a mailx variable or a freely defined variable.
value Any string or numeric value. \n within value is interpreted as a newline character, \t as a tab.

value not specified:
name is set to the null string.

No argument specified:
Prints all set variables and their values. The values are enclosed in double quotes.

You cannot change the values of environment variables. However, if you define an internal variable of the same name, mailx will use its value until you reset it (this does not apply to the mailx echo command).

The tilde command `i variable can be used to insert the value of variable in the text of a message.

You can delete variables with unset.

shell
Invokes by default the command interpreter specified in the SHELL variable. If SHELL is not set, /bin/sh will be invoked.

You can specify a different command interpreter with the mailx variable SHELL.

The shell command is not permitted in a command file.

size[msglist]
Displays the size of the specified messages on standard output in the form message_number: number_of_characters.

source[file
Reads the specified file as a command file and executes the mailx commands in it. mailx then returns to the interactive mode (see mailx command and startup files below).

top[msglist]
Displays the first 5 lines of the header for each specified message on the standard output. You can change the number of lines displayed with the mailx variable toplines.
Mailx, Read-mode commands, Alphabetical

**touch[...]msglist**
Causes the specified messages to be treated as read, i.e. they are copied to the user-specific mailbox upon termination of the default mailbox and then deleted from the current one. The user-specific mailbox is $HOME/mbox or the file defined by the MBOX variable.

This does not apply to messages that were saved with save or Save.

*touch* cancels the effect of *hold* and vice versa.

**type[...]msglist**
Prints the specified messages on standard output (see *print* on page 502).

**Type[...]msglist**
Like *print*, except that the whole header is always displayed (see *Print* on page 502).

**unalias[...]alias-name**...
Deletes the specified alias names.

**undelete[...]msglist**
Restores the specified messages provided they were deleted during the current session. The messages are marked as read (R).

If the *autoprint* variable is set, the last restored message is displayed.

*msglist* not specified:
If *msglist* is not specified, it defaults to the first deleted message following the current message that has not been undeleted if there is one, or the last deleted message preceding the current message that has not been undeleted otherwise.

**undiscard[...]field**
**unignore[...]field**
Deletes the specified header fields from the list of fields being ignored.

*field* not specified:
Deletes the whole of the list of fields being ignored.

**unset[...]name**...
**set[...]name**...
Deletes the specified variables.

If you delete a variable with the same name as an environment variable, you can access the value of the corresponding environment variable again.

**version**
Displays the current version and release number of *mailx*.
mailx, Read-mode commands, Alphabetical

**visual[...msglist]**
Invokes the editor specified by the *mailx* variable *VISUAL* (default editor: *vi*) and loads the indicated messages.

The edited message is placed in the mailbox at the end of the editing session.

The text is processed in a temporary file named `/tmp/Re$$`, where $$ is the process ID of the *mailx* process.

The **visual** command is not permitted in a command file.

**write[...msglist]..file**
Writes the specified messages in the named *file*. If the file already exists, it is extended.

*write* does not copy the header and the trailing blank line.

The messages are marked as saved (S) in the header summary. They are deleted from the default mailbox as soon as you quit *mailx*, unless you have set the *keepsave* variable.

**x**it
Exits from *mailx* without changing the current mailbox (see *exit* on page 498).

**z[±]** Scrolls the header display one page forward (**z+**) or back (**z-**). The number of lines per page is set by the *screen* variable. If *screen* is not set, 20 lines are displayed by default.

± not specified:
Same as **z+**.
mailx, Read-mode functionality

Functionality in read mode

At start-up time, mailx will take the following steps in sequence:

1. Establish all variables at their stated default values.
2. Process command-line options, overriding corresponding default values.
3. Import any of the DEAD, EDITOR, MBOX, LISTER, PAGER, SHELL or VISUAL variables that are present in the environment, overriding the corresponding default values.
4. Read mailx commands from an unspecified system start-up file, unless the -n option is given to initialize any internal mailx variables and aliases.
5. Process the start-up file of mailx commands named in the user MAILRC variable.

If no mail is present, mailx issues the message:

No mail for login_name

If mail has been received, mailx responds with a message line, an overview of all messages currently in the mailbox, and the mailx prompt ?. You can now enter mailx commands. You can use the ?, help and list command to get a list of all the available commands.

Headers

When you call mailx or use one of the mailx commands from, headers or z, mailx displays a header line for each message present. A header can have up to 9 blank-separated fields, e.g.:

N   1 hadea    Mon Sep    21 13:05    10/164    Beeblebrox

These fields refer to:

N          Processing status (see next subsection)
1          Message number. The messages are renumbered each time mailx is invoked. The oldest message is numbered 1.
hadea      Sender
Mon Sep 21  Date message received
13:05      Time message received
10/164     Size of message in lines/characters
Beeblebrox Title (first 25 characters of subject entry)
mailx, Read-mode functionality

Processing status

The processing status is the entry in the first header field. The status will be one of the following:

O  (old) The message has been read by a previous mailx call. It will be stored in
   $HOME/mbox, when you quit mailx or close the default mailbox.

U  (unread) The message has been present in the system mailbox for more than one
   invocation of mailx and has not yet been read. It will be preserved in the current mailbox
   when you quit mailx using quit.

R  (read) The message has been read. It will be saved in $HOME/mbox, if you quit mailx or
   the default mailbox.

N  (new) The message has arrived since you last called mailx or changed mailboxes.
   Messages in state new when mailx quits will be retained in the system mailbox.

M  (mbox) The message has been saved with mbox.

H  (hold) The message has been marked by the hold or preserve command. It will stay in
   the default mailbox when you close it.

S  (save) The message has been saved with a save, Save or write. It will be deleted from
   the default mailbox when you close it.

>c  This is the current message. This is the message referenced by mailx commands if you
   do leave msglist unspecified. The character c stands for any of the above status
   characters.

User-specific mailbox

Messages are written to the user-specific mailbox if
– you have read them but not deleted or explicitly saved them
– you have manipulated them with mbox or touch
– you have switched from the default mailbox to another mailbox using the file or folder
  command
– you have quit mailx with quit (except where the variable hold is set, then the messages
  remain in the standard mailbox).

The user-specific mailbox is $HOME/mbox or the file defined by the mailx variable MBOX.
The file is extended if it already exists. If you use the -f when you call mailx, you can process
this file with mailx commands in exactly the same way as the default mailbox.
mailx, Format 2: Send mode

Format 2  **Send mode**

mailx[..option].....recipient.....

No option specified

mailx behaves as described in “Functionality in send mode” on page 514.

**option**

-F  (file) mailx records all outgoing messages in a file named after the first specified recipient. This file is created in your home directory and can be processed with mailx like a mailbox.

-F not specified:

mailx searches for the record file defined in the mailx variable record. Nothing is recorded if this variable is not set.

-i  mailx ignores the SIGINT signal (see also the mailx variable ignore).

-n  mailx does not initialize from the global startup file /etc/mail/mailx.rc (see “mailx command and startup files” on page 515 below).

-s...subject

(subject) mailx enters subject in the Subject: header field of the header summary. This allows you to indicate the subject of the message.

subject

Any string. If the string includes blanks or special characters, it must be enclosed in double quotes.

**recipient**

One or more recipients. recipient can be:

- E-mail address (if you are using MAIL of the interNet Services)
- a login name on the local system
- an alias group (see mailx read-mode command alias)
- a pipe symbol followed by a shell command

If recipient begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through.
mailx commands in send mode (tilde commands)

Input format

Apart from being preceded by an escape symbol, mailx send-mode commands have the same format as read-mode commands.

```
[command][msglist][argument]...
```

~ Tilde as escape symbol. The mailx variable escape can be used to define a different character as the escape symbol. But the different character must not be a character which describes a command (e.g. ? or !).

command msglist argument
As described above under “Input format” on page 491.

Functional overview

This section provides an overview of all mailx send-mode commands, grouped by function. Some commands may appear more than once. The overview is followed by descriptions of all the commands in alphabetical order.

Help functions

```
~? Display summary of tilde commands
~p Display message being entered
```

Terminating/aborting text input

```
~ Terminate input and send
~x Abort input and do not send
~q Abort input and save but do not send
```

Inserting values of variables, old messages and files

```
~a Insert value of sign variable
~A Insert value of Sign variable
~i Insert value of mailx or environment variable
~d Insert contents of $HOME/dead.letter
~f Insert old messages
~F Insert old messages
~m Insert old messages
~M Insert old messages
~r Insert contents of named file
~< Insert contents of named file
```
mailx, Send-mode commands, Overview

**Invoking command interpreters, running shell commands**

- `!` Escape to shell
- `<!` Run shell command and insert output in text
- `|` Pass text to shell command and replace with command output

**Displaying text**

- `p` Display message being entered

**Editing text**

- `e` Call editor to edit text (default: `ed`)
- `v` Call editor to edit text (default: `vi`)

**Modifying mailing lists**

- `b` Add names to Bcc list
- `c` Add names to Cc list
- `t` Add names to To list
- `h` Edit To, Subject, Cc and Bcc fields

**Manipulating the message header**

- `c` Add names to Cc list
- `t` Add names to To list
- `h` Edit To, Subject, Cc and Bcc fields
- `s` Replace contents of Subject field

**Recording text**

- `w` Write message being entered to file, omitting header
- `q` Abort input and save text but do not send

**Send-mode commands which expect mailx to have been called in read mode**

- `.` Execute mailx command
- `:` Execute mailx command
- `f` Insert old messages
- `F` Insert old messages
- `m` Insert old messages
- `M` Insert old messages

**Executing mailx read-mode commands**

- `.` Execute mailx read-mode command
- `:` Execute mailx read-mode command
Descriptions in alphabetical order

`mailx` commands in send mode (tilde commands) must start with an escape symbol in column one. The default escape symbol is a tilde (˜), but this can be redefined with the `mailx` variable `escape`.

Tilde commands are not permitted in a command file.

`~!` `shell_command`

Executes the specified `shell_command`. Invokes by default the command interpreter specified in the `SHELL` variable. If `SHELL` is not set, `/bin/sh` will be invoked.

`~.`

Terminates message input and sends the message.

If you are working via `rlogin` on a remote computer, this tilde command will be interpreted as an instruction to clear down the connection, with the result that the remote session will be terminated immediately. Here are some possible solutions:

- redefine the escape symbol with the `mailx` variable `escape`
- set the `mailx` variable `dot`, so that you can terminate input simply with a dot in column one
- terminate input with `[END]`

`~:` `mailx-read-command`

Executes the specified `mailx` read-mode command.

You must have invoked `mailx` in read mode (and then switched to send mode with a `mailx` command such as `mail`). Otherwise, `mailx` will only execute commands that have nothing to do with the processing of a mailbox (e.g. `set` or `exit`).

The underscore in the second format is mandatory.

`~?` Displays a summary of all tilde commands.

`~a` (autograph) Inserts the value of the `mailx` variable `sign` in the message.

`~A` (autograph) Inserts the value of the `mailx` variable `Sign` in the message.

This enables you to define an alternate sign-off string, for example.

`~b...name...` (blind carbon copy) Adds one or more names to the blind carbon copy (Bcc) list. The Bcc list contains the names of additional recipients of the message. These names are not included in the header.

`~c...name...` (carbon copy) Adds one or more names to the carbon copy (Cc) list. The Cc list contains the names of additional recipients. These names are included as part of the header information (the Cc entry).
mailx, Send-mode commands, Alphabetical

`~d` (dead letter) Reads the contents of the file `$HOME/dead.letter` into the message. This file contains messages which `mailx` could not send or which you aborted with `~q`.

`~e` (ed editor) Invokes the editor specified by the `mailx` variable `EDITOR` (default editor: `ed`) and loads the partial message. Input of the partial message may be continued after the editing session.

`~f[...msglist]` (file) Inserts the specified messages, without alteration, into the message text. This command is only executed if `mails` was invoked in read mode (Format 1).

`~F[...msglist]` works like `~f`, but inserts always the whole message header. `discard, ignore` and `retain` will be ignored.

`~h` (header) Prompts successively for the following information:

- **To:** Recipient
- **Subject:** Subject of the message
- **Cc:** Carbon copy list containing additional recipients of the message; the names in this list appear in the Cc field of the header
- **Bcc:** Blind carbon copy list. Like Cc, except that the names do not appear in the header

The fields are displayed with existing values (if any), which you can edit as if you had just entered them.

`~i` variable
Inserts the value of the named variable into the message text. The named variable can be a `mailx` variable or an environment variable.

`~m[...msglist]` (move) Inserts the indicated messages into the text, shifting each line one tab stop to the right.
This command is executed only if `mailx` was invoked in read mode (Format 1).

`~M[...msglist]` works like `~m`, but inserts always the whole message header. `discard, ignore` and `retain` will be ignored.

`~p` (print) Displays the message being entered.

`~q` Quits input mode by simulating an interrupt. The input text is not aborted but saved in the `$HOME/dead.letter` file. This tilde command has the same effect as the `[DEL]` key except that it cannot be suppressed with the `ignore` variable.
mailx, Send-mode functionality

```
~r..file
~r..!shell_command
~<..file
~<..!shell_command
  (read) inserts the contents of file or the output of shell_command into the message text.
~s..string...
  (subject) Sets the Subject field of the header to the specified string. Multiple blank-separated strings do not need to be quoted.
~t..recipient...
  (to) Adds the indicated names of one or more recipients to the "To" field of the header. Multiple names must be separated by blanks.
~v  (vi editor) Invokes the screen editor identified by the mailx variable VISUAL (default editor: vi) and loads the partially entered message. Input of the edited message can be continued on completion of the editing session.
~w..file
  (write) Copies the partially entered message, without the header, into the specified file. If the file does not exist, it will be created. Otherwise the message is appended to the specified file.
~x  Exits, aborting the message being entered. The partially entered message is neither sent nor saved.
~l..shell_command
  Pipes the current text of the message to the standard input of shell_command. If the shell command returns an exit status of 0, the current text is replaced by the output of the command.
  Invokes by default the command interpreter specified in the SHELL variable. If SHELL is not set, /bin/sh will be invoked.
```

**Functionality in send mode**

When you call mailx, it first processes startup files. These files may be used to initialize mailx variables, for example (see mailx command and startup files).

Then, unless you use the -s option to specify a message subject, mailx displays:

```
Subject:
```

and expects you to enter the subject of the message. This line, which may consist of up to 1024 characters, is written by mailx into the Subject: field of the message header. If the subject is too long, mailx will print the message mail: ERROR signal 10, and the mail will not be delivered.
mailx will now be in send mode, which means that you can enter your message text. All tilde commands are permitted during text input. They must start in column 1. Once you have sent off a tilde command with \( \tilde{a} \), mailx redispalyes the whole of the command you have entered in the same line, followed by the string \( \text{continue} \) when it has finished executing the command. If you use one of the commands for inserting text into your message text (such as \( \tilde{a} \)), mailx will not echo the text on the screen. You can view the original text and the inserted text with the \( \tilde{p} \) command.

mailx stores the input text in a temporary file in the \( /tmp \) directory.

The command \( \tilde{.} \) or the \( \text{[END]} \) key signals the end of input.

Tilde commands in read mode

Some tilde commands only offer their full functionality if you call mailx in read mode (Format 1) and temporarily switch from there to send mode. The commands in question are:\n\( \tilde{a} \) and \( \tilde{f} \): (execute mailx command) and \( \tilde{f} \) and \( \tilde{m} \) (insert old messages).

The read commands you can use to switch temporarily to send mode so as to send or reply to a message are followup, Followup, mail, Mail, reply, Reply, respond and Respond.

mailx command and startup files

Command files

Command files are files that contain mailx commands. Each mailx command must be entered in a separate line. You can execute command files by using the source command during a mailx session, or you can use them as startup files (see below).

Tilde commands are not permitted in command files, and nor are the commands \(!\), edit, followup, Followup, mail, Mail, reply, Reply, respond, Respond, shell and visual.

The copy, Copy, hold and preserve commands are permitted, but then any subsequent commands which are intended to operate on a message list will be ignored.

If an error occurs in a command file, mailx ignores all subsequent commands in the file. An error also occurs if a message list refers to a non-existent message (see Example 2 on page 525).

Startup files

Startup files are command files that mailx processes every time it is invoked, unless you call it in read mode using the \(-e\) or \(-n\) options.

mailx first processes the global startup file \( /etc/mail/mail.rc \), followed by the private startup file \( $HOME/.mailrc \), provided such files exist.

You can redefine the path name of the private startup file with the MAILRC variable.
The `mailx` commands in the startup files may not use `msglist`, because this information is not yet provided when executing the startup files.

**Variables**

`mailx` utilizes environment variables, `mailx` variables and freely defined variables.

All variables can be imported, and during a `mailx` session they can be set and reassigned with the `set` command and deleted with `unset`.

`mailx` variables, i.e. all variables consisting of lowercase letters only, can only be set within `mailx` (e.g. in startup files). Contents of shell variables with the same names are not inherited by `mailx` variables.

If you use `set` to reassign an imported variable, the new value applies until you change it again, delete it with `unset` or end your `mailx` session.

The `mailx` command `echo` always references the original value of an imported variable.

`asksub`, `header` and `save` are enabled by default.

Of the variables which can be assigned values, the following have default values:

- `DEAD=$HOME/dead.letter`
- `EDITOR=ed`
- `escape=˜`
- `MBOX=$HOME/mbox`
- `LISTER=ls`
- `PAGER=more`
- `prompt=?`
- `screen=20`
- `SHELL=/bin/sh`
- `toplines=5`
- `VISUAL=vi`

The references to `Associated commands` in the following list relate to the `mailx` commands particularly affected by the setting or deletion of the variable in question.
mailx variables

allnet  
mailx treats all network names ending with matching login names as identical. Addresses in the form ...computer_name!computer_name!login_name are treated as network names. See also the metoo variable.

Default value: The variable is not set.

append  
Appends messages saved in the user-specific mailbox ($HOME/mbox by default) to the end of the file.

Associated commands: copy, Copy, file, folder, mbox, next, print, Print, type, Type, quit, touch

Default value: The variable is not set.

ask  
asksub  
Causes mailx to prompt for the subject when invoked (see also option -s).

Associated commands: "h, "s

Default value: The variable is set.

askbcc  
mailx prompts for the Bcc list after the subject is entered.

Associated commands: "c, "h

Default value: The variable is not set.

askcc  
mailx prompts for the Cc list after the subject is entered.

Associated commands: "c, "h

Default value: The variable is not set.

autoprint  
Displays the next message after the delete command, and the restored message after the undelete command.

Default value: The variable is not set.

bang  
Causes mailx to remember the last command executed with !shell_command. You can then repeat the command by entering !!.

Default value: The variable is not set.

cmd=shell_command  
Sets the default command used by pipe and | to the specified shell_command
(only used if no command is specified in pipe).

Default value: None.
**mailx, mailx variables**

- **crt=number**  
  Pipes message output having more than `number` lines through the command specified by the `PAGER` variable (default: `PAGER=more`).  
  Associated commands: `dp`, `dt`, `next`, `print`, `Print`, `type`, `Type`, `p`  
  Default value: The variable is not set.

- **debug**  
  Enables diagnostics for debugging. If you set this variable, mail will not be delivered.  
  Default value: The variable is not set.

- **dot**  
  Causes a line consisting solely of a dot in column one to terminate input (instead of the command `.`).  
  Default value: The variable is not set.

- **escape=c**  
  Substitutes `c` for the tilde escape symbol. `c` may not be a character describing a command (e.g. `?` or `!`).  
  Default value: `~`

- **flipr**  
  The effects of the commands `reply`, `respond` and `Reply`, `Respond` are exchanged.  
  Associated commands: `reply`, `Reply`, `respond`, `Respond`  
  Default value: The variable is not set.

- **folder=directory**  
  If both `folder` and `outfolder` are set, reply texts with the `followup` and `Followup` commands will be recorded in `directory`, not in the current directory. If `directory` does not begin with a slash, `mailx` sets the directory name to `$HOME/directory`.  
  You can also use the form `+filename` to reference these record files in any `mailx` command which accepts file names. `mailx` will then expand the name by prepending `directory`.  
  Default value: The variable is not set.

- **header**  
  Causes the header summary, the current `mailx` version string and the number of messages to be displayed when `mailx` is invoked.  
  Default value: The variable is set.

- **hold**  
  Preserves read messages in the default mailbox instead of putting them in the user-specific mailbox file (also refer to `MBOX` variable on page 522).  
  Associated commands: `copy`, `hold`, `mbox`, `next`, `preserve`, `print`, `Print`, `quit`, `touch`, `type`, `Type`  
  Default value: The variable is not set.
mailx, mailx variables

ignore  Ignores the SIGINT signal during message input.
Associated commands: followup, Followup, mail, Mail, reply, Reply, respond, Respond, "q ([DEL])
Default value: The variable is not set.

ignoreeof  The end-of-file signal (EOF, [END] key) is to be ignored during message input (also refer to the dot variable on page 518).
Associated commands: followup, Followup, mail, Mail, reply, Reply, respond, Respond, ".
Default value: The variable is not set.

indentprefix=string  During the insertion of a message into a text each line of the message begins with string.
Associated commands: "m, "M
Default value: Tab character.

keep  The mailbox should not be deleted when it is empty.
Default value: The variable is not set.

keepsave  Messages that are marked as saved (S) in the header are not to be deleted from the default mailbox.
Associated commands: save, Save, write
Default value: The variable is not set.

metoo  If your own login name appears in the list of recipients ("To" list), it is not deleted from the list. See also the mailx command alias and the allnet variable.
Associated commands: alias, alternates, followup, group, reply, respond, "h
Default value: The variable is not set.

outfolder  If both outfolder and folder are set, reply files with the followup and Followup commands will be stored in the directory defined by folder (also refer to record variable on page 520).
If only outfolder (or only folder) is set, reply files will be stored in the current directory.
Default value: The variable is not set.
mailx, mailx variables

**page**
Causes a form feed character (FF = CTRL L = X'0C') to be inserted after each message routed through a pipe or | command.
Default value: The variable is not set.

**prompt=string**
Sets the input prompt for mailx read-mode commands to string.
Default value: ?

**quiet**
Suppresses the display of the opening message and version identifier when mailx is invoked.
Default value: The variable is not set.

**record=file**
Records all outgoing mail in the named file. The file is extended if it already exists.
Associated commands: mail, Mail, reply, Reply, respond, Respond, ~.
Default value: The variable is not set.

**save**
Saves messages that could not be sent, e.g. due to an error or an interrupt from [DEL] during input. These messages are saved in the file specified by the DEAD variable.
Associated commands: followup, Followup, mail, Mail, reply, Reply, respond, Respond, "d, "q ([DEL])
Default value: The variable is set.

**screen=number**
mailx displays number header lines.
Associated commands: header
Default value: Dependent on the terminal type specified by TERM, usually 20

**sendwait**
After a read command, waits for the background mailer to finish before returning control to the user.
Associated commands: followup, Followup, mail, Mail, reply, Reply, respond, Respond
Default value: The variable is not set.

**showto**
If you send a message and include yourself among the recipients, the name of the first recipient in the recipient list (To list) will appear in the header list displayed, not your login name.
Associated commands: from, headers, z+, z-
Default value: The variable is not set.
sign=string  Defines an autograph string to be inserted into the text of a message.

Associated commands: 'a, 'i
Default value: The variable is not set.

Sign=string  Defines an (additional) autograph string to be inserted into the text of a message.

Associated commands: 'A, 'i
Default value: The variable is not set.

toplines=number
mailx displays number lines from the message header.

Associated commands: top
Default value: 5

Error  mailx’s error messages are largely self-explanatory.

File  
/etc/mail/mail.rc
Global startup file

$HOME/.mailrc
Private startup file

/var/mail/$USER
The default mailbox that is searched by mailx for incoming messages.

$HOME/mbox
User-specific mailbox in which mailx saves messages that have been read.

$HOME/dead.letter
File that is used by mailx to save messages that could not be sent, e.g., due to an error or an interrupt with the [DEL] key during input. If the file already exists, it is overwritten.

./<username>
Files created in the current directory with names derived from a login name. These files are created by mailx in response to the following commands:
Copy, followup, Followup, Save.

You can also select some other directory instead of the current directory (see the mailx variables folder (page 518) and outfolder (page 519)).

/tmp/R[emrxz]*
Temporary files.

/tmp/Rc$$
Temporary file used by the commands edit, visual, 'c, and 'v. $$ is the process ID of the mailx process.
mailx, Files

/usr/share/lib/mailx/mailx.help*
Help message files

Variable  Environment variables

DEAD=file  The named file is used by mailx to save messages that could not be sent, e.g. due to an error or an interrupt during input. This file is extended if it already exists.

Associated commands: followup, Followup, mail, Mail, reply, Reply, respond, Respond, `d, `q ([DEL])

Default value: $HOME/dead.letter

EDITOR=shell_command
In a mailx session, message texts can be edited with the editor named as shell_command (also refer to mailx variable VISUAL (page 523)).

Default value: ed.

HOME=directory
Home directory.
Specifies the default directory in which mailx creates or searches for the files dead.letter, mbox, .mailrc, and the files used to record outgoing messages (see the commands followup (page 499), Followup and `f).

LISTER=shell_command
Uses shell_command to list files in the folder directory.

Associated commands: folders

Default value: ls

MAILRC=file  file defines the name of the private startup file (default: $HOME/mailrc; see "mailx command and startup files" on page 515).

MBOX=file  file designates the user-specific mailbox in which mailx is to save read messages before removing them. Each additional message extends the file (also refer to the hold variable on page 518).

Associated commands: copy, hold, mbox, next, preserve, print, Print, quit, save, touch, type, Type

Default value: $HOME/mbox

PAGER=shell_command
Uses the specified shell_command for the paging of output that exceeds the number of lines defined in the crl variable.

Associated commands: dp, dt, next, print, Print, type, Type, `p

Default value: more
SHELL=shell_command
Defines the command interpreter used by mailx to execute POSIX
commands.
Associated commands: !, shell
Default value: /bin/sh

TERM
Contains information about the terminal type. The TERM variable will be
evaluated if the mailx variable screen is not set.

USER
The USER variable tells mailx the login name of the user, to enable it to
identify the default mailbox etc.

VISUAL=shell_command
In a mailx session, message texts can be edited with the editor named as
shell_command.
Associated commands: visual, ˜v
Default value: vi.

Locale
The following environment variables affect the execution of mailx:

LANG
Provide a default value for the internationalization variables that are unset
or null. If LANG is unset or null, the corresponding value from the implementa-
tion-specific default locale will be used. If any of the internationalization
variables contains an invalid setting, the utility will behave as if none of the
variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other inter-
nationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data
as characters (for example, single- as opposed to multi-byte characters in
arguments) and the handling of case-insensitive address and header-field
comparisons.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents
of diagnostic messages written to standard error and informative messages
written to standard output.

NLSPATH
Determine the location of message catalogs for the processing of
LC_MESSAGES.
Example 1  Replying to a message

User marvin receives the message you have mail, so he calls mailx without options. He sees a few lines of text and some header lines, has all his messages displayed one after the other (using Ñ, for example) and finally replies to the fourth message. The text of his reply is to be recorded in the current directory (followup or fo). While in send mode, he changes the Subject entry assigned automatically by mailx (˜s) and inserts his sign-off string, as defined in his sign variable, at the end of the text (˜a). Before sending his letter (˜.), he has it redisplayed (˜p) and then quits mailx with .xit or x so as to retain all his messages in the mailbox.

```
mailx
mailx version 4.0 Type ? for help
/var/mail/MARVIN*: 4 messages 4 new
N 1  ARTHUR  Fri Sep  6  9:21  13/373  betelgeuse
N 2  FRODO  Fri Sep  6 12:00  13/365  sysadm
N 3  PETER  Mon Sep 16 10:01  9/232  qed
N 4  BENNY  Tue Sep 17 16:43  21/593  Project S

fo BENNY
To: <BENNY>
Subject: Re: Project S

˜s Final report on P S˜s Final report on P S

Hi there, Ben!
Thanks for the minutes. What I really need is the final report itself, and not just Real Soon Now, but tomorrow at the latest. I need a printed version, not just a file.
Regards,
(-: marvin :-) (continue) 

˜.

? x
Held 4 messages in /var/mail/MARVIN
```

$
Example 2  Example of a startup file

The following startup file sets variables and prints all messages from `winnie` (with `lp`) when `mailx` is invoked in read mode.

```
# Processing variables
set page crt=24 cmd=lp VISUAL
set sign="\n\tFord Prefect\n\tSales Division\n\tDetroit"

# Sender: Network system administrator
alias sys root@orlando root@annapolis root@chicago

# Prints specific mail
if r
  pipe winnie lp
  from winnie
endif
```

Note that `mailx` terminates the script if any command cannot be executed. This might be the case here with the `pipe` command if no messages from `winnie` were present. In other words, `mailx` then would not execute the `from` command (or any others that might follow).

See also  `ed, ls, more, sh, vi`
**make**  

**make** maintain, update and regenerate groups of programs

In modular programming, programs are typically made up of a number of files. *make* is a tool for updating programs of this type.

*make* uses a *makefile*, in which you can define targets and dependencies between targets. If one source file has been modified, *make* regenerates the program by recompiling only those parts which are directly or indirectly dependent on the modified file.

*make* regenerates the target if it is older than at least one of the files on which it is dependent. *make* allows for the dependency relationships between the file and checks the date and time when a file was last modified.

The *makefile* is normally called *makefile*, *Makefile*, *s.makefile* or *s.Makefile*. If you follow this naming convention, you can call *make* without specifying any arguments. *make* will look for the *makefile* in the current working directory or in the SCCS directory and will regenerate the target if at least one modification has been made.

**Syntax**

```plaintext
make[...-einpqrst][-f..makefile]...[-kl-S] [macro=name]... [target]...
```

**Options**

- **-e**  Cause environment variables to override macro assignments within *makefiles.*
- **-f..makefile**  Specify a different *makefile*. *makefile* is a pathname of a description file, which is also referred to as the *makefile*.
- **-i**  Ignore error codes returned by invoked commands. This mode is the same as if the special target *IGNORE* were specified without prerequisites.
- **-k**  Continue to update other targets that do not depend on the current target if a non-ignored error occurs while executing the commands to bring a target up-to-date.  
  - *k* overrides earlier -S options.
- **-n**  Useful for debugging. Write commands that would be executed to standard output, but do not execute them. Even lines with an at sign (@) character prefix will be written to standard output. Lines with a plus sign (+) prefix will be executed.
- **-p**  Write to standard output the complete set of macro definitions and target description. The output format is unspecified.
- **-q**  Return a zero exit value if the target file is up-to-date; otherwise an exit value of 1. Targets will not be updated if this option is specified. However, a command line (associated with the targets) with a plus sign (+) prefix will be executed.
- **-r**  Clear the suffix list and do not use the built-in rules.
make

-s  Do not write command lines or touch messages to standard output before execution. This mode is the same as if the specified target `SILENT' were specified without prerequisites.

-S  Terminate `make' if an error occurs while executing the commands to bring a target up-to-date. This will be the default and the opposite of `-k'.

-t  Update the modification time of each target as though a touch target had been executed. A command with a plus sign (+) prefix will be executed.

Creating a makefile

The `makefile' specified by the `-f' option is a carefully structured file containing explicit instructions for updating programs. The file contains a sequence of entries defining dependencies.

The first line of each entry is a blank-separated, non-empty list of targets, followed by `:', then by a list (which may be empty) of required files or dependencies. Text following `;' and all following lines that begin with a tab are shell commands that are to be executed in order to update the target.

The first non-blank line which does not begin with a tab or `#' starts a new dependency or macro definition. Shell commands may be continued across several lines with a `\<newline>' sequence. Everything that `make' outputs (apart from the initial tab) is passed directly to the shell unmodified. Thus

```bash
echo a\b
```

will cause `ab' to be output just as the shell would.

Comments start with a number sign (`#') and continue until an unescaped newline character is reached.

The following `makefile' says that `pgm' depends on two files, `a.o' and `b.o', and that they in turn depend on their corresponding source files (`a.c' and `b.c'), and a common file `incl.h':

```bash
pgm:  a.o  b.o
      c89  a.o  b.o  -o  pgm
a.o:  incl.h  a.c
      c89  -c  a.c
b.o:  incl.h  b.c
      c89  -c  b.c
```

Executing a makefile

Command lines are executed one at a time, each by its own shell. The SHELL environment variable or the SHELL macro can be used to specify the shell that make should use to execute commands. The default is /usr/bin/sh.

The following directives (special targets) can be included in a makefile to control the behavior of make:

**.POSIX** This special target must be specified without prerequisites or commands. If it appears before the first non-comment line in the makefile, make will process the makefile specified by this section; otherwise the behavior of make is unspecified.

**.DEFAULT** If the makefile uses this special target, it must be specified with commands, but without prerequisites. The commands will be used by make if there are no other rules available to build a target.

**.IGNORE** Prerequisites of this special target are targets themselves; this will cause errors from commands associated with them to be ignored in the same manner as specified by the -i option.

**.PRECIOUS** Prerequisites of this special target will not be removed if make receives the quit or interrupt signal. If no prerequisites are specified, all targets will be ignored.

**.SILENT** Prerequisites of this special target are targets themselves; this causes commands associated with them to not be written to the standard output before they are executed.

Command lines can have one or more of the following prefixes: an at sign @, a hyphen (-), or a plus sign (+). These modify the way in which make processes the command. When a command is written to standard output, the prefix is not included in the output.

- the command will not be written to standard output before it is executed.
- any error found while executing the command will be ignored
- a command line will be executed even if the -n, -q or -t option is specified.

A line is output when it is executed unless the -s option is present or the .SILENT directive applies to the file or the initial character sequence includes a @. The -n option causes the command to be output without being executed unless there is a + prefixed to the command line (in which the command will always be executed). The -t option updates the modified date of a file without executing any commands except where there is a + prefixed to a command.)
Normally (or when the \texttt{-S} option is set), commands which return a non-zero status cause \texttt{make} to terminate. The exit status is ignored if the \texttt{-i} option is present or if the \texttt{.IGNORE} directive applies to the file or if the initial character sequence of the command includes \texttt{-}. If the \texttt{-k} option is present, work is abandoned on the current entry, but continues on other branches which are not dependent on that entry.

Interrupt and quit signals cause the target file to be deleted unless the \texttt{.PRECIOUS} directive applies to it.

Environment

The environment is read by \texttt{make}. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any \texttt{makefile} and immediately after the predefined rules. Thus macro assignments in \texttt{makefile} override environment variables. The \texttt{-e} option causes the environment to override macro assignment in a \texttt{makefile}. File name suffixes and their associated rules in a \texttt{makefile} override predefined rules for any identical suffixes.

The \texttt{MAKEFLAGS} environment variable may contain macros and any input options other than \texttt{-f} and \texttt{-p} which are valid for the command line. \texttt{make} interprets the variable before the command line. When \texttt{make} is invoked, the identically named macro \texttt{MAKEFLAGS} (and the variable, if undefined) is automatically supplied with the current options and macros and passed on to invocations of commands. Consequently the \texttt{MAKEFLAGS} always contains the latest definitions. This proves very useful for “super-makes”. In fact, when the \texttt{-n} option is used, $(\texttt{MAKE}) is always executed anyway. That means that you can run \texttt{make -n} recursively on a whole software system to see what would have been executed. This is possible because the \texttt{-n} is added to \texttt{MAKEFLAGS} and passed to further invocations of $(\texttt{MAKE}). This is one way of debugging all the \texttt{makefiles} for a software project without actually running anything.

The environment variable \texttt{PROJECTDIR} provides a directory to be used to search for SCCS files not found in the current directory. If the value of \texttt{PROJECTDIR} begins with a slash, it is considered an absolute pathname; otherwise, the home directory of a user of that name is examined for a subdirectory \texttt{src} or \texttt{source}. If such a directory is found, it is used. Otherwise, the value is used as relative pathname.

Include files

If \texttt{include} followed by a blank or a tab appears at the start of a line in a \texttt{makefile}, the rest of the line is interpreted as a file name, and the associated file will be read by the current invocation after substitution of any macros.
Macro definitions

Macro definitions are in the form \( \text{string1} = \text{string2} \). \( \text{string2} \) is defined as all characters, if any, after the equal sign, up to a comment character (#) or an unescaped newline character.

Subsequent appearances of \( \$(\text{string1}[:\text{subst1=}\text{subst2}]) \) are replaced by \( \text{string2} \). The parentheses are optional if a one-character macro name is used and there is no substitution rule. The optional \( :\text{subst1=}\text{subst2} \) is a substitution rule. If a rule is specified, all non-overlapping occurrences of \( \text{subst1} \) in the specified macro are replaced by \( \text{subst2} \). Strings for this type of substitution are delimited by blanks, tabs, newline characters and beginnings of lines. An example of the use of a substitution rule is shown in the Libraries section.

Internal macros

The \texttt{make} utility maintains five internal macros that can be used in target and inference rules.

\( \$^* \) evaluates to the current target name with its suffix deleted. It is evaluated at least for inference rules.

\( \$@ \) evaluates to the full target name of the current target, or the archive filename part of a library archive target. It is evaluated for both target and inference rules.

\( \$< \) in an inference rule, \( \$< \) evaluates to the file name whose existence allowed the inference rule to be chosen for the target. In the .DEFAULT rule, the \( \$< \) macro evaluates the current target name. The \( \$< \) macro is evaluated only for inference rules. For example, in the .c.a. inference rule, \( \$< \) represents the prerequisite .c file. An example for making optimized .o-files from .c-files is:

\[ \text{.c.o:} \]
\[ \text{c89 -c -O } ^* \text{.c} \]

or:

\[ \text{.c.o:} \]
\[ \text{c89 -c -O } \$< \]

\( \$? \) evaluates to the list of prerequisites that are newer than the current target. It is evaluated for both target and inference rules.

\( \$% \) evaluates only when the current target is an archive library member of the form \texttt{libname(member.o)}. In these cases, \( \$@ \) evaluates to \texttt{libname} and \( \$% \) evaluates to \texttt{member.o}.
Each of the internal macros has an alternative form. When an upper-case D or F is appended to any of the macros, the meaning is changed to the directory part for D and filename part for F. The directory part is the path prefix of the file without a trailing slash; for the current directory, the directory part is ".". When the $? macro contains more than one prerequisite filename, the $(?D) and $(?F) (or $?D and $?F) macros expand to a list of directory name parts and filename parts respectively.

**Default rules**

Certain file names, such as those ending in .o, have inferable prerequisites (dependency relationships) such as .c or .s. If no update commands for such a file are defined in the makefile, make looks for and compiles files matching the default prerequisites in order to make the target. For this purpose make has inference rules that allow it to build files from other files by examining suffixes and determining an appropriate inference rule to use. The following are the default inference rules:

```
.c .c .f .f .s .s .sh .sh .C .C
.c.o .c.a .c.c .c.o .f.a .f.o .f.a .f.f .f.o
.h .h .l.c .l.o .l.c .l.o .a .s .s.o .s.a .s.o
.s.s .sh .sh .y.c .y.o .y.c .y.o .y.y .C.a .C.o .C.a
.s.s .s.s .s.s .s.sh .sh .sh .y.c .y.o .y.c .y.o .y.y .C.a .C.o .C.a
```

The user can add rules to this list by entering them in the makefile.

The inference of prerequisites can be controlled. The rule for creating a file with the suffix .o from a file with the suffix .c is specified as an entry with c.o: as the target and no dependents. Shell commands associated with the target define the rule for creating a .o file from a .c file. A target with no slashes in it and beginning with a dot is identified as a rule and not as a true target.

The default rules for make appear in the rules.c source file for the make program. They can be modified locally. The following command is used to output the rules compiled into the make on any machine in a form suitable for recompiling:

```
make -pf - 2>/dev/null </dev/null
```

A tilde in the above rules refers to an SCCS file. Thus the rule .c.o would convert an SCCS C source file to an object file (.o). Since the s. of the SCCS files is a file name prefix, it is incompatible with the make suffix concept. Thus the tilde is a way of changing a file reference to an SCCS file reference.
A rule with only one suffix (e.g., .c:) defines how to build x from x.c. In effect, the other suffix is null. This is useful when building targets from only one source file (for instance, shell scripts or simple C programs).

Additional suffixes can be defined as a list with \texttt{.SUFFIXES}. The order of the list is significant: the first possible name for which a file and a rule are present is selected. The default list is:

\begin{verbatim}
.SUFFIXES .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~
\end{verbatim}

The above command for outputting the internal rules also displays this list of suffixes implemented on the current machine. Multiple suffix lists are cumulative; \texttt{.SUFFIXES:} with no dependencies clears the list of suffixes.

Thus the example shown under “Creating a makefile” on page 527 can be formulated more concisely:

\begin{verbatim}
pgm: a.o b.o
    cc a.o b.o -o pgm
a.o b.o: incl.h
\end{verbatim}

The default inference rules use certain macros to permit the inclusion of optional elements in the resultant command sequence. For example, \texttt{CFLAGS}, \texttt{LFLAGS} and \texttt{YFLAGS} are used for compiler options for \texttt{cc}[5], \texttt{lex} and \texttt{yacc} respectively.

The \texttt{.SCCS\_GET} directive allows you to modify the default commands for accessing SCCS files which are not in the current working directory. The default rule is:

\begin{verbatim}
.SCCS\_GET: sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@
\end{verbatim}

**Libraries**

If a target or prerequisite contains parentheses, it will be treated as a member of an archive library, where the string in parentheses refers to a member in the library.

Thus \texttt{lib(file.o)} and \texttt{\$(LIB)(file.o)} refer to an archive library that contains \texttt{file.o}. This assumes that the \texttt{LIB} macro has been defined previously. The expression \texttt{\$(LIB)(file1.o file2.o)} is not valid. Rules associated with archive libraries take the form \texttt{.XX.a}, where \texttt{XX} is the suffix of the file from which the archive member is to be made. Unfortunately, the current implementation requires \texttt{XX} to be different from the suffix of the archive member. Thus it is not possible to specify that \texttt{lib(file.o)} is dependent on \texttt{file.o}. 


The most common use of the archive interface follows. Here, it is assumed that the source files are all C-language source:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
     @echo lib is now up-to-date
  .c.a:
     $(CC) -c $(CFLAGS) $<
     $(AR) $(ARFLAGS) $@ $*.o
     rm -f $*.o
```

In fact, this `.c.a` rule is predefined in `make` and is superfluous in this example. A more interesting but more limited example of archive library maintenance is:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
     $(CC) -c $(CFLAGS) $(?:.o=.c)
     $(AR) $(ARFLAGS) lib $?
     rm $? 
     @echo lib is now up-to-date .c.a;:
```

In this case the macro substitution mechanism is used. The `$?` list is defined as the set of object file names (within `lib`) whose C source files are outdated. The macro substitution mechanism replaces `.o` with `.c`. Unfortunately it is not yet possible to translate to `.c~`, but this transformation may be implemented in the future. Also note the disabling of the `.c.a:` rule, which would have created each object file one after the other. This special construct considerably accelerates archive library maintenance, but it does become rather cumbersome if the archive library contains a mix of assembly programs and C programs.

**File**

[Mm]akefile and s.[Mm]akefile

```
/usr/bin/sh
```

**Hint**

Some commands return an inappropriate non-zero status. You can overcome this problem by using `-i` or the command line prefix `--`. File names containing the characters `= : @` cannot be processed. Commands which are directly executed by the shell, particularly `cd`, are ineffectual across new-lines in `make`. The syntax `lib(file1.o file2.o file3.o)` is invalid. It is not possible to build `lib(file.o)` from file.o.
Locale  The following environment variables affect the execution of `make`:

`LANG`  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding default value from the internationalized environment is used. If any of the internationalization variables contains an invalid setting, the command behaves as if none of the variables have been defined.

`LC_ALL`  If set to a non-empty value, override the values of all the other internationalization variables.

`LC_CTYPE`  Determine the locale for the interpretation of sequences of bytes of text data as characters (e.g. single-byte characters as opposed to multi-byte characters in arguments and input files).

`LC_MESSAGES`  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

`NLSPATH`  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

See also  `ar`, `lex`, `yacc`, `c89` [5]
The `man` command allows you to use the POSIX online documentation, in other words to have the syntax of a POSIX commands written to standard output.

**Syntax**

```bash
man[-k]...text...
```

- `-k` Scans the POSIX-internal database. This database contains a brief description of every command. Every line which contains the string `text` is output.

**text**

A keyword or the name of a standard utility. If `-k` is not specified and `text` does not represent one of the POSIX commands, `man` writes the following to standard output (see Example 2):

```
command text not supported
```

**Locale**

The following environment variables affect the execution of `man`:

- `LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding default value from the internationalized environment is used. If any of the internationalisation variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `LC_ALL` If set to a non-empty string value, override the values of all the other internationalisation variables.

- `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as characters (e.g. single-byte characters as opposed to multi-byte characters in arguments and in the summary database). The value of `LC_TYPE` need not affect the format of the information written about the text operand.

- `LC_MESSAGES` Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

- `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example 2**

```bash
man [-k] command
```
Example 1 Information about the `mkdir` utility:

```bash
$ man mkdir
Usage: mkdir [-m mode] [-p] dirname ...
(END) [a]
$
```

Example 2 Displaying all the commands which contain the string `mode`:

```bash
$ man mode
command mode is not supported
(END) [a]
$
$ man -k mode
Usage: lp [-cs] [-d destination] [-n copies]
[-o line-spacing='line-spacing-options']
[-o control-char-pos='control-char-pos-options']
[-o control-mode='control-mode-options']
[-t title] [ file-name ... | - ]
Usage: mkdir [-m mode] [-p] dirname ...
Usage: mkfifo [-m mode] file ...
Usage: stty [-a | -g]
stty modes
(END) [a]
$
```

See also `cat, more, lp`
mesg  

**mesg** permit or deny messages

`mesg` controls the receipt of messages on terminals that have logged on to POSIX with `rlogin` or `telnet`. You can use `mesg` to check whether your terminal can receive messages or to grant or deny other users the permission to send messages to your terminal with `write`.

**Syntax**

```bash
mesg[-y|n]
```

No option specified

`mesg` reports the current setting for your terminal and returns an exit status of 0 if messages may be sent to it, or 1 if they cannot.

**Option**

`y` Grants other users permission to write messages to your terminal.

This corresponds to the old argument `-y` which is still supported.

`n` Denies other users permission to write messages to your terminal. Messages sent via `wall` or `write` by the POSIX administrator override this denial.

This corresponds to the old argument `-n` which is still supported.

**Exit status**

- **0** Messages are receivable
- **1** Messages are not receivable
- **>1** Error

**File**

`/dev/tty*` and `/dev/term/tty*`

Files containing terminal-specific information.

**Locale**

The following environment variables affect the execution of `mesg`:

- **LANG**
  
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**
  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
**msg**

**LC_MESSAGES**

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**

Determine the location of message catalogs for the processing of LC_MESSAGES.

**Example 1**  User heidi, who is logged in at terminal tty001, checks the current setting for this terminal:

```
$ mesg
 is n
```

Messages are not receivable at terminal tty001. Thus no messages can be sent to heidi while she is working at terminal tty001:

```
$ write heidi
Permission denied.
```

**Example 2**  heidi can change the setting at terminal tty01 by entering:

```
$ mesg y
```

**See also**  tty, write, wall [12]
mkdir  make directories

`mkdir` is used to create new directories.

`mkdir` automatically makes the following standard entries in the new directory:

- . (dot) for the directory itself
- .. (dot dot) for the parent directory

To be able to use `mkdir` you have to have write permission in the parent directory.

Syntax

```
mkdir[[-m..mode][-p]..directory.....]
```

No option specified

`mkdir` creates the named directories in mode 777 (see section “chmod change file modes” on page 204) unless the `umask` command has been used to change the file-creation mode mask (see section “umask get or set the file mode creation mask” on page 778).

option

- `-m` (mode) The access permissions specified in `mode` are used for the new `directory` (see section “chmod change file modes” on page 204).

- `-p` (parent) `mkdir` creates any non-existing parent directories that are given in the path name of `directory` before creating the directory itself.

directory

Name of the directory that you wish to create. You can also create more than one directory at a time.

`directory` can be given either as a relative path name or as an absolute path name.

The user ID and the group ID of the new directory are set to the real user ID and real group ID of the calling process.

Exit status

- 0 If all directories given in the command line were made successfully.
- ≠ 0 If an error occurs. `mkdir` also prints an error message.

Locale

The following environment variables affect the execution of `mkdir`:

- `LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
mkdir  

If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example**  
Creating a new directory named *letters* in the directory `/home/sisyphus/general`:
First you check which directory you are in. The default directory mode is set to 777.

```bash
$ pwd
/home/sisyphus
```

Next you list the contents of this directory:

```bash
$ ls -l
total 145
-rw-r--r--   1 SISYPHUS   group1      5329   Nov 03 09:54 diff.rc.1
drwx--x--x   2 SISYPHUS   group1      2589   Aug 03 15:08 general
```

Then you create the new directory:

```bash
$ mkdir general/letters
```

Finally you check whether a directory named *letters* has been created:

```bash
$ cd general
$ ls -l
total 5
drwx---x---  2 SISYPHUS   group1      520   Jan 22 16:21 letters
```

**See also**  
`rm`, `rmdir`, `umask`
mkfifo  

make FIFO special files

`mkfifo` creates the FIFOs special files specified in its argument list. For every file argument, the `mkfifo` command behaves as if the `mkfifo` function (see `mkfifo` [4]) was called with the following arguments:

- the path argument of `mkfifo[4]` is set to file.
- the mode argument of `mkfifo[4]` has the value 0666 or the value of mode if the -m option is specified.

If an error occurs during the creation of a FIFO special file, `mkfifo` writes a diagnostic message to the standard error output and then continues with the remaining arguments, if any.

Syntax

```
mkfifo [-m mode]..file...
```

No option specified

`mkfifo` creates the FIFO special files specified in file with the access rights 666, modified by the current file-creation mode mask (see section “umask get or set the file mode creation mask” on page 778).

-m mode

`mkfifo` creates the new FIFO special file with the access rights specified in mode (see section “chmod change file modes” on page 204). If specified symbolically, the operands + and - are interpreted relative to a default value of a=rw.

file

The name of the FIFO special file that you would like to create. You can specify a number of FIFOs.

Locale

The following environment variables affect the execution of `mkfifo`:

- `LANG`
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `LC_ALL`
  If set to a non-empty string value, override the values of all the other internationalization variables.

- `LC_CTYPE`
  Is used for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
**mkfifo**

*LC_MESSAGES*  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

*NLSPATH*  
Determine the location of message catalogs for the processing of *LC_MESSAGES*.

See also  
`chmod`, `umask`  
`mkfifo [4]`
**mkfs**  

**make file system**

`mkfs` creates a ufs file system by writing on the special file `file`. `mkfs` waits 10 seconds before starting to build the file system. During this time you can abort the command by pressing the DEL key.

**Syntax**

```
mkfs[...-F ufs][-V][...-m|-s][-q][...-y][...-o specific options].file[size]
```

**Caution!**

The functionality of the `mkfs` command is incorporated in the POSIX installation program (see the manual "Basics for Users and System Administrators" [1]), so it is not described in further detail here. If you nonetheless need to make explicit use of `mkfs`, you will find a description of the command and its options in [12].
mknod make an inode

The command `mknod` creates a directory entry for a special file.

Syntax

**Format 1:**

```plaintext
mknod..name[..blc]..device_class..device_number
```

**Format 2:**

```plaintext
mknod..name..p
```

**Format 1**

`mknod..name[..blc]..device_class..device_number`

Only the POSIX administrator may enter this format.

- `name`
  - Name of the special file which is to be created

- `option`
  - `b` for a block-oriented special file
  - `c` for a character-oriented special file

- `device_class..device_number`
  - Number of the device class. This may be entered in octal or decimal notation. You must enter a leading zero if you choose octal notation. The device number assignment is system-specific.

Example of possible device classes (major device number):

- **58** Terminal file (`/dev/term/...`)
- **59** SF terminal file (`/dev/sf/...`)
  - (SYSFILE is used to access terminals from within BS2000 batch procedures)

It is not sufficient merely to create the TERM and SF special files. The appropriate POSIX parameters also have to be set if these files are to be used:

File: `$TSOS.SYSSSI.POSIX-BC.mnn` (`mnn = version, see the POSIX manual "Basics for Users and System Administrators" [1])

**Parameters:**

- **NOTTY** Maximum number of terminal connections that can be used
- **NOSSTY** Maximum number of SF terminal connections that can be used

These parameters only take effect when you restart the POSIX subsystem.
Format 2  \texttt{mknod \_name\_p}

\begin{itemize}
\item \texttt{name} \\
\hspace*{1em} Name of the special file
\item \texttt{p}  \hspace*{1em} Generates a FIFO file (known as a pipe)
\end{itemize}

Example  
Creating an additional special file (/dev/term). This example was conducted under the POSIX administrator's ID.

\begin{verbatim}
# cd /dev/term
/dev/term # ls -l /dev/term/511
crw-rw-rw- 1 SYSROOT TTY  58,511 Jan 14 2008 /dev/term/511
/dev/term # mknod /dev/term/512 c 58 512
/dev/term # chmod a+w /dev/term/512
/dev/term # ls -l /dev/term/512
crw-rw-rw- 1 SYSROOT SYSROOT 58,512 Jan 27 14:14 /dev/term/512
\end{verbatim}

See also  \texttt{mknod} [4]
more  

**display files on a page-by-page basis**

The `more` command is used to page through the contents of one or more files on the terminal.

`more` displays its output by scrolling up lines on the screen.

**Syntax**

| Format 1: more[\[-cdefisu\]][\[-n: number\]][\[-p: command\]][\[\] file]... |
| Format 2: more[\[-cdefisu\]][\[-n: number\]][\[+command\]][\[-lines\]][\[+line: number\] [\[+pattern\][\] file]... |

No option specified

The output is displayed one screenful at a time by default and can be controlled further with the commands described below. If you do not specify any of the options or built-in commands, `more` pauses after each screenful.

After the first screenful, (<filename>..%) appears as a prompt at the bottom of the screen.

If you press the space bar .. (on block-mode terminals ..[7]), `more` displays another screenful. If you press only .., the screen is scrolled one more line down.

`more` provides a two-line overlap between screens for continuity. If `more` is reading input from a file, the percentage of lines displayed so far is also shown (..%). This does not occur when `more` is used in combination with a pipe (\(\mid\))

Option

- `c` `more` clears the screen before displaying a new page; useful on faster displays. The `-c` option is ignored if the terminal does not have the ability to clear the screen.

- `d` On block terminals, comments and warnings referring to unavailable screen control facilities are suppressed.

- `e` Causes `more` to automatically exit the file the first time it reaches the end of the file. By default, `more` exits when it reaches the end of the file for the second time.

- `f` `more` does not fold long lines that extend beyond the right edge of the screen. The `-f` option cannot prevent line breaks automatically generated by the terminal hardware. `more` continues to count the displayed lines as if no line breaks have occurred.

- `i` Case is ignored during a search, i.e. lowercase and uppercase are regarded as the same. A search can also be conducted for overscored or underscored text. This option is ignored if the search pattern itself contains uppercase letters.

- `s` `more` compresses the output file by replacing multiple blanks with a single blank.
more

-u  more suppresses generation of underlining escape sequences.

-\texttt{u} not specified:
more handles underlining produced by text formatters such as \texttt{nroff} in a manner appropriate to the terminal. If the terminal can do underlining or has a stand-out mode, more supplies appropriate escape sequences as called for in the text file.

-n..number
Specify the number of lines per screenful. The \textit{number} argument is a positive decimal integer.

-p..command
The -p option in the command line is the same as specifying +/command, i.e. more executes \textit{command} each time at new file is displayed.
This is the same as the + \textit{command} that is also supported.

-lines
The value which you specify for \textit{lines} defines the number of lines in each screenful.

+line_number
more starts output of the file at the given \textit{line_number}.

+/pattern
\textit{pattern} is a regular expression which may be used to browse through a text file. The output begins two lines above the string which matches \textit{pattern}.
Unlike patterns used with editors, this construct should not end with a / (slash). If it does, the trailing slash will be interpreted as a character in the search pattern.

file
Name of the file or files to be displayed. If multiple files are given, the name of the current file is displayed in a header line before each file.

Commands
When more pauses at the end of a screenful, you can control subsequent output with the following commands:

n space_bar
more displays \textit{n} more lines. If you press the space bar (on block-mode terminals -\texttt{[ ])}
without a value for \textit{n}, the next screenful is displayed.

n\texttt{[} more displays \textit{n} more lines. If you press \texttt{[} without a value for \textit{n}, the next line is displayed.

n\texttt{CTRL]}d
nd
more displays \textit{n} more lines and sets the scroll size to \textit{n}.
more

**nz**  more sets the number of lines per screenful to \( n \) and displays the next screenful.

**ns**  more skips \( n \) lines of output and prints the text that follows. The omission is indicated by a comment at the skipped position.

**nf**  The number you specify for \( n \) defines how many screenfuls are to be skipped. The number of lines omitted as a result is indicated at the skipped position.

**CTRL** \( b \)  
**nb**  more skips back \( n \) screenfuls.

**q**  
**Q**  
**:q**  
**:Q**  
   The above four forms of Q terminate *more*.

**=**  more displays the current line number.

**v**  more calls the *vi* editor and skips to the current line of the current file. The current line is the last line displayed by *more*.

**h**  more displays a tabular overview of all *more* commands with a brief description of their functions.

**n/pattern**  
more searches forward for the \( n \)th occurrence of the regular expression *pattern*. If the given pattern occurs fewer than \( n \) times and if *more* is reading from a file rather than a pipe, the position in the file remains unchanged. If *more* is reading from a pipe, it will terminate.  
   If the search succeeds, *more* displays a new screenful starting two lines before the line containing the \( n \)th match.

**/n**  more searches for the \( n \)th occurrence of the last regular expression entered.

'  (single quote)  more goes to the point from which the last search for a regular expression was started. All subsequent single quotes are ignored. If no search has been performed yet, *more* goes to the start of the file.

**!shell_command**  
more invokes a new shell to execute the specified *shell_command*. Two additional variables may be used when defining *command*:

  %  is expanded to the current file name;

  !  is expanded to the previous shell command.  
   If you need to include a % or ! in the command, you must escape it by preceding it with a backslash.
more

n:n more skips forward to the nth next file named in the command line. If fewer than n files are listed, more skips to the last file; without a value for n, it skips to the next file in the given order. If you are already in the last file, more terminates.

n:p more skips back to the nth previous file named in the command line. If fewer than n files are listed, more skips to the first file. This command will not work if more is reading from a pipe.

:f more displays the current file name and line number.

. (dot) more repeats the previous command.

Functionality

The more command set the terminal to noecho mode. In other words, more's built-in commands are not displayed on your terminal when you enter them, except for regular expressions, the exclamation point, and the slash.
more operates in cbreak mode, which means that the commands listed above are processed as soon as they are completed and need not be confirmed with \]. This mode also causes both commands to automatically terminate at the end of the text file. Since more is generally more efficient at skipping forward than back, it is not very suitable for performing reverse searches on large files.
more discovers the terminal's display characteristics by reading the terminfo files.

File

/usr/share/lib/terminfo/*
File containing display characteristics.

/usr/lib/more.hlp
Help file

Variable

COLUMNS
Override the system selected horizontal screen size.

EDITOR
Used by the v command to select an editor.

LINES
Override the system selected vertical screen size. The -n option takes precedence over the LINES variable for determining the number of lines in a screenful.
Locale

The following environment variables affect the execution of `more`:

- **LANG**
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_COLLATE**
  Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.

- **LC_CTYPE**
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

- **LC_MESSAGES**
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example

Displaying the file named `test`, starting at the location of the word `example` and without folding long lines:

```
$ more -f +/example test
```

The output begins two lines above the first line which contains the word `example`.

See also `cat`, `ed`, `man`, `sh`
mount

mount a file system

`mount` integrates aufs file system into the file system hierarchy at the path name position `mountpoint`. This position must already exist. If `mountpoint` possesses any contents prior to the mount operation these remain hidden until the file system is dismounted again.

`mount` (Format 4 and Format 5, see page 554) mounts a bs2fs file system at a particular position in the POSIX file system. A bs2fs file system is understood to be a selectable set of files in BS2000 which are made available transparently in POSIX so that they can be accessed using POSIX means (commands, program interfaces). The files are selected via the user and catalog ID and wildcard symbols.

In addition, `mount` (Format 1) can be used to output a list of all the mounted file systems.

Please refer to the manual “NFS (BS2000/OSD)” [8] for details of how to mount nfs file systems.

Syntax

<table>
<thead>
<tr>
<th>Format 1: mount[...-v...-p]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2: mount[...-F..ufs][...-V][...-r][...-o..spec_options][...resource...mountpoint]</td>
</tr>
<tr>
<td>Format 3: mount[...-F..ufs][...-V][...-r][...-o..spec_options].resource..mountpoint</td>
</tr>
<tr>
<td>Format 4: mount[...-F..bs2fs][...-V][...-r][...-o..spec_options] {...resource...mountpoint...}</td>
</tr>
<tr>
<td>Format 5: mount[...-F..bs2fs][...-V][...-r][...-o..spec_options].resource..mountpoint</td>
</tr>
</tbody>
</table>

Display a list of mounted file systems

<table>
<thead>
<tr>
<th>Format 1: mount[...-v...-p]</th>
</tr>
</thead>
</table>

No option specified

`mount` displays a list of all mounted file systems (see Example).

option

- `-v` Displays a new presentation of the output. The new output contains the file system type and options in addition to the information in the old output. The fields `mountpoint` and `resource` change places (see Example).

- `-p` Displays a list of the mounted file systems in the `/etc/vfstab` format (see Example).
mount

mount ufs file systems

Format 2

```
mount[-F.ufs][-V][-r][-o.spec_options][.resource|.mountpoint]
```

Format 3

```
mount[-F.ufs][-V][-r][-o.spec_options].resource.mountpoint
```

The descriptions of Format 2 and Format 3 have been combined, since they differ only in terms of the (optional) specifications resource and mountpoint.

Format 2 can be used only if an entry for the relevant file system already exists in the /etc/vfstab file. The missing specification for resource or mountpoint is then added from this.

Formats 2 and 3 can be entered only by the POSIX administrator.

No option specified

mount displays a list of all mounted file systems.

-F_ufs

Specifies ufs as the file system type.

-V Displays the entire command line on screen but does not execute the command. The command line is displayed together with the options and arguments entered by the user and with the values derived from /etc/vfstab. This option allows you to check the general validity of the command line.

-r Mounts the file system with read access.

-o Specifies ufs file system-specific options. Multiple options should be comma-separated. If invalid options are specified a warning is issued and the invalid options are ignored. The following options may be selected:

- f Imitates an /etc/mntab entry but does not mount a file system. The parameters are not checked.

- n Mounts the file system without making an entry in /etc/mnttab.

- journal If there is no journal, one is created when the file system is mounted. This journal is used for a quick restart after a system crash. When the ro option is specified, the journal specification has no effect, i.e. there is no journaling in the event of read-only accesses.

- rw | ro Read/write or read only access. The default value is rw.

- nosuid By default, the file system is mounted in such a way that the s-bit is set for users. If you specify nosuid then the default value is deactivated and the file system is mounted without the s-bit being set for users.

- remount Is used together with rw. A file system which has been mounted with read access only can be remounted with read/write access. This option fails if the file system is not currently mounted or has been mounted with rw.
**bs2fscontainer**

Specifies the file system which is to be mounted as the bs2fs container, i.e. as a file system which temporarily accommodates files from bs2fs file systems.

This option may only be specified for a single ufs file system. Any other mount commando with this option is rejected.

The -r, -o ro, -o journal and -o remount options may not be specified together with the bs2fscontainer option.

When the POSIX installation program is used, this option can be entered via the option line.

This option can only be specified for a file system which was flagged as the bs2fs container when it was created with the POSIX installation program. When the append function is applied to a ufs file system which is to be created or overwritten, the bs2fscontainer option must be specified in the option line for this purpose.

If this is not done, the mount is aborted with an error and the file system, together with its content, is retained.

The ufs file system that serves as the bs2fs container is expected to be an empty file system. If it is not empty, its content is deleted using the –o bs2fscontainer option when the mount command is executed.

After a successful mount, two bs2fsd copy daemons are started automatically.

**resource**

specifies the file system which is to be mounted.

**mountpoint**

Specifies the local position for mounting resource. You must specify an absolute path name.
The descriptions of Format 4 and Format 5 have been combined, since they differ only in terms of the (optional) specifications resource and mountpoint.

Format 4 can be used only if an entry for the relevant file system already exists in the /etc/vfstab file. The missing specification for resource or mountpoint is then added from this (see also the note on page 556).

Formats 4 and 5 can be entered only by the POSIX administrator.

A prerequisite for entering a mount command of Format 4 or 5 is that a bs2fs container is already mounted.

No option specified
mount displays a list of all mounted file systems.

-F bs2fs
Specifies bs2fs as the file system type.

-V Displays the entire command line on screen but does not execute the command. The command line is displayed together with the options and arguments entered by the user and with the values derived from /etc/vfstab. This option allows you to check the general validity of the command line.

-r Mounts the file system with read access.

-o Specifies bs2fs file system-specific options. Multiple options should be comma-separated. If invalid options are specified a warning is issued and the invalid options are ignored. The following options may be selected:

rw | ro Read/write or read only access. The default value is rw.

nosuid The file system is mounted without setting the s bit for users. This option is enabled by default for bs2fs file systems and cannot be disabled.

remount Is used together with rw. A file system which has been mounted with read access only can be remounted with read/write access. This option fails if the file system is not currently mounted or has been mounted with rw.
**mount**

**ftyp=(text|binary|textbin)**

This option has the same effect as the *ftyp* command when copying files using the *bs2cp* command. It defines whether BS2000 SAM files and text-type POSIX library elements (element type other than L) are interpreted in POSIX as text or binary files. PAM files are always interpreted as binary files, ISAM files always as text files.

This option should only be specified once. If it is specified more than once, the specification with the highest priority applies, where *ftyp=textbin* has the highest priority, *ftyp=text* the next highest priority and *ftyp=binary* the lowest priority.

The default is *ftyp=text*.

**ftyp=text**

SAM files and text-type library elements are interpreted as text files. When writing to a bs2fs file, end-of-line characters (X‘15’) are converted to a record change and tabulator characters (X‘05’) to the corresponding number of blanks.

**ftyp=binary**

SAM files and text-type library elements are interpreted as binary files. A 1:1 transfer takes place without interpreting and converting data (record change/end-of-line characters, tabulator/blanks, etc.).

**ftyp=textbin**

SAM files and text-type library elements are interpreted as binary text files. When writing to a bs2fs file, only end-of-line characters (X‘15’) are converted to a record change. Tabulator characters (X‘05’) are not converted to blanks.

**resource**

Defines which BS2000 files are to be mounted. The following syntax applies for the option:

```
:cat: $user. filename-with-wild
```

The option can be entered in upper- or lowercase or in a mixture of both. Special characters of the POSIX shell such as ‘$’ or ‘*’ must be escaped explicitly.

<table>
<thead>
<tr>
<th>cat</th>
<th>Catalog ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>BS2000 user ID</td>
</tr>
</tbody>
</table>
filename-with-wild

BS2000 file name with wildcard symbols

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Replaces an arbitrary (even empty) string.</td>
</tr>
<tr>
<td>/</td>
<td>Replaces precisely one arbitrary character.</td>
</tr>
<tr>
<td>Terminating period</td>
<td>Partially-qualified entry of a name.</td>
</tr>
<tr>
<td><a href="">sx:sy</a></td>
<td>Replaces a string that meets the following conditions:</td>
</tr>
<tr>
<td></td>
<td>- It is at least as long as the shortest string (sx or sy)</td>
</tr>
<tr>
<td></td>
<td>- It is not longer than the longest string (sx or sy)</td>
</tr>
<tr>
<td></td>
<td>- It lies between sx and sy in the alphabetic collating sequence; numbers are sorted after letters (A…Z, 0…9)</td>
</tr>
<tr>
<td></td>
<td>- sx can also be an empty string which is in the first position in the alphabetic collating sequence</td>
</tr>
<tr>
<td></td>
<td>- sy can also be an empty string which in this position stands for the string with the highest possible code (contains only the characters X'FF')</td>
</tr>
<tr>
<td>&lt;s1,…&gt;</td>
<td>Replaces all strings that match any of the character combinations specified by s. s can also be an empty string. Any such string can also be a range specification “sx:sy”</td>
</tr>
<tr>
<td>-s</td>
<td>Replaces all strings that do not match the specified s. The minus sign may only appear at the beginning of the string.</td>
</tr>
</tbody>
</table>

The file set defined with resource can consist of both existing files and files which are to be created. When a new file is to be created, the required file name must match the wildcard pattern of the corresponding mount command.

mountpoint

Specifies the local position for mounting resource. You must specify an absolute path name. If mountpoint is a symbolic reference then the file system is mounted in the directory to which the symbolic reference points rather than being mounted alongside the symbolic reference.

Hint

If an entry for the file system concerned exists in the /etc/vfstab file, one of the options resource or mountpoint can be omitted (Format 2 or 4). When a bs2fs file system is used, the following must be observed in this case:

- If the mountpoint option is specified, an entry in /etc/vfstab can then be identified unambiguously and the corresponding file system is mounted.
– If only the resource option is specified, multiple suitable entries for a bs2fs file system can be contained in the /etc/vfstab file as this file system can be mounted in parallel at multiple locations. In this case only the first bs2fs file system entered in the /etc/vfstab file is mounted.

Only entries with an identical wildcard string are recognized as suitable entries. Entries with a different wildcard string are also not taken into account even if they define the same set of files.

File

/etc/mnttab
Table of mounted file systems.

/etc/dfs/fstypes
The default type of distributed file system.

/etc/vfstab
Table of automatically mounted file systems.

/etc/mnttab Table of mounted file systems

The file /etc/mnttab contains information about all the file systems mounted on the local computer. This file contains information which is generated by the mount command.

Each line contains the following information; items are separated by any number of blanks and/or tabs:

Format

<table>
<thead>
<tr>
<th>resource</th>
<th>mountp</th>
<th>fstype</th>
<th>spec-options</th>
<th>time</th>
</tr>
</thead>
</table>

resource Absolute path name of the mounted file system or, in the case of bs2fs file systems, mounted BS2000 files in wildcard syntax.

For bs2fs file systems the entry differs from the entry in the mount command as follows:
– It is converted completely into uppercase notation
– Escaped characters are displayed without the associated escape character

Example:

mount -F bs2fs -o ftyp=text :v70a:\$bach.sys\* /home/bach/bs2.1

generates the following entry in /etc/mnttab:

:V70A:\$BACH.SYS* /home/bach/bs2.1 bs2fs ftyp=text,suid,rw,noquota ...

mountp Absolute path name of the mount point.
Mount

fstype File system type.
spec-options Options as specified for the mount command.
time Mount time, given in seconds since 1.1.1970

Entries in the /etc/mnttab file are deleted again if the mount or umountall command is executed for corresponding file systems or file system types.

Example
Enter the following in the POSIX shell: cat /etc/mnttab

/dev/root / ufs rw,suid 802532552
/proc /proc proc rw, 802532553
/dev.fd /dev/fd fdfs rw 802532553
/dev/dsk/3 /var ufs suid,rw,noquota 802532558
/dev/dsk/2 /home1 ufs suid,rw,noquota 802532588
SINTEST1:/nfs /nfsclient ufs rw 802536261

/etc/vfstab Table of defined file systems
The /etc/vfstab file describes every file system which is defined on the local computer. You can edit this file with an editor.

The file systems for which yes is entered in the automnt column of the /etc/vfstab file are mounted automatically when POSIX starts or by the mountall command.

In addition, the entries in the file are used to complement any missing details for resource or mount options when a mount command is executed.

Corresponding entries are generated automatically in the /etc/vfstab file for ufs file systems which are defined using the POSIX installation program. For all other file systems (e.g. bs2fs or nfs file systems) the entries must be created manually when required.

In contrast to the /etc/mnttab file, execution of the mount and umount commands for the /etc/vfstab file has no repercussions. Corresponding entries are retained.

The fields in the table are separated by blanks and/or tabs. A hyphen (−) indicates a blank entry in the relevant field. The table contains the following fields:

Format

special fsckdev mountp fstype fsckpass automnt mntopts

special Describes the resource to be mounted.
The following must be borne in mind in the case of (manual) entries for bs2fs file systems:

- Letters may only be specified in uppercase notation
- Special characters may not be escaped, nor is it permissible to enclose the string in quotes

`fsckdev` Name of the block-oriented device or of the resource of the character-oriented device.

`mountp` Mount point: absolute path name of the directory in which the resource is to be mounted.

`fstype` File system type.

`fsckpass` The pass number to be used for multiple `fsck` commands

`automnt` Specifies whether (yes) or not (no) the resource is to be mounted automatically by `mountall` at POSIX startup time.

`mntopt` List of options separated by commas for mounting the file system. The options are the same as the specific options of the `mount` command.

**Example**

Enter the following in the POSIX shell:

```
cat /etc/vfstab
```

```plaintext
/dev/root /dev/rroot / ufs 1 yes -
/proc - /proc proc - no -
/dev/fd - /dev/fd rdfs - no -
/dev/dsk/3 /dev/rdsk/3 /var ufs 1 yes -
PGB00004:home2/froede/SHARE - /home/froede/RETSINA nfs - no soft
PGB00004:home2/froede/SHARE - /home/froede/PGB0004 nfs - no soft
/dev/dsk/4 /dev/rdsk/4 /home/froede ufs 1 yes -
/dev/dsk/10 /dev/rdsk/10 /home/gast ufs 1 yes -
/dev/dsk/13 /dev/rdsk/13 /mnt/ascii ufs 1 no -
/dev/dsk/8 /dev/rdsk/8 /mnt/dat1 ufs 1 no -
/dev/dsk/23 /dev/rdsk/23 /bs2fscont ufs 1 no -
/dev/dsk/24 /dev/rdsk/24 /home/bach mount3 ufs 1 no -
/dev/dsk/25 /dev/rdsk/25 /home/bach/mountxxx ufs 1 no -
/dev/dsk/26 /dev/rdsk/26 /home/bach/mountyyy ufs 1 no -
/dev/dsk/5 /dev/rdsk/5 /home/bach ufs 1 yes -
/dev/dsk/2 /dev/rdsk/2 /home/bach/mount99 ufs 1 yes -o
/dev/dsk/6 /dev/rdsk/6 /suderlan ufs 1 no -
:V70A:BACH.ASS.*.S - /home/bach/bs2.1 bs2fs 1 yes ftyp=binary
:V70A:BACH.CCC.*.C - /home/bs2.2 bs2fs 1 yes ftyp=text
:V70A:BACH.PLAMLlib* - /home/bach/bs2.2 bs2fs 1 yes ftyp=textbin
:V70A:BACH.SEM*.C - /home/bs2000 bs2fs 1 yes -
```
Example 1  Displaying the mounted file systems (Format 1) and mounting a new file system (Format 2). This example was conducted under the POSIX system administrator’s ID.

```
# mount
/dev/root on / type ufs read/write/setuid on Thu Jun 4 09:17:49 2009
/proc on /proc type proc read/write on Thu Jun 4 09:17:49 2009
/dev/fd on /dev/fd type fdfs read/write on Thu Jun 4 09:17:49 2009
/dev/dsk/2 on /tmp type ufs setuid/read/write/noquota on Thu Jun 4 09:17:50 2009
/var on /dev/dsk/3 type ufs setuid/read/write/noquota on Thu Jun 4 09:17:49 2009
/home on /dev/dsk/4 type ufs setuid/read/write/noquota on Thu Jun 4 09:17:51 2009
/home1 on /dev/dsk/5 type ufs setuid/read/write/noquota on Thu Jun 4 09:17:51 2009
/home2 on /dev/dsk/6 type ufs setuid/read/write/noquota on Thu Jun 4 09:17:51 2009

# mount -p
/dev/root - / ufs - no rw,suid
/proc - /proc proc - no rw
/dev/fd - /dev/fd fdfs - no rw
/dev/dsk/2 - /tmp ufs - no suid,rw,quota
/var - /var ufs - no suid,rw,quota
/home - /home ufs - no suid,rw,quota
/home1 - /home1 ufs - no suid,rw,quota
/home2 - /home2 ufs - no suid,rw,quota

# mount -v
/dev/root on / type ufs read/write/setuid on Thu Jun 8 09:17:49 2009
/proc on /proc type proc read/write on Thu Jun 4 09:17:49 2009
/dev/fd on /dev/fd type fdfs read/write on Thu Jun 4 09:17:49 2009
/dev/dsk/2 on /tmp type ufs setuid/read/write/noquota on Thu Jun 4 09:17:50 2009
/var on /var type ufs setuid/read/write/noquota on Thu Jun 4 09:17:49 2009
/home on /home type ufs setuid/read/write/noquota on Thu Jun 4 09:17:51 2009
/home1 on /home1 type ufs setuid/read/write/noquota on Thu Jun 4 09:17:51 2009
/home2 on /home2 type ufs setuid/read/write/noquota on Thu Jun 4 09:17:51 2009

# mount -F ufs /dev/dsk/17 /home3
# mount -p
/dev/root - / ufs - no rw,suid
/proc - /proc proc - no rw
/dev/fd - /dev/fd fdfs - no rw
/dev/dsk/2 - /tmp ufs - no suid,rw,quota
/var - /var ufs - no suid,rw,quota
/home - /home ufs - no suid,rw,quota
/home1 - /home1 ufs - no suid,rw,quota
/home2 - /home2 ufs - no suid,rw,quota
/dev/dsk/17 - /home3 ufs - no suid,rw,quota
```
Example 2  Mounting the bs2fs container and a bs2fs file system. The example executes under the
POSIX administrator ID.

    # mount -F ufs -o bs2fscontainer /bs2fscont
    # mount -F bs2fs ':V70a:$sysaudit.sys.conslog.2007-06*'  /home/bs2.conslog
    # mount

See also  umount, mountall
          mount(), umount() [4]
mountall

mountall mount file systems

mountall is used to mount file systems on the basis of a file_system_table (/etc/vfstab is the
default file system table). The special file name "-" reads from the standard input. If you
specify the hyphen then the standard input must possess the same format as /etc/vfstab.

Before the individual file systems are mounted, fsck performs a plausibility test to determine
whether the system appears to be viable for mounting (not in the case of file systems of the
type bs2fs or nfs). If the system is not viable for mounting, fsck corrects it before an attempt
is made to mount it.

If only file_system_type is specified then mountall applies only to file systems of the specified
type.

The file systems are mounted in the order ufs - bs2fs - nfs. This ensures that when the bs2fs
file systems are mounted, the bs2fscontainer file system required for this purpose is already
mounted in ufs.

Syntax

```
mountall [-F file_system_type] [-l | -r] file_system_table
```

No option specified

mountall mounts all file systems for which the field automnt in the file system table is set
to yes.

Options

- `-F file_system_type`
  Specifies the type of file system to be mounted.

- `-l`
  Limits the process to local file systems (ufs and bs2fs).

- `-r`
  Limits the process to remote file system types (nfs).

file_system_table

If file_system_table is not specified, mountall refers to /etc/vfstab.

Hint

If the `-F` option is specified together with one or more of the options `-l`, `-r` and `-b` and
the options are mutually compatible, the `-l`, `-r` and `-b` options have priority. For example,
mountall -F bs2fs -l and mountall -F ufs -l have the same effect as mountall -l: all local file
systems (i.e. all ufs and bs2fs file systems) are mounted. The entries mountall -F bs2fs and
mountall -b also lead to the same result: all bs2fs file systems are mounted.

Error

If the file systems are viable for mounting and error-free, no message is output. Error and
warning messages are issued by fsck and mount or by mountall in the case of incorrect
syntax.
mountall

File

/etc/fstab
Default file system table.

See also
fsck, mount, umountall
mv

**move files**

`mv` is used to change the name of a file, or to move a file from one directory to another within the file tree. To be able to use `mv` you have to have write permission for the directory in which the file resides or the directory to which you want to move it, as appropriate.

`mv` does not make copies of files that are moved or renamed within the same file system. Instead, it simply modifies the appropriate entries in the parent directory, retaining any links to other files.

If a file is moved to another file system, however, `mv` uses the `cp` command. In this case the original file is first copied and then deleted, and all links to other files are lost.

**Syntax**

| Format 1: | `mv[[-f][-i]]..file..newfile` |
| Format 2: | `mv[[-f][-i]]..file......dir` |
| Format 3: | `mv[[-f][-i]]..dir..newdir` |

**Format 1**

**Renaming a file**

`mv[[-f][-i]]..file..newfile`

No option specified

If you specify an existing file for `newfile` and do not have write permission for it, `mv` displays the mode of `newfile` and prompts you to confirm whether it should proceed. It only does so if your answer begins with `y`.

⚠️ **Caution!**

If the standard input is not a terminal, no prompt for confirmation is displayed and `newfile` will not be overwritten.

**options**

- `-f` If a file named `newfile` already exists, `mv` will overwrite the existing file even if you do not have write permission for it.

- `-i` (interactive) If you specify an existing file for `newfile`, you will always be asked to confirm whether `mv` should really proceed.

If you enter more than one specification for `-f` or `-i` then no error occurs. The final option to be specified determines the behavior of `mv`.

**file** Name of the file to be renamed.
newfile
New name for the file; must not be the same as file. If a file named newfile already exists, it is overwritten by the contents of file if you have write permission for newfile (see also -i).

If you specify an existing file for newfile and do not have write permission for it, mv will display the mode of newfile and prompt you to confirm whether it should proceed. The existing newfile is not overwritten unless you answer y. If the -f option is specified, confirmation will not be requested and newfile will be overwritten.
If the standard input is not a terminal, confirmation will not be requested and newfile will not be overwritten.

If the parent directory of newfile is writable but has the sticky bit (the t bit) set, one or more of the following conditions must be satisfied:
- the user must own the file
- the user must own the directory
- the user must have write permission for the file
- the user must be a privileged user

Format 2  Moving files and directories to another directory

mv[\[-f][\[-i]\]..file...dir

option
see format 1

file Names of files or directories to be moved to directory dir. If you name a directory as source, all the files and directories under it are moved recursively.

dir Name of the directory to which the files and/or directories are to be moved. You need write permission for this target directory.

If dir is writable but has the t bit (sticky bit) set, one or more of the following conditions must be satisfied:
- the user must own the file
- the user must own the target directory
- the user must have write permission for the file
- the user must be a privileged user
Format 3  Renaming a directory

```
mv[-f][-i]...dir...newdir
```

option

- see format 1

- **dir** Name of the directory to be renamed.
- **newdir** New name for the directory.

- *dir* and *newdir* must belong to the same physical file system; however, they do not have to share the same parent directory.

- If a directory named *newdir* already exists, the directory named *dir* is moved to the *newdir* directory.
- If *newdir/dir* already exists then *mv* only moves the directory *dir* if *newdir/dir* is empty.

Error

```
mv: Cannot rename <dir> to <newdir> : Permission denied
You do not possess write permission for the directory <dir> which is to be renamed.
mv: Cannot create <file> : Permission denied
You do not possess write permission for the directory to which <file> is to be moved.
```

Locale

The following environment variables affect the execution of *mv*:

- **LANG** Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL** If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_COLLATE** Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements used in the extended regular expressions defined for yes/no queries.

- **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files), the behavior of character classes used in extended regular expressions defined for yes/no queries.

- **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
Built-in sh command

**mv**

**NLSPATH** Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example 1** The file *songs* in the current working directory is to be renamed *popsongs* and moved to the directory `/home/joanne/arts/music`.

```
$ mv songs /home/joanne/arts/music/popsongs
```

**Example 2** The files *daisy*, *rose* and *violet* in the current working directory are to retain their names and be moved to the directory `/home/joanne/flowers`.

```
$ mv daisy rose violet /home/joanne/flowers
```

**See also** `chmod`, `cp`, `find`, `ln`, `rm`
newgrp change to a new group

The POSIX shell built-in `newgrp` overlays the current shell with `/bin/newgrp`. The `/bin/newgrp` command makes the specified group ID number your current group ID and overlays itself with a new shell. You terminate the shell in which you called `newgrp` by hitting the [END] key.

⚠️ **Caution!**

If `newgrp` is killed with the [DEL] key while `/bin/newgrp` is replacing the current shell, the shell from which `newgrp` was called is terminated as well.

The `newgrp` command can thus be used to switch to another user group. This means:
- your access permissions for existing files will be changed to those of the new group affiliation;
- group access permissions for new files that you create will be those of the group to which you have switched.

Only the variables that you exported earlier (see section “export set export attribute for variables” on page 363) will be known in the (new) current shell after the group has been changed. Variables that have not been exported are either considered undefined or assigned a default value by the shell (see section “sh shell, the standard command language interpreter” on page 679). Shell variables such as `PATH` and `HOME` are also assigned default values, unless they have already been exported by the system or were explicitly exported by you.

**Before the call**

The `/etc/group` file must contain an entry for the group that you wish to switch to. Otherwise, `newgrp` will terminate with an error message.

The `newgrp` command can be used to switch to any group to which you belong, i.e. any group for which your login name appears in the `/etc/group` file under the corresponding group entry.

If the error messages "Sorry" and/or "Unknown group" occur then `newgrp` aborts. In the event of any other error messages the shell terminates.

**Syntax**

**Format 1:** `newgrp[-l][...group]`

**Format 2:** `newgrp[-][...group]`

The two formats are described together since the option `-l` in format 1 is equivalent to option `-` in Format 2.

No argument specified

Changes you back to the group whose group ID (GID) has been entered for your login name in the `/etc/passwd` file.
the newgrp command overlays the current shell with a login shell. Before this shell displays its prompt, it first executes the /etc/profile and your $HOME/.profile (if present) and switches to your home directory.

In other words, except for the fact that you are now a member of a new group (the one specified on invoking newgrp), you continue working in the same environment as the one that applied after you logged into the system.

– or -l not specified:
newgrp overlays the current shell with /bin/newgrp. The current directory is not changed; however, the new shell will not be aware of variables that have not been explicitly exported. Unexported variables are either undefined or assigned a default value by the shell.

group
Name of the group you wish to switch to. There must be an entry for this group name in the /etc/group file. The associated group ID (GID) must already be associated with a login name in the /etc/passwd file.

If you are not a member of the specified group, there must be a password defined for the group in the /etc/group file. newgrp expects you to enter this password before it switches groups.

If no password is entered for your login name then a password for the group must be present in the file /etc/group. The command newgrp expects a password to be input before changing to the relevant group.

If you wish to change back to the user group which is entered for you, call newgrp without specifying a group name.

group not specified:
You change back to the group whose group ID (GID) is entered for your login name.

Exit status always 0

Error
Unknown group
This name has not been entered in the /etc/group file.

Sorry
You are not permitted to switch to this group, since you are not a member of the group, and no password has been defined for it either.

File /etc/group
Defines a name for the entered group IDs and determines all the members of this group.
newgrp

The following environment variables affect the execution of newgrp:

**LANG**  
Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**  
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  
Determine the location of message catalogs for the processing of LC_MESSAGES.

**Example**  
Change to the group with the group name council:

```
$ newgrp council
$ >newfile
$ chmod 640 newfile
$ ls -lg newfile
-rw-r----- 1 ROSE council 162 Mar 19 18:34 newfile
```

The new file (newfile) created after the group change is available to members of the council group as a read-only file.

**See also** exec, export
nice

invoke a utility with an altered system scheduling priority

⚠️ Caution!
The `nice` command is supported only on grounds of compatibility. It has no effect on BS2000 task priorities. For that reason only the syntax chart is given here. The options, arguments and so forth are not described.

Syntax

| Format 1: nice[-n..incrementl].command[...argument] |
| Format 2: nice[-..increment].command[...argument] |
**nl** line numbering filter

The *nl* command reads lines from a file or the standard input and writes them on the standard output with line numbering.

*nl* views the text it reads in terms of logical pages. A logical page consists of a header, a body, and a footer section. Empty sections are valid.

The start of a header, body, and footer of a logical page is normally signaled by an input line containing nothing but one of the following strings:

<table>
<thead>
<tr>
<th>Line</th>
<th>Start of</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>header</td>
</tr>
<tr>
<td>--</td>
<td>body</td>
</tr>
<tr>
<td>-</td>
<td>footer</td>
</tr>
</tbody>
</table>

If the input text does not contain any delimiter characters, *nl* assumes that the text being read is in a single logical page body.

Line numbering is reset at the start of a logical page (exception: option `-p`). Different line numbering options are independently available for header, body, and footer (e.g. no numbering of header and footer lines, while numbering blank lines only in the body).

**Syntax**

```
nl[...option][...file]
```

No option specified

*nl* numbers all logical page body lines that contain printable text, but not header and footer lines.

Line numbering is reset to 1 at the start of each logical page.

Within a logical page, *nl* numbers lines in increments of 1.

Each line number may be up to 6 positions long and is output right justified without leading zeros. A tab is used to separate line numbers from text.

**option**

The options must be entered individually, i.e. separated from other options by blanks.

The name of the file may be specified before, between, or after the options. The position of the file name in the command line has no effect on the operation of the *nl* command.
Selecting the lines to be numbered

-b..type
   (b -body) Specifies which logical page body lines are to be numbered.
   *type* can be any of: *a*, *n*, *preg_expr*, or *t*.
   
   *a*     All lines are numbered.
   *n*     No lines are numbered.
   *preg_expr*
      Numbers all lines containing strings that match the given *reg_expr*.
      *reg_expr* is a simple regular expression (see *Tables and Directories, Regular
      POSIX shell expressions*). If the specified regular expression contains shell
      metacharacters, it must be enclosed in single quotes:
      *p'reg expr'.
   *t*     All lines with printable text are numbered.

   -b not specified:
      Only the lines with printable text are numbered. Option -b with the *t* argument is thus the
      default.

-f..type
   (f -footer) Specifies which logical page footer lines are to be numbered.
   *type* may be: *a*, *n*, *preg_expr*, or *t*.

   -f not specified:
      Logical page footer lines are not numbered. Option -f with the *n* argument is thus the
      default.

-h..type
   (h - header) Specifies which logical page header lines are to be numbered.
   *type* may be: *a*, *n*, *preg_expr*, or *t*.

   -h not specified:
      Logical page header lines are not numbered. Option -h with the *n* argument is thus the
      default.
Restarting line numbering:

- **p** Line numbering is not restarted at logical page delimiters.
- **v** startnum
  Line numbering is restarted at `startnum` at the start of each logical page.
  `startnum` may be any number greater than or equal to 0.
- **v** not specified:
  The line counter is reset to 1 at the start of each logical page.

Defining the increment:

- **i** incr
  `incr` is the increment value used to number logical page lines.
  - **i** not specified:
    The default increment is 1.

Defining the output format:

- **n** format
  `format` is the line numbering format.
  The recognized values are: `ln`, `rn`, or `rz`.
  - **n** not specified:
    Line numbers are printed right justified, without leading zeros. In other words, the `-n` option with argument `rn` is the default.
- **s** sep
  Defines `sep` as the separator between line numbers and the corresponding text lines.
  `sep` may be one or more characters.
  - **s** not specified:
    By default, line numbers and text lines are separated by a tab character.
- **w** n
  Defines `n` as the number of positions for line numbers.
  The maximum value for `n` is 100. If you specify a higher value, a value of 100 is assumed.
  - **w** not specified:
    Individual line numbers may be up to 6 positions long.
Numbering blank lines

-\( n \)

`nl` treats \( n \) consecutive blank lines as a single blank line.

Example

\$ nl -b a -l 2

results in only every second blank line being numbered when there are several consecutive blank lines (none of the header and footer lines are numbered).

-\( l \) not specified:
Each blank line is interpreted as one full line (\( n = 1 \)).

Defining delimiters for header, body and footer

-\( d \)[x]y]

Changes the delimiter characters that specify the start of a header, body or footer section from `:` to the string `xy`.
If a backslash (`\`) is to be specified for `x` or `y`, it must be escaped by single quotes or a second backslash, e.g. `-d'\*'` or `-d\*'`.

y not specified:
The start of a header, body or footer section is identified by the string `x:` instead of `:`; i.e. the second character retains its default value.

file
Name of the input file.

file not specified:
`nl` reads from standard input.

Locale
The following environment variables affect the execution of `nl`:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_COLLATE**
Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, and for deciding which characters are in a certain character class for the -b, -f and -h options.

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example**

The file *poems* has the following contents:

```
\:\:\:\:
Taken from: Do you love me?
\:\:\:
The trouble with you
' s you' ve lost a screw
I' m sorry it' s you
but there' s nothing to do
There' ll be no abatements
there are no replacements
don' t make a to-do
just say toodle-oo
I' m sorry I can' t help you
you' d cost too much to redo
you' ll have to be abolished
report to be demolished
\:
Ronald Laing
\:\:\:\:
Limerick
\:\:\:
A limerick packs laughs anatomical
into space that is short and economical;
But the good ones I' ve seen
so seldom are clean,
And the clean ones so seldom are comical.
\:
Author unknown
```
Call `nl` without options

```
$ nl poems

Taken from: Do you love me?
1  The trouble with you
2  's you've lost a screw
3  I'm sorry it's you
4  but there's nothing to do
5  There'll be no abatements
6  there are no replacements
7  don't make a to-do
8  just say toodle-oo
9  I'm sorry I can't help you
10 you'd cost too much to redo
11 you'll have to be abolished
12 report to be demolished

Ronald Laing

Limerick
1  A limerick packs laughs anatomical
2  into space that is short and economical;
3  But the good ones I've seen
4  so seldom are clean,
5  And the clean ones so seldom are comical.

Author unknown
```
Number lines in increments of 10

$ nl -v10 -i10 poems

Taken from: Do you love me?

10 The trouble with you
20 's you've lost a screw
30 I'm sorry it's you
40 but there's nothing to do
50 There'll be no abatements
60 there are no replacements
70 don't make a to-do
80 just say toodle-oo
90 I'm sorry I can't help you
100 you'd cost too much to redo
110 you'll have to be abolished
120 report to be demolished

Ronald Laing

Limerick

10 A limerick packs laughs anatomical
20 into space that is short and economical:
30 But the good ones I've seen
40 so seldom are clean,
50 And the clean ones so seldom are comical.

Author unknown

See also ed, pr
**nm** write the name list of an object file

The utility *nm* displays symbolic information appearing in the object file, executable file or object-file library named by *file*.

If no symbolic information is available for a valid input file, the *nm* utility will report that fact, but not consider it an error condition.

**Syntax**

```
   nm [\[-APv\][-efox][-g][-u][-t_format]..file...
```

**options:**

- **-A** Write the full pathname or library name of an object on each line.
- **-e** Write only external (global) and static symbol information.
- **-f** Produce full output. Write redundant symbols (.text, .data and .bss), normally suppressed.
- **-g** Write only external (global) symbols.
- **-o** Write numeric values in octal (equivalent to `-t-o`)
- **-P** Write information in a portable output file, as specified in Standard output.
- **-t_format**
  Write each numeric value in the specified format. The format is dependent on the single character used as the *format* option-argument:
  - **d** The offset is written in decimal (default)
  - **o** The offset is written in octal (corresponds to `-o`)
  - **x** The offset is written in hexadecimal (corresponds to `-x`)
- **-u** Write only undefined symbols.
- **-v** Sort output by value instead of alphabetically.
- **-x** Write numeric values in hexadecimal (equivalent to `-t-x`).

**file**

A pathname of an object file, executable file or object-file library.
Standard output

If symbolic information is present in the input files, for each file or for each member of an archive, the nm utility will write the following information to standard output:

- Library or object name, if -A is specified.
- Symbol name.
- Symbol type, which is one of the following single characters:
  - A: Global absolute symbol
  - a: Local absolute symbol
  - B: Global "bss" symbol
  - b: Local "bss" symbol
  - D: Global data symbol
  - d: Local data symbol
  - T: Global text symbol
  - t: Local text symbol
  - U: Undefined symbol
- Value of the symbol
- The size in bytes associated with the symbol, if applicable.

If the -P option is specified, the previous information is displayed using the following portable format. The three versions differ depending on whether was specified in decimal, octal or hexadecimal format.

If -t is not specified, the format is as if -t..x has been specified.

```
%s%s %s %d %d
```

where library-/object name is formatted as follows:

- If -A is not specified, library-/object name is an empty string.
- If -A is specified and the corresponding file operand does not name a library:
  
  ```
  "%s: ': , file
  
  - If -A is specified and the corresponding file operand names a library. In this case, object file names the object in the library containing the symbol being described:
  
  "%s[%s]: ': , file , object file
```
Locale

The following environment variables affect the execution of `nm`:

**LANG**

Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**

If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_COLLATE**

Determine the locale for character collation information for the symbol-name and symbol-value collation sequence.

**LC_CTYPE**

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**

Determine the location of message catalogs for the processing of `LC_MESSAGES`.

See also

`ar`

`c89 [5]`
**nohup invoke a utility immune to hangups**

`nohup` executes a command or shell script, ignoring the signals SIGHUP (signal number 1) and SIGQUIT (signal number 3). `nohup` can be used to prevent a process from being terminated when you log out from a terminal.

**Before the call**

If you wish to use `nohup` with a pipe or a list of commands, you must put the relevant pipeline or command list in a file and run the file as a shell script under `nohup`'s management.

**Syntax**

```
nohup command[...argument...]
```

Any command or shell script. You can pass arguments to `command` by entering them after the command as usual. If the standard output and standard error of `command` have not been redirected, they will be written to a file called `nohup.out` in the current directory.

If you are not permitted to create the file `nohup.out` in the current directory (i.e. you have no write permission for this directory) or if you do not have write permission for the file `nohup.out` in the current directory, the output will be redirected to the file `$HOME/nohup.out`.

If the file `nohup.out` or `$HOME/nohup.out` does not exist, a corresponding file is created and the file's permission bits will be set only for the calling user; otherwise, the output is appended to the existing file.

**Exit status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>The utility specified by <code>command</code> was found but could not be invoked.</td>
</tr>
<tr>
<td>127</td>
<td>An error occurred in the <code>nohup</code> utility or the utility specified by <code>command</code> could not be found.</td>
</tr>
</tbody>
</table>

**Error**

`nohup: cannot open/create nohup.out`

You are not allowed to create a file called `nohup.out` in the current working directory or in `$HOME`, or you are not allowed to write to existing `nohup.out` files in those directories.

**File**

`nohup.out`

File located in the current directory which receives the standard output and standard error output from commands executed under `nohup`. 
$HOME/nohup.out
The standard output and standard error output of commands executed under nohup are sent to this file if the nohup.out file in the current directory is write-protected or if you are not allowed to create this file in the current directory (because you do not have write permission there).

Variable
HOME
The value of the environment variable HOME is used to determine the directory in which the file nohup.out is to be created in cases where nohup.out in the current directory is write-protected or where you are not allowed to create files in the current directory.

PATH
Determine the search path that will be used to locate the utility invoked.

Locale
The following environment variables affect the execution of nohup:

LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH
Determine the location of message catalogs for the processing of LC_MESSAGES.

Example
You initiate the following job and set it to run in the background:

```bash
$ nohup programm &
```

sending output to nohup.out

You then log out. programm is not terminated, since it is immune to the quit signal. The output is written to a file called nohup.out.

See also
chmod, kill, nice, sh
signal() [4]
The *od* command dumps (i.e. writes the contents of) a file on the standard output in an output format that you select with command-line options.

The first column of each output line specifies the position of the first character in this line. Characters can be specified in octal, decimal or hexadecimal, depending on the indicated output format.

Syntax

**Format 1:** `od[-v][-A_addr_base][-j_skip][-N_count][-t_type_string][...file...]

**Format 2:** `od[-bcdDfoosSvxX][...file][+[offset][.]b]]`

**Format 1**

- **-v** (verbose) Shows all data.
  - If not specified, all output lines which are identical to the immediately preceding output lines are replaced by a line which contains only an asterisk (*).

- **-A_addr_base**
  - Identifies the offset base for the input. `addr_base` is a character. The characters o, d and x mean that the offset base is written in octal, decimal or hexadecimal notation. The character n means that no offset is written.

- **-j_skip**
  - Identifies the starting position for the output. The next `skip` bytes from the start of the input are jumped. If the input is not at least `skip` bytes in length, an error message is output. By default, `skip` is interpreted as a decimal number. If `skip` begins with 0x or 0X then the offset is interpreted as a hexadecimal number. If `skip` begins with a leading zero then the offset is interpreted as an octal number. If this is followed by one of the characters b, k or m then the offset is interpreted as a multiple of 512, 1024 or 1048576 bytes.

- **-N_count**
  - Only `count` bytes of the input are formatted. By default, `count` is interpreted as a decimal number. If `count` begins with 0x or 0X then the offset is interpreted as a hexadecimal number. If `count` begins with a leading zero then the offset is interpreted as an octal number. If fewer than `count` bytes are present, no error message is output and *od* simply formats the available input.

- **-t_type_string**
  - Identifies one or more output types. `type_string` consists of a string which identifies the types used for input. The string must consist of the type-specific characters a (named character), c (character), d (decimal), f (floating), o (octal), u (unsigned decimal) und x (hexadecimal).
file
Name of the file to be dumped.

file not specified:
od reads from standard input.

Format 2  od[--bcdDfFoOsSvxX][[-][+][offset[.]][b]]

No option specified
Each group of 2 bytes is interpreted as an unsigned octal number (same as option -o).

option
If you use several options in order to combine different output formats, the option
sign (-) may only be specified once, and the option names must be grouped together
without intervening blanks, e.g: od -bcs file.

-b   Interprets each byte as an octal number.
-c   Interprets each byte in accordance with the current setting for LC_CTYPE.
-d   Interprets each group of 2 bytes as an unsigned decimal number.
-D   Interprets each group of 4 bytes as an unsigned decimal number.
-f   Interprets each group of 4 bytes as a floating-point number.
-F   Interprets each group of 8 bytes as an extended precision number.
-o   Interprets each group of 2 bytes as an unsigned octal number.
-O   Interprets each group of 4 bytes as an unsigned octal number.
-s   Interprets each group of 2 bytes as a signed decimal number.
-S   Interprets each group of 4 bytes as a signed decimal number.
-v   (verbose) Shows all data.

   -v not specified
All output lines which are identical to the immediately preceding output lines are
replaced by a line which contains only an asterisk (*).

-x   Each 2-byte sequence is interpreted as an unsigned hexadecimal number.
-X   Each 4-byte sequence is interpreted as an unsigned hexadecimal number.

file
Name of the file to be dumped.

file not specified:
od reads from standard input.
The offset argument specifies the offset in the file where dumping is to commence. Offset is normally interpreted in octal bytes. If a period (.) is appended to the offset argument, the offset is interpreted in decimal. If b is appended, the offset is interpreted in blocks of 512 bytes.

If the file argument is omitted, the offset argument must be preceded by a plus sign (+) to prevent it from being interpreted as a file name.

Offset not specified:
Dumping commences at the start of the file.

Locale The following environment variables affect the execution of od:

- **LANG** Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL** If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

- **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **LC_NUMERIC** Determine the locale for selecting the radix character used when writing floating point formatted output.

- **NLSPATH** Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1 Unsigned octal dump of the contents of file text:

```bash
$ cat text
Keep a stiff upper lip.
$ od text
  0000000 151205 102627 040201 040242 121611 103206 040244 113627
  0000020 102631 040223 104627 012400
  0000027
```
Example 2  Display the contents of the file *text* in octal form and as ASCII characters from the sixth byte onward:

```
$ od -bc text 5.
0000005 201 100 242 243 211 206 206 100 244 227 227 205 231 100 223 211
  a       s       t       i       f       u       p       p       e       r       l       i
0000021 227 025
  p       \n
0000023
```

See also  *Tables and directories, EBCDIC character set*
**paste**

**merge corresponding or subsequent lines of files**

`paste` can be used to horizontally merge the n-th corresponding lines from two or more input files (parallel merging, Format 1) or to successively merge all lines within a single file (serial merging, Format 2). The result is written to standard output.

**Syntax**

**Format 1:** `paste[...-d..list]..file.....`

**Format 2:** `paste-ss[...-d..list]..file.....`

**Format 1**

**Joining the n-th lines of several files (parallel merging)**

`paste[...-d..list]..file.....`

`paste` concatenates the n-th line of each input file, treating each file as a column or columns of a table. The corresponding lines are pasted together horizontally and displayed on the standard output (see Example 1 on page 590).

**No option specified**

The tab character acts as a delimiter between columns

**-d..list**

(delimiter) Uses a character from `list` as a delimiter between output columns. The characters in `list` are used consecutively and circularly, i.e. `paste` returns to the top of the list after using the last character in it. In parallel merging, lines from the last input file are always terminated with a newline character, not with a character from the `list`.

Any string of arbitrary characters can be specified for `list`. The following escape sequences may also be used: `\n` (newline), `\t` (tab), `\\` (backslash), and `\0` (empty string, not a null character). If `list` contains escape sequences, blanks, or shell metacharacters, it must be enclosed in double quotes "...".

**file**

Name of the input file.

This format of `paste` is only meaningful if you specify several files.

If you use a dash (-) as the name for `file`, `paste` reads from standard input.
Format 2  Joining successive lines (serial merging)

```
paste -s[-d..list]..file....
```

-s  (subsequent) For each input file, `paste` joins the lines together to form a single line and writes this line to standard output. By default, the lines from the input file are separated by tabs (see option `d`). Each output line is terminated by a newline character

-d..list
(d - delimiter) One of the characters from `list` is used in the output line instead of a tab to mark the joins between the input lines.

The characters in `list` are used consecutively and circularly, i.e. `paste` returns to the top of the list after using the last character in it.

Any string of arbitrary characters can be specified for `list`. The following escape sequences may also be used: \n (newline), \t (tab), \ (backslash), and \0 (empty string, not a null character). If `list` contains escape sequences, blanks, or shell metacharacters, it must be enclosed in double quotes "..".

file
Name of the input file. You may name more than one file.
If you use a dash (-) as the name for `file`, `paste` reads from standard input.

Error

```
paste: line too long
Output lines must not exceed 511 characters.
paste: too many files - limit 12
A maximum of 12 input files may be specified in format 1
paste: cannot open file: no such files or directory
file does not exist.
paste: cannot open file: Permission denied
User does not have read permission for file.
```

Locale

The following environment variables affect the execution of `paste`:

- `LANG`  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `LC_ALL`  If set to a non-empty string value, override the values of all the other internationalization variables.

- `LC_CTYPE`  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).
paste

**LC_MESSAGES**

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**

Determine the location of message catalogs for the processing of *LC_MESSAGES*.

### Examples of Format 1

**Example 1**

The corresponding lines from the files *numbers* and *letters* are to be pasted together:

The *numbers* file contains numbers from 1 to 100:

```
1
2
3
...
100
```

The *letters* file contains lowercase letters from *a* to *z*:

```
a
b
c
...
z
```

```
paste number letters
```

```
1 a
2 b
3 c
...
25 y
26 z
27
...
100
```
Example 2  The current directory contains the following files:

```
$ ls -C
  corr  jokes  names  plan  probe
  prog.c  tst  words
```

The following command numbers these files (see the `numbers` file in Example 1):

```
$ ls | paste numbers -
  1  corr
  2  jokes
  3  names
  4  plan
  5  probe
  6  prog.c
  7  tst
  8  words
  9
  10
```

Example 3  The current directory contains the same files as in Example 2. The following command lists the contents of the current directory in three columns. However, the columns will only be justified properly if the individual file names do not extend beyond the next tab stop.

```
$ ls | paste - - -
  corr  jokes  names
  plan  probe  prog.c
  tst  words
```

How is this output produced? Compare the above command with the command

```
$ paste file1 file2 file3
```

In this case, `paste` first reads the initial lines from all three files and pastes them into one line. When this is done, the second lines are read, etc.

In the command `ls | paste - - -`, the first file name that `paste` reads from standard input (`corr`) corresponds to the first line from `file1`; the second file name `jokes` corresponds to the first line from `file2` etc.

Example 4  The current directory contains the same files as in example 2. As in example 3, you now wish to display the file names in three columns, but the second and third columns are to be separated by a colon instead of a tab.

```
$ ls | paste -d"\t:" - - -
  corr  jokes:names
  plan  probe:prog.c
  tst  words:
```
Example of Format 2

Example 5  The *customers* file contains the following:

```bash
hansen
smith
cologne
koch
schulz
london
tornio
meyer
perth
```

$ `paste -s customers`

hansen  smith   cologne koch    schulz  london  tornio  meyer   perth

The following command joins only three lines of the *customers* file at a time. This is because a newline character is specified as a delimiter after every third input line:

```bash
$ `paste -s -d"\t\t\n" customers`
```

hansen  smith   cologne koch    schulz  london  tornio  meyer   perth

See also  *cut*, *grep*, *pr*
**patch** apply changes to files

**patch** will take a patch file containing any of the three forms of difference listing (normal, with context or in the ed format), produced by the **diff** (1) program and apply those differences to an original file, producing a patched version. By default, the patched version is put in the place of the original. If you specify the **-b** option, the original file is stored under the same name with the extension ".orig".

You may also specify where you want the output to go with a **-o** option.

Upon startup, **patch** will attempt to determine the type of the diff listing, unless overruled by a **-c**, **-e** or **-n** option. Context diffs and normal diffs are applied by the **patch** program itself, while **-e** diffs are simply fed to the **ed** editor via a pipe.

**patch** will try to skip any leading extraneous material, apply the diff, and then skip any trailing extraneous material. If the entire diff is indented by a consistent amount, this will be taken into account.

**Syntax**

```
patch[-bINR][-c[-e][-n][-d..dir][-D..define][-i..patchfile][-o..outfile][-p..num]
[-r..rejectfile][file]
```

**Options**

- **-b** saves all original files with a .orig suffix before the patch is applied. The backup file is overwritten if it already exists; if **patch** is executed several times, a backup file is only created on the first execution. The additional option **-o** does not save the original file, but saves the output file if it already exists.

- **-c** Interprets the patch file as a context diff (output from **diff**, if **-c** or **-C** is specified).

- **-d..dir**
  
  **patch** changes to the dir directory before further actions take place.

- **-D..define**
  
  marks changes with the instruction
  
  ```
  #ifdef define
  ...
  #endif
  ```
  
  Note that, unlike with the C compiler, there must be a space between the **-D** option and the argument.

- **-e** Interprets the patch file as an ed script.
patch

-i patchfile
  *patch* reads the patch from the *patchfile* file. If you enter a dash for *patchfile*, *patch* reads from the standard input.

A patch file contains one or more patches and maybe some additional information. If a patch file contains several patches, each patch should contain information about filenames (like with *diff* -c), so that *patch* can find affected files automatically.

*patch* evaluates the following information:

**Index:** pathname
  *pathname* names the file to be corrected

***** pathname
  *pathname* specifies the "old" file on which the patch is based

**--- pathname
  *pathname* specifies the file to be corrected (has priority over **Index:**)

Each patch contains patch instructions, which correspond to one of the three kinds of diff.

-i not specified: *patch* reads from the standard input.

-I Any sequence of tabs and blanks in the patch file will match any sequence of tabs and blanks in the input file. However, normal characters must still match exactly.

-n Interprets the patch file as a normal diff.

-N Ignores patches that have already been applied. Such patch instructions are rejected by default.

-o outfile
  The corrected version is written to *outfile*. Several corrected files are appended one after the other. If different patches affect the same original file, the following scripts are applied to a temporary file.

  Several versions of the original file are then written to *outfile* accordingly.

-p num
  Controls the handling of pathnames found in the patch file. *num* specifies how many slashes are to be stripped from the front of the pathname. (Any intervening directory names are also removed.) For relative pathnames, the search takes place in the current directory or in the directory specified by the -d option.

-p not specified:
  Only the basic name, without path, is used.
**patch**

*Example*

Supposing the filename in the patch file was

/u/howard/src/blurfl/blurfl.c

Setting the `-p 0` option gives the entire pathname unmodified, while the `-p 1` option gives

u/howard/src/blurfl/blurfl.c

without the leading slash, the `-p 4` option gives:

blurfl/blurfl.c

and not specifying the `-p` option at all just gives you `blurfl.c`.

-R patch swaps the patch instructions before they are applied to the original file. This is necessary if the old and new files were exchanged when creating the patch. `patch` attempts to swap each hunk before it is applied. Rejects are written to the error file in swapped format.

The `-R` option cannot be used for `ed` scripts, as the information is insufficient for reconstructing the swapped operation.

If the first hunk of a patch fails, `patch` swaps the patch instructions to see if they can be applied in this way. If this is possible, you are asked whether the `-R` option should be set.

If this is not possible, `patch` enters the hunk in the error file and continues as normal.

(Note: reversed patch instructions cannot be detected with this method in the case of a normal diff and where the first command is an append, i.e. it should in fact have been a delete.)

-r...rejectfile

Specifies the file to which the rejected patches should be sent.

-r not specified:

The rejected files are sent to one file, which has the same name as the output file, but with the suffix `.rej`.

file

Pathname of the file to be patched.

file not specified:

patch attempts to determine the filename from the leading extraneous material.

**Hint**

Notes for patch senders

If you create patch files on a regular basis, it is recommended that the revision level be included as the first patch instruction. If you add the line "Prereq: " to the patch file, you can prevent users applying incorrect patches without being warned.

Check whether the filenames have been specified correctly in the context diff header or in the "Index:" line. If you wish to make corrections in a subdirectory, inform the patch user that the `-p` option must be set.
Avoid reversed patches where possible.

Place patches that need to be together in the event of an error in separate files where possible.

If patch files are created with rejected hunks, `patch` terminates with a non-zero exit status. If you want to apply a set of patches in a loop, you should check this exit status so that a subsequent patch will not be performed on a partly corrected file.

If code was duplicated (e.g. using "#ifdef OLDCODE ... #else ... #endif"), `patch` cannot correct both versions. If the command can be executed at all, it may correct the wrong version and inform you that the corrections have been made successfully.

If you use an already applied patch again, `patch` assumes that the patch in question is a reversed patch and offers to un-apply the patch.

### Locale

The following environment variables affect the execution of `patch`:

- **LANG**: Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**: If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **LC_TIME**: Determine the locale for recognising the format of file timestamps written by the `diff` utility in a context-difference input file.

- **NLSPATH**: Determine the location of message catalogs for the processing of `LC_MESSAGES`.

### Exit status

- **0**: Successful completion.
- **1**: One or more rejected patch instructions were written to the error file.
- **>1**: An error occurred.

### See also

`diff`, `ed`
**pathchk**

**check pathnames**

The *pathchk* command checks that one or more pathnames are valid (i.e. they can be used to access or create a file without causing syntax errors) and portable (i.e. the name need not be adjusted). More extensive portability checks can be carried out with the *-p* option.

By default, the *pathchk* command checks the components of all *pathname* arguments based on the underlying file system. An error message is output for the *pathname* argument if:

- It is longer than the maximum permitted pathname length (`{PATH_MAX}` bytes).
- It contains a component that is longer than the maximum permitted filename length (`{NAME_MAX}` bytes) in the relevant directory.
- It contains a component in a directory that is not searchable.
- It contains a component with characters that are invalid in the directory.
- The name length of a file or directory in a bs2fs file system does not contradict the rules for a bs2fs file system.

*pathchk* does not consider it an error if one or more components of a *pathname* argument do not exist, as long as the file with the specified pathname can be created and does not violate any of the checks described above.

**Syntax**

```
pathchk[\[-p\]..pathname...]
```

**Option**

`-p` *pathchk* does not perform a check on the underlying file system, but on generic portability conditions. An error message relating to the *pathname* argument is output if:

- It is longer than the maximum permitted length for portable pathnames (`{POSIX_PATH_MAX}` bytes).
- It contains a component longer than the maximum length for portable filenames (`{POSIX_NAME_MAX}` bytes).
- It contains a component with characters not contained in the portable character set for filename.

**pathname**

The pathnames to be checked.

**Locale**

The following environment variables affect the execution of *pathchk*:

- **LANG**
  Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contain an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalization variables.
pathchk

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Hint**  Using the *test* command you can check whether a certain pathname specifies an existing file. However there is no information about whether a component of the pathname was truncated (in a directory where the \(_{\text{POSIX\_NO\_TRUNC}}\) function is not activated. The *pathchk* command does not check for the existence of a file. It only checks if a certain pathname exists or if it can be created without truncating the name.

The *noclobber* option in the shell (compare with the description of *set*) can create a unique file.

**Example**  You can check if all pathnames in an imported data interchange archive are legitimate and unambiguous on the current system as follows:

```bash
pax -f archive | sed -e \*(h0/ == .*/s///\*(h0 | xargs pathchk
if [ $? -eq 0 ]
then
  pax -r -f archive
else
  echo Investigate problems before importing files.
  exit 1
fi
```

You can check whether all files in the current directory hierarchy can be transferred to another system, that the general portability conditions are met, and that the *pax* command is available as follows:

```bash
find . -print | xargs pathchk -p
if [ $? -eq 0 ]
then
  pax -w -f archive .
else
  echo Portable archive cannot be created.
  exit 1
fi
```

You can check whether a specified pathname names a readable file and whether an application can create a file by extending the filename without truncating the pathname and without overwriting any existing files.
case $- in
  *C*) reset='';
  *) reset="set +C"
   set -C;;
esac

if [ $? -ne 0 ]; then
  printf "%s: %s not found or %s.out fails \creation checks.\n" $O "$path" "$path"
  $reset # reset the noclobber option in case a trap
  # on EXIT depends on it
  exit 1
fi

This example is based on the following:

1. PROCESSING displays the code used by the application in order to use $path once a check has been made that $path.out meets the required conditions.

2. The state of the noclobber option is unknown if this code was called, and should be reset on exit to the state it was in when the code was called. (The reset variable is used to restore the initial state in this example.)

3. Note the usage of the following construction:

   rm "$path.out" > "$path.out"

   a) The pathchk command has by now already checked that $path.out is not truncated.

   b) If the noclobber option is set, the shell checks that $path.out does not already exist before rm is called.

   c) If the shell has successfully created $path.out, rm removes it again so that the application can create the file in the PROCESSING step.

   d) If the PROCESSING step wants the file to exist already when it is called, rm "$path.out" > "$path.out" should be replaced by > "$path.out"

   This confirms that the $path.out file does not exist yet, and it will be created for use by PROCESSING.

See also test
pax portable archive interchange

The pax command writes files to archive files, reads them or creates lists of these files and copies directory hierarchies. A number of archive formats are supported; see the -x. format option.

The action executed depends on which of the -r and -w options are specified. The four possible combinations of -r and -w give the four operating modes: list mode, read mode, write mode, and copy mode, corresponding respectively to the four forms shown in the Syntax.

list  In list mode (i.e. neither -r nor -w is specified), pax reads an archive file from the standard input and writes the name of the stored files, whose pathnames match the specified pattern, to the standard output. If a specified file is a directory, the file hierarchy rooted at that file will be written out also.

read  In read mode (i.e. -r is specified, but not -w), pax reads an archive file from the standard input and extracts the files whose pathnames match the specified pattern. If the extracted file is a directory, the file hierarchy rooted at that file is also extracted. The extracted files are created relative to the current directory.

write In write mode (i.e. -w is specified, but not -r), pax writes the contents of the file operands to the standard output in archive format. If no file operand is specified, a list of files to copy, one per line, is read from the standard input. When specifying a directory, all files in the file hierarchy rooted at that file are copied.

copy In copy mode (i.e. both -r and -w are specified), pax copies the file operands to the destination directory directory. If no file operands are specified, a list of all files to be copied, one per line, is read from the standard input. When specifying a directory, all files in the file hierarchy rooted at that file are copied.

The copying process runs as if the copied files were written to an archive file and subsequently extracted, except that there may be hard links between the original and the copied files.

If temporary directories are needed to extract a file in read or copy mode, these are created by pax.

At least one file must match the specified pattern or file operands, otherwise pax writes a diagnostic message to the standard message output for each operand that is unmatched, and terminates with a non-zero exit status.

The archive formats supported are automatically recognized when reading the file. The standard output format for archives when writing (no -x option specified) is the extended tar format.

An archive can span multiple files. The pax command determines what file to read or write as the next file.
If the selected archive format supports the specification of links between files, `pax` returns an error if no link can be created between these files during extraction.

Syntax

<table>
<thead>
<tr>
<th>Format 1:</th>
<th>pax[-cdnv][-f...archive][-s...instruction] ...[pattern...]</th>
</tr>
</thead>
</table>
| Format 2:          | pax-rc[-cdiknuv][-p...string] ... [-s...instruction] ...
| Format 3:          | pax-w[-dituvX][-b...blocksize][-a][-f...archive][-s...instruction] ...
| Format 4:          | pax-rc-w[-diklntuvX][-p...string] ... [-s...instruction] ...

```
directory options
-r     Reads an archive file from the standard input.
-w     Writes files in the specified archive format to the standard input.
-a     Appends files to the end of the archive.
-b...blocksize
       Puts the data in blocks in the archive file, and specifies the block size as a positive
decimal integer. Devices and archive formats may impose restrictions on blocking.
       Blocking is automatically determined on input. Default blocking during the creation of
       archives depends on the archive format (see the -x option below). To maintain a
       portable archive, the block size may not be bigger than 32 KB.
-c     Selects all files apart from those specified by the pattern or file operands.
-d     The file hierarchy rooted at this file in a directory is not copied, read, or archived.
-f...archive
       Specifies the path name of the input archive (in list or read mode) or the output archive
       (write mode). This archive is used instead of the standard input or output.
-i     Renames files interactively. For every file archived whose name matches the pattern
       operand, or for every file that matches a file operand, a prompt is written to /dev/tty.
       This prompt contains the name of the file. A line is read from /dev/tty. If this line is
       blank, the file is skipped. If the line consists of a single period, the file is processed
       without the name being modified. Otherwise the name is replaced by the contents of the
       line. If an end-of-line character is found when reading a response, or if /dev/tty cannot
       be opened for reading or writing, the pax command terminates immediately with a non-
       zero exit status.
-k     Prevents the overwriting of existing files.
-l     Creates links between files. Hard links are created whenever possible between source
       and target file hierarchies. 
```
-n  Selects the first file archived that matches the pattern operand. Only one matching file is selected for every pattern operand (file hierarchies rooted at the directories are not affected by this restriction).

-p ..string
Determines file characteristics (privileges). The string argument contains the file characteristics that must be retained or ignored during extraction. The string consists of the a, e, m, o and p specification characters. Several characters can be concatenated in the same string, and the -p option can be specified more than once.
The specification characters mean the following:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Date and time of the last file access are set to the current date.</td>
</tr>
<tr>
<td>e</td>
<td>User ID, group ID, access permissions (see section “chmod change file modes” on page 204), date and time of the last access and modification are preserved.</td>
</tr>
<tr>
<td>m</td>
<td>Date and time of the last file modification are not preserved.</td>
</tr>
<tr>
<td>o</td>
<td>User ID and group ID are preserved.</td>
</tr>
<tr>
<td>p</td>
<td>Access rights are preserved.</td>
</tr>
</tbody>
</table>

In the list above, "preserved" means that an attribute saved in the archive will be assigned to the extracted file, depending on the privileges of the invoking process. Otherwise, the attribute is defined as part of the normal file creation action.

pax does not set the s bits for the file mode.

If for any reason, one of these attributes cannot be preserved, pax writes a diagnostic message to the standard output. If an attribute cannot be preserved, the extracted file is not deleted, although it does effect the exit status.

If specification characters are duplicated in the string argument or are in conflict with the other arguments, the last argument entered has priority. If for example -p..e..e is specified, the date and time of the last file modification are preserved.

-s ..instruction
Modifies files specified by the pattern or file operands, according to the instruction. The syntax of ed is used for this. The concepts "address" and "line" mean nothing in the context of the pax command. The following format is used: -s /old/new/[gp]

As in ed, old is a simple regular expression and new can contain & signs, \n (where \n is a number), references, and subexpressions. The old string can also contain a newline character.

All characters can be used as delimiters (e.g. /). A number of -s expressions are allowed. The expressions are evaluated in the specified order ending with the first successful substitution. The optional trailing g is as defined in ed. The optional trailing p writes successful substitutions to the standard error output.
Filenames replaced by a null string are ignored when reading and writing archives.

- **t**  The date and time of the last access to archived files are set to the value they had before being accessed by `pax`.

- **u**  Ignores files that are an older version (with an older last modification date/time) of a file of the same name that already exists. In read mode, an archived file that has the same name as a file in the file system is extracted if the archived file is more current than the one in the file system. A file is only saved in write mode if there is no more recent version with the same name.

- **v**  In list mode, a detailed table of contents is written to the standard output. Otherwise the pathnames of the stored file are written to the standard output.

- **x**  `pax` recognizes the following formats:

  - **cpio**  The extended `cpio` interchange format. The default block size for this character-oriented archive file format is 512. Implementations of `pax` support all values for the block size up to 32256 that are a multiple of 512.

  - **ustar**  The extended `tar` interchange format. The default block size for this character-oriented archive file format is 1024. Implementations of `pax` support all values for the block size up to 32256 that are a multiple of 512.

  `pax` terminates immediately with a non-zero exit status if an attempt is made to append data to an archive file in a different format than the existing archive format.

- **X**  `pax` will not switch to a directory in a different file system when traversing a file hierarchy specified by a path name.

  The options that operate on the names of files (`-c`, `-i`, `-n`, `-s`, `-u` and `-v`) are evaluated one after the other as described below:

In read mode, the files are selected according to `pattern` operands specified by the user, and the modifications made using the `-c`, `-n` and `-u` options. The `-s` and `-i` options in this order then modify the names of the selected files. The `-v` option outputs the names resulting from these modifications.

In write mode, the files are selected according to pathnames specified by the user, and the modifications performed using the `-n` and `-u` options. The `-s` and `-i` options then modify the names of the selected files. The `-v` option outputs the names resulting from these modifications.

If both the `-u` option and the `-n` option are specified, `pax` does not consider a file unless it is more current than the file to which it is compared.

- **directory**  Pathname of the target directory for copy mode.
file
   Pathname of the file to be copied or archived.

pattern
   A pattern that matches one or more pathnames of archived files. By default (if no pattern
   is specified), all files in the archive are selected.

Standard input (stdin)

In write mode, the standard input is only used if no file operands are specified. The standard output must then be a text file containing a list of the pathnames. The file must contain one pathname per line, and there may be no preceding or trailing blank characters.

In list and read mode, the standard input must be an archive file.

Otherwise the standard input is not used.

File
   The input file specified by archive or the standard input if the archive is read from it, is a file formatted according to one of the archive formats listed in the -x option.

Prompts are written and responses read to or from stdin/stdout.

Standard output (stdout)

If -f is not specified in write mode, the standard output is the archive formatted according to one of the archive formats listed in the -x option.

In list mode, the table of contents of the selected files are written to the standard output in the following format:
'\%s\n', pathname

If -v is specified in list mode, the table of contents of the selected files is written to the standard output in the following format:

Pathnames that represent hard links to previous files are written in this format:
'\%s == %s\n', ls_-l_listing, linkname

All other pathnames are written in this format:
'\%s\n', ls_-l_listing

ls_-l_listing is the format created by the ls command with the -l option.

Standard error (stderr)

If -v is specified in read, write or copy mode, pax writes the pathnames it processes to the standard error output in the following format:
'\%s\n', pathname

These pathnames are output at the start of file processing. The final newline character is output after the file is read or written.
If the `-s` option is specified, and there is a trailing `p` in the substitution string, substitutions are written to the standard error output in the following format:

```
%s >> %s
```

original_pathname, new_pathname

Any messages about the archive format are also written to the standard error output.

Output files

In read mode, the files extracted or copied are of the archived file type.

In write mode, the output file named by the `-f` option is a file formatted according to archive formats listed in the `-x` option.

Consequence errors

If `pax` cannot create a file or a link when reading an archive, cannot find a file when writing an archive, or cannot preserve the user ID, group ID, or access rights with the `-p` option set, a diagnostic message is written to the standard error output and a non-zero exit status is returned. Processing is continued nonetheless. If `pax` cannot create a link to a file, by default it does not create a second copy of the file.

If the extraction of a file from an archive is prematurely ended by a signal or an error, `pax` may only have extracted a part of the file, or (if the `-n` option is not specified) a file under the name specified by the user, that is not the file the user wanted. Extracted directory access rights may also have additional bits from the file-creation mode mask as well as incorrect information about the date and time of the last file access and modification.

Locale

The following environment variables affect the execution of `pax`:

- **LANG**
  - Specifies a default value for the locale variables that are unset or null. If `LANG` is unset or null, the corresponding default value of the locale is used. If the locale variable contains an invalid setting, `pax` behaves as if no variable were set.

- **LC_ALL**
  - If this variable has a value, i.e. is not empty, this value overwrites the values of all other locale variables.

- **LC_COLLATE**
  - Determines the locale for the behavior of ranges, equivalence classes and collating elements used in the pattern matching expressions for the `pattern` operand, in the simple regular expression for the `-s` option, and in the extended regular expression defined for yes/no queries.

- **LC_CTYPE**
  - Determines the locale for the interpretation of byte sequences (e.g. single-byte as opposed to multibyte characters in arguments and input files), for the behavior of character classes used in the extended regular expressions defined for yes/no queries, and for pattern matching.
**pax**

*LC_MESSAGES*  
Determines the locale for the processing of yes/no responses, as well as the format and language of diagnostic messages, output by `pax`.

*LC_TIME*  
Determines the format and the content of date and time specifications if the `-v` option is specified.

*NLSPATH*  
Determines the position of the message catalog for the processing of `LC_MESSAGES`.

**Hint**  
This hint is only for users who have created archives in SINIX using the SINIX tar and cpio commands.

The `-p` option was introduced to reconcile differences between the conventional `tar` and `cpio` implementations. In particular these two commands use `-m` in very different ways. The `-p` option also provides extended facilities for the consistent addressing of future file attributes, e.g. for extended security systems or high performance archives. Of the many combinations, only two modes are usually used:

- `-p.e` "Preserve everything" is used by the conventional superuser with all the appropriate privileges to preserve all file attributes as they are recorded in the archive. The `e` flag is the sum of `o` and `p`.

- `-p.p` "Preserve" the access rights. This is used by users with regular privileges who would like to preserve all file attributes apart from the owner. The file date and time specifications are preserved by default. However this can be deactivated with two other flags, and the date and time of extraction used.

Some of the functionalities described require appropriate privileges for whoever calls `pax`, particularly when creating block-oriented or character-oriented special files, when restoring the date and time of file access (`-t` option), provided that the user is not the owner of the file, or when preserving the group, and the mode (`-p` option).

**Example**  
The following command creates an archived named `archiv` containing the files `file1` through `filen` plus the directory `dir1` and all its subdirectories:

```
$ pax -w -f archiv file1...filen dir1
```

The following commands copy the `olddir` file hierarchy to `newdir`:

```
mkdir newdir
pax -rw olddir newdir
```

The following command reads the `a.pax` archive, whereby all file directories/files rooted in `/usr` in the archive relative to the current directory are extracted.

```
pax -r -s "./old/*/usr/*" -f a.pax
```
pdbl set up and manage user-specific program cache

Any user can call this command. This command enables the current user to set up, maintain and manage user-specific program caches. There are two scope types for user-specific program caches:

SESSIONWIDE all the processes of a session are connected
USERWIDE all the processes of a user ID are connected

Syntax

pdbl {(-s[...sid] | -u) | -i}
pdbl {(-s[...sid] | -u) | -e | -size}
pdbl {(-s[...sid] | -u) | -a | -d}
pdbl {(-s[...sid] | -u) | -D}
pdbl {(-s[...sid] | -u) | -b | path}
pdbl {(-s[...sid] | -u) | -l[...element]}
pdbl {(-s[...sid] | -u) | -r | element}
pdbl | -h

options

-s sid
The program cache of a session (SESSIONWIDE scope) is selected. sid is the ID of the desired session. If sid is not entered, the current session will be automatically selected.

If the option -s is selected, all the following options operate on the program cache of the selected or current session.

-u The program cache of the user ID (USERWIDE scope) is selected.

If the option -u is selected, all the following options operate on the program cache of the current user.

-i The status of the program cache and statistical data about size and allocation percentages is written to standard output in the following format:

Cache name CREATED: date time STATE: status
SIZE: size MB ENTRIES: entries
FREE PAGES: pages

name The name of the program cache is formed with the letters DBL, the scope (S for SESSIONWIDE or U for USERWIDE) and the corresponding ID of the session or the user. For example, the program cache of session 504 has the name DBLS504.
Date of program cache set-up.
Time of program cache set-up.
Current status of program cache (*active*, *inactive* or *in delete*).
Total cache size in megabytes.
Current number of stored core images.
Number of memory pages still available in cache. In the worst case, there is one memory page less available to core images because the extension of the cache catalog occupies one page.

The program cache is set up and activated in the size indicated (in megabytes). The maximum size of the cache is not determined by *pdbl* but by system- and task-specific settings. The size of the cache cannot exceed the ADDRESS-SPACE-LIMIT of the user ID.

The program cache is activated and used immediately in loading processes.

The program cache is deactivated and ignored immediately in loading processes.

The program cache is resolved if no currently stored core images are present (this is analogous to option `-D`). The program cache remains in the *inactive* status if core images are still present.

The program cache is resolved and no longer used in loading processes.

If the program cache is locked because the loading process has already started, it will remain in the *in delete* state until all current loading processes have been finished.

The core image of a program identified by its *path* is stored in the program cache. The program entered under *path* must be executable.

A list of all core images currently stored in the program cache is written to standard output in the following format:

Name of the program element in the PLAM library or the plain file name of the program in UFS.
Number of memory pages occupied by the core image.
Date of the last access to the core image.
Time of the last access to the core image.
Current status of the program cache (*active*, *inactive* or *in delete*).
library  Name of the PLAM library from which the core image was loaded or the path name of the LLM in UFS.

-I element
Detailed information about the core image element in the program cache is written to standard output in the following format:

element CREATED : cdate ctime ACCESS: adate atime
START AT: staddress CACHESIZE: csize kB
USECOUNT: number
-----------------------------------------------
SLICES : sl LOADADDR: SIZE: 
loadaddress ssize kB
.
.
.
.
.
.
.
.
.
.

info

Meaning of output

element  Name of the program element in the PLAM library or the plain file name of the program in UFS.
cdate  Creation date of the core image.
ctime  Creation time of the core image.
adate  Date of the last access to the core image.
atime  Time of the last access to the core image.
staddress  Start address of the core image during processing.
csize  Number of memory pages occupied by the core image.
number  Number of load accesses to the core image.
sl  Number of slices.
loadaddress  Load address of the slices.
ssize  Number of memory pages occupied by the slices.
info  Information about the core image origin.

-r element
The core image element will be deleted from the program cache. All core images of the program cache will be deleted if "*" is indicated as an element.

-h  An overview of all options and parameters is given.
Example

```
# pdbl -u -e 16  # set up program cache

# pdbl -u -l     # show status
Cache DBLU101     CREATED: 01/27/09 16:04:01  STATE: active
SIZE: 16 MB    ENTRIES: 0
FREE PAGES: 4095

# cd /usr/demo/bin

# ls -l hello    # show LLM in UFS
-rwxr-xr-x 1 ROOT SYSROOT 364544 Feb 20 11:09 hello

# pdbl -u -b hello  # create and save core image

# pdbl -u -l     # show core images in program cache
hello        57 Jan 27 16:05:37 /usr/demo/bin/hello
```
pkginfo  

show information on software packages in POSIX

The `pkginfo` command shows information on software packages which are installed in POSIX. A software package installed in POSIX is defined by:

- the name of the software product
- the package from the software product (optional)
- the version of the software product
- the path under which the software product is installed (default: `/`)
- the BS2000 library (SINLIB) from which the software product was installed
- the date of the (last) installation.

Syntax

```
pkginfo[-l][-q][-v version][-P package][-I ipath] product
```

No option specified

An overview of all the installed software packages is output, see example 1.

options

- `-l` (l - long) Detailed outputs
- `-q` (q - quiet) No outputs; only the exit status is set
- `-v` version
  
  Only product(s) of the specified version are displayed
- `-P` package
  
  Only the specified package of a product is displayed
- `-I` ipath
  
  (I - Installation) Only products with the specified installation path are displayed

Exit status

- `0` Installed product(s) complying with the specifications have been found
- `1` No installed product complying with the specifications has been found
- `>1` Errors

Error

The `/var/sadm/pkg/instlog` file cannot be read or does not match the expected format.

Errors reported by the runtime system (CRTE).

File

`/var/sadm/pkg/instlog` - Log file of the POSIX package installations
**pkginfo**

**Hint**

The `pkginfo` command always outputs the information in English, regardless of the language set. This avoids problems which could otherwise occur during the analysis of outputs in shell scripts.

The names of the software product and package can be entered in upper or lower case.

**Example 1** Outputting an overview of all the software packages installed:

```
$ pkginfo
PRODUCT                        PACKAGE         VERSION  INSTALLATION-PATH
POSIX-BC                       -               080      /
POSIX-SH                       -               070      /
NFS                            -               030      /
POSIX-NSL                      -               070      /
POSIX-SOCKETS                  -               070      /
CRTE                           -               028      /
IMON-BAS                       -               031      /
POSIX-HEADER                   -               018      /
TCP-IP-SV                      PRNGD           031      /opt/TCP-IP-SV/prngd
TCP-IP-SV                      OPENSSH         031      /opt/TCP-IP-
SYMAPI                         -               066      /opt/emc
SBA-BS2                        -               062      /
SCCA-BS2                       -               020      /opt/emc/sccabs2
```

**Example 2** Outputting details of all packages of a software product:

```
$ pkginfo -l tcp-ip-sv
PRODUCT NAME              : TCP-IP-SV
PRODUCT PACKAGE         : PRNGD
PRODUCT VERSION         : 031
INSTALLATION PATH       : /opt/TCP-IP-SV/prngd
INSTALLATION LIBRARY    : $TSOS.SINLIB.TCP-IP-SV.031.PRNGD
INSTALLATION DATE       : Fri Oct 10 08:08:56 2008
```

```
Example 3  Outputting details of a particular package of a software product:

```
$ pkginfo -l -P openssh tcp-ip-sv
PRODUCT NAME        : TCP-IP-SV
PRODUCT PACKAGE      : OPENSSH
PRODUCT VERSION      : 031
INSTALLATION PATH    : /opt/TCP-IP-SV/openssh
INSTALLATION LIBRARY : $TSOS.SINLIB.TCP-IP-SV.031.OPENSSH
INSTALLATION DATE    : Fri Oct 10 08:06:08 2008
```

Example 4  Checking the installation of particular versions of a software product:

```
$ pkginfo -q -v 027 crte && echo "INSTALLED." || echo "NOT INSTALLED."
NOT INSTALLED.
$ pkginfo -q -v 028 crte && echo "INSTALLED." || echo "NOT INSTALLED."
INSTALLED.
$
**posdbl**  

**set up and manage global program cache**

Only the super user can call this command. A global program cache of scaleable size is kept to store ready-to-run core images of POSIX programs. These core images are stored implicitly during the call of a POSIX TOOL from the shell library or are stored explicitly using this command. The global program cache is available to all users to load a stored program.

### Syntax

```
posdbl{-s|-h|-S|-D|-n}  
posdbl{-e|-d}{loader|linker|both}  
posdbl.-b.path  
posdbl.-l[.element]  
posdbl.-r.element
```

### options

- **-s** The status of the global program cache, the implicit linker process and statistical data about size and allocation percentage is written to standard output in the following format:

  ```
  POSIX-DBL linker status loader status  
  Cache POSIX@DBL CREATED: date time  
  SIZE: size MB ENTRIES: entries  
  FREE PAGES: pages
  ```

  _Meaning of output_

  **status** Current status of the implicit linker and loader process (ON, OFF).

  **date** Creation date of the program cache.

  **time** Creation time of the program cache.

  **size** Total cache size in megabytes.

  **entries** Current number of stored core images.

  **pages** Number of memory pages still available in cache. In the worst case, there is one memory page less available to core images because the extension of the cache catalog occupies one page.

- **-h** An overview of all options and parameters is output.

- **-S** A script which enables the current content of the program cache to be restored is output to stdout.

- **-D** The program cache is deleted.

- **-n** A new, empty program cache is generated.
**posdbl**

- **e / d**
  The loader process (*loader*), the implicit linker process (*linker*) or both processes (*both*) are activated (option *e*) or deactivated (option *d*).

- **b path**
  The core image of a program identified by its path name is stored in the program cache. The program indicated under *path* must be executable.

- **l**
  A list of all core images currently stored in the program cache is written to standard output in the following format:

  element size date time library

  **element** Name of the program element in the PLAM library or the plain file name of the program in UFS. If the core image was saved with the command call

  `posdbl -b path`

  the name of the program element will be preceded by a plus sign (+).

  **size** Number of memory pages occupied by the core image.

  **date** Date of the last access to the core image.

  **time** Time of the last access to the core image.

  **status** Current status of the program cache (*active*, *inactive* or *in delete*).

  **library** Name of the PLAM library from which the core image was loaded or the path name of the LLM in UFS.
-I element

Detailed information about the core image element in the program cache is written to standard output in the following format:

```
element     CREATED : cdate ctime     ACCESS: adate atime
START AT: staddress  CACHESIZE: csize kB
USERCOUNT: number
```

```
-----------------------------------------------
SLICES : sl LOADADDR: SIZE:
    loaddress ssize kB
    . . . . . .
```

```
-----------------------------------------------
```

info

Meaning of output

- element Name of the program element in the PLAM library or the plain file name of the program in UFS.
- cdate Creation date of core image.
- ctime Creation time of core image.
- adate Date of the last access to the core image.
- atime Time of the last access to the core image.
- staddress Start address of the core image during processing.
- csize Number of memory pages occupied by the core image.
- number Number of load accesses to core image.
- sl Number of slices.
- loaddress Load address of the slices.
- ssize Number of memory pages occupied by the slices.
- info Information about the core image origin.

-r element

The core image element is deleted from the program cache. All core images of the program cache will be deleted if "**" is indicated as an element.
Example

```bash
# posdbl -s # show status
POSIX-DBL:
linker ON  loader ON
Cache POSIX@DBL
CREATED: 07/18/02 13:06:11
SIZE: 24 MB  ENTRIES: 9
FREE PAGES: 2688

# posdbl -d linker # deactivate implicit load process
POSIX-DBL:
linker OFF  loader ON

# posdbl -l # show core images in program cache
SH   202 Feb 19 11:05:14  $TSOS.SINLIB.POSIX-SH.070
RM   38  Feb 19 11:02:33  $TSOS.SINLIB.POSIX-SH.070
LS   40  Feb 19 10:56:15  $TSOS.SINLIB.POSIX-SH.070
... ...

# cd /usr/demo/bin

# ls -l hello # show LLM in UFS
-rwxr-xr-x 1 ROOT SYSROOT 364544 Feb 20 11:09 hello

# posdbl -b hello # create and save core image

# posdbl -l # show core images in program cache
SH   202 Feb 19 11:05:14  $TSOS.SINLIB.POSIX-SH.070
RM   38  Feb 19 11:02:33  $TSOS.SINLIB.POSIX-SH.070
LS   40  Feb 19 10:56:15  $TSOS.SINLIB.POSIX-SH.070
... ...
+hello 22 Feb 20 11:10:55 /usr/demo/bin/hello
```
pr

**prepare files for printing**

The `pr` command formats and writes the contents of files on the standard output.

Output is either in single columns (default) or in multiple columns. Multi-column output can be produced by using either the `-a` option or the `-m` option.

### Syntax

```
pr[...option][...file]...
```

No option specified

The files are divided into pages which are separated by a sequence of line feed characters. The page length is 66 lines, which includes 10 lines of header and trailer output.

Headers are made up of two blank lines, one line of text containing the page number, date and time that the file was last modified, and the file name, and two more blank lines. Trailers consist of five blank lines.

Files are output in single columns. Overlong lines are split.

If the standard output is associated with a terminal, error messages are not reported until all the specified files have been output.

### Option overview

The following overview indicates the various options you can use to modify the output format:

<table>
<thead>
<tr>
<th>Function</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set first page to be displayed</td>
<td>+</td>
</tr>
<tr>
<td>Split text into columns</td>
<td>-</td>
</tr>
<tr>
<td>Define how columns are to be filled</td>
<td>-a</td>
</tr>
<tr>
<td>Define page width for multi-column output</td>
<td>-w</td>
</tr>
<tr>
<td>Prevent truncation of lines in columns</td>
<td>-s</td>
</tr>
<tr>
<td>Output files simultaneously in columns</td>
<td>-m</td>
</tr>
<tr>
<td>Double-space the output</td>
<td>-d</td>
</tr>
<tr>
<td>Convert tabs to spaces</td>
<td>-e</td>
</tr>
<tr>
<td>Convert spaces to tabs</td>
<td>-i</td>
</tr>
<tr>
<td>Number lines</td>
<td>-n</td>
</tr>
<tr>
<td>Indent text</td>
<td>-o</td>
</tr>
<tr>
<td>Change page length</td>
<td>-l</td>
</tr>
<tr>
<td>Change file name in header</td>
<td>-h</td>
</tr>
</tbody>
</table>
**Function** | **Option**
---|---
Display file one page at a time | -p
Use form-feed characters to separate pages | -f
Use form-feed characters to separate pages | -F
Suppress error messages | -r
Suppress output of header and trailer | -t

**Description of options in alphabetical order**

**+page_number**

Starts output at the specified `page_number`.

+ not specified: Output begins at the first page.

**-columns**

Outputs the file in `columns` columns. Output appears as if `-e` and `-i` have been set with their default values.

This option cannot be combined with `-m`.

The columns of a page are filled with input lines from top to bottom. This setting can be changed with `-a`.

The default page width for multi-column output is 72. This setting can be changed with `-w`.

`pr` determines the width of each column by dividing the page width by the number of columns. If a line is too long, it is truncated to the right. Truncation can be suppressed with `-s`.

- not specified: Single-column output.

**-a (a - across)** Arranges multi-column output across the page.

The number of columns must be greater than 1. This number can be defined with `-columns` or `-w`.

If a line is too long to fit in a column, it is truncated to the right. Truncation can be suppressed with `-s`.

The `-a` option cannot be combined with the `-m` option.

-`a` not specified: `pr` fills columns down the page from top to bottom.

**-d (d - double-space)** Double-spaces the output. Blank lines that result from double-spacing are deleted when they occur at the top of a page.
-e[tab_char][spacing]
Replaces each tab character in the input by an appropriate number of spaces.

  tab_char Character which pr interprets as a tab character. May be any non-numeric character.

    tab_char not specified:
    pr uses the horizontal tab character (see Tables and Directories, ISO 646 character set).

  spacing Spacing between tabs. The first tab stop in a line is always set in column 1.
    If spacing is 0, the default value of 8 characters is assumed.

    spacing not specified:
    The tab spacing is 8 characters.

-f (f - form feed) Uses a single form-feed character to separate output pages.

    If the output is to a terminal, pr pauses before the first page and sounds the terminal bell. You then start the output by pressing [J].

    -f not specified:
    Pages are separated by a sequence of line feed characters.

-F (F - form feed) Pages are separated by a single form feed character.

    -F not specified:
    Pages are separated by a sequence of line feed characters.

Caution:
    In previous versions, the -F option had a different meaning. This is now supported by pr by default.

-h ..header
    Writes a text of your choice in the header instead of the file name. This option is ignored if you also specify -t or -l with a page length of 10 or less.

-i[tab_char][spacing]
Replaces white space in the input text with the tab character tabchar at appropriate positions in the output text.

  tab_char Character which pr interprets as a tab character. May be any non-numeric character.

    tab_char not specified:
    pr uses the horizontal tab character (see Tables and Directories, ISO 646 character set).

  spacing Spacing between tabs. The first tab stop in a line is always set in column 1.
    If spacing is 0, the default value of 8 characters is assumed.

    spacing not specified: The tab spacing is 8 characters.
-l length
  Sets the length of an output page.
  The page length includes a total of 10 lines for the header and footer. Consequently, if
  you specify a value of 10 or less for length, the header and trailer will not be output (see
  the -t option).
  -l not specified:
  Each page is made up of 66 lines.

-m Merges and outputs all named files simultaneously, one column per file. The maximum
  number of files that may be specified is nine.
  -m cannot be used together with -columns.
  Otherwise, the same rules apply as for -columns.
  The -m option cannot be combined with the -a option.
  -m not specified:
  pr outputs files one after the other.

-n[separator][char_pos]
  Numbers the lines. For multi-column output, the lines of each individual column are
  numbered. The line number occupies the first char_pos+1 character positions of each
  line or each column line.
  separator Character used to separate the line number from the start of the line. May
  be any non-numeric character.
  separator not specified:
  A horizontal tab is used as the separator (see section “ASCII character set
  (ISO 646)” on page 889).
  char_pos Number of character positions occupied by line numbers.
  char_pos not specified:
  Defaults to 5.

-o...offset
  Offsets (indents) each output line by offset character positions.

-p If the output is directed to a terminal, pr sounds the terminal bell and pauses before
  beginning each page. The page is displayed after you press the [J] key.

-r Suppresses diagnostic reports if pr is unable to open a file.
  -r not specified:
  After completing the entire output, pr issues an error message to report files that could
  not be accessed.


-\texttt{s[separator]}
Separates columns by the single character \texttt{separator} instead of a tab. If you do not specify \texttt{-w} at the same time, \texttt{-s} also prevents the truncation of overlength lines (up to 512 character) on multi-column output. \texttt{separator}: Character used to separate columns.

\texttt{separator not specified:}
The default separator is a horizontal tab (see section “ASCII character set (ISO 646)” on page 889).

-\texttt{t} Suppresses the headers and trailers. Terminates printing after the last line of each file instead of padding with spaces to the end of the page.

-\texttt{w...width}
Sets the page width for multi-column output. Multi-column output can be produced with \texttt{-columns} or \texttt{-m}.
\texttt{width}: Number of characters in a line.

-\texttt{w} not specified:
The default page width for multi-column output is 72 characters.

\texttt{file}
Name of the file to be prepared for printing. You may specify more than one file. If more than one file is specified, \texttt{pr} outputs them in succession.
If you use a dash as the name for \texttt{file}, \texttt{pr} reads from standard input.
If you use the \texttt{-m} option for multi-column output, no more than 9 files may be specified.

\texttt{file} not specified:
\texttt{pr} reads from standard input.

\texttt{File}
/\texttt{dev/tty}*  
/\texttt{dev/term/tty}*
Special files for individual terminals.

If standard output is directed to one of the special files \texttt{/dev/tty*}, other output directed to this terminal is delayed until standard output is completed. This prevents error messages from being interspersed throughout the output.

\texttt{Variable}
\texttt{TZ}
Determine the timezone for use in writing header lines.
Locale

The following environment variables affect the execution of `pr`:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and which characters are defined as printable. Non-printable characters still will be written to standard output, but are not counted for the purpose for column-width and line-length calculations.

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**LC_TIME**
Determine the format of date and time for use in writing header lines.

**NLSPATH**
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example 1
Print `file1` and `file2` sequentially in three-column format (`-3`) using the form-feed character (`-f`) at the end of each page so as to ensure that the output is independent of the printer type.

```
$ pr -3f file1 file2 | lp
```

Example 2
The tab spacing of 8 positions in `file1` is to be changed to 6 in `file2`. First you convert tabs in `file1` to spaces (`-e`; tab spacing = 8), then you reconvert the spaces back to tabs (`-i6`; tab spacing = 6), and then you write the result to `file2`. The `-t` option suppresses the header and trailer for both `pr` commands.

```
$ pr -et file1 | pr -i6t > file2
```
Example 3  The file *months* contains the name of a month in each line. This file is now to be displayed with *pr* in three columns (*-3*) with two-digit numbering (*-n2*). The columns are to be filled from left to right (*-a*):

```
$ pr -3n2a months
Jan 27 16:21 2009 months Page 1

  1   January       5   May         9   September
  2  February       6   June        10  October
  3   March        7   July        11  November
  4    April       8   August      12  December
```

See also  *cat, fold, more*
**print** write arguments to standard output

*print* is the output mechanism of the POSIX shell.

**Syntax**

```bash
print[[-Rnprsu][filedes]][arg]...
```

If it is invoked with no options or with the option `-` or `--`, the arguments are printed on standard output as explained in the *echo* command description.

**option**

- `-R` or `--`
  In raw mode, the escape conventions of *echo* are ignored. The `-R` option prints all subsequent arguments and options except for the `-n` option.

- `-n`
  No newline is added to the output.

- `-p`
  The arguments are written to the pipe of the process spawned with `|&`, not to standard output.

- `-s`
  The arguments are written to the history file, not to standard output.

- `-u filedes`
  Arguments are written to the single-digit file descriptor `filedes`, not to standard output.

**argument**

see *echo*

**Example 1** Various output possibilities for the string *abcdef*.

```bash
$ print "abc\tdef"
abc  def
$ print -r "abc\tdef"
abc\tdef
$ print -R "abc\tdef"
abc\tdef
$ print -n abc; print def
abcdef
```

**Example 2** Output to a file via the file descriptor.

```bash
$ exec 4>print.out
$ printf -u4 abc
abc
```

For further examples, see *echo*.

**See also**  *echo, read*
printf formatted output

The `printf` command outputs the arguments you specify in formatted form. `printf` supports all format specifications for strings as in the `printf()` function in C.

**Syntax**

```
printf format[...arg]...
```

- **format**
  - Character string that can contain three different types of objects:
    - plain characters, which are output without any modifications.
    - Escape sequences for metacharacters, which are converted into the corresponding characters in the output, e.g. `\n` is converted to a newline character.
    - Format elements from which each one of the specified arguments `arg` is processed.

- **arg**
  - String to be written to standard output in the format specified by `format`.
  - If there are fewer arguments than expected by `format`, the missing arguments are set to 0 or the null string. If there are more arguments than expected by `format`, `format` is applied more than once (unless `arg_no$` is specified, in which case the excess arguments are ignored).

  `arg` not specified:
  - The result is undefined.

**Metacharacters**

The following metacharacters are interpreted by `printf`:

- `\` Backslash (for distinguishing octal characters)
- `\a` Warning, alert
- `\b` Backspace
- `\f` Form Feed
- `\n` Newline
- `\r` Carriage Return
- `\t` Tab
- `\v` Vertical tab
- \octal Octal number, whereby octal consists of one, two, or three digits

1) These metacharacters are supported only on character terminals (i.e. if you are accessing the POSIX shell via rlogin)
Format elements

A format element comprises:

```
%[arg_no$][field_width][.precision]conversion_character
```

%  Always located at the beginning of the format element. If the % character is not to be interpreted as a part of the format element but as an ordinary character to be output, it must be escaped by preceding it with another % (%.%).

arg_no$

Decimal integer with which you specify the position of the argument to be processed. The number must be followed by a $ character.

%arg_nr$ and % should not be used in combination.

arg_no not specified: The argument following the last converted argument is used.

If you use arg_nr$ for one argument, you should also use arg_nr$ for all the other arguments.

field_width

Decimal integer with which you specify the minimum field width. If the string to be converted has fewer characters than field_width, it is padded on the left to the field width and output right-adjusted. If left-adjustment is desired, the decimal integer must be preceded by a dash (-). The padding is with blanks unless the field_width integer starts with a zero, in which case padding for right-adjusted output is done with zeros.

A field_width may also be indicated by an asterisk (*) instead of an integer. In this case, an integer argument supplies the field width. This argument must appear before the string to be converted. The asterisk does not work in combination with arg_no$.

If the string is longer than the field width, the field is automatically expanded.

.precision

Decimal integer with which you specify the maximum number of characters to be output from the string to be converted. This number must be preceded by a dot (.). If the precision argument is zero, nothing is output. The number of characters output is always controlled by the precision, even if some other value has been specified for the field width.

A precision may also be indicated by an asterisk (*) instead of an integer. In this case, an integer argument supplies the precision. This argument must appear before the string to be converted. The asterisk does not work in combination with arg_no$.

conversion_character

The following konversion_characters can be used for printf:

- b  character string with metacharacters
- c  single character
- d  signed decimal integer
printf is a built-in sh command.

- e: floating-point number in exponential notation, e.g. 5.234e+2
- E: floating-point number in exponential notation, e.g. 5.234e+2
- f: floating-point number, e.g. 52.34
- g: %e or %f, whichever is shorter
- G: %E or %f, whichever is shorter
- o: signed octal integer (base 8)
- s: character string
- u: unsigned decimal integer
- x: unsigned hexadecimal integer (base 16)

With s, all characters from the character string are output until the number of characters specified for precision is reached. If precision is not specified, the entire character string is output.

With b, the character string in arg can contain metacharacters. printf supports all escape sequences interpreted by the echo command in this case, i.e. the escape sequences specified above with the exception of octal numbers specified as \0octal, and \c. \c causes printf to abort output at this point and not to terminate it with a newline character.

Locale

The following environment variables affect the execution of printf:

- LANG: Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
- LC_ALL: If set to a non-empty string value, override the values of all the other internationalization variables.
- LC_CTYPE: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
- LC_MESSAGES: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- LC_NUMERIC: Determine the locale for numeric formatting. It will affect the format of numbers written using the e, E, f, g and G conversion characters.
- NLSPATH: Determine the location of message catalogs for the processing of LC_MESSAGES.
Built-in sh command

printf

Example 1  Output the string "Good Morning Helen" on the screen:

```bash
$ printf '%s %s %s\n' Good Morning Helen
```

The following command produces the same output:

```bash
$ printf '%2$s %3s %1$s\n' Helen Good Morning
```

Example 2  Output the first 6 characters of your home directory /usr/kathy with an appropriate message:

```bash
$ printf 'The first 6 characters of %s are %.6s.\n' $HOME $HOME
```

The first 6 characters of /usr/kathy are /usr/k.

Example 3  The examples that follow write a 4-digit number in an 8-character field right-adjusted, right-adjusted with leading zeros, and left-adjusted, respectively:

```bash
$ printf '%8s\n' 1860
1860

$ printf '%08s\n' 1860
00001860

$ printf '%-8s\n' 1860
1860......
```

See also  awk, bc, echo

printf() [4]
ps

**report process status**

*ps* outputs information about active processes. This report represents a snapshot of the system or process status at a given moment and could be outdated within a split-second, so it may not reflect the actual situation by the time it is displayed.

**Syntax**

```bash
ps[-aAcdefj][.-g..groplist][.-G..groplist][.-n..namelist][.-o..format][.-p..proclist][.-s..sesslist][.-t..termlist][.-T][.-u..userlist][.-U..userlist]
```

No option specified
  
  *ps* outputs information about processes associated with the current terminal. The output consists of a short listing comprising:
  
  - the process ID (PID)
  - the terminal identifier (TTY)
  - the cumulative execution time (TIME)
  - the command name (COMD).

  The significance of the output columns is explained in the section entitled *Output*.

**Option**

- **-a** Outputs information about all processes associated with a terminal, except process group leaders.

- **-A** Information about all own processes is output. This option is equivalent to the *-e* option. The POSIX administrator is shown information about all processes.

- **-c** Also outputs information relating to process class and priority.

- **-d** Outputs information about all processes associated with a terminal as well as those which are not. Process group leaders are not taken into account.

- **-e** Outputs information about all own processes. This option is equivalent to the *-A* option.

**BS2000:**

-only the POSIX administrator is shown information about all processes.
  
  If other users are also to obtain information on all the processes, the POSIX administrator must set the s bit for the */sbin/ps* file after POSIX-BC has been installed (command: `chmod +s /sbin/ps`).

- **-f** (full list) Outputs a full listing with supplementary information on each process. The columns displayed in a full listing are explained in more detail in the section entitled *Output*.

  If `-f` is specified, *ps* outputs the command name and arguments. However, the arguments are displayed only if the process belongs to the user who called *ps*, or if *ps* was called by the POSIX administrator.
If the command name for the process contains non-printing characters, the command name is enclosed within square brackets [...]. If the \(-f\) option is specified without other options, the information output refers to processes associated with the controlling terminal.

\(-g\). \texttt{grplist}

Restricts listing to data about processes whose process group leaders are given in \texttt{grplist}.

\texttt{grplist}

\texttt{grplist} is a list containing the process ID numbers of process group leaders. This list can be specified in one of two forms:

a comma-separated list of numbers,
or a list of numbers enclosed in double quotes with the numbers separated by commas and/or blanks.

\(-G\). \texttt{grplist}

Information is only output about processes whose real process leader is specified in \texttt{grplist}.

\texttt{grplist}

\texttt{grplist} is a list of the process ID numbers of process group leaders. \texttt{grplist} has the following format:

The numbers are separated by commas or the list is enclosed in quotes "...". In the latter case, the numbers may also be separated by commas and/or spaces.

\(-j\)

Outputs the session ID and the process group ID.

\(-l\) (long list) Outputs a long listing with detailed information on each process. The output columns in a long listing are explained in the section entitled \textit{Output}. If the \(-l\) option is specified without other options, the information output refers to processes associated with the controlling terminal.

\(-n\). \texttt{namelist}

The system file specified in \texttt{namelist} is used instead of the default file.

\(-o\). \texttt{format}

Outputs information in accordance with the definitions specified in \texttt{format} (see the section "User-defined output formats" on page 635).

\(-p\). \texttt{proclist}

Restricts listing to data about processes whose process ID numbers are given in \texttt{proclist}.

\texttt{proclist}

\texttt{proclist} is a list of process ID numbers.
The numbers must be separated by commas, or alternatively, the whole list can be enclosed in double quotes with the numbers separated by commas and/or blanks.

-s.. sesslist
Restricts listing to data about processes associated with a session listed in sesslist.

sesslist
sesslist is a list of session ID numbers. This list can be specified in one of two forms:
a comma-separated list of numbers,
or a list of numbers enclosed in double quotes with the numbers separated by commas and/or blanks.

-t.. termlist
Restricts listing to data about the processes associated with the terminals named in termlist.

termlist
termlist is a list of terminal identifiers which may be specified in one of two forms: either the device’s file name (e.g. term/tty04) or, if the device’s file name is constructed with tty, just the digit identifier (e.g. 004).

The terminal identifiers must be separated by commas, or alternatively, the whole list can be enclosed in double quotes with the entries separated by commas and/or blanks.

-T The BS2000 TSN of the processes is also output. -T can be specified together with other options. -T is effective with all options except -o.

-u.. userlist
Restricts listing to data about processes whose process owner is given in userlist.

userlist
userlist is a list of user ID numbers or login names.

The entries in userlist must be separated by commas or, alternatively, the whole list can be enclosed in double quotes with the entries separated by commas and/or blanks.

-U.. userlist
Information is only output about processes whose real process leader is specified in userlist.

userlist
userlist is a list of user IDs or login names.

The specifications in userlist must be separated by commas, or alternatively, the whole list can be enclosed in double quotes with the entries separated by commas and/or blanks.
Output

The following describes the headings and meanings of the `ps` output columns if option `-o` is set. The letters in parentheses indicate the option that causes the corresponding column to appear. `all` means that the column appears for all options. Note that options `-c`, `-j`, `-f` and `-l` determine only what information is provided for a process; they do not determine which processes will be listed.

F (l)
Flags (hexadecimal and additive) associated with the process. These flags are machine-dependent and have therefore been omitted here.

S (l)
State of the process.

0: process running
S: Sleeping: process waiting for an event
R: Runnable: process is runnable
I: Idle: process being created
Z: Zombie state: process is terminated, but its exit status has not yet been queried by the parent with a `wait()` system call
T: Traced: traced process stopped by a parent process
X: SXBRK state: process waiting for more memory

UID (f, l)
(User ID) The user ID number of the process owner. If the `-f` option is set, the login name is output instead of the UID.

Only the first 7 characters of the user ID are output.

PID (all)
(Process ID) The process ID number of the process. Every process is assigned a unique PID when it is created. You can use this number in a `kill` command, for example, if you want to terminate a particular process.

TSN (f)
Task sequence number of the BS2000 process (task).

PPID (f, l)
(Parent Process ID) The process ID of the parent process.

PGID (j)
Process group ID.

SID (j)
Session ID.
C (f, l)
Processor utilization for scheduling.

CLS (c)
Scheduling class (processes handled by the scheduler).

PRI (l,c)
Priority of the process. Higher numbers normally mean lower priority. However, if the -c option is specified, higher numbers mean higher priority.

NI (l)
Nice value, i.e. value by which process priority was changed (see nice). Only processes in the time-sharing class have a nice value.

ADDR (l)
Core address (physical page frame number) of the user area if resident; the disk address if swapped out.

SZ (l)
Size in blocks of the core image of the process.

WCHAN (l)
Address of the event for which the process is waiting. If the column is blank, the process is running.

STIME (f)
Starting time of the process. The time is output within the first 24 hours; thereafter, the date.

TTY (all)
The controlling terminal for the process. A question mark (?) is output when there is no controlling terminal.

TIME (all)
The cumulative execution time for the process in minutes and seconds.

COMD (all)
The command name. If the -f option is specified, the full command name and its arguments are output.

Processes which have terminated, but whose exit status has not yet been queried by the parent with a wait() system call, is marked <defunct>.

If termist, proclist, userlist, or grplist is not specified, ps will try to determine the controlling terminal by checking the standard input, standard output, and standard error in that order. It will then report on the processes associated with the controlling terminal. If the above three channels are all redirected, ps will not find a controlling terminal and hence not produce a report.
User-defined output formats

You can use the option `-o format` to define your own output (see also example 2). `format` is a list of variables which can be entered as individual arguments or separated by commas or blanks.

Each variable has a default header. The default header can be renamed, i.e., you can assign it a new text by appending to it the equals sign (=) and the new name.

The variables specified in `format` are written to the standard output where they appear next to each other. The individual field widths depend on the length of the default header. If the header text is void, e.g., `-o user=`, then at least the width of the default header is used.

You may specify the following variables in `format`, a default header is assigned to the variable if no further `-o` specifications are entered:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default header</th>
<th>Meaning of variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruser</td>
<td>RUSER</td>
<td>the real login name of the process. If this can be determined and the field width permits it, this is displayed as text, otherwise in decimal form.</td>
</tr>
<tr>
<td>user</td>
<td>USER</td>
<td>the effective login name of the process. If this can be determined and the field width permits it, this is displayed as text, otherwise in decimal form.</td>
</tr>
<tr>
<td>rgroup</td>
<td>RGROUP</td>
<td>the real group ID of the process. If this can be determined and the field width permits it, this is displayed as text, otherwise in decimal form.</td>
</tr>
<tr>
<td>group</td>
<td>GROUP</td>
<td>the effective group ID of the process. If this can be determined and the field width permits it, this is displayed as text, otherwise in decimal form.</td>
</tr>
<tr>
<td>pid</td>
<td>PID</td>
<td>the decimal value of the process ID</td>
</tr>
<tr>
<td>ppid</td>
<td>PPID</td>
<td>the decimal value of the parent process ID</td>
</tr>
<tr>
<td>pgid</td>
<td>PGID</td>
<td>the decimal value of the parent group ID</td>
</tr>
<tr>
<td>pcpu</td>
<td>%CPU</td>
<td>the percentage value of utilized CPU time</td>
</tr>
<tr>
<td>vsz</td>
<td>VSZ</td>
<td>the decimal value of the storage space required by the process in kilobytes</td>
</tr>
<tr>
<td>nice</td>
<td>NI</td>
<td>the decimal value of the system scheduling priority</td>
</tr>
<tr>
<td>etime</td>
<td>ELAPSED</td>
<td>the time the process has been running, specified in [[dd-]hh:mm:ss</td>
</tr>
<tr>
<td>time</td>
<td>TIME</td>
<td>the total period of CPU utilization for the process</td>
</tr>
<tr>
<td>tty</td>
<td>TT</td>
<td>name of the terminal on which the process is running</td>
</tr>
<tr>
<td>comm</td>
<td>COMMAND</td>
<td>name of the command which is being executed, specified in text form (it may contain spaces)</td>
</tr>
<tr>
<td>args</td>
<td>COMMAND</td>
<td>command with all spaces, in text form (it may contain spaces)</td>
</tr>
<tr>
<td>tsn</td>
<td>TSN</td>
<td>BS000 TSN (task sequence number) of the process</td>
</tr>
</tbody>
</table>
Variable

**COLUMNS**

Override the system-selected horizontal screen size, used to determine the number of text columns to display.

Locale

The following environment variables affect the execution of *ps*:

- **LANG**
  
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**
  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**
  
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **LC_TIME**
  
  Determine the format and contents of the date and time strings displayed.

- **NLSPATH**
  
  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example 1

*ps* is called with the `-l` option to display a long listing of information on all active processes on the controlling terminal:

```
$ ps -l
F S   UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN TTY      TIME CMD
10 S   110  1455  1453  0  30 20 c0fec9d8     23 d114f200 term/001  0:02 sh
10 O   110  1862  1455 12  50 20 c0fec870     60          term/001  0:00 ps
18 S   110  1858  1455 10  20 20 c0feca8f     55 d115f280 term/001  1:03 find
```

The `find` command is now to be terminated by calling `kill` with the process ID number displayed in the PID column. A subsequent call to `ps` then confirms that the corresponding process no longer exists.

```
$ kill 1858
$ ps

  PID  TTY      TIME CMD
  1455 tty004  0:02 sh
  1873 tty004  0:00 ps
  1858 Terminated
```
Example 2  User-defined output format of the ps command: output of USER, PID, PPID (if the new name is to contain FATHER) and output of arguments.

$ ps -o user,pid,ppid=FATHER -o args

<table>
<thead>
<tr>
<th>USER</th>
<th>PID</th>
<th>FATHER</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELEN</td>
<td>62</td>
<td>0</td>
<td>[sh]</td>
</tr>
<tr>
<td>HELEN</td>
<td>122</td>
<td>62</td>
<td>[ps]</td>
</tr>
</tbody>
</table>

Example 3  Only the BS2000 TSN of the current shell (without a header) is to be output.

$ ps -o tsn= -p $$

7R6C

By way of comparison:

$ ps -T

<table>
<thead>
<tr>
<th>PID</th>
<th>TSN</th>
<th>TTY</th>
<th>TIME</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>7R69</td>
<td>term/002</td>
<td>0:00</td>
<td>ps</td>
</tr>
<tr>
<td>148</td>
<td>7R6C</td>
<td>term/002</td>
<td>0:11</td>
<td>sh</td>
</tr>
</tbody>
</table>

See also  kill, nice
pwd  

**return working directory name**

The POSIX shell built-in `pwd` writes the absolute path name of the working (current) directory on the standard output.

**Syntax**

```
pwd
```

If the error message

```
Cannot open ...
```

or

```
Read error in ...
```

is displayed, a file system error has occurred. Inform the POSIX administrator.

**Locale**

The following environment variables affect the execution of `pwd`:

- **LANG**
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_MESSAGES**
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example**

You want to define your current directory as the home directory:

```
$ pwd
/usr/art/cobol/prg
$ HOME=pwd`
$ echo $HOME
/usr/art/cobol/prg
```

**See also**

`cd`
rcp  

**rcp** remote file copy

The *rcp* can be used to copy files and directories
– from a local host to a remote host
– from a remote host to a local host
– between two remote hosts

You can use this command, for example, when you want to work with files located on a remote host. The *rcp* command can be used symmetrically by BS2000-POSIX and remote UNIX systems.

**Format 1** of the command copies a single file from one host to a file on another host.

**Format 2** of the *rcp* command copies multiple files or a directory of another host.

The *rcp* command can only be used by users with access authorization on the remote host.

In other words, the *rhosts* file in the *HOME* directory of the ID on the remote host must have an entry containing the name of the local host or a plus character (+) and the user ID. The user ID must have a standard account number for *rlogin* access. This standard account number can be assigned using the ADD-USER, MODIFY-USER-ATTRIBUTES or ADD-POSIX-USER command.

The files *etc/hosts* and *etc/hosts.equiv* usually used in UNIX systems are not used in POSIX. The assignment of host names to IP addresses is handled via BCAM or DNS.

**Syntax**

**Format 1:**  
*rcp*[\[-p\][\-b\]]*file1*..*file2*

**Format 2:**  
*rcp*[\[-pr\][\-b\]]*file*....*dir*

**Format 1 Copy individual files**  

*rcp*[\[-p\][\-b\]]*file1*..*file2*

No option specified

The modification and access times of each copy are set to the current time. The access permissions are set in accordance with *umask*.

-\-p  Every copy receives the same modification and access times and the same access permissions as the original file.

-\-b  The file is transferred in binary mode.

*file1*  
is the file to be copied. This file may be stored on the local or remote host.  

If the file is stored on the local host, it may be specified with *path*, where *path* is the pathname of *file1*. If *path* is not an absolute pathname, it is interpreted as a relative path to the current directory.
If the file is stored on a remote host, file1 must be specified in the following format:

```
login@remote_host:path or
remote_host:path or
login@remote_host.domain:path
```

- `remote_host` is the name of the remote host or its IP address in IPv4 or IPv6 dot notation:
  - IPv4: n.n.n.n = 0..255(decimal)
  - IPv6: x:x:x:x:x:x:x:x = 0..FFFF(hexadecimal)

The alternative presentation methods described in RFC 2373 are permissible for IPv6 dot notation.

If for the name of the remote host its address is specified in IPv6 dot notation, this must be enclosed in square brackets, e.g.:

```
rcp test hansi@[::ffff:AC19:7D37]:hansi_test
```

Otherwise the first colon in the IPv6 address would be incorrectly interpreted as a separator between the host name and the path name.

- `login` is the user ID on the remote host.
- `path` is the pathname of file1.
- `domain` is the name of the DNS domain to which the remote host belongs.

Pathnames may include the usual metacharacters (e.g. *, ?) for constructing file names. These metacharacters are evaluated on the local host. path must not contain a colon ":", and the string for `login@remote_host` must not include a slash "/".

file2 is the file to which the source file is to be copied. This file may be placed on the local or remote host. If the file already exists, it will be overwritten.

The format is the same as for file1.
Format 2  Copy multiple files or directories

```
rcp[-pr][-b]...file...dir
```

No option specified

The modification and access times of each copy are set to the current time. The access permissions are set in accordance with `umask`.

- **p** Every copy receives the same modification and access times and the same access permissions as the original file.

- **r** All subdirectories of `dir` are copied recursively. In this case, `file` must be a directory.

- **b** The file is transferred in binary mode.

**file**

may be

- a file,
- multiple files, or
- a directory (with `-r`) on the local or remote host.

The following format applies on the local host:

```
filename ...
```

or

```
directory
```

*filename* is the respective name or pathname of one or more files.

*directory* is the respective name or pathname of the directory.

If *file* is located on the remote host, the following applies:

```
login@remote_host:path or
remote_host:path or
login@remote_host.domain:path
```

*remote_host* is the name of the remote host.

*login* is the user ID on the remote host.

*path* is the pathname of *file1*.

*domain* is the name of the DNS domain to which the remote host belongs.

Pathnames may include the usual metacharacters (e.g. *, ?) for constructing file names. These metacharacters are evaluated on the local host.
Example

$ rcp remote_host:path1/ * path2/dir

copies all files that match the first pathname into the specified directory.

dir

is the directory to which the files or directories are to be copied. This directory may be located on the local or remote host. Files of the same name, if any, are overwritten.

The format for dir on the remote host is:

directory

The format for dir on the remote host is:

login@remote_host:directory

or

login@remote_host.domain:directory

Error

remote_host: unknown host

The host is not known to BCAM or only known under another alias.

remote_host: Connection timed out

No acknowledgment was received from the remote host within a specified period.

File

$HOME/.rhosts

List of host names and user IDs that can log on to them under those IDs.

Example 1 The POSIX user john wants to copy the file test from his current directory to the user john on the remote Solaris host rainbow. The copied file is to be named john_test on the remote host rainbow. The required access permissions for the copy operation are present.

$ rcp test john@rainbow:john_test

Example 2 The POSIX user john copies the directories test and practice from the Solaris login name john on the host rainbow to the POSIX directory /home/john/solaris_copy.

$ rcp -r john@rainbow:test john@rainbow:practice /home/john/solaris_copy

See also rsh
**read**  read a line from standard input

`read` is a POSIX shell built-in command that reads a line from standard input and sequentially assigns the individual input line arguments as values to the shell variables specified in the call.

The only argument separators that `read` recognizes are the characters assigned to the `IFS` shell variable, the defaults being blanks, tabs and newline characters.

If `read` appears in a shell script and standard input has not been redirected, the script halts execution in order to read your next input from standard input. The script resumes execution as soon as you enter a newline character (see also Examples on page 645).

The command does not read from the standard input but from the pipeline for the process created using `|&`. When the end-of-file is reached in the pipeline, the input is cleared to permit `|&` to generate a new process.

In raw mode the backslash \ at line end has no special function, i.e. the line is not continued.

The input is written as a command to the `history` file.

The command reads from the single-figure `file_descriptor` instead of from the standard input. The file descriptor can be opened using the `exec` command. The default value of `file descriptor` is 0.

Name of the shell variable to which the corresponding input line argument is assigned. The first argument is assigned to the first name, the second argument goes to the second, and so on, with the last name assigned whatever remains on the input line.

The names of shell variables must start with a letter or an underscore (_) and must consist of letters, underscores and digits only.

Any leftover arguments in the input line are assigned to the last variable specified in the `read` command line.

Any leftover variables of the `read` command are assigned the null string.

If the first argument contains a ?, the rest of the word is written to standard error as a prompt.

`name` not specified: `REPLY` is used for `name`. 

**Syntax**

```
read [-option][...name?prompt][...name]...
```

- **option**
  - `-p`  The command does not read from the standard input but from the pipeline for the process created using `|&`. When the end-of-file is reached in the pipeline, the input is cleared to permit `|&` to generate a new process.
  - `-r`  In raw mode the backslash \ at line end has no special function, i.e. the line is not continued.
  - `-s`  The input is written as a command to the `history` file.
  - `-u file_descriptor`  The command reads from the single-figure `file_descriptor` instead of from the standard input. The file descriptor can be opened using the `exec` command. The default value of `file descriptor` is 0.
Exit status
  0 when `read` executes successfully
  >0 when no input is received, i.e. EOF is encountered.

Error
  `sh: text: not an identifier`
  This error message may have the following causes:
  – either you did not specify a variable name on the command line, or
  – the name you specified contains illegal characters.

  `read: missing arguments`
  You called `read` without arguments.

Variable
  `IFS`
  Input field separator (argument delimiter). The default values are blank, tab and newline.

  `PS2`
  Provide the prompt string that an interactive shell will write to standard error when a line ending with a backslash is read and the `-r` option was not specified, or if a here-document is not terminated after a newline character is entered.

Locale
  The following environment variables affect the execution of `read`:

  `LANG`
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

  `LC_ALL`
  If set to a non-empty string value, override the values of all the other internationalization variables.

  `LC_CTYPE`
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

  `LC_MESSAGES`
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

  `NLSPATH`
  Determine the location of message catalogs for the processing of `LC_MESSAGES`.
Example 1  The read command is invoked in a script named readtest, which contains the following:

```bash
: Invoked with sh readtest, halts for input
echo Please enter a customer name:
read customer1 customer2 customer3
if [ -z "$customer1" ]
then exit 5
else echo Customer1: $customer1
    echo Customer2: $customer2
    echo Customer3: $customer3
fi
```

Invocation of the readtest script file:

```
$ sh readtest
Please enter a customer name:
Shaw Bowden Pitman Potter
Customer1: Shaw
Customer2: Bowden
Customer3: Pitman Potter
```

After invocation, the shell script issues the message specified in the echo command and invokes read. The script halts, and the entered customer names are then read in. The newline character terminates the input line for read. The third variable customer3 is assigned two names, since four arguments were specified in the input line.

Example 2  Use of the read command to read in the first line from a file:

```
$ read line < /etc/group
$ echo $line
root::0:root
```

In this case, the first line of the file /etc/group will always be read, even if read is invoked repeatedly.

Example 3  The following shell script makes use of the read command in order to read in lines from a file successively:

```bash
: Invoked with sh readinall
exec < /etc/group
for i in 1 2 3 4 5 6 7
done
```

U22794-J-Z125-6-76
In the shell script `readinall`, the shell built-in `exec` redirects the standard input to the file `/etc/group` for the following `read` command. Owing to the `for` loop, `read` is invoked seven times in the script. Each invocation positions the read pointer on the next line, thus causing `echo` to output the first seven lines of the `/etc/group` file in succession:

```
$ sh readinall
record1: root::0:root
record2: daemon::1:daemon
record3: sys::2:sys:
record4: bin::3:bin,admin
record5: uucp::4:
record6: ces::5:
record7: other::10:gast,mgast,tele
```

To evaluate the argument `\${record[$i]}` correctly, the shell has to interpret the `echo` command line twice; hence the inclusion of `eval`. At the first attempt the shell only interprets `$i`, as the first dollar sign is escaped by the backslash. At the second attempt the shell interprets `$record[1-7]`.

See also `exec`
**readonly**  
set read-only attributes for variables

The POSIX shell built-in *readonly* marks the specified shell variables as read-only, i.e. protects them from being changed by reassignment in the current shell. An error message is issued if any such attempt is subsequently made.

This protection only applies to the current shell. This means that the variable may be reassigned in a subshell or in a parent shell (i.e. when you terminate the current shell). Within the current shell, however, it is not possible to undo this protection.

If you invoke *readonly* without arguments, a list of all existing read-only variables in the current shell is written to standard output.

**Syntax**

<table>
<thead>
<tr>
<th>Format 1</th>
<th>readonly[,.name[=value]]...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2</td>
<td>readonly..-p</td>
</tr>
</tbody>
</table>

**Format 1**  
*readonly[,.name[=value]]...*

- `name`: Name of the shell variable which you want to protect against change. You may enter any number of shell variables each separated by a space.

  `name` not specified:
  
  *readonly* writes the names of all shell variables which are protected in the current shell to the standard output. The output has the following form: `name=value`

**Format 2**  
*readonly..-p*

*readonly* writes the names of all shell variables which are protected in the current shell to the standard output. The output has the following form: `readonly name=value`

**Error**

- `name`: is read only

  This error message is issued when you try to assign a value to a protected shell variable.

**Locale**

The following environment variables affect the execution of *readonly*:

- **LANG**: Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**: If set to a non-empty string value, override the values of all the other internationalization variables.
**readonly**

**Built-in sh command**

- **LC_CTYPE**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**: Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example**

Marking the shell variable **HOME** as read-only in the current shell:

```
$ readonly HOME
$ readonly
HOME=/USER1
$ sh
$ readonly
$ [END] bzw. @@d
$ readonly
HOME=/USER1
```

The **HOME** variable is not protected in the subshell.

**See also**

`export, env, set, typeset`
renice  set system scheduling priorities of running processes

Caution!
The `renice` command is supported only on grounds of compatibility. It has no effect on BS2000 task priorities. For that reason only the syntax chart is given here. The options, arguments and so forth are not described.

Syntax

Format 1: `renice[-n...increment][...g][...p][...u]...ID...
Format 2: `renice...nice_value...[-p]...pid...[-g]...gid...[-p]...pid...[-u]...user
Format 3: `renice...nice_value...[-g]...gid...[-g]...gid...[-p]...pid...[-u]...user
Format 4: `renice...nice_value...[-u]...user...[-g]...gid...[-p]...pid...[-u]...user`
rm  

**remove directory entries**  

*rm* removes the entry (link) for one or more files from a directory. You must have write permission for a directory before you can remove a file from it.

**Syntax**  

```
rm[...option]...file...
```

**No option specified**  

If you have write permission for *file*, *rm* removes the entry without issuing a warning. If you do not have write permission for *file* and the standard input is a terminal, *rm* prompts you by displaying the permissions and a query ? and asking whether you want the file to be deleted. The entry is not deleted unless you answer the locale’s equivalent of yes. No such confirmation is requested if the standard input is not a terminal.

**options**  

- **-f**  Removes the entries without any questions. Files will not be removed if you do not have write permission for the directory.

- **-i**  *rm* removes files interactively, requesting confirmation for each write-protected file (or directory, if -r is in effect) before removing it. The -i option overrides the -f option and remains in effect even if the standard input is not a terminal.

- **-r**  Accepts a directory name as an argument for *file*. The usual error message is not issued. *rm* recursively deletes the entire contents of the directory and also removes the directory itself. You cannot remove the parent directory (..) in this way. Symbolic links that are encountered with this option are not traversed. If the removal of a non-empty, write-protected directory is attempted, the command will always fail (even if the -f option is used), resulting in an error message.

- **-R**  Analogous to -r.

- **file**  Name of the file that is to be removed. If you include the -r option, *file* may also be a directory. You can include a number of file/directory arguments.

  If you specify a file that has several links, only the specified link is removed; the file itself remains intact (the link counter is decremented by one). The file itself is not deleted until you remove the final link.

  You must have write permission for the directory before you can remove a file from it; but you need not have read or write permission for the file itself.
Locale

The following environment variables affect the execution of `rm`:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_COLLATE**
Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements in extended regular expressions defined for yes/no queries.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments) and the behavior of character classes within regular expressions used in extended regular expressions defined for yes/no queries.

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example 1
Removal of all files that end in `.prog` with confirmation:

```
$ rm -i *.prog
exec.prog:? (y/n) y
code.prog:? (y/n) yuppie
input.prog:? (y/n) n
lucky.prog:? (y/n) no
a.prog:? (y/n) tomorrow
$
```

The links to files `exec.prog` and `code.prog` are removed; the others are retained.

Example 2
Removal of the directory `norm` with all files and subdirectories.

```
$ rm -r norm
```

See also `rmdir`
**rmdir**  
remove directories

*rmdir* removes one or more empty directories. Directories containing files cannot be removed with *rmdir*. To remove a directory together with everything it contains you can use the *rm* command, specifying option `-r`.

**Syntax**

```
rmdir[-p]directory....
```

No option specified

*rmdir* removes the specified directories.

option

- `p` (*p* - parents) The specified directory is removed, and all empty parent directories in the specified path are removed recursively.

directory

Name of the directory you want to remove.  
You can name any number of directories.

**Error**

*rmdir: dir1: Directory not empty*  
You have attempted to use *rmdir* to remove a directory *dir1* which still has files in it.  
You can use the *rm* command with option `-r` to remove directories that contain files.

*rmdir: dir1: Directory does not exist*  
The directory named *dir1* does not exist.

*rmdir: ../.: Can't remove current directory or ..*  
The current directory or its parent cannot be removed. Change to the parent directory.

**Locale**

The following environment variables affect the execution of *rmdir*:

`LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

`LC_ALL` If set to a non-empty string value, override the values of all the other internationalization variables.

`LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
**Built-in sh command**

**rmdir**

---

**LC_MESSAGES**

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**

Determine the location of message catalogs for the processing of **LC_MESSAGES**.

---

**Example**

Removing the directories **pro** and **proc**.

Your current directory has the following contents:

```
  drwxr-xr-x  11 SUSAN    other       5720   Nov 18 14:16 ./
  drwxr-xr-x  13 ROOT     root        3380   Nov 04 11:48 ../
-rw-------   1 SUSAN    other         79   Jul 19 14:21 .profile
-rwx------   1 SUSAN    other        125   May 25 10:29 begin
  drwx------   2 SUSAN    other         32   Oct 11 15:36 pro/
  drwx--x--x   2 SUSAN    other         32   Nov 07 10:43 proc/
```

The directories **pro** and **proc** are empty, so you can remove them with **rmdir**.

```
$ rmdir pro proc
```

```
  drwxr-xr-x  11 SUSAN    other       5720   Nov 18 14:16 ./
  drwxr-xr-x  13 ROOT     root        3380   Nov 04 11:48 ../
-rw-------   1 SUSAN    other         79   Jul 19 14:21 .profile
-rwx------   1 SUSAN    other        125   May 25 10:29 begin
```

---

**See also** **rm**
rmpart  remove partition

The `rmpart` removes partition `unitnumber` from the partition table. The associated BS2000 file is not removed.

Syntax

```
rmpart[\-V]..unitnumber
```

⚠️ **Caution!**
The functionality of the `rmpart` command is incorporated in the POSIX installation program (see POSIX manual "Basics for Users and System Administrators" [1]), so it is not described in further detail here.
**rsh**

**remote shell**

`rsh` can be used to execute a command on a remote UNIX system.

You can use this command if, for example:

– you want to determine the contents of a directory on a remote UNIX system, or
– you want to print a file on the remote UNIX system.

The `rsh` command can only be used by users with access authorization on the remote host. In other words, the `.rhosts` file in the `HOME` directory of the user ID on the remote UNIX system must have an entry containing the name of the local host or a plus character (+) and the user ID. The user ID must have a standard account number for `rlogin` access. This standard account number can be assigned using the ADD-USER, MODIFY-USER-ATTRIBUTES or ADD-POSIX-USER command.

The `rsh` command can be used symmetrically by BS2000-POSIX and remote UNIX systems.

The files `/etc/hosts` and `/etc/hosts.equiv` usually used in UNIX systems are not used in POSIX. The assignment of host names to IP addresses is handled via BCAM or DNS.

The command can only be executed if the daemon `rshd` is active on the local and remote hosts. The daemon `rshd` is started by `inetd`.

The actual command interpreter called to execute the command on the remote host is determined by the entry of the user in the `/etc/passwd` file.

If `rsh` is executed from a file that is not named `rsh`, it will use the name of that file as the remote host. In other words, if you create a symbolic link to `rsh` under the name of the remote host, you can call `rsh` by simply specifying that host name.

Stop signals interrupt only the `rsh` process on the local host.

The environment variables of the local host are not passed to the command interpreter on the remote host.

**Syntax**

```
  rsh..host[...-n][...-l..login][...-x][...command][...option]
```

**host**

The BCAM or DNS name of the remote host on which a Shell command is executed. The name of the remote host must be known to local BCAM or to the DNS server.
Instead of the name of the remote host you can also specify its IP address in IPv4 or IPv6 dot notation. The IP address must be known to BCAM.

IPv4: \[ n.n.n.n \quad n = 0..255 \quad \text{(decimal)} \]
IPv6: \[ x:x:x:x:x:x:x:x \quad x = 0..FFFF \quad \text{(hexadecimal)} \]

The alternative presentation methods described in RFC 2373 are permissible for IPv6 dot notation.

-\( n \)  
Instructs the \texttt{rsh} command to not redirect its standard input to the command on the remote host. The standard input of \texttt{rsh} is redirected to \texttt{/dev/null} instead.

This is useful, for example, if the output of the \texttt{rsh} command is being piped to a program that is itself reading from standard input (see Example 1).

Note, however, that \texttt{-n} is also required when an \texttt{rsh} command is started in the background.

For example, the call

\begin{verbatim}
rsh host dd if=/dev/nrmt0 bs=20b | tar xvpf -
\end{verbatim}

causes \texttt{tar} to terminate before \texttt{rsh}. The \texttt{rsh} command then tries to write to the broken pipe and competes with the command interpreter for the standard input instead of exiting. The \texttt{-n} option can prevent this from occurring.

This problem only occurs when \texttt{rsh} is located at the start of a pipe and is not reading from standard input. If you want \texttt{rsh} to actually read from standard input, you must not specify the \texttt{-n} option.

For example, if you call the command

\begin{verbatim}
tar cf - . | rsh host dd of=/dev/rmt0 obs=20b
\end{verbatim}

with the \texttt{-n} option, \texttt{rsh} will incorrectly read from \texttt{/dev/null} instead of the pipe.

-\( l..login \)  
The user ID (or login name) with which the user has logged on at the remote host. This user ID must be specified if you want to work on the remote host with a different user ID than on the local host.

-\( x \)  
The \texttt{rsh} command returns the termination status of the remote command if the \texttt{rsh} server on the remote system also supports this feature (BS2000-POSIX, UNIX system).
**rsh**

command

The command to be executed on the remote host.

*command* not specified:

*rsh* uses *rlogin* to log on at the remote host.

option

Arguments to be used with the command on the remote host.

**Mode of operation**

The *rsh* command transfers

- its standard input to the command on the remote host
- the standard output of the command on the remote host to the standard output of the *rsh* command
- the standard error of the command on the remote host to the standard error of the *rsh* command

The *interrupt*, *quit* and *terminate* signals are passed on to the command on the remote host.

The current directory of the started command is set to the *HOME* directory of the user on the remote host.

Metacharacters (e.g. `<`, `>`, `&`) that have not been escaped are evaluated on the local host.

Escaped metacharacters are interpreted on the remote host.

The *rsh* command exits on completion of the command on the remote host.

Screen-oriented programs such as ced, for example, cannot be activated with the *rsh* command. The command *rlogin* must be used instead.

**Example 1**

The POSIX user *john* wants to view the contents of the directory `/home/john/test` on the remote host *rainbow*. The output is to be displayed on the screen in pages. The user ID *john* is also present on the remote host *rainbow*.

```
$ rsh rainbow -n ls -l /home/john/test | more
```

```
total 167232
drwxrwxr-x   2 john 99          1024 Mar 18 09:24 archive
drwxr-xr-x   2 john 99          1024 Apr 14 14:15 de
drwxr-xr-x   2 john 99          1024 Apr 22 08:19 en
...
```

**Example 2**

The POSIX *john* wants to append the contents of the remote file `/home/john/hello_1` on the remote host *rainbow* to the local file *test_1*. The user ID *john* is also present on the remote host *rainbow*.

```
$ rsh rainbow cat /home/john/hello_1 >> test_1
```
Example 3  The POSIX john wants to append the remote file /home/john/travel_1 to the remote file /home/john/holland. Both files are located on the host rainbow. The user ID john is also present on the host rainbow.

$ rsh rainbow cat /home/john/travel_1 ">>" /home/john/holland

Example 4  The POSIX john wants to print the remote file /home/emil/test on the host rainbow at that host. The POSIX john has been entered in the /home/emil/.rhosts file on the host rainbow.

$ rsh rainbow -l emil lpr /home/emil/test

Example 5  A POSIX user wants to print the local file /home/john/dat on the remote host rainbow.

$ rsh rainbow lpr < /home/john/dat

File  /$HOME/.rhosts
      List of host names and user IDs that can log on to the hosts under the listed IDs.

See also  rcp
**sed**

*stream editor*

*sed* is a non-interactive stream editor that provides a similar set of functions to the interactive line editor *ed*.

*sed* is a versatile tool that allows you to:

- perform multiple global editing functions efficiently in one pass through the input
- easily edit piped command output
- apply a sequence of editing commands that is too complicated for interactive input.

*sed* reads files sequentially, edits each line with *sed* commands you have specified in a *sed* script, and writes the results on the standard output. The *sed* script is either read straight from the command line or taken from a file. *sed* acts as a filter, i.e. it does not change the original input file. If you want to save the changes, you will have to redirect the standard output to a file.

**Syntax**

**Format 1:**

```
[ -n ] [script[...file...]]
```

- **-n** Suppresses the default output, which is to pass every input line to the standard output after processing (see Example 6 on page 669).

  - not specified: *sed* copies each processed input line to the standard output, even if the *sed* script does not contain an output command such as *p*. The commands in the *sed* script determine whether a line is modified by editing instructions. Input lines that are processed by an output command such as *p* are thus shown twice in succession. First, as part of the default output of all processed lines, and then as the output of the special editing command which causes them to be displayed again.

  **script**

  - Uses the command-line *script* to edit the input file. If the script contains blanks, newlines, or shell metacharacters, it must be enclosed in single quotes: `'script'`.

  **file**

  - Name of the file whose contents are to be processed by *sed*. The file must be a text file.

  - not specified:

    - *sed* reads from standard input.
Format 2  \texttt{sed[-n][-e..script][-f..scriptfile][..file...]} 

-\texttt{n}  Suppresses the default output, which is to pass every input line to the standard output after processing (see Example 6 on page 669).

-\texttt{n} not specified: \texttt{sed} copies each processed input line to the standard output, even if the \texttt{sed} script does not contain an output command such as \texttt{p}. The commands in the \texttt{sed} script determine whether a line is modified by editing instructions. Input lines that are processed by an output command such as \texttt{p} are thus shown twice in succession. First, as part of the default output of all processed lines, and then as the output of the special editing command which causes them to be displayed again.

-\texttt{e..script}  Uses the command-line \texttt{script} to edit the input file. If the script contains blanks, newlines, or shell metacharacters, it must be enclosed in single quotes: '\texttt{script}'.

The -\texttt{e script} option may be included a number of times with different scripts and may also be combined with the -\texttt{f scriptfile} option. In this case \texttt{sed} applies the commands from all scripts on each input line.

If the command line contains the -\texttt{e} option only once and does not include option -\texttt{f}, you may omit the -\texttt{e} when you specify \texttt{script}.

-\texttt{f..scriptfile}  Reads editing instructions for the input file from the file named \texttt{scriptfile}.

You can include the -\texttt{f scriptfile} option a number of times with different script files and also combine it with -\texttt{e script}. \texttt{sed} then reads the \texttt{sed} commands from all specified \texttt{sed} scripts.

\texttt{file}  Name of the file whose contents are to be processed by \texttt{sed}. The file must be a text file.

\texttt{file} not specified: \texttt{sed} reads from standard input.
Functionality

`sed` works on copies of the input lines, successively copying each line into a temporary work area called a "pattern space".

Each line of input is normally processed in the following cycle:

Step 1: The next (or first) line of input is copied into the pattern space.

Step 2: All `sed` script commands that select the pattern space (i.e. address the last line copied into it) are successively applied to its contents. Depending on the editing instruction in the script, the contents of the pattern space are then altered as required, or left unchanged.

Step 3: The contents of the edited pattern space are then sent to standard output, and the pattern space is cleared.

This cycle is repeated until all input lines have been processed.

The pattern space may occasionally contain multiple lines (see commands `g`, `G`, `N`). However, it is always the address of the last line copied into the pattern space that serves as the pattern space address.

Some `sed` commands (`g`, `G`, `h`, `H`) copy text from the pattern space to a temporary storage area called the "hold space". The hold space saves all or part of the pattern space for subsequent retrieval.

Format of a `sed` script

A `sed` script consists of command lines in the form:

```
[range]sed_command[flags]...
```

or

```
[range]{sed_command[flags]...
   sed_command[flags]...
   .
   .
   .
}
```

The braces are required only if you specify an address `range`. The right brace must be positioned at the start of a line, i.e. only preceded by blanks or tabs.

No blanks are permitted between the `range` and the `sed_command`. 
A *range* serves to select particular input lines and can be specified as one or two comma-separated addresses. When the given *range* "selects" the pattern space, i.e. addresses the last line copied to it, the associated *sed* command or command list is applied to the pattern space.

For *range* you can specify one address or two addresses separated by a comma.

```bash
range = address
```

The pattern space that matches *address* is selected.

```bash
range = address1,address2
```

An inclusive range from the first pattern space that matches *address1* through the next pattern space that matches *address2* is selected. However, if *address2* is a line number in the input file that comes before the one selected with *address1*, only one line is selected (*address1*).

*range* not specified:
The pattern space is always selected, i.e. the last input line copied to it is always addressed.

### Addresses

<table>
<thead>
<tr>
<th>address</th>
<th>Meaning:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>last line of the last input file</td>
</tr>
<tr>
<td>n</td>
<td>nth input line, where n is a positive integer. Input lines are numbered consecutively across all files.</td>
</tr>
<tr>
<td>/pattern/</td>
<td>Input line containing a string matching the specified <em>pattern</em>. Any slash / appearing in <em>pattern</em> must be preceded by a backslash \ to escape it. <em>pattern</em> is a simple regular expression (see <em>Tables and Directories, Regular POSIX shell expressions</em>) with the following modifications:</td>
</tr>
<tr>
<td></td>
<td>The construction ?regular expression?, where ? is any character, is identical to /regular expression/.</td>
</tr>
<tr>
<td></td>
<td>Note that in the address \xabc\xdefx, for example, the second x is escaped and stands for itself, so that the regular expression is abc\xdef.</td>
</tr>
<tr>
<td></td>
<td>The escape sequence \n matches a newline embedded in a pattern space.</td>
</tr>
<tr>
<td></td>
<td>A period matches any character except the terminal newline of the pattern space.</td>
</tr>
</tbody>
</table>
**sed commands**

The following section contains a list of *sed* commands, described in alphabetical order. The square brackets [ ] are not to be entered; they merely indicate that the enclosed address or address range is optional.

The text argument consists of one or more lines. All but the final line must end with a backslash (\) to escape the terminating newline.

```
[address]a\text
  Append text to the pattern space that is output.
```

```
[range]b[label]
  Branch to the *sed* command :label in the *sed* script.

  label not specified:
  Branch to the end of the *sed* script.
```

```
[range]c\text
  Change. Delete the selected pattern space, send text to the output and start the next cycle.
```

```
[range]d
  Delete the contents of the pattern space and start the next cycle. Step 3 is dropped, i.e. the contents of the pattern space are not sent to standard output.
```

```
[range]D
  Delete the initial segment of the pattern space up to (and including) the first newline and start the next cycle.
```

```
[range]g
  Replace the contents of the pattern space by the contents of the hold space.
```

```
[range]G
  Append the contents of the hold space to the pattern space.
```

```
[range]h
  Replace the contents of the hold space by the contents of the pattern space.
```

```
[range]H
  Append the contents of the pattern space to the hold space.
```

```
[address]i\text
  Insert text into the standard output before the contents of the pattern space.
```
[range]l
List the pattern space on the standard output, representing non-printing characters with
replacement characters (e.g. tab characters as the greater-than sign >) or with their
two-digit octal ASCII code equivalents in the form \nn (see section “ASCII character set
(ISO 646)” on page 889). Long lines with more than 71 characters are split into two or
more lines (folded). A backslash at the end of a screen line indicates that the text line
continues in the next output line.

[range]n
Next. Copy the contents of the pattern space to the standard output and replace them
with the next line of input. The address of the last line of input becomes the address of
the pattern space.

[range]N
Append the next line of input to the pattern space with an embedded newline. The
address of the last appended line becomes the address of the pattern space.

[range]p
Print the contents of the pattern space on the standard output. Non-printing characters
are not represented.

[range]P
Print the initial segment of the pattern space, up to the first newline, on the standard
output. Non-printing characters are not represented.

[address]q
Quit sed. If you have specified multiple sed scripts, sed quits at the first q encountered in
any of the scripts.

[range]r...rfile
Read the contents of rfile and send them to the standard output before copying the next
input line to the pattern space. rfile must be separated from the sed command r by
exactly one space and must come at the end of the command line.

[range]s/RE/repstring/[flags]
Substitute repstring for strings that match the regular expression RE in the pattern
space. RE can be specified in the form of a simple regular expression (see section
“Regular POSIX shell expressions” on page 877). The delimiter does not have to be i:
most other characters are accepted. For a fuller description see the s command in ed.
flags
n where n is an integer between 1 and 512. Substitute repstring for just the nth
instance of RE in a line.

g Globally substitute repstring for all instances of RE in a line.
**sed**

**p**
Print the contents of the pattern space on the standard output if a replacement was made. This applies even if *sed* was invoked with the -n option.

**w wfile**
Write. Write the pattern space to the file *wfile* whenever a replacement is made. Any file named *wfile* that was already present before you called *sed* will be overwritten. If a number of *w* commands in a *sed* script write to the same *wfile*, the contents of the pattern space will be appended to the existing contents of *wfile* in each case. *wfile* must be separated from the *sed* command *w* by exactly one space and must come at the end of the command line. A maximum of 10 different files can be used for *wfile* in any one invocation of *sed*.

⚠️ **Caution!**
If you use the name of your input file as *wfile*, you will destroy your input file!

No flags specified:
Only the first instance of *RE* in the line is replaced by *repstring*.

**[range] t label**
Test. Branch to the *sed* command :label in the *sed* script if any substitutions have been made since the last input line was copied to the pattern space or since the most recent execution of a *t* command.

*label* not specified: Branch to the end of the *sed* script.

**[range] w wfile**
Write. Write the pattern space to the file *wfile*. Any file named *wfile* that was already present before you called *sed* will be overwritten. If a number of *w* commands in a *sed* script write to the same *wfile*, the contents of the pattern space will be appended to the existing contents of *wfile* in each case. *wfile* must be separated from the *sed* command *w* by exactly one space and must come at the end of the command line. A maximum of 10 different files can be used for *wfile* in any one invocation of *sed*.

⚠️ **Caution!**
If you use the name of your input file as *wfile*, you will destroy your input file!

**[range] x**
Exchange the contents of the pattern and hold spaces.

**[range] y/string1/string2/**
Transform. Replace each occurrence of a character in *string1* with the corresponding character in *string2*. *string1* and *string2* must be of the same length and must be specified explicitly. Regular expressions cannot be used.
[range] command
[range] ([commandlist])

Don't command. Apply command to all lines not selected by the specified address range. command may be any sed command or a sed command list enclosed in braces {...}.

:label
Set a label for b and t commands to branch to. label is any string up to 8 characters long.

[address]=
Prints the current line number on the standard output on its own line.

[range] [sed_command
    sed-command
    .
    .
    .
]
Successively execute all sed commands enclosed within the braces {} if the address range selects the current pattern space.
The { can be preceded with blank characters and can be followed with white space. The commands can be preceded by white space. The termination } must be preceded by a newline character and then zero or more blank characters.

The newline character is treated as a null command and is ignored. This allows you to produce more transparent sed scripts by using blank lines.

# If # is the first character entered in the first line of a script file, the entire line is interpreted as a comment.

#n If the first line of a script file begins with the character sequence #n, the default output of the pattern space is suppressed (as with the -n option). The entire line is treated as a comment, i.e. not interpreted as an sed command.

Error
sed: command garbled: ...
The sed script contains a syntax error. The colon is followed by the script location at which sed terminated.

Can't open file
You have specified a nonexistent input file or a file for which you do not have read permission.

Unrecognized command: ...
The sed script contains an unknown command.
Locale

The following environment variables affect the execution of `sed`:

- **LANG**
  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_COLLATE**
  Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.

- **LC_CTYPE**
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions. LC_CTYPE also governs which characters the `sed` command `l` treats as non-printing.

- **LC_MESSAGES**
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1
Write the string XXXXX in all blank lines of a file and redirect the output to another file:

```bash
$ sed '/^$/s/^/XXXXX/' file > newfile
```

`/^$/` selects all blank lines, i.e. lines that contain nothing (not even a blank) between start of line and end of line. `sed` searches for the beginning of the line (`^`) and replaces it with the string XXXXX.

Example 2
Indent by 4 spaces all lines that begin with a digit, and redirect the output to another file:

```bash
$ sed '/^[0-9]/s/^/ËËËË/' file > newfile
```

`/^[0-9]/` selects all lines that begin with a digit. `sed` locates the start of the line (`^`) and replaces it with 4 blanks (`ËËËË`), i.e. shifts the rest of the line 4 positions to the right, padding positions 1 to 4 with spaces.

Example 3
Print all the non-blank lines in a file:

```bash
$ sed '/^$/d' file
```

All blank lines are selected by `/^$/` and deleted by `d.`
Example 4  Make a file double-spaced by adding a blank line after each line in the file:

```bash
$ sed 's/$/\n>' file
```

Since no address is specified, `sed` searches for the end of the line ($) in every line and replaces it with a newline character `\n`, thus adding a blank line after every line.

Example 5  Print the second and third column of a file in which individual columns are separated from one another by a colon. The third column is to be placed before the second:

```bash
$ sed 's/\([^:]*:\([^:]*\):\([^:]*\):.*\)/\2:\1/' file
```

Explanation:

```
s/  / //

   The string between the first and second slash is searched for in each line and replaced by the string between the second and third slash.

[^:]*
   Any number (*) of characters other than a colon ([^:])

:   Colon

\  
   Brackets off part of expression for reuse in the replacement string which is between the second and third slash.

\2   The replacement string is to begin with the part of the expression which is in the second bracket `\(...\)`.  

:   There is to be a colon between the first and second part of the replacement string.

\1   The second part of the replacement string is to be the part of the expression which is in the first bracket `\(...\)`.  
```
Example 6  Filter all professions out of the file personnel and write them into the file named professions under a new heading. The following example works on the assumption that the used search key (Profession:) does not appear in the first line of the file.

The personnel file has the following format:

Name: John Miller
Marital status: Divorced
Profession : Journalist

Name: Catherine Baker
Marital status: Married
Profession: Programmer
etc.

sed program:

```
$ sed -n '1{$/^.*/Professions:/
   h
   /^Profession:/s/^Profession: *\(.\)*/\1/
   H
}g
p
}' personnel > professions
```

$ cat professions
Professions:
Journalist
Programmer
etc.

Explanation:

Line 1 in the pattern space is replaced by the string Professions: and then stored in the hold space by the h command. Each line in the pattern space that begins with Profession: is replaced by the string that follows it and then appended to the hold space by the H command. The last line of the file ($) is replaced in the pattern space by the entire contents of the hold space. Finally the pattern space is printed on the standard output by the p command.

The -n option changes the default output so that the input lines copied to the pattern space are not automatically sent to standard output after editing. Only the p command actually prints the pattern space.

See also  awk, ed, grep

Tables and Directories, Regular POSIX shell expressions
**set**

**set or unset options and positional parameters**

The POSIX shell built-in command `set` has three functions:

- If no operand is specified, `set` writes on standard output the names and values of all shell variables defined for the current shell.
  
  By contrast, the `env` command writes on standard output the names and values of all shell variables available to all commands and all subshells: variables such as `IFS`, `MAILCHECK`, `PATH`, `PS1` and `PS2` are not included unless they have been exported (with `export`).

- Any options assigned to `set` govern the behavior of the current shell.
  
  The same options are also available for the `sh` command, but in this case they govern the behavior of the subshell thus generated.

- Arguments used instead of an option or after the option list are handled by `set` as follows:
  
  - it assigns the first nine arguments to positional parameters `$1` through `$9`,
  
  - it assigns all the arguments to the shell parameters `$*` and `$@`,
  
  - it assigns the number of arguments to the shell parameter `#$`.

If there are fewer than nine arguments, the remaining positional parameters are set to the null string.

To access the tenth and subsequent command-line arguments directly you need to use the `shift` command.

If you want to use `set` to control the execution of a shell script, you have to include `set` in the script. This is because shell scripts are always executed by a subshell.

**Syntax**

**Format 1:** `set[-+AabCefhkmnuvx][[-t][-o..option...]][argument...]`

**Format 2:** `set--argument...`

**Format 3:** `set-argument...`

**Format 1**

No operand specified

`set` writes on standard output the names and values of all shell variables defined for the current shell.

No option specified

`set` sequentially assigns the first nine arguments to positional parameters `$1` through `$9`.

It assigns all command-line arguments collectively to the shell parameters `$@` and `$*`, and it assigns the total number of arguments to the shell parameter `#$`. If there are fewer than nine arguments, the remaining positional parameters are set to the null string.

To access the tenth and subsequent command-line arguments directly you need to use the `shift` command.
option

The options govern the behavior of the current shell.

If you want to use more than one option, you have to group the code letters together with no spaces between them and with a single minus sign at the beginning. The -- option cannot be combined with other options.

You can use the command `echo $-` to find out which options have already been set with the `set` or `sh` command. Set options other than `-n` and `-t` can also be unset without the need to terminate the current shell.

The following options are available:

- **-A** name
  Assigns values to fields:
  Deletes the value of the variable `name` and sequentially assigns it values from the argument list.

+**A**  If you use the plus sign `+A`, the values are not deleted before assignment.

-**a**  Automatically exports all shell variables that you define or redefine in the current shell.
  -**a** not specified and not set previously:
    Shell variables that you define or redefine in the current shell are not automatically exported. In other words, a shell variable will not be known in a subshell unless you explicitly export it using the shell built-in `export`.

+**a**  Turns off option `-a`. Any shell variables defined earlier while the `-a` option was active will continue to be exported automatically.

-**b** (background) All background jobs are displayed. The following message is written to standard error output:

```
"[%d] %c %s%s\n", <job-number>,<current>,<status>,<job-name>
```

- **<current>**  The + sign identifies the job that would be used as a default for the `fg` or `bg` utilities; this job can also be specified using the `job_id %+` or `%+.`
  The - sign identifies the job that would become the default if the current default job were to exit; this job can also be specified using the `job_id %-`.

  All other jobs are represented by a blank character. Only one job can be identified with a + and one with a - at any given time.

- **<job-number>**  A number that can be used to identify the process group to the `wait`, `fg`, `bg` and `kill` utilities. Using these utilities, the job can be identified by prefixing the job number with %.

+**b**  `-b` is reset.
-C Prevents files from being overwritten when output is redirected with "">". If the output is
redirected using "">|" then the noclobber option is overridden for the file in question.

+C -C is reset.

-e (exit) The current shell exits immediately if a command returns a non-zero exit status.

-e not specified and not set previously:
An interactive shell exits when you press [END]. A shell running a shell script only exits
prematurely if it encounters a syntax error. Otherwise it exits on reaching the end of the
script, regardless of the exit status of individual commands.

+e Turns off option -e.

-f (file name) Disables the file name generation mechanism. The characters *, ?, and [...] are not treated as
metacharacters and expanded to generate all the matching file
names.

-f not specified and not set previously:
Matching file names are substituted for the above patterns.

+f Turns off option -f.

-h (hash) This option changes the behavior of the current shell when you define shell
functions:
The commands you include in a shell function are entered in the hash table at the time
of function definition (see hash).

-h not specified and not set previously:
Commands used in a shell function are not entered in the hash table until the function
is executed.

+h Turns off option -h.

-k (keyword) Variable assignments, i.e. arguments in the form name=value, are allowed at
any point on the command line. The shell performs the assignment and makes the
resultant keyword parameter available to the environment of the command (see Example 1 on page 67).

-k not specified and not set previously:
Variable assignments must precede the command name. The shell makes the appro-
priate keyword parameters available to the environment of the command.

+k Turns off option -k.

-m Background commands are executed in an individual process group. Command ter-
nination is reported in a line. The exit status of background jobs is acknowledged by a
termination message. If the POSIX shell is inactive this option is automatically selected.

+m -m is reset.
-n (no execution) The current shell reads and interprets all commands but does not execute them, thus omitting the last step in the processing of a command line (see the section “Processing commands using the POSIX shell” on page 40).

The -n option can be used to check a shell script for syntax errors. The -n option is ignored in an interactive shell.

-n not specified and not set previously:
All commands are read, interpreted, and executed.

+n -n is reset (except in interactive shells).

-o argument

\textit{argument} may be one of the following option names:

- \texttt{allexport} Corresponds to \texttt{-a}
- \texttt{errexit} Corresponds to \texttt{-e}
- \texttt{bgnice} All background jobs are executed with low priority. This is the default presetting.
- \texttt{ignoreeof} The POSIX shell is not terminated at end-of-file. To terminate the shell, use the built-in command.
- \texttt{keywords} Corresponds to \texttt{-k}
- \texttt{markdirs} All directories which are created when the file name is generated are terminated by a backslash `/`.
- \texttt{monitor} Corresponds to \texttt{-m}
- \texttt{noclobber} No existing files are overwritten if output is redirected using `>`. 
- \texttt{noexec} Corresponds to \texttt{-n}
- \texttt{noglob} Corresponds to \texttt{-f}
- \texttt{nolog} Function definitions are not stored in the \textit{History} file.
- \texttt{nounset} Corresponds to \texttt{-u}
- \texttt{privileged} Corresponds to \texttt{-p}
- \texttt{verbose} Corresponds to \texttt{-v}
- \texttt{trackall} Corresponds to \texttt{-h}
- \texttt{vi} Depends on the value of the \texttt{TERM} variable. If \texttt{TERM=BLOCK} or if \texttt{TERM} is not set then the command assumes that a BS2000 block-mode terminal is in use and the argument \texttt{vi} is not supported. Any other value of \texttt{TERM} indicates a character-mode terminal.
argument \texttt{vi} changes the input mode to that of a \texttt{vi} type line editor until you press \texttt{[ESC]}. This switches you to command mode. The line sends a newline character.

\textbf{viraw} Depends on the value of the \texttt{TERM} variable. If \texttt{TERM=BLOCK} or if \texttt{TERM} is not set then the command assumes that a BS2000 block-mode terminal is in use and the argument \texttt{vi} is not supported. Any other value of \texttt{TERM} indicates a character-mode terminal. If \texttt{vi} mode is active then the \texttt{viraw} argument causes each character to be processed as soon as it is typed.

\textbf{xtrace} corresponds to \texttt{-x}

\textbf{-t} (terminate) The current shell exits after executing one command, i.e. the command line that contains the \texttt{set -t} command.

\texttt{-t} not specified:
An interactive shell exits when you press \texttt{[END]}. A shell running a shell script only exits prematurely if it encounters a syntax error. Otherwise it exits on reaching the end of the script.
If the \texttt{-e} option is set in the current shell, this shell exits immediately if a command returns a non-zero exit status.

\textbf{-u} (unset) Causes the current shell to issue an error message on encountering an unset shell variable in a command line. The command in question is not executed.
Shell scripts are terminated as soon as the shell encounters an undefined shell variable.

\texttt{-u} not specified and not set previously:
The shell does not issue an error message for an unset shell variable, but substitutes the null string as its value by default.

\textbf{+u} Turns off option \texttt{-u}.

\textbf{-v} (verbose) The current shell displays each subsequent input on standard error before executing the corresponding command.
In combination with the \texttt{-x} option, this option is useful for debugging shell scripts.

\texttt{-v} not specified and not set previously:
The current shell does not echo its input.

\textbf{+v} Turns off option \texttt{-v}.

\textbf{-x} (execute) The current shell interprets its input and writes it on the standard error, marking each command that it processes with a plus sign (+) at the start of the line. A command consisting solely of a variable assignment is output without the plus sign. The shell then executes the command.
In contrast to the output with option \texttt{-v}, the \texttt{-x} option shows you how the shell has replaced metacharacters and shell variables.
Thus you can test a shell script as follows:

- either include `set -xv` as the first command in your shell script, or
- run the shell script with the command `sh -xv script`.

-x not specified and not set previously:
The current shell does not echo its input.

+\textbf{x}  \hspace{0.5em} \textbf{Turns off option}  -x.

\textbf{argument}

Any string which is delimited by a space or a tab. The final argument is terminated by a command separator.

You may enter any number of arguments provided that each is delimited by at least one space or tab.

\textit{set} assigns the first nine arguments to the positional parameters $1$, $2$, ... and $9$. \textit{set} also assigns all the specified arguments to the shell parameters $*$ and $@$ and writes the total number of specified arguments to the shell parameter $#. If fewer than nine arguments are specified, empty strings are assigned to the remaining positional parameters.

Use \textit{shift} to address the tenth and other remaining call arguments directly.

\textbf{argument not specified:}
The positional parameters and the shell parameters $@$, $*$ and $# are not modified.

\textbf{Format 2} \hspace{1em} \textbf{set} -- \textbf{[...argument...]}  

\textbf{- -}  \textbf{(Dashes) Not combinable with other options.}

The first argument specified after -- is not interpreted as an option, even if it begins with a minus sign. This enables you to use \textit{set} to assign values beginning with a minus sign to positional parameters (see Example 3 on page 678).

\textit{set} -- has no effect on any other options currently in effect.

\textbf{argument}

Any string delimited by blanks or tabs. The last argument is terminated by a command separator. You can include any number of arguments, with at least one blank or tab between them.

\textit{set} sequentially assigns the first nine arguments to positional parameters $1$ through $9$. It assigns all command-line arguments collectively to the shell parameters $@$ and $*$; and it assigns the total number of arguments to the shell parameter $#. If there are fewer than nine arguments, the remaining positional parameters are set to the null string.

To access the tenth and subsequent command-line arguments directly you need to use the \textit{shift} command.

\textbf{argument not specified:}
The positional parameters and the shell parameters $@$, $*$ and $# are left unchanged.
set

Built-in sh command

Format 3  set - [...argument...]

- Options -x and -v are reset.

argument

set assigns the first nine arguments to the positional parameters $1, $2, ... and $9. If fewer than nine arguments are specified, empty strings are assigned to the remaining positional parameter.

Use shift to address the tenth and other remaining call arguments directly.

argument not specified:
The positional parameters are left unchanged.

Locale

The following environment variables affect the execution of set:

LANG  Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL  If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH  Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1  The shell script archive contains the following:

    set | grep customer
    echo $customer

The -k option is set in the current shell, and the shell script is then executed as follows:

    $ set -k
    $ sh archive customer=Meyer
    customer=Meyer
    Meyer

After set -k, the shell variable customer may be defined after the script name in the command line. This variable, which is accessed in the example by the echo and set commands, is only known in the subshell that runs the script.
When the script finishes, the subshell that runs it also exits. The variable `customer` is not known in the (now current) parent shell. The command below will therefore output nothing:

```bash
$ set | grep customer
```

If the `-k` option is now turned off with `+k` and the shell script is invoked as before, the `customer` variable will be undefined within the script. The `echo` command simply outputs a blank line. In other words, if the `-k` option is not set, the variable definition must precede the script name on the command line.

```bash
$ set +k
$ sh archive customer=Meyer
$ customer=Meyer sh archive
customer=Meyer
Meyer
```

**Example 2** The shell script `evaltest` has the following contents:

```bash
set -xv
eval echo \$$#
echo $0
```

The `set` command sets the `-x` and `-v` options in the subshell that executes this shell script. The shell script is then called as shown below:

```bash
$ sh evaltest today is friday
eval echo \$$#
+ eval echo $3
+ echo friday
friday
echo $0
+ echo evaltest
evaltest
```

The `-v` option (verbose) causes each command to be displayed before it is processed. The `-x` option causes the shell to interpret each command line and output it with a `+` in the first column. The output shows that the command-line arguments of the `eval` command are interpreted twice by the shell.
Example 3  The following shell script counts all arguments read from the standard input. It is, however, somewhat slower than the `wc -w` command:

```
1  : sh count counts all arguments on standard input
2  num=0
3  while read line
4      do
5       set -- $line
6       num=`expr $num + $#`
7      done
8  echo $num
```

Line 3: The `sh` built-in `read` reads a line from the standard input and assigns it to the variable `line`.

Line 5: The `sh` shell built-in `set` assigns the arguments which make up the value of the variable `line` to the positional parameters `$1`, `$2` and so on; `$#$` is the total number of arguments in `Sl`ine.

In this example the `--` option has two functions:
- Even if `Sl`ine is empty, `set` still does not output the current environment.
- If `Sl`ine starts with a minus sign, `set` does not interpret its first argument as an option.

Line 6: For each line that is read, `expr` increments the value of the `num` variable by the number of arguments in the input line (`#$`).

The commands in lines 5 and 6 are repeated until the last input line has been read in by the `read` command. The `echo` command then outputs the value of the `num` variable, which is the total number of arguments read (total number of words).

See also  `env`, `getopts`, `typeset`
sh  shell, the standard command language interpreter

In the login shell, commands are read sequentially from the file /etc/profile and then, if the files exist, either from the .profile of the current directory or from $HOME/.profile. Following this, commands are also read from the file whose name is defined by the value of the shell variable ENV following parameter substitution.

Syntax

```
sh[...option...][..file][..argument]...
```

No option specified

A subshell with the option -s is called. The shell process which was already current becomes the parent of the new shell. Depending on which terminal you are using, you can terminate the subshell either with [END] or @@d or with the `exit` command.

option

When called, `sh` interprets the options below as well as the options which are described under the POSIX shell built-in `set` command:

- -c command_string
  The commands are read from `command_string`.

- -s
  If the option -s is specified or if `file` and `argument` are specified then the POSIX shell reads the commands from the standard input and writes output to the standard output. Only diagnostic messages are written to the standard error output.

- -i
  If the option -i is specified or if the standard input and output are connected to a video display terminal then an interactive POSIX shell is called. In this case, TERM is ignored to prevent `kill 0` from terminating an interactive shell and INTR is intercepted and ignored so that `wait` can be interrupted. In all cases, the POSIX shell ignores QUIT.

- -r
  The -r option calls a restricted POSIX shell, in which the following restrictions apply:
  \- The shell built-in command `cd` is rejected, which means you cannot leave your current working directory
  \- You cannot change the value of the PATH variable.
  \- Commands are rejected if the command file name contains a slash `/`. You can only execute commands which are in your current working directory or in directories on the paths assigned to the PATH shell variable.
  \- Commands are rejected if the command line includes `>` or `>>`. In other words, you cannot use `>` or `>>` to redirect command output to a file.

To exit a restricted subshell you use [END] or @@d or the `exit` command, depending on terminal type.
file

If option -s has not been specified but file has been, the shell procedure file is searched for at the location specified in the search path. The read permission must be set for the procedure. If the s-bit is set for the owner or group, then it is ignored. Commands are read as described below.

argument

You may enter commands as arguments. The command name is passed as argument zero. The commands are described below.

If the POSIX shell is called by means of the system call exec and if the first character of the zero argument is a hyphen -, then the shell is treated as a login shell.

Exit status

The POSIX shell normally returns the exit status of the last command executed (see section “exit cause the shell to exit” on page 358). Errors detected by the POSIX shell (syntax errors for example) result in an exit status which is not equal to zero. If you are not using the POSIX shell interactively, processing of the script file is interrupted. If the POSIX shell detects runtime errors it reports these by outputting the command name and error conditions. If the line number of the line which contains the incorrect command is greater than one, the line number is output in square brackets [ ... ] after the command or function name.

File

/etc/profile
/etc/suid_profile
$HOME/.profile
/tmp/sh*
/dev/null

Variable

The following variables affect sh:
FCEDIT, HISTFILE, HISTSIZE, HOME, IFS, MAIL, MAIL, MAILCHECK, MAILPATH and PATH (description see the section “POSIX shell variables and parameter substitution” on page 48ff).

Locale

The following environment variables affect the execution of sh:

LANG

Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL

If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE

Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within pattern matching.
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files), which characters are defined as letters, and the behavior of character classes within pattern matching.

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

Determine the location of message catalogs for the processing of LC_MESSAGES.

See also: cat, cd, chmod, cut, echo, env, paste, stty, test, umask, vi, dup(), exec(), fork(), getrlimit(), ioctl(), lseek(), pipe(), signal(), umask(), ulimit(), wait(), rand() [4]
**shift**  
**shift positional parameters**

The POSIX shell built-in `shift` shifts the values of positional parameters over to the left. `shift` without arguments thus results in the following:

- Shell parameter $0 (the command name) is unaffected.
- The original value of $1 is lost; this value can no longer be accessed.
- Instead, $1 takes the original value of $2, $2 that of $3, and so on until $8, which takes the value of $9.
- The tenth command-line argument is passed to the last positional parameter $9.
- $# is reduced by one.
- $* and $@ contain all command-line arguments, starting with the new value of $1. The original value of $1 is dropped.

By default, the shell provides positional parameters which give you direct access to only the first nine command-line arguments of a `set` command or a shell script. This restriction can be bypassed with `shift`, since it allows you to move values over to the left by the required number of places.

**Syntax**

```
shift[...n]
```

- **n** A positive integer; `shift` is executed $n$ times. This means that positional parameter $1$ takes the value of the $(n+1)$th command line argument, etc.

  - If the value of $n$ is too large, `shift` issues an error message as soon as no command-line argument is available for $1$; i.e. $#$ is equal to 0.
  - $n$ not specified:  
    - `shift` is executed once.
    - If $1$ already contains the last argument, the next invocation of `shift` will result in an error message.

**Error**

```
sh: shift: bad number
```

This error message is issued if `number >#$`. As a result no value could be assigned to positional parameter $1$.

**Locale**

The following environment variables affect the execution of `shift`:

- **LANG** Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL** If set to a non-empty string value, override the values of all the other internationalization variables.
Built-in sh command

**shift**

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

Example 1
The tenth command-line argument is to be accessed in a shell script:

```
: 
arg1=$1
shift
arg10=$9
:
```

The original value of `$1` is assigned to the `arg1` variable. The value thus remains accessible even after the `shift` command.

Example 2
The following interactive session demonstrates the changes that take place in the values of the shell parameters `$1`, `$*`, and `$#` when `shift` is invoked:

```
$ set 1 2 3 4 5 6 7 8 9 10 11 12
$ echo $1
1
$ echo $#
12
$ echo $*
1 2 3 4 5 6 7 8 9 10 11 12
$ shift 6
$ echo $1
7
$ echo $*
7 8 9 10 11 12
$ echo $#
6
```

See also **set**
show_pubset_export

show_pubset_export  show file system affected by pubset export

This command supplies the system administrator with information on which file systems in
POSIX are affected by the export of a pubset and must therefore be unmounted before the
EXPORT-PUBSET command is executed.

This information is particularly helpful when bs2fs file systems are used. Unlike with the ufs
and NFS file systems, it is not sufficient in this case to check whether the mount point of a
file system is located on the pubset concerned.

With bs2fs file systems the location of the BS2000 files mounted by means of bs2fs is also
relevant. The mount point of the bs2fs container also plays a role. If it is located on the
pubset to be exported, all mounted bs2fs file systems are affected by the export,
irrespective of their location.

The table below shows when a file system is affected by EXPORT-PUBSET:

<table>
<thead>
<tr>
<th>File system type</th>
<th>Affected if ... located on the pubset to be exported</th>
</tr>
</thead>
<tbody>
<tr>
<td>ufs</td>
<td>Mount point or container file</td>
</tr>
<tr>
<td>nfs</td>
<td>Mount point</td>
</tr>
<tr>
<td>bs2fs</td>
<td>Mount point or mounted BS2000 files or mount point/container file of the bs2fs container (cf. ufs)</td>
</tr>
</tbody>
</table>

Syntax

```
show_pubset_export cat-id
```

cat-id
Catalog ID of the pubset which is to be checked (without enclosing colons ":"). The entry
can be made in upper- or lowercase notation or a mixture of both; the check is
performed with the cat-id converted to uppercase notation.

Files

The following files are searched for the specified catalog ID in order to determine the file
systems affected:

```
/etc/mnttab
Table of all mounted file systems
```

```
/etc/partitions
Table of all possible partitions
```

If the catalog ID entry is missing in this file, the default ID is determined via BS200 and used
for the check.

```
SYSSSI.POSIX-BC.<version>
SYSSSI file of POSIX from BS2000
```

The BS2000 file name of the container file of the root file system is determined from this file
(ROOTFSNAME parameter) as this name is not entered in the /etc/partitions file.
Example

The file systems affected by the export of the DATA pubset are determined. The container file of the ufs file system mounted under the home directory `/home/bach` resides on the DATA pubset.

```
# show_export DaTa

the nfs filesystems on pubset DATA

nfs filesystem PGTR0157:/home4 mounted on /home/bach/nfs4
nfs filesystem PGTR0157:/home5 mounted on /home/bach/nfs5

the bs2fs filesystems on pubset DATA

bs2fs filesystem :DATA:$BACH.ASS.* mounted on /home/bach/bs2/mount.ass
bs2fs filesystem :V70A:$BACH.CCC.* mounted on /home/bach/bs2/mount.ccc
bs2fs filesystem :DATA:$BACH.PLAM* mounted on /home/bach/bs2/mount.plam

the ufs filesystems on pubset DATA

ufs filesystem /dev/dsk/4 mounted on /home/froede
ufs filesystem /dev/dsk/5 mounted on /home/bach
```

#
sleep suspend execution for an interval

The `sleep` command suspends the execution of the process that calls it for a user-defined period of time. `sleep` is used mainly in shell scripts to delay the execution of the next command.

**Syntax**

```
sleep [time]
```

- **time**
  - The time in seconds after which execution of the process is to resume.
  - `time` must be a non-negative decimal number.

**Error**

```
sleep: bad character in argument
```
You have used a negative integer or a non-numeric expression for `time`.

**Locale**

The following environment variables affect the execution of `sleep`:

- **LANG**
  - Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  - If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**
  - Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**
  - Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLS_PATH**
  - Determine the location of message catalogs for the processing of `LC_MESSAGES`. 
**Example 1**  This example makes use of `sleep` from the command line. You start a background process to remind yourself in 10 minutes (600 seconds) that you need to make a phone call:

```
$ (sleep 600; echo 'Call Mr. Jones!') &
```

**Example 2**  In this example, `sleep` is used within a shell script named `always`, which calls the `backup` program every two minutes:

```
$ cat always
while true
do
  backup
  sleep 120
done
```

If you run the `always` script as a background process, you will only be able to terminate it with `kill`, not with the [DEL] key.

See also  `alarm()`, `sleep()` [4]
sort

sort, merge or sequence check text files

sort sorts lines in an input file and writes the result on the standard output.

If you specify more than one file, sort sorts and merges the files in the same operation, i.e. the contents of all input files are sorted and printed together.

Sorting can be performed either by whole lines or by specific parts of lines, known as sort keys. If you wish to sort by whole lines, you do not specify any sort keys; one or more keys can be used to sort by particular portions of lines. A sort key is defined by specifying the positions of fields in a line in the form `+pos1 -pos2` (see "Defining specific sort keys" on page 691).

sort divides the lines of a file into fields. A field is a string of characters that is delimited by a field separator or a newline. Blanks and tabs are the default field separators. In a sequence of one or more default separators, all separators are part of the next field. Leading blanks at the beginning of a line thus by default form part of the first field.

Syntax

**Format 1:**
```
sort[-m][-o...output_file][-bdfiMnru][-t...char]
[-k..keydef][-z...recsz][-y[kmem]][-T...directory][...file...]
```

**Format 2:**
```
sort...c[-bdfiMnru][-t...char]
[-k..keydef][-z...recsz][-y[kmem]][-T...directory][...file...]
```

**Format 3:**
```
sort[-m][-o...output_file][-bdfiMnru][-t...char]
[+pos1[-pos2]][-z...recsz][-y[kmem]][-T...directory][...file...]
```

**Format 4:**
```
sort...c[-bdfiMnru][-t...char]
[+pos1[-pos2]][-z...recsz][-y[kmem]][-T...directory][...file...]
```

The formats are described together because
- options `-m` and `-o...output_file` in formats 1 and 3 are substituted for option `-c` in formats 2 and 4,
- option `-k..keydef` in formats 1 and 2 is substituted for option `+pos1[-pos2]` in formats 3 and 4.

No option specified
Input lines are sorted lexicographically by bytes (characters) in machine collating sequence.
Options that alter the behavior of sort

Multiple options may be used, provided they are specified individually, each preceded by a blank and a minus sign.

- **c**  sort checks whether the input file is already sorted according to the current ordering rules. If it is, nothing is output; otherwise, the first line that does not match the ordering rules is displayed.

  Only one file may be specified with option -c!

- **m**  sort merges input files which are already sorted.

- **o**  `output_file`  
  `output_file` is the name of a file to which the sorted contents of the input file are to be written. The file named as `output_file` can also be one of the input files, but in this case the original unsorted contents of the named file are overwritten.

  -o `output_file` not specified:  
    `sort` writes on the standard output.

- **T**  `directory`  
  Specifies a directory for temporary files.

- **u**  (unique) Causes identical lines to be output once only. Lines with identical sort keys are considered identical lines.

- **y**  
  Option -y defines the memory size that `sort` uses to start with. This initial size has a large impact on the speed with which the file is sorted. It is a waste of memory or of CPU time to sort a small file in a large amount of memory or a large file in a small amount of memory respectively.

  `kmem`  
  Amount of memory (in Kbytes) initially assigned to `sort`. If you assign a value above the maximum of 1 Mbyte or below the minimum of 16 Kbyte, the corresponding extremum will be used. Thus if you define a value of 0 (-y0), for example, `sort` will start with minimum memory.

  -y `kmem` not specified:  
    `sort` starts with maximum memory.

  -y `kmem` not specified:  
    `sort` starts with a system default memory size (32 Kbytes), and continues to use more space if required.
-z recsz

With this option you allocate correctly sized buffers for the merge phase. You only need to do this if you are using option -c or -m, i.e. if you are not actually sorting the files:

If you are sorting the files, sort records the size of the longest line read in the sort phase so that buffers of the correct size can be allocated during the merge phase.

If you are not sorting the files, sort normally uses a default value for the buffer size. Lines longer than this will cause sort to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

Options that alter ordering rules

You have two possibilities to specify the following options:

- Before the first +pos1 specification:

  They then apply globally to all sort keys specified with +pos1.

  Multiple options can either be specified as usual, each with a minus sign and delimiting blanks, or they can be grouped together without intervening spaces and with just one minus sign at the beginning.

- After a +pos1 or -pos2 specification:

  They then override global settings for the sort key to which they are attached, i.e. the altered ordering rule applies only to the preceding position specification.

  These options are directly appended to +pos1 or -pos2 without minus signs and blanks.

-b Ignores leading field separators when determining the start and end of a sort key. Note that the b option is only effective when sorting is based on sort keys (i.e. not on the whole line).

-d Performs a lexicographical sort, taking into account only the characters for which the C functions isalnum() and isspace() return a value of "true". These are the characters defined in the current locale as alphanumeric letters, digits, or characters producing white space, such as blanks or tabs.

-f Folds lowercase into uppercase before sorting, thus making no distinction between them.

-i In non-numeric comparisons, ignores all characters for which the C function isprint() returns a value of "false", i.e. all characters defined as non-printing in the current locale. If the collating sequence is based on the ASCII table, for example, characters 001 through 037 (octal) and character 0177 (octal) are ignored (see section "ASCII character set (ISO 646)" on page 889).
-M The first three characters of the sort key are converted to uppercase, treated as names of months, and collated in calendar order. The -M option implies the -b option.

-n Sorts numerically. A numeric value must come first in the sort key and may consist of: blanks, minus signs, digits 0-9, and a decimal point. The -n option implies the -b option, i.e. leading blanks are ignored.

-r Reverses the collating sequence (sorting order).

Option that alters field separators

This option must be specified separately with a minus sign.

-t..char
Uses the character you specify for char as the field separator. Unlike default field separators, char is itself not part of a field. It may, however, be part of a sort key, for example if the sort key extends from the first to the third x-separated field. Every field separator char is significant, i.e. charchar delimits an empty field.

-t..char not specified:
The default field separators apply (blanks and tabs). A sequence of one or more default field separators forms part of the following field.

Defining specific sort keys

When defining sort keys please note that sequences of letters defined as one collating element in the current locale count as a single letter. In a Spanish locale, for example, ch is a single collating element.

-k..keydef
Defines the sort fields. keydef is defined as a sort field in the following form:

\[start\text{\_of\textunderscore sort\textunderscore field[type]}[,end\text{\_of\textunderscore sort\textunderscore field[type]]}\]

where start\_of\textunderscore sort\_field corresponds to +pos1 and end\_of\textunderscore sort\_field to -pos2 (see description below). type corresponds to one of the options b, d, f, i, n or r.

+pos1[-pos2]
+pos1 and -pos2 specify the start and end of a sort key on the basis of the fields in the input lines.
+pos1 is the position of the first character in the sort key;
-pos2 refers to the first character after it. +pos1 must come before -pos2.

-pos2 not specified:
The sort key extends from +pos1 to the end of the line.
The pos1 and pos2 arguments have the form:
m[,n]
where \( m \) and \( n \) are integers with the following significance:

- **\( m \)** Skips \( m \) fields of the line, addressing field \( m+1 \).
- **\( n \)** Skips \( n \) characters plus the field separator as of the last character of field \( m \), thus addressing character \( n+1 \) within field \( m+1 \). If the \(-b\) option is in effect, field separators at the start of a field are not counted; thus, \(+m.nb\) refers to the \( n+1 \)th non-whitespace character after field \( m \).

- **\( n \) not specified:**
  Is equivalent to \( .0 \) and refers to the first character after field \( m \). If the \(-b\) option is in effect, field separators at the start of a field are not counted; thus, \( +m.0b \) refers to the first non-whitespace character in the \( m+1 \)th field.

**Example**

To specify a sort key that begins with the fourth character in the second field and ends with this field, you enter:

```
sort +1.3 -2
```

**Explanation:**

```
End   End   End
Field1 Field2 Field3
030-456537 A.Mackenzie  Dublin
```

**Sort key:**

- **\(+1.3\)** Skip field 1 and 3 characters:
  the 4th character after field 1 is the 1st character in the sort key: M

- **\(-2\)** Skip field 2 and 0 characters:
  the 1st character after field 2 is the 1st character after the sort field: blank.
  Thus the character before is the last character in the sort key: n
  Note that default field separators, unlike those defined with option \(-t\), are part of the following field. Hence the first character of field 2 is the blank, the second character is the A, and so on.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal.

**file**

Name of the file you wish to sort.

You may name more than one file. All named files are sorted and merged, and the input lines from all of them together are sorted and written to standard output. In the input files, any letter sequence defined as a collating element in the current locale counts as a single letter. Thus in a Spanish locale \( \text{ch} \) is a single collating element. When the last line in an input file is missing a newline character, \textit{sort} appends one, issues a warning, and continues.
Only one file may be specified together with the -c option.

If you use a dash (-) as the name for file, sort reads from standard input.

file not specified:
sort reads from standard input.

Exit status
The following exit status values may occur:

0  All input files were processed successfully. If option -c was set, then the input file was sorted correctly.
1  If -c was set the input file was not sorted as specified. If both -c and -u were set two identical input lines were found with the same sort field.
>1  An error has occurred.

Locale
The following environment variables affect the execution of sort:

LANG  Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL  If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE  Determine the preset collating sequence used by the sort command.

LC_CTYPE  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files). LC_CTYPE also governs how character classes are handled by the -b, -d, -f and -i options.

LC_MESSAGES  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_NUMERIC  Determine the form of the radix character (decimal point) in conjunction with the -n option.

LC_TIME  Determine the currently valid month names, their abbreviations and their collating sequence in conjunction with option -M.

NLSPATH  Determine the location of message catalogs for the processing of LC_MESSAGES.
sort

Example 1  Sorting the contents of input_file with the second field as the sort key.

    $ sort +1 -2 input_file

Example 2  You wish to sort the contents of input_file1 and input_file2 in reverse order on the second character of the second field (= 1st character which is not a space if fields are all separated by one space). The output is to be written to output_file.

    $ sort -r -o output_file +1.0 -1.2 input_file1 input_file2

Example 3  Sorting the contents of input_file1 and input_file2 in reverse order, placing the output in output_file, and using the first character in the second field as the sort key.

    $ sort -r -o output_file +1.0b -1.1b input_file1 input_file2

Example 4  Displaying the presorted file input_file, suppressing all but the first occurrence of lines having the same third field.

    $ sort -u +2 -3 input_file

See also  comm, join, uniq
split

**split**  split a file into pieces

*split* divides a file into smaller segments, storing the segments in individual output files and leaving the original file unaltered.

The output files are automatically numbered; *split* uses a suffix comprising two lowercase letters (aa, ab ... zz) from the current internationalized environment for this purpose. The last file contains the remainder of the input file and may contain fewer lines than the number specified.

If the number of output files required exceeds the maximum length allowed by the suffix, *split* does not write the last file (as this would contain more lines than specified) and terminates with the exit status >0. The files that have already been created are not deleted.

### Syntax

**Format 1:** `split [-b bytes][-a number][file][name]`

**Format 2:** `split [-l lines][-a number][file][name]`

**Format 3:** `split [lines][-a number][file][name]`

No option specified

The output files are called xaa, xab etc. up to xzz in lexicographical sequence. In this case, *split* creates a maximum of 676 output files.

**Format 1**  
`split [-b bytes][-a number][file][name]`

- `a..number`
  
  The suffix for the output file consists of `number` letters. For example, `-a 4` creates the output files xaaaa, xaaab etc. up to xzzzz.
  
  Thus the maximum theoretically possible number of file names is $26^{number}$ for $0<number<8$. If `number` is greater than 7, it defaults to `UINT_MAX`.

- `-a` not specified:
  
  The suffix consists of 2 letters.

- `b..bytes`

  *split* splits the input file into sections of size `bytes`. `bytes` can be specified as follows:

  - `n` as the number of bytes
  - `n k` as a multiple of 1024 bytes
  - `n m` as a multiple of 1048576 bytes

- `file`

  Name of the input file you want to split.
  
  If you use a dash (-) as the name for `file`, *split* reads from standard input.

  `file` not specified:

  *split* reads from standard input.
**split**

name
Name of the output files: The first output file is given the name `nameaa`, the second receives the name `nameab`, and so on lexicographically, up to `namezz`. So `name` must be at least 2 characters shorter (or `number` characters shorter if `-a` is specified) than the maximum file name length (`{NAME_MAX}` bytes) allowed in the file system.

If you specify a value for `name`, the `file` argument is mandatory.

Format 2  split[[-l...lines][-a...number][...file][...name]]

Format 3  split[[-l...lines][-a...number][...file][...name]]

- `-l...lines`  
  `split` splits the input file into sections containing `lines` lines.
  
  This corresponds to the old option `...lines`, which is still supported.
  
  - `l` not specified:
    `split` splits the input file into sections containing 1000 lines.

- `-a...number`  
  The suffix for the output file consists of `number` letters. For example, `-a...4` creates the output files `aaaa`, `aaab` etc. up to `zzzz`.
  
  Thus the maximum theoretically possible number of file names is $26^{\text{number}}$ for $0<\text{number}<8$. If `number` is greater than 7, it defaults to `UINT_MAX`.

  - `a` not specified:
    The suffix consists of 2 letters.

file
Name of the input file you want to split.

If you use a dash (`-`) as the name for `file`, `split` reads from standard input.

- `file` not specified:
  `split` reads from standard input.

name
Name of the output files: The first output file is given the name `nameaa`, the second receives the name `nameab`, and so on lexicographically, up to `namezz`. So `name` must be at least 2 characters shorter (or `number` characters shorter if `-a` is specified) than the maximum file name length (`{NAME_MAX}` bytes) allowed in the file system.

If you specify a value for `name`, the `file` argument is mandatory.
Locale

The following environment variables affect the execution of `split`:

`LANG`  
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

`LC_ALL`  
If set to a non-empty string value, override the values of all the other internationalization variables.

`LC_CTYPE`  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

`LC_MESSAGES`  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

`NLSPATH`  
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example 1  
The contents of the file `example` are to be split into several 20-line files:

```bash
$ split -l 20 example
$ ls
 example
 xaa
 xab
 xac
 xad
```

Example 2  
Every two lines from the standard input are to be written into files named `out...`. Since the names of output files are explicitly specified (`out`) in this case, the minus sign (`-`) for standard input must not be omitted!

```bash
$ split -l 2 - out
What is true was always true
and will always remain true.
But what is not true, was never reality
and will never become reality.
END
```

```bash
$ ls
 outaa
 outab
```

See also  
`csplit`
**start_bs2fsd**

### start_bs2fsd  start copy daemons

`start_bs2fsd` enables the system administrator to start additional copy daemons `bs2fsd`.

#### Syntax

```
start_bs2fsd [number]
```

No option specified

`start_bs2fsd` provides information about the number of copy daemons currently running.

**number**

Specifies how many copy daemons are to be started in addition to the copy daemons already started. Up to 8 copy daemons can execute.

#### Example

Check how many daemons are currently running

```
# start_bs2fsd
/sbin/start_bs2fsd: 2 bs2fs daemons are running
```

Start an additional daemon and repeat the check

```
# start_bs2fsd 1
/sbin/start_bs2fsd: start additional daemon 1 of 1
# start_bs2fsd
/sbin/start_bs2fsd: 3 bs2fs daemons are running
```
strings  find printable strings in files

`strings` looks for strings in binary files and writes them on standard output. A string is by default any sequence of 4 or more printable ASCII characters ending with a newline or a null byte (see section “ASCII character set (ISO 646)” on page 889). `strings` is useful for identifying random object files and many other things.

Syntax

| Format 1: `strings[-a][-o][-t..format][-n..number]...file...` |
| Format 2: `strings[-t][-o][-t..format][-n..number]...file...` |

- `-a` `strings` searches the entire file for printable strings.
  
  This corresponds to the old option `-`, which is still supported.

  `-a` not specified:

  `strings` only looks in the initialized data space of object files.

- `-n..number`

  Defines a `strings` as any sequence of `number` or more printable characters ending with a newline or a null byte.

  This corresponds to the old option `-number`, which is still supported.

  `-n` not specified:

  A string is any sequence of 4 or more printable characters ending with a newline character or a null byte.

- `-o` `strings` precedes each string by its offset in the file.

- `-t..format`

  Each string is output preceded by its offset in the file. The `format` of the positioning specification is defined as follows:

  - `d` Decimal positioning specification
  - `o` Octal positioning specification
  - `x` Hexadecimal positioning specification

  `file`

  Name of the file in which `strings` is to look for printable strings.

Locale

The following environment variables affect the execution of `strings`:

`LANG`

Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
strings

**LC_ALL**  If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and to identify printable strings.

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  Determine the location of message catalogs for the processing of **LC_MESSAGES**.

Example  Finding all printable strings in the executable binary file *a.out*:

```
$ strings a.out
L
LLM-UFS
~ 2009-01-27 16:24:55
2009-01-27 16:24:552009-01-27 16:24:55-
: hallo hallo
~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
~~~~
C(BS2000) COMPILER: V03.2B00,COMPILATION DATE: 2009-01-27,REP AREA:
:
```

This could be the output for a file that originally held the following source code:

```c
#include <stdio.h>
main()
{
 printf("Hallo, user\n");
}
```

See also  *nm, od*
### stty

**check and change terminal options (set terminal type)**

*stty* can be used to check or change terminal options.

> Depending on the implementation, some *stty* options such as *parext*, *iexten* and *siflush* are accepted but not interpreted by the driver, so these options will not have any effect.

The *stty* command is only useful when you are accessing the POSIX shell via rlogin. The *stty* command has no effect on BS2000 block-mode terminals; *stty* can only be used to change the options on character-mode terminals.

**Syntax**

```bash
stty[...option]...
```

No option specified

*stty* reports some of the option settings for the device that is its current standard input.

### Options for defining the scope of the output

- `-a` (a - all) Reports all currently valid terminal option settings.
- `-g` Reports current settings in a form that can be used as input to another *stty* command.

### Options for setting input/output characteristics

The following section describes the various options available when setting I/O characteristics for the terminal that is *stty*’s current standard input. The I/O options are listed in five main groups:

- Data transfer control options
- Input control options
- Output control options
- Local features
- Control assignments.

An additional group constitutes combinations of options from these five groups.

Explanations related to parenthesized options are also indicated within parentheses.

Note that even though many combinations of options make no sense, *stty* does no sanity checking.
Data transfer control options

As serial lines are not supported on BS2000, the data transfer control options do not work. They are summarized here for reasons of completeness:

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parenb</td>
<td>Enable parity generation and detection</td>
</tr>
<tr>
<td>parext</td>
<td>Extended parity generation and detection</td>
</tr>
<tr>
<td>parodd</td>
<td>Odd parity</td>
</tr>
<tr>
<td>csize</td>
<td>Define character size: cs5, cs6, cs7 or cs8</td>
</tr>
<tr>
<td>ispeed number</td>
<td>Set terminal input baud rate to number</td>
</tr>
<tr>
<td>ospeed number</td>
<td>Set terminal output baud rate to number</td>
</tr>
<tr>
<td>number</td>
<td>Set baud rate to number</td>
</tr>
<tr>
<td>hupcl</td>
<td>Hang up connection on last close(2)</td>
</tr>
<tr>
<td>hup</td>
<td>Same as hupcl</td>
</tr>
<tr>
<td>cstopb</td>
<td>Use two stop bits per character</td>
</tr>
<tr>
<td>cread</td>
<td>Enable receiver</td>
</tr>
<tr>
<td>clocal</td>
<td>Modem control signals not supported</td>
</tr>
</tbody>
</table>

Input control options

The following options can be used to control the input of data:

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignbrk</td>
<td>Ignore break on input</td>
</tr>
<tr>
<td>brkint</td>
<td>SIGINT on break</td>
</tr>
<tr>
<td>ignpar</td>
<td>Ignore parity errors</td>
</tr>
<tr>
<td>parmrk</td>
<td>Mark characters with parity errors</td>
</tr>
<tr>
<td>inpck</td>
<td>Enable input parity checking</td>
</tr>
<tr>
<td>istrip</td>
<td>Mask input characters to 7 bits</td>
</tr>
<tr>
<td>inlcr</td>
<td>Map newline to carriage return on input</td>
</tr>
<tr>
<td>igncr</td>
<td>Ignore carriage return on input</td>
</tr>
<tr>
<td>icrnl</td>
<td>Map carriage return to newline on input</td>
</tr>
<tr>
<td>iucic</td>
<td>Map uppercase letters to lowercase on input</td>
</tr>
<tr>
<td>ixon</td>
<td>Enable START/STOP output control</td>
</tr>
<tr>
<td>ixany</td>
<td>Allow any character to restart output</td>
</tr>
<tr>
<td>ixoff</td>
<td>Do not send STOP character when input queue is nearly full</td>
</tr>
</tbody>
</table>
ignbrk (-ignbrk)
   Ignore (do not ignore) break on input.

brkint (-brkint)
   Send (do not send) SIGINT on break.

ignpar (-ignpar)
   Ignore (do not ignore) parity errors.

parmrk (-parmrk)
   Mark (do not mark) characters with parity errors.

inpck (-inpck)
   Enable (disable) parity checking on input.

istrip (-istrip)
   Mask (do not mask) input characters to 7 bits.

inlcr (-inlcr)
   Map (do not map) newline character to carriage return on input.

igncr (-igncr)
   Ignore (do not ignore) carriage returns on input.

icrnl (-icrnl)
   Map (do not map) carriage return to newline character on input.

iuclc (-iuclc)
   Map (do not map) uppercase letters to lowercase on input.

ixon (-ixon)
   Enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (-ixany)
   Allow any character (only the ASCII character DC1) to restart output.

ixoff (-ixoff)
   Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.
Output control options

The following options can be used to control the output of data:

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opost</td>
<td>Post-process output in accordance with subsequent options</td>
</tr>
<tr>
<td>olcuc</td>
<td>Map lowercase letters to uppercase on output</td>
</tr>
<tr>
<td>ocrnl</td>
<td>Map carriage return to newline on output</td>
</tr>
<tr>
<td>onocr</td>
<td>Do not output carriage returns in column 0</td>
</tr>
<tr>
<td>onlret</td>
<td>Newline also performs the carriage return function</td>
</tr>
<tr>
<td>ofill</td>
<td>Use fill characters for data transmission delays</td>
</tr>
<tr>
<td>ofdel</td>
<td>Use abort character DEL 0x7 as fill character</td>
</tr>
<tr>
<td>cr0,1,2,3</td>
<td>Select extent of delay for carriage returns on output</td>
</tr>
<tr>
<td>nl0,1</td>
<td>Select extent of delay for newline characters on output</td>
</tr>
<tr>
<td>tab0,1,2,3</td>
<td>Select extent of delay for horizontal tabs on output</td>
</tr>
<tr>
<td>bs0,1</td>
<td>Select extent of delay for backspaces on output</td>
</tr>
<tr>
<td>f0,1</td>
<td>Select extent of delay for form feeds on output</td>
</tr>
<tr>
<td>vt0,1</td>
<td>Select extent of delay for vertical tabs on output</td>
</tr>
</tbody>
</table>

**opost (-opost)**

Post-process output in accordance with the other specified options (do not post-process output: ignore all other output control options).

**olcuc (-olcuc)**

Map (do not map) lowercase letters to uppercase on output.

**ocrnl (-ocrnl)**

Map (do not map) carriage return to newline on output.

**onocr (-onocr)**

Do not (do) output carriage return in column 0.

**onlret (-onlret)**

A newline character also performs (does not perform) the carriage return function on output.

**ofill (-ofill)**

Use fill characters (use timing) for data transmission delays.

**ofdel (-ofdel)**

Use abort character DEL 0x7 as fill character.

**cr0 cr1 cr2 cr3**

Select extent of delay for carriage returns on output.
stty, Local features

nl0 nl1
Select extent of delay for newline characters on output.

tab0 tab1 tab2 tab3
Select extent of delay for horizontal tabs on output.

bs0 bs1
Select extent of delay for backspaces on output.

ff0 ff1
Select extent of delay for form feeds on output.

vt0 vt1
Select extent of delay for vertical tabs on output.

Local features

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isig</td>
<td>Enable checking of input characters against control characters intr, quit and swtch</td>
</tr>
<tr>
<td>icanon</td>
<td>Enable canonical input</td>
</tr>
<tr>
<td>xcase</td>
<td>Enable canonical upper/lowercase presentation</td>
</tr>
<tr>
<td>echo</td>
<td>Echo back every character typed</td>
</tr>
<tr>
<td>echok</td>
<td>Output an additional newline after the kill character</td>
</tr>
<tr>
<td>echonl</td>
<td>Output newline characters</td>
</tr>
<tr>
<td>nofsh</td>
<td>Disable flushing of the I/O queue</td>
</tr>
<tr>
<td>tostop</td>
<td>Send SIGTTOU when background processes write to the terminal</td>
</tr>
<tr>
<td>iexten</td>
<td>Enable extended functions for input data</td>
</tr>
</tbody>
</table>

isig (-isig)
Enable (disable) the checking of each input character against the special control characters intr, quit, and swtch (see “Control assignments” on page 706). If a match is found, the function associated with the control character is executed (not executed).

icanon (-icanon)
Enable (disable) canonical input, i.e. the special functions of the characters erase and kill are (are not) performed on input.

xcase (-xcase)
Enable (disable) canonical uppercase/lowercase presentation.

echo (-echo)
Echo back (do not echo back) every character typed.
**stty, Local features**

**echo (−echoe)**
Echo (do not echo) erase characters as a backspace-space-backspace string.

Setting the *echo* option will erase the character overwritten with erase on many terminals; however, since this mode does not keep track of the column position, problems may arise with escaped characters, tabs, and backspaces.

**echok (−echok)**
Output (do not output) an additional newline after a kill character.

**echonl (−echonl)**
Output (do not output) newline characters.

**noflsh (−noflsh)**
Disable (enable) the flushing of input and output queues on detection of an intr, quit, or swtch signal.

**tostop (−tostop)**
Send (do not send) SIGTTOU when background processes write to the terminal.

**iexten (−iexten)**
Enable (disable) special control characters not currently controlled by *icanon, isig, ixon* or *ixoff*.

**Control assignments**

**control-character c**
Set *control-character* to *c*, where *control-character* can be: ctab, discard, dsusp, eof, eol, eol2, erase, intr, kill, lnext, quit, reprint, start, stop, susp, swtch, werase, *min* or *time* (*min* and *time* are used with *-icanon*; *ctab* is used with *-stappl*). If *c* is preceded by the ^ character, then the value used is the corresponding CTRL character (e.g. ^D corresponds to [CTRL] D). ^? is interpreted as [DEL], and ~ is interpreted as undefined.

**min number/time number**
Set the value of *min* or *time* to *number*. *min* and *time* are used with *-icanon* (see termio() [4]).

**line i**
Set line discipline to *i* (0 < *i* < 127).
Combination modes

The following combination modes may be used to control the terminal:

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>evenp</td>
<td>Enable $parenb$ and $cs7$</td>
</tr>
<tr>
<td>parity</td>
<td>Same as evenp</td>
</tr>
<tr>
<td>oddp</td>
<td>Enable $parenb$, $cs7$, and $parodd$</td>
</tr>
<tr>
<td>-evenp</td>
<td>Disable $parenb$, and enable $cs8$</td>
</tr>
<tr>
<td>-parity</td>
<td>Same as parity</td>
</tr>
<tr>
<td>-oddp</td>
<td>Disable $parenb$ and $parodd$, and enable $cs8$</td>
</tr>
<tr>
<td>raw</td>
<td>Enable raw mode</td>
</tr>
<tr>
<td>nl</td>
<td>Disable $icrnl$, $ocrnl$, $inlcr$, $igncr$, $ocrnl$, and $onlret$</td>
</tr>
<tr>
<td>lcase</td>
<td>Enable $xcase$, $iuecl$, and $oluc$</td>
</tr>
<tr>
<td>LCASE</td>
<td>Same as lcase</td>
</tr>
<tr>
<td>tabs</td>
<td>Preserve tabs when printing</td>
</tr>
<tr>
<td>ek</td>
<td>Reset erase and kill to original values</td>
</tr>
<tr>
<td>sane</td>
<td>Reset all terminal options to reasonable values</td>
</tr>
</tbody>
</table>

**evenp** or **parity**

Enable $parenb$ and $cs7$.

**oddp**

Enable $parenb$, $cs7$, and $parodd$.

**-parity** or **-evenp**

Disable $parenb$, and enable $cs8$.

**-oddp**

Disable $parenb$ and $parodd$, and enable $cs8$.

**raw** (**-raw** or **cooked**)

Enable (disable) raw input and output; raw mode is equivalent to: $cs8$, with no erase, kill, intr, quit, swtch, eof, or output post-processing, and no parity bit.

**nl** (**-nl**)

Disable (enable) $icrnl$, and $onlcr$. In addition, **-nl** disables $inlcr$, $igncr$, $ocrnl$, and $onlret$.

**lcase** (**-lcase**)

Enable (disable) $xcase$, $iuecl$, and $oluc$.

**LCASE** (**-LCASE**)

Same as **lcase** (**-lcase**).
tabs (-tabs or tab8)
  Preserve tabs (expand to spaces) when printing.

ek  Reset control characters erase and kill back to system-specific default values.

sane  Reset all terminal options to some reasonable values.

Locale

The following environment variables affect the execution of stty:

- **LANG**
  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments) and which characters are printable.

- **LC_MESSAGES**
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1

Check current settings (only on access to POSIX shell via rlogin):

```
$ stty -a
speed 38400 baud;
intr = ^r; quit = \; erase = ^h; kill = ^x;
eof = ^d; eol = <undef>; eol2 = <undef>; swtch = <undef>;
start = ^s; stop = ^t; susp = ^z; dsusp = <undef>;
rprnt = ^r; flush = ^o; weerase = ^w; lnext = ^v;
-parnod -parodd cs8 -cstopb hupcl cread -clocal -clocal -parext
-ignbrk brkint ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl -icrnl
-ixanx -ixoff -imaxbel
-isig icanion -xcase echo echoe echok -echonl -noflsh
-tostop -echoctl -echopr -echoe -defecho -flusho -pendin -iexten
apost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel -tabs
```

Example 2

If you have inadvertently performed an action that causes inputs not to be shown on the screen, you can restore the screen display by entering the following command:

```
$ stty echo
```

The input of the above command will, however, not be displayed.
Example 3  Reset all terminal settings to reasonable values:

```
$ stty sane
```

Example 4  The terminal settings can also be reset with the aid of the `-g` option:

```
$ STATUS=`stty -g`
```

The above command defines a variable with the current settings of the terminal (which may differ from those of `stty sane`). The existing terminal settings can now be changed by resetting them with:

```
$ stty $STATUS
```

See also  `ioctl()` [4]
sum  print checksum and block count of a file

sum calculates and prints a checksum for the named file, and also prints the space used by
the file, in 512-byte blocks. The output is written to standard output. sum may be useful for checking that a file has arrived complete and unchanged after a file
transfer.

Syntax  sum[-r][...file]...

-r   Uses an alternate algorithm to compute the checksum and prints a different checksum.
file
Name of the file whose checksum is to be calculated. Both ordinary files and directories
may be specified. You may name more than one file.

file not specified: sum reads from standard input.

Caution! The algorithms used in computing the checksum may not be portable. Different
checksums may thus be produced for the same file when sum is run on different
systems.

Locale The following environment variables affect the execution of sum:
LANG    Provide a default value for the internationalization variables that are unset
or null. If LANG is unset of null, the corresponding value from the implemen-
tation-specific default locale will be used. If any of the internationalization
variables contains an invalid setting, the utility will behave as if none of the
variables had been defined.

LC_ALL  If set to a non-empty string value, override the values of all the other inter-
nationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data
as characters (for example, single- as opposed to multi-byte characters in
arguments).

LC_MESSAGES Determine the locale that should be used to affect the format and contents
of diagnostic messages written to standard error.

NLSPATH  Determine the location of message catalogs for the processing of
LC_MESSAGES.
Example 1  Checksum of the file *months* using the first algorithm:

```
$ sum months
6658 1 month
```

Example 2  Checksum of the file *months* using the second algorithm:

```
$ sum -r months
62412 1 month
```

See also  *cksum, wc*
**sync**

**flush system buffers**

`sync` causes all system buffers not yet written to hard disk or floppies to be flushed out to disk, thus assuring that all file updates up to that point are saved.

A system buffer is an internal buffer that serves to improve system performance. It is not accessible to users.

**Syntax**

```
sync
```

**Application usage**

`sync` will only write local buffers to local disks; you cannot use `sync` to force buffers to be written out to disk on remote machines.
tabs | set terminal tabs

This command is only available to users accessing the POSIX shell via `rlogin`.

tabs sets the tab stops on the user’s terminal, clearing any previous settings. For tabs to be effective, the terminal must have hardware tabs that can be set remotely using escape sequences. The lowest column number is 1. For tabs, column 1 refers to the leftmost column on the screen, even on terminals whose column markers begin at 0.

### Syntax

```
tabs[-T..type][+mn][..tabspec]
```

#### -T..type
Terminal type, which needs to be known for margin setting. `type` is a name listed in `/usr/share/lib/terminfo` (siehe term [12]).

- `T..type` not specified: `tabs` uses the value of the `TERM` environment variable. If `TERM` is not defined in the current environment (see `environ` [12]), `tabs` uses a type whose attributes are valid for many terminals.

#### +mn
All tabs are moved n columns to the right, with n+1 becoming the left margin. If n is omitted, a value of 10 is assumed. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. Normally, the leftmost margin is set with `+m0`. On most terminals, the margin is reset further to the right only when `+m` is specified explicitly.

#### tabspec
Four different types of tab specification are permitted:

- `-code` tabs in a predefined pattern (canned)
- `-n` tabs at regular intervals (repetitive)
- `--file` tabs in a pattern defined in file (file)
- `n1[,n2...]` tabs at freely selectable positions (arbitrary)

`-code` You determine the pattern of tabs on a line by specifying one of the following:

- `-a` Tab stops in columns 1,10,16,36,72. This corresponds to Assembler, IBM S/370, first format.
- `-a2` Tab stops in columns 1,10,16,40,72. This corresponds to Assembler, IBM S/370, second format.


tabs

- **c**  Tab stops in columns 1,8,12,16,20,55.  
This corresponds to COBOL normal format.

- **c2**  Tab stops in columns 1,6,10,14,49.  
This corresponds to COBOL compact format. Columns 1-6 are omitted. The first character typed corresponds to card column 7, one space takes you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see `fspec` [12]):

  ```
  <:t -c2 m6 s66 d:>
  ```

- **c3**  Tab stops in columns 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67.  
This corresponds to COBOL compact format with additional tabs. Columns 1-6 are omitted. The first character typed corresponds to card column 7, one space takes you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see `fspec` [12]):

  ```
  <:t -c3 m6 s66 d:>
  ```

  This is the recommended format for COBOL.

- **f**  Tab stops in columns 1,7,11,15,19,23.  
This corresponds to FORTRAN format.

- **p**  Tab stops in columns 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61.  
This corresponds to PL/I format.

- **s**  Tab stops in columns 1,10,55.  
This corresponds to SNOBOL format.

- **u**  Tab stops in columns 1,12,20,44.  
This corresponds to UNIVAC 1100 Assembler format.

- **n**  The number specified causes tab stops to be set in columns  
1+n,1+2n,1+3n. Specifying 8 eight gives you the UNIX system standard tab setting. Specifying 0 clears all tab stops.

- **file**  `tabs` reads the first line of the file, searching for a format specification (see `fspec` [12]). If it finds one, it sets the tab stops accordingly. Otherwise, the default value 8 is assumed. This type of format may be used to ensure that a tabbed file is printed with the correct tab settings, with `tabs` being used in conjunction with `pr` (see `pr`).

- **n1[,n2,...]**  You can freely select the columns for the tab stops by means of a list containing up to 64 numbers in ascending order. A number preceded by a plus sign is added to the value of the previous number. This does not apply to the first number. The format `1,10,20,30`, for example, can be specified as `1,10,+10,+10`. 
The numbers are separated by a comma, or the list is enclosed in quotes, whereby the numbers are separated by a comma and/or a blank.\[n1[,n2,...]\] must be specified as the last argument in the command line.

tabspec not specified:
tabs uses the default setting \(-8\), which corresponds to the UNIX system standard tab setting.

**Hint**
The mechanisms for clearing tab stops and setting the left margin are not the same on every terminal. tabs can set a maximum of 64 tab stops, but will only clear 20.

**Error**
illegal tabs
When setting arbitrary tab stops you failed to maintain ascending order or you specified a zero.

illegal increment
When setting arbitrary tab stops you specified a zero or omitted an increment.

unknown tab code
The code you specified as tabspec cannot be found.

can't open
The file you specified as tabspec cannot be opened.

file indirection
The format specification in the file specified as tabspec points to another file. Indirect references of this sort are not permitted.

tabs cannot be set for this terminal
You are running the command on an unknown terminal type, e.g. on a block-mode terminal.

**File**
\(/usr/share/lib/terminfo/*\)
Files used to name and define terminals

**Variable**
TERM
Terminal type

**Locale**
The following environment variables affect the execution of tabs:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.
tabs

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1**  Setting tab stops in COBOL normal format.

```$tabs -c```

**Example 2**  Set a tab in every sixth column, i.e. in columns 1,7,13,19,...

```$tabs -6```

**Example 3**  Set tabs in columns 1, 8 and 36.

```$tabs 1,8,36```

This has the same effect as

```$tabs 1,+7,+28```

**Example 4**  Setting tab stops in accordance with the first line (format specification) of your file `$HOME/tabspec.list/file1`.

```$tabs --$HOME/tabspec.list/file1```

**See also**  `pr`, `tput`, `environ`, `fspec`, `term`, `terminfo` [12]
**tail**

**deliver the last part of a file**

`tail` prints the contents of the specified file on standard output beginning at a designated place. If no file name is specified, `tail` outputs the contents of standard input.

If `tail` is used with a character-oriented special file a problem may occur.

### Syntax

**Format 1:**

```
[ -f][ -c place | -n place][ file]
```

**Format 2:**

```
+ [place][ -l | -b | -c][ file]
```

**Format 3:**

```
- [place][ -l | -b | -c][ file]
```

**Format 4:**

```
+ | - [place][ -l][ -rl][ file]
```

**-f** (follow) If the input file is not a pipe, the program will not terminate after the last line of the input file has been displayed, but will enter an endless loop, wherein it sleeps for at least a second and then attempts to read and copy further records from the input file. You could thus use this option to monitor the growth of a file that is being written by some other process.

- `f` is ignored if no file is specified and the standard input is a pipe.

**-c** place

(character) The output of the `tail` starts at the character defined by place.

You use `place` to specify the position in the input file at which you want output to start:

```
[[+|-]number]
```

- `number` can be specified as any decimal integer value. `+number` is calculated relative to the beginning of the file; `-number` is calculated relative to the end of the file.

⚠️ **Caution!** Tails relative to the end of the file are buffered by `tail` and are thus restricted to a maximum of 4 Kbytes.

- `+` not specified: `tail` calculates relative to the end of the file.

- `-` not specified: `tail` calculates relative to the end of the file.

- `number` not specified: The default value of 10 is assumed.

- `place` not specified: The default value for `place` is -10, i.e. 10 lines from the end of the file.
**tail**

- **-n** place
  (number) This option is equivalent to `-c place` except that the output is counted line by line not byte by byte.

  **file**
  Name of the input file to be displayed by **tail**.

  **file** not specified:
  **tail** reads from standard input.

**Format 2**

```
tail...+[place][...-b|-c|-l][...-f][...file]
```

**Format 3**

```
tail...-[place][...-b|-c|-l][...-f][...file]
```

Format 2 and format 3 are described together since they differ only in the use of +/- .

  **place**
  You use **place** to specify the position in the input file at which you want output to start. Enter a decimal integer value for **place**. If you enter `+[place]` counting starts at the beginning of the file. If you enter `-[place]` the command counts from the end of the file.

  **Caution!**
  **tail** buffers sections which are read from the end of the file. The size of this buffer is limited to 4 kbytes.

  **place** not specified:
  The default value is 10, i.e. 10 units from the start of the file (format 2) or from the end of the file (format 3).

- **-b** (block) Starts display at the block defined by **place**. A block is a 512-byte unit.

- **-c** (character) Starts display at the character defined by **place**.

- **-l** (line) Starts display at the line defined by **place**.

- **-f** (follow) If the input file is not a pipe, the program will not terminate after the last line of the input file has been displayed, but will enter an endless loop, wherein it sleeps for at least a second and then attempts to read and copy further records from the input file. You could thus use this option to monitor the growth of a file that is being written by some other process.

**file**

Name of the input file to be displayed by **tail**.

**file** not specified:
**tail** reads from standard input.
Format 4  tail[-l][-[place][-l][-r][-f][...file]]

-l  (line) Starts display at the line defined by place.

Only one of the options -b, -c, or -l may be used at a time. If you omit all three options, the -l option is set by default.

-f  (follow) If the input file is not a pipe, the program will not terminate after the last line of the input file has been displayed, but will enter an endless loop, wherein it sleeps for at least a second and then attempts to read and copy further records from the input file. You could thus use this option to monitor the growth of a file that is being written by some other process.

-r  (reverse) Lines are copied from the specified starting point in the file in reverse order. The default is to output the entire file in reverse order.

If the -r option is used together with a numeric value [number], the numeric value must be negative. Thus in this case [number] must not be combined with +.

You must not combine the -f and -r options.

file  Name of the input file to be displayed by tail.

file not specified:  tail reads from standard input.

Locale  The following environment variables affect the execution of tail:

LANG  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL  If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH  Determine the location of message catalogs for the processing of LC_MESSAGES.
Example 1  The file *months* contains the name of a month in each line. The following two calls to *tail*:

$ tail -n 5 months
$ tail -n +8 months

will result in the same output:

August
September
October
November
December

Example 2  To observe the effect of the *-f* option, create a shell script that writes data to a file in an endless loop and run this script in the background.

The shell script *neverend* with the following contents would serve the purpose:

```bash
while true
do {
    date >>anne
    sleep 5
}
done
```

You can now run this script in the background and then call *tail* with option *-f* and the file *anne* as its argument:

$ neverend&
$ tail -n -5 -f anne

These five lines were already in the file before any additions.
FRI JAN 23 11:40:35 MEZ 2009
FRI JAN 23 11:40:40 MEZ 2009
.
.
.

tail first displays the last five lines of the file *anne*, followed by the data written to this file by the script *neverend*. The process generated with *tail -f* can be terminated by pressing the [DEL] key (or the key combination @ @c) in the case of block terminals).

See also  *cat, head, more*
**talk**  
*talk* talk to another user

This command is only available to users accessing the POSIX shell via *rlogin*.

*talk* enables you to communicate with another user working at a terminal on the same or a different host. *talk* copies input lines from your terminal to the recipient’s terminal. *talk* is architecture dependent; it works only between machines that have the same architecture.

**Syntax**

```
[talk][loginname][ttyname]
```

**loginname**

Login name of the user you want to communicate with. If this user is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal.

If the user is working on another system, you have to enter:

```
loginname@systemname
```

**ttyname**

Name of the terminal on which the user with whom you want to talk is working. You need only enter *ttyname* if the recipient *loginname* is logged in more than once. The *who* command enables you to see which terminals *loginname* is logged in on.

**Functionality**

In a *talk* conversation, there is an originator and a recipient. The user who calls *talk* is the originator, while the user whose login name is specified is the recipient.

When the originator calls *talk*, the following appears on the recipient’s terminal:

```
Message from TalkDaemon@target_system at time
[talk]: connection requested by originator@source_system
[talk]: respond with: talk originator@source_system
```

and the following on the originator’s:

```
Waiting for your party to respond
```

This is on condition that the recipient’s login name is defined, the recipient is logged in and the connection could be set up successfully.

At this point, the recipient can accept the call by entering:

```
talk originator@source_system
```

The screens of the originator and the recipient then divide into two windows, the upper for writing messages, the lower for reading messages. Both parties can read and write messages simultaneously.
**talk**

To refuse a call, the recipient presses **[DEL]**, at which point the shell prompt appears and the user can continue working normally.

Once communication is established, during conversation

- printable characters are passed through to the other party
- the bell character is passed through to the other party
- you can redraw the screen with **[CTRL][L]**,
- you can terminate communication by pressing **[DEL]**. The message

  Connection closing. Exiting

  is output and the shell prompt appears at the bottom of the screen, followed by the cursor.

You can use the `mesg` command to grant (**mesg -y**) or deny (**mesg -n**) other users permission to set up a communication connection to your terminal with `talk`. The default setting for `mesg` is **y**. Certain commands, such as `pr`, automatically disallow messages in order to prevent messy output.

**Error**

The connection with the recipient could not be set up yet. It is advisable to wait a while, and then try again if the message *Waiting for your party to respond* does not appear on your screen after a few seconds.

**File**

This file is required to locate the recipient's terminal. All users who are logged in are recorded in this file.

**Locale**

The following environment variables affect the execution of `talk`:

- **LANG**
  - Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
**LC_ALL**  
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

**LC_MESSAGES**  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

**NLSPATH**  
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1**  
You want to communicate with the user *walter* who is working on the same system as you:

```
$ talk walter
...
```

**Example 2**  
You want to communicate with the user *lindsay* who is working at terminal tty003 on the system *munich*:

```
$ talk lindsay@munich tty003
...
```

**See also**  
*mailx, mesg, pr, who, write*
The `tar` command is used to process archive files. Archive files can only be placed on the POSIX file system, not on magnetic tapes and cartridges.

The actions to be performed are defined by the `mainkey` operand.

```
-i
```

The `tar` command has been withdrawn from the XPG4 standard and replaced by the command `pax`.

Consequently, only the `pax` command should be used in the future, since `tar` is now supported only for compatibility reasons.

**Syntax**

```
tar mainkey[modifier...][file]...
```

**Main keys**

`mainkey` consists of a “function character”, which may be specified with or without a hyphen (-).

It may be immediately followed by one or more additional options called modifiers (see page 725).

`mainkey` can be one of the following characters:

- `c` (create) A new archive is created. The specified file or files are written at the start of the archive.

- `r` (replace) The specified file or files are appended to the end of the archive.

- `u` (update) The specified file or files are added to the archive if not already present in it or if the last modification time of the file to be archived is later than that of the archived file.

- `t` (table) `tar` prints a table, showing the contents of the archive.

If one or more existing files from the archive files are specified, the names of the specified file or files are written to standard output.

If no files are specified, the names of all files located in the archive are written to standard output.
[-]x  (extract) The specified file or files are extracted from the archive.

Files that are stored in the archive with relative pathnames are extracted to the current directory.

Files that are stored in the archive with absolute pathnames are extracted to the corresponding directory (if such a directory exists).

If the specified file is a directory, all subdirectories contained in it are extracted recursively.

If the specified file does not exist in the directory in which the extracted file is to be placed, it is created as a new file with the following attributes:

– The user ID and group ID of the owner and the last modification time are taken from the archived file.
– Access permissions are set as defined with umask.

The s-bit is taken into account only if tar is called by a user with system administrator privileges.

If the file already exists, the access permissions are not changed.

If one or more files of the same name already exist in the archive, the last extracted file will overwrite all such files.

Modifiers

The characters listed below can be directly appended to the main key as a modifier, provided the following rules are observed:

– First enter all modifiers without intervening blanks.
– Then enter the individual arguments, which must be separated by blanks. The order of the arguments is determined by the order in which the associated modifiers were entered.

+ v  (verbose) The name of each processed file is written to standard error. Each name is preceded by one of the characters below, which indicates the type of processing:

<table>
<thead>
<tr>
<th>Function character</th>
<th>Output character</th>
</tr>
</thead>
<tbody>
<tr>
<td>c, r, u</td>
<td>a</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

The individual file names are then followed by the corresponding file size (in blocks).

If the modifier v is used with the mainkey [-]t, the access permissions and the owner and group of the file are also displayed.
For each file, tar waits for a confirmation before executing the corresponding function specified with mainkey. tar displays the following:

mainkey filename:

and expects you to enter your input after the colon. It is only after you enter your confirmation (with y or j, depending on the language) that the action will be executed.

If w is not specified, the requested actions are performed without confirmation.

f (file) The first specified file operand (or the second if b has already been specified) is interpreted as the name of the archive to be processed.

If the specified name is '-' (a hyphen), the archive will be written to standard output or read from standard input. This makes it possible to use tar at both ends of a command pipe:

- If f is specified with the main keys -c, -r or -u and the associated file operand is '-', the archive is written to standard output.
- If f is specified in connection with the mainkey -t or -x and the associated file operand is '-' (a hyphen), the archive is read from standard input.

If f is not specified, the default archive file defined by the shell variable TAPE is used.

h tar follows symbolic links.

If h is omitted, tar will process only the symbolic link, but not the associated file.

l (link) tar issues a message when a reference to another file cannot be resolved.

If l is omitted, no messages displayed.

m (modification date/time) On copying the specified file from the archive, tar sets the time of last modification for the copy to the current date and time.

If m is omitted, the original date and time that was recorded in the archive are retained.

o (owner) The file copied from the archive is assigned the same owner as the user who called tar. The file is also assigned the group of that user.

If o is omitted, the original owner and group that was recorded in the archive are retained.

file

Pathname of a file or directory that is to be written to the archive (mainkey -c, -r or -u), read from an archive (mainkey -x), or listed (mainkey -t).

If file is the name of a directory, the corresponding actions are applied (recursively) on all files and subdirectories of that directory.

If the modifier f is specified, the first file operand is interpreted as the archive name.
Variable

**TAPE**

Determines the name of the archive to be used if the modifier `f` is not specified.

Locale

The following environment variables affect the execution of *tar*:

- **LANG**
  
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**
  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

- **LC_COLLATE**
  
  Determine the collating sequence.

- **LC_MESSAGES**
  
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  
  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example 1

Create an archive file:

All files contained in the current directory (`.`) and its subdirectories are to be written to the archive file `$HOME/archive/example1.TAR`.

```bash
$ tar cvf $HOME/archive/example1.TAR .
a ./sh_history 8 Blocks
a ./profile 1 Blocks
a ./test.proc 1 Blocks
a ./pos_examp/file1 1 Blocks
a ./pos_examp/file2 1 Blocks
a ./pos_examp/file3 8 Blocks
tar: ./example1.TAR : same as archive file
```
Example 2  List the contents of an archive file on standard output:

```
$ tar tvf $HOME/archive/example1.TAR
USTAR format archive
rw------- 632/−993  3866 May 26 18:12  2008, ./.sh_history
rwxr-xr-x 632/−993     48 Mar 20 12:40  2008, ./.profile
rw-r--r-- 632/−993     29 Apr 23 13:31  2008, ./test.proc
rw-r--r-- 632/−993  52 May 26 18:07  2008, ./pos_examp/file1
rw------- 632/−993   3602 May 26 18:07  2008, ./pos_examp/file3
```

Example 3  Read files from an archive file of another user into the current directory:

The files stored in the archive file have a different user ID and group ID than that of the user who calls the `tar` command. On reading the files from the archive, this user also wants the file ownership to be changed to his/her user ID and group ID, respectively:

```
$ id
uid=3010(VSC0) gid=3030(VSCG0)
$ tar tvf /home/vsx/vsx1/km/example.TAR
USTAR format archive
rw-r--r-- 3001/3002  883 May 28 13:29  2008, ./pos_examp/file1
rw-r--r-- 3001/3002  1766 May 28 13:30  2008, ./pos_examp/file2
rw-r--r-- 3001/3002  2649 May 28 13:30  2008, ./pos_examp/file3
$ ls -lR
```

(1) The user VSC0 begins by displaying his/her own user ID and group ID.

(2) A table of contents of the archive file `example.TAR`, which is owned by some other user, is displayed. This indicates that the files and directories in the archive have a different user ID and group ID than that of the user VSC0.
(3) The user VSC0 sets the -o option in the `tar` command to have his/her user ID and group ID transferred on unpacking the archive file to the current directory.

(4) To verify that the new user ID and group ID have actually been entered, the user VSC0 issues the `ls` command.

**Hint**

**Exchanging tar archives between UNIX systems and BS2000**

In UNIX systems, `tar` archives are created in ASCII format. Since POSIX processes files in EBCDIC format by default and no normal ASCII-EBCDIC conversion of archive files is possible, the exchange of archive files can only occur via ASCII file systems. The following two variants briefly explain how this can be achieved:

**Variant 1 (with NFS)**

The `tar` archive is stored on the UNIX system and can be accessed in POSIX (mounted via NFS). If the shell variable `IO_CONVERSION` is set to `YES`, the `tar` archive can be unpacked with automatic conversion to the POSIX file system.

**Variant 2 (without NFS)**

An ASCII file system must be created in POSIX (see the description for `POSIX filesystem marker = n` under Install POSIX subsystem in the POSIX manual “Basics for Users and System Administrators” [1]. Setting the shell variable `IO_CONVERSION=YES` causes the files in such a file system to be treated as ASCII files.

The `tar` archive must now be copied from the UNIX system to the POSIX file system by means of a binary transfer. It can then be automatically converted to a POSIX EBCDIC file system when it is unpacked.

**See also** `ar`, `cpio`, `pax`
**tee**  

**join pipes and make copies of input**

`tee` transcribes data from standard input to standard output and simultaneously copies this data to the specified file.

**Syntax**

```
    tee[-ai][..file...]
```

- **-a**  (append) `tee` appends its output to the original contents of `file` if `file` already exists when `tee` is invoked.
  - `-a` not specified:  
    If `file` exists when `tee` is invoked, the original contents of `file` are overwritten.

- **-i**  (ignore) The signal SIGINT (see `kill`) is ignored.

- **file**  Name of the file to which `tee` is to write its output. If `file` does not exist when you call `tee`, a new file is created; if it does exist, `tee` either overwrites its contents or appends the output to it, depending on whether or not the `-a` option is used. If you specify more than one file, `tee` writes its entire output to each file.
  - `file` not specified:  
    `tee` writes its output to standard output only.

**Locale**

The following environment variables affect the execution of `tee`:

- **LANG**  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**  Determine the location of message catalogs for the processing of `LC_MESSAGES`. 

---

730
Example  Use of the `tee` command in a pipe:

```bash
$ (date; who) | tee security| wc -l
3
$ cat security
Mon Mar 9 16:19:41 MEZ 1995
QM212JNA  term/003   Mar 8 14:06:28
user2     pts/0       Mar 9 16:02:54
```

The `date` and `who` commands send the current date and information on all users currently logged in to standard output. The first pipe redirects this output to `tee`'s standard input. `tee` reads this input and copies it to the file `security` and to standard output. The second pipe sends the standard output of `tee` to the standard input of the `wc -l` command; `wc -l` writes the number of read lines to standard output: 3 (2 users logged in and 1 line for the date).
**test**  evaluate expression

The POSIX shell built-in `test` is used to check whether specific conditions are satisfied. Such conditions may be:

- file attributes,
- characteristics and comparisons of strings, and
- algebraic comparisons of integers.

Conditions can be negated and several conditions may be combined with one another.

Depending on the exit status, you can execute various commands, terminate loops, etc.

The POSIX shell built-in `test` has two forms (see syntax below). The effect is the same in both cases..

Syntax

```
[expression]
[[expression]]
```

The square brackets as well as the delimiting blanks before and after `expression` are mandatory. This is an alternative method of executing the POSIX shell built-in `test`.

`expression`

One or more primary expressions which may be combined (see the section “Combining conditions” on page 736).

`test` checks the following conditions:

**File attributes**

- `-r..file`
  (read) true if `file` exists and you have read permission.

- `-w..file`
  (write) true if `file` exists and you have write permission.

- `-x..file`
  (execute) true if `file` exists and you have execute permission.

- `-f..file`
  (file) true if `file` exists and is a regular file.

- `-d..file`
  (directory) true if `file` exists and is a directory.

- `-e..file`
  true if `file` exists.
-h..file
  true if file exists and is a symbolic link. Normally symbolic links are followed by all other conditions.

-c..file
  (character device) true if file exists and is a character special file.

-b..file
  (block device) true if file exists and is a block special file.

-p..file
  (pipe) true if file exists and is a named pipe (FIFO).

-u..file
  (set user ID) true if file exists and its set-user-id bit is set.

-g..file
  (set group ID) true if file exists and its set-group-id bit is set.

-k..file
  (sticky bit) true if file exists and its sticky bit is set.

-s..file
  (size) true if file exists and is not empty.

-t[..file]..filedescr
  (terminal) true if the specified file descriptor is open and assigned to a terminal.
  filedescr not specified:
  The file descriptor 1 is taken by default.

file
  Name of the file or directory whose attributes are to be tested. Relative or absolute path names may also be specified.
  If you use shell metacharacters in the file names, test only checks the first file that matches this name.
  If you specify the null string for file, i.e. a pair of single quotes "" or double quotes "", test interprets file as the name of your current directory.
  If you do not specify file, test issues an error message and terminates with exit status 1.
  A shell parameter may also be specified for file. This parameter should always be enclosed in double quotes "". If the corresponding shell variable has not been defined, the null string will be passed as an argument to test. The quotes thus guarantee that test is always supplied an argument during parameter substitution.
Characteristics and comparisons of strings

Any arbitrary sequence of characters may be specified as a string. Blanks and tabs included in the string must be escaped. If the string is to be protected from interpretation by the shell, the relevant metacharacters can be escaped by preceding them with a backslash \ or by enclosing the entire string within single or double quotes.

The null string can be specified by a pair of consecutive double quotes or single quotes. If you do not specify a string, `test` issues an error message and terminates with exit status 1.

A shell parameter may also be specified as a string. This parameter should always be enclosed in double quotes. If the corresponding shell variable has not been defined, the null string will be passed as an argument to `test`. The double quotes thus guarantee that `test` is always supplied an argument during parameter substitution.

`[-n..]string`
(non zero) true if the specified `string` is not the null string, i.e. has a non-zero length.
This option allows you to test whether there is a value assigned to a shell variable. The corresponding shell parameter must be enclosed within double quotes.

-`n` not specified:
The meaning is the same as above, but the condition is easier to read if `-n` is specified.

`[-z..]string`
(zero) true if the specified `string` is the null string, i.e. has a length of zero.
This enables you to test whether there is no value assigned to a shell variable. The corresponding shell parameter must be enclosed within double quotes.

`string1 ..!=.. string2`
True if the two strings are not identical. The blanks surrounding the `!=` (not equal to) sign are required, since `test` expects this character as an independent argument. If shell parameters are being compared, they must be quoted.

`string`
True if `string` is not the null string.
Algebraic comparison of integers

Integers can either be specified directly or as values of shell variables. There is no limit to the size of integer values you can specify, nor to the size of values you can define for a shell variable.

If you specify a shell parameter as a number, you should always enclose it within double quotes "..". This ensures that test receives the null string as an argument if the corresponding shell variable has not been defined. Quoting thus guarantees that an argument will be supplied to test when parameter substitution is performed.

```
int1..op..int2
```

The two integers int1 and int2 are algebraically compared by test on the basis of the operator specified in op. test expects operators as independent arguments; they must therefore be given between two blanks.

op can be any of the following:

```
-eq (equal) true if the two integers are algebraically equal.
-ne (not equal) true if the two integers are algebraically not equal.
-ge (greater than or equal) true if int1 is algebraically greater than or equal to int2.
-gt (greater than) true if int1 is algebraically greater than int2.
-le (less than or equal) true if int1 is algebraically less than or equal to int2.
-lt (less than) true if int1 is algebraically less than int2.
```

Negating conditions

```
!..condition
```

True if the specified condition is not satisfied. The exclamation mark must be followed by a blank.

Example

```
$ ! -r file
```

test returns an exit status of 0 (i.e. true) if you are not permitted to read the specified file.
Combining conditions

Primary conditions can be linked with one another to form a compound expression. The condition itself can also be negated. Since the corresponding relational operators are expected as independent arguments by test, they must be enclosed by one blank on either side.

The find command searches directories for files that satisfy given conditions. Conditions for test are combined in a similar way.

The following operators can be used to combine conditions:

condition.\(-a..condition\\)

(and) Logical AND operator, i.e true if all conditions combined in this way are satisfied. Each -a operator must be preceded and followed by a blank.

condition.\(-o..condition\\)

(or) Logical OR operator, i.e true if at least one of the conditions is satisfied. Each -o operator must be preceded and followed by a blank.

\( (..compexpr..) \)

compexpr represents a compound expression comprising two or more arbitrarily grouped conditions. The operator that precedes conditions grouped within parentheses applies to the whole compound expression (i.e. all conditions) and not just to the condition that immediately follows it. As the shell has its own interpretation of parentheses, they should be escaped with a backslash when used within test. Each operator and expression in the compound expression compexpr must be separated from one another by blanks.

Example

```
$ ! \( -r file -o -w file \) | OR
```

The expression within parentheses is true if you have read or write permission for the specified file. The negation operator (!) negates the whole expression. Thus, if you have neither read nor write permission for the specified file, test will return an exit status of zero (true).

Precedence of operators

The order of precedence for operators used with test is as follows:

Parentheses over negation over AND over OR.

The conditions within parentheses in the preceding example are thus evaluated first and then negated.
Exit status

0  If the specified expression is true
1  Although the specified expression is syntactically correct, its evaluation returns the value false. Alternatively, you have not entered an expression.
>1 Error (e.g. syntactically incorrect expression)

Error
test: argument expected
This error message is issued if you have failed to specify a condition fully, i.e. if a file, string, or number is missing in the specification.
Shell parameters must therefore always be enclosed within double quotes. Otherwise, an argument will be missing whenever the corresponding shell variable is undefined.

Locale

The following environment variables affect the execution of test:

LANG  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL  If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH  Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1

The following shell script tests whether the specified positional parameter is the name of a file or of a directory.

```bash
if test -f "$1" # or: if [ -f "$1" ]
then
echo $1 is a file
elif test -d "$1" # or: elif [ -d "$1" ]
then
echo $1 is a directory
fi
```

Since the $1 positional parameter has been quoted, test substitutes the current directory for it if you call the shell script without further arguments.
If the quotes were omitted in this case, test would terminate with an error message.
Example 2  The following script makes use of the operator \texttt{-gt} to test whether the first file specified in the command line, i.e. \texttt{$1$}, contains more lines than the one specified after it \texttt{($2$)}:

```bash
if \[ ` wc -l "$1$" -gt ` wc -l "$2$" \]
   then echo $1 contains more lines than $2
fi
```

\texttt{cat} passes on the contents of the file as input for the \texttt{wc -l} command via a pipe. The \texttt{wc -l} command counts and outputs the number of lines in each of the two files. The shell substitutes the specification in the backquotes for the appropriate number of lines.

Example 3  A test is to be performed to check whether a value has been assigned to the shell variable \texttt{TAPE}. This can be done in various ways:

\textit{Method A}

```bash
if \[ ! -z "$TAPE$" \]
   then echo a value is assigned to the TAPE variable
   else ....
fi
```

\textit{Method B}

```bash
if \[ -n "$TAPE$" \]
   then echo a value is assigned to the TAPE
   else ....
fi
```

The \texttt{-n} is optional and may be dropped.

The shell parameter \texttt{$TAPE$} is enclosed within double quotes to prevent \texttt{test} from issuing an error message in case no value has been assigned to the corresponding variable.

\textbf{See also}  \texttt{find}
time

time a simple command

time can be used to measure the execution time of a program or a shell script. After the program or shell script is executed, time writes the following times to standard error: real, user, sys.

- real is the elapsed time during the invoked process and its child processes, i.e. the time between program call and program termination.
- user is the time spent by the process or one of its child processes when executing user code. A process executes user code when it executes machine instructions from its own code segment.
- sys is the time spent by the process or one of its child processes when executing system code. A process executes system code when it executes machine instructions from system calls.

The output format for time is hh:mm:ss.tt, where hh stands for hours, mm for minutes, ss for seconds, and tt for tenths of a second.

Syntax

time[-p]prog[...]

-p Writes the results of the measurement to the standard error output.

prog Name of the program (or shell script) to be timed.

arg Optional arguments that may be passed to prog exactly as if prog were called without time.

If time is called on a multiprocessor system, the sum of user and system time may be greater than real time. A figure of more than 100% for the apparent CPU load is the result of child processes being split between a number of processors.

Exit status Corresponds to the exit status of prog.

1-125 Error in time.

126 prog cannot be executed.

127 prog was not found.

Variable

PATH

Determine the search path that will be used to locate the utility to be invoked.
Locale

The following environment variables affect the execution of `time`:

- **LANG**: Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**: If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**: Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example

Measure the execution time of the `ls` command. The standard output of `ls` is redirected to the file `list`.

```bash
$ time ls -l >list
```

```
real    0m0.04s
user    0m0.57s
sys     0m0.08s
```

See also

- `times`
- `time()`, `times()` [4]
Built-in sh command

**times**  
write process times

The built-in `times` command in the POSIX shell `sh` outputs the total time consumed by the processes which the shell has started so far. The time required for child processes is also output.

The time is subdivided into shell user time and system time (1st line) and child process user time and system time (2nd line). The time is specified in minutes (m) and seconds (s).

The user time is the time which has elapsed during the user phase of processes, while the system time is the time which has elapsed during the system phase.

Use `time` if you want to know the time consumed by a particular command.

**Syntax**

```plaintext
times
```

**Locale**

The following environment variables affect the execution of `times`:

- **LANG**
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example**

Finding out the accumulated user and system times of all processes run from the current shell:

```plaintext
$ times
0m8.68s 0m2.5s
0m22.74s 0m10.14s
```

In the shell user phase, the processes have taken 8.68 seconds and in the shell system phase 2.5 seconds. The child user phase has so far taken 22.74 seconds and the child system phase 10.14 seconds.

**See also**

`times`, `time()`, `times()` [4]
touch  change file access and modification times

touch is used to set the access and modification times of files to the current or specified date.

Syntax

Format 1: touch[-acm][-r..ref__file|.-t..time]..file.....
Format 2: touch[-acm][-MMDDhhmm[yy]]..file.....

Format 1  touch[-acm][-r..ref__file|.-t..time]..file.....

No option specified

touch sets the last modification and access times for the specified files to the specified or current date. If a file does not yet exist then touch creates it. touch without an option therefore has the same effect as touch -am.

option

-a  (access time) touch sets the the last access time for the specified files to the specified or current date.

If neither -a nor -m is specified:
touch sets the access and modification times of the named files to the specified or current date.

-c  touch does not create files which do not exist. In such a case, no message is output.

-m  ( modification time) touch sets the last modification time for the specified files to the specified or current date.

Neither -a nor -m specified:
touch sets the last access and modification time for the specified files to the specified or current date.

-r..ref_file

Instead of the current time, touch uses the corresponding date and time from ref_file.

-t..time

Instead of the current time, touch uses the time specified in time. time is output as a decimal number with the following format:

[[CC]YY]MMDDhhmm[.SS]

CC: The first two figures of the year specification (century). The century specification is optional. If you do not specify a century, the current century is used.
YY: The last two figures of the year specification (decade and year). The decade and year specification is optional. If no decade and year are specified, the current decade and year are used.

If the decade and year are specified but the century $CC$ is not, then $CC$ is calculated as follows:

for $69 \leq YY \leq 99$ $CC$ is 19

for $00 \leq YY \leq 38$ $CC$ is 20

MM: Month (01-12)

DD: Day (01-31)

hh: Hours (00-23)

mm: Minutes (00-59)

SS: Seconds (00-61). The seconds specification is optional. If you do not specify $SS$ then the value 00 is used.

If you wish to specify the seconds, the value of $SS$ must be in the range 00-61 (instead of the usual range 00-59). The values 60 and 61 should be considered as reserve seconds.

The highest date you can specify for time is:

20380119031407

CCYYMMDDhhmmss

file

Name of the input file. touch processes all types of files, including directories. Several file names may be specified in one call.

Completely numeric file names may cause problems, as touch may interpret them as date arguments.

Format 2 touch[[-acm][[-MMDDhhm[yy]]..file...]]

No option specified

touch sets the access and modification times of the named files to the specified or current date. If a file does not exist, it is created. Calling touch without options is thus equivalent to touch -am.

option

-a (access time) updates only the access time of the named files to the specified or current date.

If neither -a nor -m is specified:

touch sets the access and modification times of the named files to the specified or current date.
-c  Prevents `touch` from creating a file if the named file did not previously exist. No corresponding message is issued.

-m   (modification time)Updates only the modification time of the named files to the specified or current date.
    If neither -a nor -m is specified:
    `touch` sets both the access and modification times of the named files to the specified or current date.

MMDDhhmm[YY]
    Date and time to which the access and/or modification times of the specified files are to be set. The date/time argument comprises eight to ten digits with the following significance:
    month (MM) - day (DD) - hour (hh) - minute (mm) - year (YY)
    If you specify the year YY but not the millennium CC, CC is inferred as follows:
    with 69<YY<99, CC is 19
    with 00<YY<38, CC is 20
    The highest date you can specify for time is:
    0119031438
    MMDDhhmmYY
    YY not specified:
    `touch` assumes you mean the current year.
    MMDDhhmm[YY] not specified:
    The access and/or modification times of each named file are set to the current date and time by default.

file
    Name of the input file. `touch` processes all types of files, including directories. Several file names may be specified in one call.
    Completely numeric file names may cause problems, as `touch` may interpret them as date arguments.

Error
date: bad date conversion
You have specified an illegal date, e.g. 13010000.

Variable  TZ
    Determine the timezone to be used for interpreting the time option argument.
Locale

The following environment variables affect the execution of `touch`:

- **LANG**
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

Example 1

Set the access and modification times of `file` to the current date. No new file is to be created if this file does not exist.

```bash
$ touch -c file
```

The `ls -l` command can be used to display the time of the last modification; `ls -lu` outputs the access time.

Example 2

Set the access time of `file` to 9 o’clock on 8/26:

```bash
$ touch -a 08260900 file
$ ls -lu file
-rw-r--r-- 1 BERT  qm2 736 Aug 26 09:00 file
```

See also `date`, `ls`, `utime()` [4]
tput  

change terminal characteristics

This command is only helpful for users accessing the POSIX shell via `rlogin`.

You can use `tput` to

- indicate one capability of a terminal (Format 1)
- list a number of capabilities of a terminal (Format 2)
- initialize a terminal (Format 3)
- reset a terminal (Format 4)
- display the long name of a terminal (Format 5)
- clear the screen (Format 6).

Syntax

Format 1: `tput[-T..type]..capname[..parameter...]`

Format 2: `tput-S`

Format 3: `tput[-T..type]..init`

Format 4: `tput[-T..type]..reset`

Format 5: `tput[-T..type]..longname`

Format 6: `tput[-T..type]..clear`

Format 1  

Indicating one capability of a terminal

`tput[-T..type]..capname[..parameter...]`

`tput` indicates capabilities of a terminal as defined in the `terminfo` database (see “System administrator's reference manual” [12], `terminfo()`).

-T..type

`type` indicates the type of terminal whose capabilities are to be queried.

If the -T..type option is specified, the shell variables `LINES` and `COLUMNS` and the layer size will not be referenced.

-T..type not specified

`type` defaults to the value of the environment variable `TERM`.

capname

`capability name` `capname` is the alias (abbreviated form) of a terminal capability as defined in the `terminfo` source file. To find out whether your terminal has a particular capability (attribute), you specify the appropriate alias `capname` (see “System administrator's reference manual” [12], `terminfo()`).
Depending on the type of attribute, *tput* responds as follows:

- **Boolean type attribute:** *tput* simply sets an exit status (0 for *true* is the terminal has the attribute, 1 for *false* if it does not). In the POSIX shell you can inspect the exit status with `echo $?`.

- **string type attribute:** *tput* returns the appropriate string.

- **integer type attribute:** *tput* returns an integer.

If *capname* is an attribute for which no value is assigned to the specified terminal type in the *terminfo* database, *tput* returns -1.

**parameter**

If the *capname* attribute is a string that takes parameters, you can enter the parameters in *parameter*. *capname* and *parameter* are passed to *tput* as a compound string. An all numeric parameter is passed as a number.

### Format 2

**Listing a number of capabilities of a terminal**

*tput -S*

*tput* can be used to list multiple capabilities of the user’s terminal. You pass the capabilities from standard input rather than from the command line (see Example 4). Only one terminal capability is permitted per line. If the `-T` option is specified, the shell variables *LINES* and *COLUMNS* will not be referenced. The meanings of the exit statuses 0 and 1 change (see Error on page 750).

**capname**

*(capability name)* *capname* is the alias (abbreviated form) of a terminal capability as defined in the *terminfo* source file. To find out whether your terminal has a particular capability (attribute), you specify the appropriate alias *capname* (see “System administrator's reference manual” [12], *terminfo*). Depending on the type of attribute, *tput* responds as follows:

- **Boolean type attribute:** *tput* simply sets an exit status (0 for *true* is the terminal has the attribute, 1 for *false* if it does not). In the POSIX shell you can inspect the exit status with `echo $?`.

- **string type attribute:** *tput* returns the appropriate string.

- **integer type attribute:** *tput* returns an integer.

- **simple input redirection is possible:**
  
  *tput -S* < *filename*

  Each line in *filename* is assumed to contain an entry in the form *capname[.parameter]*.

  If only *tput -S* is specified, \[DEL\] or @@d must be used to indicate the end of input.

  If *capname* is an attribute for which no value is assigned to the specified terminal type in the *terminfo* database, *tput* returns -1.
parameter

If the capname attribute is a string that takes parameters, you can enter the parameters in parameter. capname and parameter are passed to `tput` as a compound string. An all numeric parameter is passed as a number.

Format 3  Initializing a terminal

`tput[-T..type].init`

If the terminfo database contains an entry for the user’s terminal, `tput` initializes the terminal in accordance with the terminal type specified in -T..type. `tput` performs the following individual activities:

- If present, the terminal’s initialization strings are output (capname is1, is2, is3, if, iprog)
- Any delays (e.g. newline) specified in the entry are set in the terminal driver.
- Tabs expansion is turned on or off according to the specification in the entry.
- If tabs are not expanded, the standard tabs are set (every 8 spaces).

If the terminfo entry does not contain the information needed for any of these activities, that activity is silently skipped.

- -T..type
  This indicates the type of terminal whose capabilities you want to use to initialize your terminal.

- -T..type not specified:
  type defaults to the value of the environment variable TERM.

Format 4  Resetting a terminal

`tput[-T..type].reset`

`tput` outputs the terminal’s reset strings (capname rs1, rs2, rs2, rf). If the reset strings are not present but initialization strings are, the initialization strings are output. Otherwise, `reset` behaves identically to `init`.

- -T..type
  This indicates the type of terminal whose capabilities you want to use to reset your terminal.

- -T..type not specified:
  type defaults to the value of environment variable TERM.
Built-in sh command

**Built-in sh command**

**tput**

Format 5  **Displaying the long name of a terminal**

```bash
tput[...-T...type]..longname
```

If the **terminfo** database is present and contains an entry for the user’s terminal (see Format 1, **-T..type** option), the long name of the terminal is output. This name is the last name in the first line of the description of the terminal in the **terminfo** database (see “System administrator’s reference manual” [12], terminfo()).

Format 6  **Clearing the screen**

```bash
tput[...-T...type]..clear
```

`tput` outputs the clear-screen sequence.

- **-T...type**
  - This indicates the type of terminal whose capabilities you want to use to reset your terminal.
  - **-T...type** not specified:
    - `type` defaults to the value of environment variable **TERM**.

**Exit status**

When **capname** is of type boolean and the **-S** option is not set:

- 0 if the specified terminal type has the capability
- 1 if the specified terminal type does not have the capability

When **capname** is of type string and the **-S** option is not set:

- 0 if the **capname** capability is defined for the terminal type
- 1 if the **capname** capability is not defined for the terminal type (nothing is written to standard output)

When **capname** is of type boolean or string and the **-S** option is set:

- 0 if all lines could be processed successfully
- 1 exit status 1 can never occur as no indication of which line failed can be given

When **capname** is of type integer:

- 0 The exit status is always 0. You can test the value of the standard output to determine whether **capname** is defined for the specified terminal type or not. -1 on standard output indicates that **capname** is not defined.

Error situations are indicated by an exit status of 2, 3 or 4.
Error

Depending on the exit status, `tput` issues one of the following error messages:

- `tput: unknown terminal type`
- `terminal type type unknown or no terminfo database present, exit status 3`
- `tput: unknown terminfo capability capname`

Self-explanatory, exit status 4

Variable

`TERM`
Default value used for the type of terminal when `-T..type` is not specified.

File

`/usr/share/lib/terminfo/*/`
Directory containing binary versions of the terminal type descriptions.

`/usr/include/curses.h`
curses() header file

`/usr/include/term.h`
terminfo() header file

`/usr/lib/tabset/*`
Information on tab handling (see "System administrator's reference manual" [12], terminfo(), Tabs and initialization)

Locale

The following environment variables affect the execution of `tput`:

`LANG`
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

`LC_ALL`
If set to a non-empty string value, override the values of all the other internationalization variables.

`LC_CTYPE`
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

`LC_MESSAGES`
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

`NLSPATH`
Determine the location of message catalogs for the processing of `LC_MESSAGES`.
Example 1  Initializing a terminal

```bash
$ tput init
```

A sequence of control characters that initializes the screen is passed to the user’s terminal. The sequence used is the one appropriate to the terminal type defined in the `TERM` environment variable.

Example 2  Displaying the number of columns for the current terminal:

```bash
$ tput cols
80
```

Example 3  Indicating whether the current terminal is a hardcopy terminal:

```bash
$ tput hc
$ echo $?
1
```

Since `hc` is a Boolean type value, `tput` simply returns an exit status, which you can inspect with `echo $?`. The returned value of 1 indicates that the terminal is not a hardcopy terminal.

Example 4  Displaying several capabilities in one `tput` invocation

```bash
$ tput -S <<here
> clear
> cup 10 10
> bold
> here
```

The screen is cleared, the cursor moved to position 10,10 and bold (extra bright) mode turned on. The list is terminated by `here` in the last line.

Example 5  The file `input` has the following contents:

```bash
cols
lines
```

Using this file as input to `tput` for terminals of types 97801 and 97808 would, for example, produce the following result:

```bash
$ tput -S < input
89
24
$
```

See also  `stty`, `tabs`
**tr** translate characters

`tr` reads an input text from standard input, replaces (Format 1 and 2) or deletes (Format 3 and 4) selected characters from it, and writes the result to standard output.

### Syntax

| Format 1: `tr[[-cs]]..string1..string2` |
| Format 2: `tr[-s[[-c]]..string1`       |
| Format 3: `tr[-d[[-c]]..string1`       |
| Format 4: `tr[-ds[[-c]]..string1..string2` |

If you specify more than one option in the command line for either of these formats, these options can be preceded by a single minus sign with no intervening blanks (e.g. `-cs` or `-dc`).

**Replace characters**

**Format 1**  
`tr[[-cs]]..string1..string2`

**Format 2**  
`tr[-s[[-c]]..string1`

**Format 3**  
`tr[-d[[-c]]..string1`

**Format 4**  
`tr[-ds[[-c]]..string1..string2`

- `tr` reads the input text, replacing characters that appear in `string1` with the corresponding characters in `string2`. In other words, the nth character in `string1` is replaced in the input text by the nth character in `string2`. If `string2` contains fewer characters than `string1`, those characters in `string1` which have no corresponding character in `string2` are not replaced (see Example 1 on page 756).

- `-c` Complements `string1` with respect to the currently applicable character set (octal values 001 through 377). The complemented `string1` then contains all characters of the currently applicable character set except for those specified in the original `string1`. `tr` then replaces the nth character of the input text in the complemented `string1` with the nth character in `string2`.

- `-s` (squeeze) After replacement, `tr` reduces all strings of repeated output characters in `string2` to a single character (see Example 2 on page 757).

string1..[string2]  
In `string1` you specify the characters to be replaced; `string2` provides the replacement string.

The characters in both strings must be specified without intervening blanks or other delimiters.

If a string contains metacharacters that have a special meaning for the shell, these metacharacters must be escaped by enclosing the entire string in single quotes `'..'` or by preceding each such character with a backslash `\`.
The strings can contain the following specifications:

**character**  any printable character.

\`{octal_number}`

whereby *octal_number* is a one, two, or three-digit octal number. The backslash must be escaped so that the digits are recognized as an octal number.

*tr* also processes the NUL character (000 in octal).

**Warning:** In previous versions, the NUL character was always deleted as an input character.

**metacharacters**

as escape sequences (same as for the *printf* command). You can enter the following escape sequences:

\`{\}\`   Backslash (for distinguishing octal numbers)
\`{a}`   Warning, alert \(^1\)
\`{b}`   Backspace \(^1\)
\`{f}`   Form Feed
\`{n}`   New Line
\`{r}`   Carriage Return
\`{t}`   Tab
\`{v}`   Vertical tab \(^1\)

\(^1\) These metacharacters are supported only on character-mode terminals (i.e. if you are accessing the POSIX shell via rlogin)

\`{a-z}` or \`{[a-z]}\`

Stands for the set of characters from *a* to *z* inclusive. Characters are sorted in the currently applicable collating sequence. Unlike in internationalized regular expressions, *a* and *z* must be ordinary characters, i.e. not equivalence class expressions \`{[=c=]}\` or collating symbols \`{[.cc.]}\`.

In the current collating sequence (see *LC_COLLATE*), the character used for *a* must precede the character used for *z*.

"b-a" is an illegal range and is rejected.

In the notation without brackets, "a-a" stands for "a" and so on, but "---" is interpreted as three ".-" characters. In the notation with brackets, \`{[a-a]}\` and \`{[---]}\` lead to undefined results.
Depending on the locale, "a-c" means "abc" or "aäbc", "n-p" means "nop" or "noöp" or "nöop", "t-v" means "tuv" or "tuüv" or "tüuv", and "r-t" can mean "rst", "rsßt" or "rßst".

**[:class:]**  
*class* specifies a character class, similar to internationalized regular expressions. The following values are possible for *class*:

- `alnum`: alphanumeric
- `alpha`: alphabetic
- `blank`: blank (white space)
- `cntrl`: control character
- `digit`: digit
- `graph`: graphic
- `lower`: lowercase
- `print`: print
- `punct`: punctuation
- `space`: space
- `upper`: uppercase
- `xdigit`: hexadecimal

Character classes must not be specified in a replacement string. Exception: The classes `lower` and `upper` are permitted if the corresponding character class is specified on the same side in `string1`.

**[=equivalence=]**  
*equivalence* specifies an equivalence class, similar to internationalized regular expressions.

Equivalence classes must not be specified in a replacement string.

**[a*n]**  
Stands for `n` repetitions of the `a`, e.g. `[a*3]` stands for `aaa`. Only useful in a replacement string.

- If the first digit of `n` is 0, `n` is considered octal; otherwise, it is taken to be decimal.
- If `n` is 0 or is omitted, it is taken to be "huge", meaning that the preceding character is to be repeated as often as required to pad `string2` to the length of `string1` (see Example 1).

`string2` not specified:
`string1` (possibly complemented, see `-c`) is used for `string2`.

`string1` and `string2` not specified:
`string1` is the null string. Either the null string (without option `-c`) or the entire current character set (with option `-c`) is taken for `string2`.

**Delete characters**

**Format 3**  
`tr -d[[-c]...] string1`

**Format 4**  
`tr -ds[[-c]...] string1...string2`

This format is only useful when `string1` is specified. If the `-s` option is not set, `string2` will be ignored.

- `-d` *(d - delete)* Deletes all input characters that appear in `string1`.
  - If the `-s` option is not specified, `string2` is ignored.
-c  Complements \texttt{string1} with respect to the current character set. The complemented \texttt{string1} then contains all characters of the current character set except for those specified in the original \texttt{string1}.

\texttt{tr} then deletes all input characters which occur in the complemented \texttt{string1}.

-\texttt{s} (squeeze) \texttt{tr} reduces all strings of repeated output characters in \texttt{string2} to a single character. The -\texttt{s} option is meaningless if \texttt{string2} is not specified.

\texttt{string1}..[\texttt{string2}]

In \texttt{string1} you specify the characters to be deleted; \texttt{string2} contains the characters that are to be reduced to a single character if they appear two or more times in succession in the output (see option -\texttt{s}).

Details with respect to how these strings are to be specified are given on page 752.

\texttt{string1} and \texttt{string2} not specified:

If the -\texttt{c} option is not specified, all input characters are copied unaltered to standard output. If option -\texttt{c} is specified, \texttt{tr} deletes all input characters, i.e. prints nothing on standard output.

\textbf{Locale}

The following environment variables affect the execution of \texttt{tr}:

\texttt{LANG}

Provide a default value for the internationalization variables that are unset or null. If \texttt{LANG} is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

\texttt{LC_ALL}

If set to a non-empty string value, override the values of all the other internationalization variables.

\texttt{LC_COLLATE}

Determine the locale for the behavior of range expressions and equivalence classes (eg. \{a-z\}).

\texttt{LC_CTYPE}

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments) and the behavior of character classes. \texttt{LC_CTYPE} also specifies which characters are included in the currently valid character set in conjunction with the -\texttt{c} option.

\texttt{LC_MESSAGES}

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

\texttt{NLSPATH}

Determine the location of message catalogs for the processing of \texttt{LC_MESSAGES}.
Ranges of uppercase and lowercase letters cannot always be unambiguously mapped to each other in locales which contain the 'ß' character (see above under "Replace characters"). The ranges “a-z” and “A-Z” are exceptions here and are treated separately as of this POSIX version.

Nevertheless, do not try to convert lowercase letters to uppercase using the command below. The result is not defined in a locale other than the POSIX or C locale and can vary in older POSIX versions or on other platforms:

```
$ tr 'a-z' 'A-Z' <file
```

Instead, use the following character classes to convert lowercase letters to uppercase (or vice versa):

```
$ tr '[:lower:]' '[:upper:]' <file
$ tr '[:upper:]' '[:lower:]' <file
```

However, `LC_CTYPE` and `LC_COLLATE` must refer to the same locale for conversion to work correctly. You can ensure that they do by assigning `LANG` or `LC_ALL`.

---

**Example 1**

`tr` without options: simple examples to demonstrate how `tr` works:

```
$ cat days
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
t
$ tr TS ts <days
Monday tuesday Wednesday thursday Friday saturday sunday

Here `tr` replaces all occurrences of `T` with `t` and of `S` with `s`.
```

```
$ tr TSF ts <days
Monday tuesday Wednesday thursday Friday saturday sunday
```

The second string has fewer characters than the first in this case. `tr` replaces `T` by `t` and `S` by `s`, but leaves `F` unaltered.
Now let us try replacing all lowercase letters with x. The following solution is not suitable:

```
$ tr '[-a-z]' x <days
Mondxy Tuesdxy Wednesdxy Thursdxy Sxturdxy Sundxy
```

`tr` has clearly only replaced occurrences of `a` with `x`. To replace all lowercase letters, we need to call `tr` as follows:

```
$ tr '[-a-z]' '[-x*]' <days
Mxxxxx Txxxxxx Wxxxxxxx Txxxxxx Fxxxxx Sxxxxxxxx Sxxxxx
```

Each character in string 1 now has a corresponding `x` in string 2, as the asterisk (*) causes string 2 to be padded with `x` as often as required. The single quotes are essential, since the strings include shell metacharacters.

**Example 2** We want to create a list of all the words that appear in `textfile`, one word to a line, a word being defined as any consecutive string consisting only of letters.

```
$ cat textfile
'When shall we three meet again?
In thunder, lightning, or in rain?'
$ tr -cs '[:A-Za-z:][[:alnum:]]' '[:\025*:]' <textfile
When
shall
we
three
meet
again
In
thunder
lightning
or
in
rain
```

**Delete characters (Format 2)**

**Example 3** Deleting a non-printing character from a file (`tr -d`)

```
$ tr -d '\016' <file
```

`tr` deletes from the file the character with the octal code 016 and displays the result on the standard output.

See also `ed`, `sh`, `sed`
trap  

**trap signals**

The POSIX shell built-in `trap` is used to define how the current shell is to react to subsequent signals received. You can thus use `trap` in shell scripts to have specific “clean-up” tasks performed before the script terminates, or to define positions at which no interrupts are to take place. `trap` has two functions:

- **`trap`** defines how the shell is to react to a signal:
  - The shell executes the commands specified in the `trap` command line.
    Any command possibly terminated earlier will not be resumed after the command list is executed. Shell scripts are resumed with the command that follows the one that was interrupted.
  - The shell ignores the specified signal.
  - The shell reverts to default actions for the specified signals. `trap` can be used to reset signal handling to the default behavior.

- **`trap`** displays all signals for which signal handling defaults were modified in the current shell.

**Syntax**

```
trap[...command_list...signal_number|...signal_name...]
```

- **`command_list`** determines how the shell is to react to the signals specified, i.e. whether it is to
  - execute commands if the signal specified is received
  - ignore the specified signal
  - revert to default actions for the signal specified.

**Executing commands**

You can specify one or more commands for `command_list`. These commands are to be executed if the signal specified is received. If you specify more than one command, you should separate these by semicolons. The semicolon must be quoted to prevent immediate interpretation by the shell.

The command list must be a single argument, so it must always be enclosed in single quotes ‘...’ or double quotes “...” if argument separators or semicolons appear in it.

If the given `command_list` is not a null string, the signal handling behavior it defines will only be valid in the current shell. `trap` must be explicitly called in each subshell; otherwise, default actions are automatically restored (see the section “Signal handling in the shell” on page 760).

It is relevant to note here that the specified commands are interpreted twice by the shell:
- once when the shell sets the `trap`, and
- once when the corresponding signal occurs, and the shell invokes `trap` to execute the defined commands.
It can thus make a difference whether single or double quotes are used:

'command_list'
Special characters are not interpreted until the shell actually invokes `trap` and executes the commands. Shell variables are thus evaluated only when the `trap` routines are executed.

"command_list"
The characters $, \, and `...` are interpreted by the shell when the `trap` is set. Shell variables are often still undefined at this stage, however.

If you want to prevent the shell script from being executed further when a signal has been received, specify `exit` as the last command in the command list.

Ignoring signals
If a null string ("" or ") is specified for `command_list` the specified signals are ignored.
The corresponding signals are also ignored in every subshell.

Resetting signal handling to default
If `command_list` is not specified the shell reverts to default actions for the signals specified (see the section “Signal handling in the shell” on page 760).

<table>
<thead>
<tr>
<th>signal_number</th>
<th>signal name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Terminate the shell (EOF)</td>
</tr>
<tr>
<td>1</td>
<td>SIGHUP</td>
</tr>
<tr>
<td>2</td>
<td>SIGINT</td>
</tr>
<tr>
<td>3</td>
<td>SIGQUIT</td>
</tr>
<tr>
<td>15</td>
<td>SIGTERM</td>
</tr>
</tbody>
</table>

The following signals are relevant for the shell:

If 0 is specified for `signal_number` this causes the specified `command_list` to be executed on exiting the current shell; 0 is not a signal. This means
– if you have called `trap` interactively, `command_list` will be executed as soon as you press the END key.
– if `trap` is specified in a shell script, `command_list` will be executed after this script.

Signal 9 (SIGKILL) always kills the process, so `trap "9` will not work.
No argument specified

The `trap` command with no arguments prints the list of commands associated with each signal number for which signal handling has been altered in the current shell. The list is written to standard output in the following format:

```
signal_number: command_list
```

Note, however, that only the signal numbers for which the default actions were modified earlier (with the command `trap`) are displayed (see the section “Signal handling in the shell” below).

### Signal handling in the shell

A process can receive a signal at any time during its execution. Such signals may either be generated by the process itself, by another process or by the user at the terminal (e.g. by pressing [DEL]. The shell then reacts to the signal in one of the following ways:

- It ignores the incoming signal.
- It terminates.
- It calls a function to deal with the signal.

The following signals are relevant to users who have used `rlogin` to access the POSIX shell:

<table>
<thead>
<tr>
<th>Signal number</th>
<th>Signal name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIGHUP</td>
<td>Disconnection of link to terminal</td>
</tr>
<tr>
<td>2</td>
<td>SIGINT</td>
<td>[DEL]</td>
</tr>
<tr>
<td>3</td>
<td>SIGQUIT</td>
<td>[CTRL]</td>
</tr>
<tr>
<td>9</td>
<td>SIGKILL</td>
<td>The command <code>kill -9 PID</code>, where PID is the process identification number of the corresponding shell</td>
</tr>
<tr>
<td>15</td>
<td>SIGTERM</td>
<td>The command <code>kill -15 PID</code>, where PID is the process identification number of the corresponding shell</td>
</tr>
</tbody>
</table>
Depending on whether or not you have defined signal handling with `trap`, the shell will react to these signals as follows:

<table>
<thead>
<tr>
<th>Signal number</th>
<th>Interactive shell</th>
<th>Shell script in the background</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ignore</td>
<td>abort</td>
</tr>
</tbody>
</table>
| 2             | take defined action after next command or after \( \text{PID} \)
otherwise: ignore | ignore |
| 3             | take defined action after next command or after \( \text{PID} \)
otherwise: ignore | ignore |
| 9             | abort             | abort                          |
| 15            | ignore            | ignore                          |

Depending on whether or not you have defined signal handling with `trap`, a shell script or the current foreground process in a shell script will react as follows (where \( \text{PID} \) is the process ID of the shell script):

Signal handling undefined for shell script and current foreground process in shell script:

<table>
<thead>
<tr>
<th>Signal number or key</th>
<th>Shell script</th>
<th>Foreground process in shell script</th>
</tr>
</thead>
<tbody>
<tr>
<td>kill -1 ( \text{PID} )</td>
<td>abort immediately</td>
<td>continue running</td>
</tr>
<tr>
<td>kill -2 ( \text{PID} )</td>
<td>abort after normal termination of foreground process</td>
<td>continue running</td>
</tr>
<tr>
<td>( \text{DEL} )</td>
<td>abort immediately</td>
<td>abort immediately</td>
</tr>
<tr>
<td>kill -3 ( \text{PID} )</td>
<td>abort after normal termination of foreground process</td>
<td>continue running</td>
</tr>
<tr>
<td>( \text{CTRL} )[\text{L}]</td>
<td>abort immediately</td>
<td>abort immediately, writing core dump to disk</td>
</tr>
<tr>
<td>kill -9 ( \text{PID} )</td>
<td>abort immediately</td>
<td>continue running</td>
</tr>
<tr>
<td>kill -15 ( \text{PID} )</td>
<td>abort after normal termination of foreground process</td>
<td>continue running</td>
</tr>
</tbody>
</table>
### Signal handling undefined for shell script, defined for current foreground process in shell script

<table>
<thead>
<tr>
<th>Signal number or key</th>
<th>Shell script</th>
<th>Foreground process in shell script</th>
</tr>
</thead>
<tbody>
<tr>
<td>kill -1 PID</td>
<td>abort immediately</td>
<td>continue running</td>
</tr>
<tr>
<td>kill -2 PID</td>
<td>abort after normal termination of foreground process</td>
<td>take defined action</td>
</tr>
<tr>
<td><strong>DEL</strong></td>
<td>abort after normal termination of foreground process</td>
<td>take defined action</td>
</tr>
<tr>
<td>kill -3 PID</td>
<td>abort after normal termination of foreground process</td>
<td>take defined action</td>
</tr>
<tr>
<td><strong>CTRL U</strong></td>
<td>abort after normal termination of foreground process</td>
<td>take defined action</td>
</tr>
<tr>
<td>kill -9 PID</td>
<td>abort immediately</td>
<td>continue running</td>
</tr>
<tr>
<td>kill -15 PID</td>
<td>abort after normal termination of foreground process</td>
<td>take defined action</td>
</tr>
</tbody>
</table>

### Signal handling defined for shell script, undefined for current foreground process in shell script

<table>
<thead>
<tr>
<th>Signal number or key</th>
<th>Shell script</th>
<th>Foreground process in shell script</th>
</tr>
</thead>
<tbody>
<tr>
<td>kill -1 PID</td>
<td>take defined action after normal termination of foreground process; then resume execution</td>
<td>continue running (signal not delivered)</td>
</tr>
<tr>
<td>kill -2 PID</td>
<td>take defined action after normal termination of foreground process; then resume execution</td>
<td>continue running (signal not delivered)</td>
</tr>
<tr>
<td><strong>DEL</strong></td>
<td>take defined action; then resume execution</td>
<td>abort immediately</td>
</tr>
<tr>
<td>kill -3 PID</td>
<td>take defined action after normal termination of foreground process; then resume execution</td>
<td>continue running (signal not delivered)</td>
</tr>
<tr>
<td><strong>CTRL U</strong></td>
<td>take defined action; then resume execution</td>
<td>abort immediately, writing core dump to disk</td>
</tr>
<tr>
<td>kill -9 PID</td>
<td>abort immediately</td>
<td>continue running</td>
</tr>
<tr>
<td>kill -15 PID</td>
<td>take defined action after normal termination of foreground process; then resume execution</td>
<td>continue running (signal not delivered)</td>
</tr>
</tbody>
</table>
Signal handling defined for shell script and current foreground process in shell script

<table>
<thead>
<tr>
<th>Signal number or key</th>
<th>Shell script</th>
<th>Foreground process in shell script</th>
</tr>
</thead>
<tbody>
<tr>
<td>kill -1 PID</td>
<td>take defined action after normal termination of foreground process; then resume execution</td>
<td>continue running (signal not delivered)</td>
</tr>
<tr>
<td>kill -2 PID</td>
<td>take defined action after normal termination of foreground process; then resume execution</td>
<td>continue running (signal not delivered)</td>
</tr>
<tr>
<td>DEL</td>
<td>take defined action; then resume execution</td>
<td>take defined action; then resume execution</td>
</tr>
<tr>
<td>kill -3 PID</td>
<td>take defined action after normal termination of foreground process; then resume execution</td>
<td>continue running (signal not delivered)</td>
</tr>
<tr>
<td>CTRL + 1</td>
<td>take defined action; then resume execution</td>
<td>take defined action; then resume execution</td>
</tr>
<tr>
<td>kill -9 PID</td>
<td>abort immediately</td>
<td>continue running</td>
</tr>
<tr>
<td>kill -15 PID</td>
<td>take defined action after normal termination of foreground process; then resume execution</td>
<td>continue running (signal not delivered)</td>
</tr>
</tbody>
</table>

The `signal()` function can be used in C programs to define how signal handling is to be implemented (see `signal()` [4]).

**Locale**

The following environment variables affect the execution of `trap`:

- **LANG**: Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**: If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**: Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
**trap**  

**Built-in sh command**

**NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1** To ensure that signal 2 is ignored in a shell script, the following line is included in it:

```
trap '' 2
```

The two single quotes, i.e. the null string, cause signal 2 to be ignored. This means that the shell script cannot be terminated externally by the \[DEL\] key or by the command `kill -2 process_id`.

**Example 2** Use of the command *trap* in an interactive shell:

```
$ trap 'echo Last logged out: `date` >>$HOME/logfile' 0
$ trap 0, echo Last logged out: `date` >>$HOME/logfile
$ [END]
```

```
login: rose
Password:
```

```
$ cat logfile
Last logged out: Mon Mar 9 18:17:23 MEZ 2009
```

The *trap* in the above example defines that a message is to be written to the file $HOME/logfile on exiting the current shell. The command list must be enclosed in single quotes here, since *date* is only to be executed when the shell terminates.

**Example 3** The shell script *traptest* illustrates how temporary files can be deleted if signals are received during execution. The script contains the following lines (without the line numbers):

1. `TMP=/usr/rtmp/$$
2. `/usr/rtmp/$$`
3. `trap "rm -f $TMP; trap 0; exit 1" 1 2 3 15`
4. `trap "rm -f $TMP; exit 0" 0`
5. `ls > $TMP`

**Line 1:**

The file name /usr/rtmp/$$ is assigned to the variable TMP. The shell substitutes the process ID of the current shell for $$, thus creating a unique file name.

**Line 2:**

The command list is enclosed in double quotes, since the TMP variable has already been assigned a value. The following actions are defined for signals 1, 2, 3, and 15: delete the file /usr/rtmp/$$, reset the end of script definition (0) (see line 3), and terminate the script with exit status 1.
Built-in sh command

**trap**

Line 3:

0 is the only signal-number specified, i.e. the definition for the end of the script is: delete the file /usr/rtmp/$$, and terminate the script with exit status 0. This definition must be reset in line 2 with `trap 0`, since the command *exit* terminates the shell script. Otherwise, line 3 would cause the script to terminate with exit status 0.

Line 4:

The file /usr/rtmp/$$ is created. This line must not precede the *trap* command; otherwise, if the script terminates before the shell executes the first *trap* command, the file will not be deleted.

In this case the command *exit* should be explicitly invoked at the end of the command list, otherwise the commands that follow *trap* in the script could execute in an undefined state.

See also  
*exit*, *kill*,  
signal() [4]
true  return true value

The `true` command simply returns a zero exit status and does nothing else.

You can use `true` in shell scripts to generate the condition `true`. You can use the companion command `false` to generate the condition `false` (non-zero exit status).

Syntax  

```
true
```

Exit status  0

Example  The following shell script initiates an endless loop which you can terminate by pressing [DEL]:

```
while true
do
  :
  :
  done
```

See also  `false`, `:` (colon)
**tsort**

**topological sort**

The *tsort* utility writes to standard output a totally ordered list of items consistent with a partial ordering of items containing in the input.

The input consists of pairs of items (non-empty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

The actions of *tsort* are not affected by the locale *LC_COLLATE*.

**Syntax**

```
   tsort [file]
```

*file*  
Name of the file to order.

*file* not specified:  
The standard input is used.

**Error**  
Odd data: There is an uneven number of fields in the input file.

**Locale**

The following environment variables affect the execution of *tsort*:

- **LANG**  
  Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

- **LC_MESSAGES**  
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**  
  Determine the location of message catalogs for the processing of *LC_MESSAGES*.
Example

The command

```
$ tsort <<@d
  > a b c c d e
  > g g
  > f g e f
  > g g
  > h h
@d
```

produces the output:

```
a
b
c
d
e
f
g
h
```
tty  

output path name of current terminal

tty outputs the path name of the terminal with which the process is linked. The exit status indicates whether or not standard input is a terminal.

If the process is linked with a virtual terminal, tty returns the name of the virtual terminal, not the real terminal.

Syntax

```
tty[[-s]]
```

- **-s**  Inhibits output of the terminal path name, returning the exit status only.

  **-s** not specified:

  *tty* reports an error if standard input is not a terminal.

Exit status

- **0**: Standard input is a terminal.
- **1**: Standard input is not a terminal.
- >1: An invalid option was specified.

Error

not a tty

The standard input is not a terminal and the **-s** option was not specified.

Locale

The following environment variables affect the execution of *tty*:

- **LANG**  Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the default locale will be used. If any of the variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- **LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

- **NLSPATH**  Determine the location of message catalogs for the processing of **LC_MESSAGES**.
**tty**

Example 1  Output the name of the current terminal:

```bash
$ tty
/dev/term/003
```

Example 2  The following script is to send output to the screen even if standard output is redirected to a file:

```bash
if tty -s
    read input
else
    echo 'Output to the terminal' > `tty`
fi
```

Example 3  The following script is to generate an error message if standard input is not the terminal:

```bash
if tty -s
    read input
else
    echo 'Standard input is not a terminal' >&2
fi
```
Built-in sh command

**type**

**write a description of command type**

The POSIX shell built-in *type* is used to indicate how the shell would interpret a given name if it were entered as a command. The command type (and some other information) is written to standard output.

In the POSIX shell *sh*, *type* is an exported alias for *whence -v*; *whence* is a POSIX shell built-in command.

**Syntax**

```
    type name...
```

*name*

Name of the command whose type is to be queried. Several blank-separated names may be specified at a time.

If *name* is neither a shell function nor a POSIX shell built-in, the shell searches for an executable file with this name in the directories whose path names are defined in the *PATH* variable.

The *type* command indicates whether *name* is a shell function or a POSIX shell built-in. If *name* designates an executable file, the absolute path name or a path name in the form `.name` is displayed.

If *name* is a shell function, *type* will additionally print the function definition. In the POSIX shell, the definition is not shown.

If *name* is not an executable file, *type* issues an error message. The same error message is also output if *name* does not exist in the directories whose path names are assigned to the *PATH* variable.

**Error**

*name not found*

This error message means:

- either there is no file with this name in the directories that can be accessed via the *PATH* variable
- or the file with this name is not executable.

**Variable**

*PATH*

Determine the location of *name*.

**Locale**

The following environment variables affect the execution of *type*:

*LANG*

Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
**type**  
Built-in sh command

---

**LC_ALL**  
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

---

**Example**  
The following series of commands illustrates the kind of information supplied by the shell built-in `type`:

```
$ type mount
mount is /etc/mount
$ type echo
echo is a shell builtin
$ type ls
ls is a tracked alias for /usr/bin/ls
$ type typetest
typetest is ./typetest
```

---

**See also**  
`whence`, `file`
The `typeset` command can be used for two purposes:
- to list variable names and other optional information on standard output
- to set attributes and values for shell variables.

**Syntax**

```
typeset[..option][..name[=value]]...
```

If neither `option` nor `name[=value]` is given, the names and attributes of all variables are listed on standard output.

If there are options but no `name` arguments, all variables which have these options set are listed along with their attributes.

Using a plus sign instead of a minus sign stops the values from being printed.

`option` can be used to assign one or more attributes to the variable `name[=value]`. When `typeset` is invoked inside a function, a new instance of the variable `name` is created. The value and its attribute are restored when the function completes.

Using a minus sign before the option letters turns on the corresponding attributes; a plus sign turns them off.

You may enter the following list of attributes as options:

- `-L[num]`
  - Left-justifies `value` and removes leading blanks.
  - If `num` is non-zero, it defines the length of the field; otherwise, the field length is determined by the length of the first value assignment.
  - This length is used for subsequent assignments. When the variable is reassigned, it is padded to the right with blanks or truncated as necessary.
  - Leading zeros are removed if the `-Z` option is also set. The `-L` option turns off the `-R` option.

- `-R[num]`
  - Right-justifies `value` and pads it with leading blanks.
  - If `num` is non-zero, it defines the length of the field; otherwise, the field length is determined by the length of the first value assignment. This length is used for subsequent assignments.
  - When the variable is reassigned, the field is either padded to the left with blanks or truncated from the end.
  - The `-R` option turn off the `-L` option.

- `-Z[num]`
  - Right-justifies the `value` and pads it with leading zeros if the first non-blank character is a digit and the `-L` option has not been set. If `num` is non-zero, it defines the length of the field; otherwise, the field length is determined by the length of the first value assignment.
The names refer to function names rather than variable names. The POSIX shell stores the functions in the \texttt{.sh\_history} file. Consequently, you cannot display a function definition on the screen if \texttt{.sh\_history} does not exist or if the \texttt{nolog} option was not set when the function was read. No assignments can be made, and the only other valid options in combination with \texttt{-f} are:

\texttt{-t} Turns on execution tracing for the function.
\texttt{-u} Causes the function to be marked undefined. The \texttt{FPATH} variable is searched to find the function definition the next time the function is referenced.
\texttt{-x} Allows the function definition to remain in effect across POSIX shell scripts invoked by name. In other words, the function is exported.

\texttt{-l[num]}
The variable is an integer. This option makes arithmetic faster. If \texttt{num} is non-zero, it defines the output arithmetic base; otherwise, the first assignment determines the output base.

\texttt{-l} All uppercase characters are converted to lowercase. The \texttt{-u} option is turned off.
\texttt{-r} The given variables are marked readonly, which means that their values cannot be changed by subsequent assignment.
\texttt{-t} Tags the variables. Tags are user-definable and have no special meaning to the POSIX shell.
\texttt{-u} All lowercase characters are converted to uppercase characters. The \texttt{-l} option is turned off.
\texttt{-x} The given names are marked for automatic export to any new environment.

**Error**

\texttt{sh: variable: bad number}

if the number assigned to the variable in the \texttt{-i} option was not an integer.

**Example**

```
$ typeset -L var1="...31"
$ typeset -L
  var1='31...'
  ...
$ typeset -l var2=VALUE2
$ typeset -l
  var2=value2
```

**See also**  \texttt{readonly, export, set, env}
Built-in sh command

ulimit  set or report file size limit

The POSIX shell built-in `ulimit` enables you to

- check the file size limits imposed for the current shell and its child processes
- set or change the individual file size limits for the current shell and all its child processes.

Normal users without POSIX administrator privileges may decrease this limit, but not increase it. The new limits apply to files written by the current shell and all its child processes.

You cannot increase a limit that has been decreased until you have terminated the shell in which you have decreased the limit.

Syntax

Format 1

```
ulimit[,...H][,...S][,...option]...
```

`ulimit` writes the limits checked by `option` to standard output.

- **H** Checks a **hard** limit.
- **S** Checks a **soft** limit.

Neither **-H** nor **-S** specified: `ulimit` writes the **soft** limits to standard output.

**option**

You can use options to specify the limits to be checked. You can combine the options however you want.

No option specified: `ulimit` uses the `-f` option (see Format 2 below).

- **-a** Checks all limits.

The other options are described under Format 2 below.

Format 2

Set limits

```
ulimit[,...H][,...S][,...option]...limit
```

`ulimit` sets the limit defined by `option` to `limit`. You can only set one new limit per command call.

- **-H** Sets a **hard** limit. Normal users without POSIX administrator privileges may reduce any **hard** limit. However, only the POSIX administrator may increase a **hard** limit.

Format 1: ulimit[,...H][,...S][,...option]...
Format 2: ulimit[,...H][,...S][,...option]...limit
-S  Sets a soft limit. Any user can set a soft limit to a value less than the hard limit.

Neither -H nor -S specified:
ulimit sets hard and soft limits to the specified value.

Option
You can use options to specify the limits to be set. The following limits, described in more detail under getrlimit() [4], are available for your current shell and all its child processes:

-a  (all) Query all limit values.

-f  (file size) Maximum file size (in 512-byte blocks) that you may create (write); there is no limit on reading. If file size is 0, no files can be created. If you exceed the default value, you either receive an error message from the appropriate command (depending on the command you have used to create the file) or the new file only contains as much data as the imposed limit can accommodate.

Example for file size
After the command ls -lR >file, file will contain as many bytes as are permitted by the current size limit.
The command cp, on the other hand, will issue an error message if the file to be copied exceeds the currently set file size limit.

-m  (memory) Maximum size of the data segment or heap (in Kbytes) in a process.

-n  (number of filedescriptors) Maximum number of (open) file descriptors permitted in a process plus 1.

-t  (time) Maximum CPU time (in seconds) permitted for a process.

No option specified:
ulimit uses the -f option.

Limit
Sets the size limit for the current shell and all its child processes in accordance with the specified option. Normal users without POSIX administrator privileges may only specify values less than the current size limit defined for limit. As a POSIX administrator, however, you may also increase this limit with limit. If you specify the string unlimited for limit, the limit is set to the maximum possible value.

Error
sh: ulimit: exceeds allowable limit
You have tried to increase the currently set file size limit. This privilege is reserved for POSIX administrators only.

Exit status
0 If the command is executed successfully
>0 Rejection of higher limit or error
Built-in sh command

The following environment variables affect the execution of `ulimit`:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example 1**
The current soft limit value for the maximum number of file descriptors is increased. All users may do this provided that the new limit value remains below the hard limit value. The new limit value is also valid in a subshell.

```bash
$ ulimit -Sn 80
$ ulimit -Sn
80
```

**Example 2**
You want to check and then reduce the maximum file size. This limit can only be increased again by the POSIX administrator.

```bash
$ ulimit
unlimited
$ ulimit 20000
$ sh
$ ulimit
20000
```

The new limit is also valid in subshells. From now on, only files smaller than 20,000 * 512 bytes can be created.

See also `getrlimit()`, `signal()`, `ulimit()` [4]
umask  get or set the file mode creation mask

The POSIX shell built-in _umask_ sets or displays the current user file-creation mode mask. This mask defines the access permissions to be assigned to all new files and directories which you subsequently create in the current shell or in one of its subshells.

Changes made in the file-creation mode mask remain in effect until a new value is set with _umask_ or the shell in which _umask_ was originally called is terminated.

The POSIX administrator can use _umask_ to define the value of the file-creation mode mask in the _/etc/profile_ file. Since _/etc/profile_ is executed by every login shell, the access permissions set in this way are valid for every used logged in on the system (see Example 1 on page 780).

**Syntax**

\[
\text{umask}\{-S\}[\ldots\text{mask}]\\n\]

- `S` The mask is symbolically output in the following form:

  \[u=\text{access permissions},g=\text{access permissions},o=\text{access permissions}\]

  where \(u\) = user, \(g\) = group, \(o\) = others and access permissions = r, w, x

  mask

  Three octal digits comprising the file-creation mode mask. This mask defines the permissions to be assigned to all new files or directories that the user subsequently creates in the current shell or in any of its subshells (see „Setting permissions with the file-creation mode mask“ below).

  Since _umask_ can only withhold existing permissions from the base settings, you cannot use it to have execute permissions directly assigned to files, regardless of what you specify for _mask_. Use the command _chmod_ instead.

  _mask_ not specified:

  _umask_ displays the current user file-creation mode mask in the following output format

  \[0\text{nnn}\] (\(0\) indicates octal notation and \(\text{nnn}\) indicates the current file-creation mode mask in octal).

**Setting permissions with the file-creation mode mask**

When creating files and directories the following permissions are generally assigned as base settings (see _open()_ [4]):

- `rw-rw-rw-` to files, i.e. 110110110 in binary notation
- `rwxrwxrwx` to directories, i.e. 111111111 in binary notation

_umask_ is only capable of withholding existing permissions from these base settings. This means that you cannot use _umask_ to have execute permission automatically assigned to new files. You may, however, set the appropriate x-bit with the _chmod_ command.
The file-creation mode mask comprises three octal digits, which are specified when calling `umask`. The permissions they refer to are derived as follows:

1. Convert the digits in the mask to their binary equivalents.
2. Create the complement to each of these binary numbers, i.e., replace the zeros by ones, and ones by zeros.
3. Add this binary complement to the binary value of the base mode setting by logically ANDing the two; the result thus only contains ones in positions where both pairs have ones; in all other positions it contains zeros.

**Example**

The file-creation mode mask 022 changes permissions as follows:

- **Files:**
  The base mode setting for files is 110110110.
  
  1. The binary equivalent of octal 022 is 000010010
  2. The complement of this value is 111101101
  3. AND operation: 110110110
      ------------
      110100100

  The permissions for all newly created files will thus be `rw-r--r--`

- **Directories:**
  The base mode setting for directories is 111111111.

  AND operation: 111101101
                  111111111
                  111101101

  All new directories created will thus have the permissions `rwxr-xr-x`

**File**

`/etc/profile`

File executed by each login shell; used to set a shell environment.

The POSIX administrator normally defines one mask value for users without special privileges and one for `root` in this file.

**Locale**

The following environment variables affect the execution of `umask`:

- `LANG`
  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
umask  

**LC_ALL**  
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  
Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1** The following lines are often included in the `/etc/profile` file:

```bash
if [ "$USER" != "admin" -a "$USER" != "root" ]
then
  umask 066
fi
```

Since every login shell executes the `/etc/profile` file, the file-creation mode mask 066 applies to all users except *root* and *admin*.

The following permissions are thus assigned by default:
- for new files created: `rw-------`
- for new directories created: `rwx--x--x`

**Example 2** Change the file-creation mode mask and display it:

```bash
$ umask 033
$ > new
$ mkdir newdir
$ ls -ld new newdir
-rw-r--r--  1 ANNE     other          0   Mar 22 09:49 new
drwxr--r--  2 ANNE     other        520   Mar 22 16:40 newdir
$ umask
0033
```

The output of the `ls -ld ...` command shows which access permissions are assigned to new files and directories when the file-creation mode mask has been set to the value 033.

**See also**  
`chmod`, `chmod()`, `creat()`, `open()`, `umask()` [4]
umount unmount a file system

The command `umount` unmounts a file system which was mounted using `mount`. The file system entry is deleted from the table `/etc/mnttab`.

Syntax

```
umount[.-V..{(resource | mountpoint)}]
```

**option**

- `-V` Outputs the entire command line on screen but does not execute the command. The command line is generated with all the options and arguments specified by the user as well as the values taken from `/etc/vfstab`. You should select this option in order to subject a command line to a general check and validity check.

**resource**

Specifies the resource which is to be unmounted. Format as for `mount`.

For bs2fs file systems the option must be specified in UPPERCASE in accordance with the entry in the internal tables. Special characters of the POSIX shell such as `$` or `*` must be escaped explicitly.

In the case of an NFS resource this is replaced by the name of the source server. This must be followed by a colon and the path name of the resource.

**mountpoint**

Specifies the local position at which `resource` must be unmounted. You must specify an absolute path name.

**Hint**

Specification of the `mountpoint` option is always recommended for unmounting bs2fs file systems. If a bs2fs file system is mounted in multiple positions in the POSIX file system (i.e. identical `resource` specification in `mount`) and only `resource` is specified for `umount`, only the file system mounted last is unmounted. However, if `mountpoint` is specified, the corresponding file system is always unmounted.

The `umount` command is rejected if it refers to a file system mounted using the `bs2fscontainer` option and at least one bs2fs file system is still mounted. In the case of a successful `umount` for the bs2fs container, the bs2fsd copy daemons are terminated automatically.

**File**

- `/etc/mnttab`
  Table of mounted file systems.

- `/etc/vfstab`
  Table of automatically mounted file systems.

See also

`mount`, `umountall`, `mount`, `umount` [4]
umountall  

**umountall** unmount file systems

`umountall` unmounts all mounted file systems with the exception of `/root`, `/proc`, `/var` and `/usr`. If only `file_system_type` is specified, `umountall` relates only to file systems of the specified type. The file systems are unmounted in the order `nfs - bs2fs - ufs`. This ensures that the bs2fscontainer file system required for bs2fs file systems is only unmounted in the ufs when it is no longer needed, i.e. when no further bs2fs file systems are mounted.

**Syntax**

```
umountall [-F file_system_type] [-k] [-l] [-r]
```

- `-F` Specifies the file system type to be unmounted.
- `-k` Sends the SIGKILL signal to processes which have opened files in the file system.
- `-l` Limits the operation to local file systems (`ufs` and `bs2fs`).
- `-r` Limits the operation to remote file systems (`nfs`).
- `-b` Limits the operation to bs2fs file systems.

**Error**  
If the file systems can be unmounted no message is output. Error and warning messages are issued by `fsck` and `mount`.

**Hint**  
If the `-F` option is specified together with one or more of the options `-l`, `-r` and `-b` and the options are mutually compatible, the `-l`, `-r` and `-b` options have priority. For example, `mountall -F bs2fs -l` and `mountall -F ufs -l` have the same effect as `mountall -l`: all local file systems (i.e. all ufs and bs2fs file systems) are unmounted. The entries `mountall -F bs2fs` and `mountall -b` also lead to the same result: all bs2fs file systems are unmounted.

**File**

`/etc/mnttab`  
Table of mounted file systems.

`/etc/vfstab`  
Table of automatically mounted file systems.

**See also**

`fsck`, `mount`, `mountall`, `umount`
Built-in sh command

unalias  remove alias definitions

The built-in unalias command in the POSIX shell sh deletes the alias variables entered using alias.

Syntax

| Format 1: unalias name..... |
| Format 2: unalias-a |

Format 1  unalias name.....

name       The alias definition specified in name is deleted.

Format 2  unalias-a

-a   All alias definitions are deleted.

Exit status

0   The command was executed successfully.

>0 Error or one of the name operands is an invalid alias name.

Locale

The following environment variables affect the execution of unalias:

LANG   Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL   If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES   Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH   Determine the location of message catalogs for the processing of LC_MESSAGES.
Example

The existing alias definitions are output and the alias variable defined with the name lx is deleted.

$ alias
autoload='typeset -fu'
cat=/usr/bin/cat
command='command '
false='let 0'
functions='typeset -f'
hash='alias -t -'
history='fc -l'
integer='typeset -i'
ll='ls -al'
local=typeset
ls=/usr/bin/ls
lx='ls -l'
nohup='nohup '
...

$ unalias lx
$ alias
autoload='typeset -fu'
cat=/usr/bin/cat
command='command '
false='let 0'
functions='typeset -f'
hash='alias -t -'
history='fc -l'
integer='typeset -i'
ll='ls -al'
local=typeset
ls=/usr/bin/ls
nohup='nohup '
...

See also  alias
The `uname` command writes information related to the current operating system to standard output.

Syntax: \texttt{uname[...option]...}

No option specified

The name of the operating system is output, e.g. \texttt{POSIX-BC}.

Option

\begin{itemize}
  \item \texttt{-a} \texttt{(all)} Outputs the following information: the name of the operating system, the system’s node name, the version number of the operating system, the revision status of the operating system version, the machine hardware name and the processor type, e.g. \texttt{POSIX-BC D016ZE07 2.0A K10 BS2000 BS2000}
  \item \texttt{-m} \texttt{(machine type)} Outputs the machine hardware name, e.g. \texttt{BS2000}.
  \item \texttt{-n} \texttt{(node)} Outputs the node name of the operating system. This is the name by which a system is known in a communication network, e.g. \texttt{D016ZE07}.
  \item \texttt{-p} \texttt{(processor type)} Outputs the processor family of the system that you are currently using, e.g. \texttt{BS2000} for Intel-based systems.
  \item \texttt{-r} \texttt{(release)} Outputs the version number of the operating system, e.g. \texttt{2.0A}.
  \item \texttt{-s} \texttt{(system)} Outputs the name of the operating system. This is the name by which the operating system is known in the local installation, e.g. \texttt{POSIX-BC}.
  \item \texttt{-v} \texttt{(version)} Outputs the revision status of the POSIX system version, e.g. \texttt{K10}.
\end{itemize}

Locale

The following environment variables affect the execution of `uname`:

\begin{itemize}
  \item \texttt{LANG} Provide a default value for the internationalization variables that are unset or null. If \texttt{LANG} is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
  \item \texttt{LC_ALL} If set to a non-empty string value, override the values of all the other internationalization variables.
  \item \texttt{LC_CTYPE} Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
  \item \texttt{LC_MESSAGES} Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
\end{itemize}
uname

**NLSPATH**

Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example**

You want to find out the node name of the operating system as well as the version number of the operating system you are running:

```bash
$ uname -nr
D016ZE07 2.0A
```

**See also**

`uname()` [4]
uncompress expand compressed files

uncompress expands files compressed by compress and restores them to their original form. If the calling process has the appropriate privileges, the owner, access permissions, and the access and modification times are not changed.

Whenever possible, each file specified is replaced by a file with the same name but without the .Z extension.

Expansion is not performed if
- the file to be created already exists and uncompress is running in the background
- the specified file was compressed with a higher value for maxbits (see compress -b on page 225) than your current system can process.
- there are links to the compressed file.

Syntax

uncompress[[-cfv][...file...]]

No option specified
The specified files are expanded.

-c uncompress writes the expanded files to standard output. No files are modified or created. uncompress -c has exactly the same effect as zcat.

-f (force) file.Z is expanded even if a file named file already exists. file is overwritten.

-f not specified:
uncompress asks whether the existing file should be overwritten or not. You are not asked for confirmation, however, when uncompress is running in the background. The specified file is not expanded.

-v (verbose) uncompress issues a message confirming successful expansion.

file.Z: -- replaced with file

file
Name of a file compressed with compress. You can specify more than one file. Each file name must include a .Z suffix which you do not have to specify when invoking uncompress. The expanded file is assigned the name file. The file.Z is deleted after being successfully expanded (unless the -c option is specified). Access permissions, access and modification dates, and the owner of the file remain unchanged.

Error

file.Z: not in compressed format
The specified file is not available in compressed format or is a directory.

file.Z: compressed with xxbits, can only handle yybits
The specified file was compressed with too high a value for maxbits (see compress -b on page 225). The system on which you called uncompress cannot process this value. Compress the file again using the value specified as yy in the error message or with a lower value.
uncompress

file.Z: -- has xx other links: unchanged
There are xx links to the specified file. The file is not expanded.

uncompress: corrupt input
The SIGSEGV signal (segmentation violation; an addressing error caused by unauthorized
segment access) was received, which normally indicates that the input file is corrupted.

Locale
The following environment variables affect the execution of uncompress:

**LANG**  Provide a default value for the internationalization variables that are unset
or null. If LANG is unset of null, the corresponding value from the implementa-
tion-specific default locale will be used. If any of the internationalization
variables contains an invalid setting, the utility will behave as if none of the
variables had been defined.

**LC_ALL**  If set to a non-empty string value, override the values of all the other inter-
nationalization variables.

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data
as characters (for example, single- as opposed to multi-byte characters in
arguments).

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents
diagnostic messages written to standard error.

**NLSPATH**  Determine the location of message catalogs for the processing of
LC_MESSAGES.

Example  You want to expand the file topsecret.Z. The same directory already contains a file called
topsecret, but this can be overwritten.

```
$ ls -l
total 62
-rw-------  1 felix    group1    4862 Mar 12 10:16 topsecret
-rw-------  1 felix    group1   26326 Feb 08 10:27 topsecret.Z
```

```
$ uncompress -v topsecret.Z
```
topsecret already exists; do you wish to overwrite topsecret (y or n)? y
topsecret.Z: -- replaced with topsecret

```
$ ls -l
total 138
-rw-------  1 felix    group1   69845 Feb 08 10:27 topsecret
```

See also  compress, zcat
The `unexpand` command writes files or the standard input to the standard output. Blanks at the beginning of each line are converted to the maximum number of tabs, followed by the minimum number of blanks needed to pad the line to the next tab stop, which were originally filled by the converted characters.

A tab stop is set after every 8 column positions by default. All backspace characters are copied to the output and cause the column position count for for tab stop calculations to be decremented. The column position count will not be decremented below zero.

**Syntax**

```
unexpand[.\[-a\][-t\-tablist][\-file...]]
```

- **-a** Not only are the blanks at the start of each line converted, but also all sequences of two or more blanks directly in front of a tab stop are converted into the maximum number of tabs followed by the minimum number of blanks needed to pad the line to the next tab stop, which were originally filled by the converted characters.

- **-t..tablist**
  Specifies the tab stops. The argument `tablist` must consist of one or more numbers, separated by blanks or commas, in ascending order. A list separated by blanks must be enclosed in quotes. If only one number is specified, the tab stops will be set to `tablist` column positions instead of the default 8 column positions.
  If multiple numbers are given, the tabs will be set at the specified column positions.
  Each tab stop position `N` must be an integer value greater than zero, and the specifications must be in ascending order. This means that tabbing from the start of the line of output to position `N` causes the next character output to be in the `(N+1)`th column position in that line.
  For characters that are one position behind the last position defined in a list with multiple tab stops, there is no conversion of blanks into tabs.
  If `-r` is specified, conversion of blanks is not restricted to the leading blanks. `-a` is ignored in this case.

- **file**
  The file whose blanks are to be replaced by tab characters.

**Locale**

The following environment variables affect the execution of `unexpand`:

- **LANG**
  Specifies a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding default value from the internationalized environment is used. If one of the internationalization variables contains an invalid setting, the command behaves as if none of the variables have been defined.
unexpand

If this variable has been assigned a value, i.e. it is not a null string, this value overrides the values of all the other internationalization variables.

Determines the internationalized environment for the interpretation of byte sequences (e.g. single-byte characters as opposed to multi-byte characters in arguments and input files). The internationalized environment for the processing of tab characters and blanks and for the specification of the width in column positions each character would occupy on a constant-width output device.

Determines the format and contents of error messages.

Determines the position of message catalogs for the processing of LC_MESSAGES.

The standard output is equivalent to the input files with blanks converted into the appropriate number of tabs.

The default behavior of unexpand, whereby only leading blanks are taken into consideration, may not be desired in some cases. Users who always want to convert all blanks in a file should use an alias definition to create a version of unexpand that is always called with the options -a or -t..8.

See also expand, tabs
uniq

uniq report or filter out repeated lines in a file

The command `uniq` searches a file for sequences of identical lines, and writes the file to standard output, removing all but one of repeated lines in the process. Note that repeated lines must be adjacent in order to be found, i.e. the input file must be sorted.

Syntax

Format 1: `uniq[-c|-d|-u][[-n][-m]][input_file][output_file]`

Format 2: `uniq[-c|-d|-u][-f_field][-s_char][input_file][output_file]`

The two formats are defined together since option `-n` in format 1 is equivalent to the option `-f_field` in format 2 and option `-m` in format 1 is equivalent to option `-s_char` in format 2.

No option specified

The named `input_file` is output without repeated lines.

option

-c Outputs all lines without repetitions, starting each line with a decimal number to indicate how often it occurred repeatedly in `input_file`. `uniq` ignores the `-u` and `-d` options if set with the `-c` option.

-d Outputs one copy each of only those lines that are repeated in `input_file`.

-u Outputs only the lines that are not repeated in `input_file`.

-n Causes the first `n` characters from the beginning of the line to be ignored when comparing for duplicates. If the `-n` option is combined with the `-m` option, the first `n` characters after the `m`th field are ignored. Blanks following the `n`th field are not ignored: they must be allowed for in the value of `n`.

-n not specified:

Lines are compared from the beginning of the line or beginning with field `m+1`.

Option `-n` is equivalent to option `-f_field` in format 2.

+m Ignores the first `m` fields from the beginning of the line, plus any tabs or blanks located in front of a field, when comparing for duplicates. A field is a string of non-blank characters separated from its neighbors by tabs or blanks.

Option `-m` is equivalent to option `-s_char` in format 2.

-m not specified:

Lines are compared from the beginning of the line or beginning with character `n+1`.

`input_file`

Name of the file that is to be examined.

`input_file` not specified:

`uniq` reads from standard input.
uniq

output_file
Name of the file to which the output is to be written.
output_file not specified: uniq writes to standard output.

Locale  The following environment variables affect the execution of uniq:

LANG    Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL    If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE    Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files) and which characters constitute a blank character in the current locale.

LC_MESSAGES    Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH    Determine the location of message catalogs for the processing of LC_MESSAGES.

Example 1  You want to search a file for identical lines, regardless of where they are located in the file. A count showing how often each of these lines occurs is also to be output.
$ sort file | uniq -c
Example 2  You want to output the 10 most frequently occurring words in the file `text`.

```
$ cat text \
    | sed 's/ */ /g' \
    | tr ' ' '
' \
    | sed '/^$/d' \
    | sort \
    | uniq -c \
    | while read N W: do printf "%06d %s\n" $N "$W"; done \
    | sort -r \
    | head -n 10
```

Explanation:
- `sed` generates a list from `text` in which one or more blanks are replaced by one blank.
- `tr` replaces blanks in this list by newline characters.
- `sed` removes empty lines from this list.
- `sort` sorts this list according to EBCDIC.
- `uniq -c` outputs all lines without repetitions and in front of each one enters how frequently it occurs.
- The `while` loop replaces the frequency by a 6-digit number with leading zeros.
- `sort -r` sorts this frequency list backward, i.e. the most frequent line is contained in the first line.
- `head` outputs the first 10 lines of this list.

See also  `comm, sort`
unset unset values and attributes of variables and functions

The POSIX shell built-in `unset` removes the specified shell function or shell variable from the current environment.

The built-in environment variables `IFS, MAILCHECK, PATH, PS1,` and `PS2` cannot be deleted using `unset`.

Syntax

```
unset[-f][-v]...name....
```

`-f` If option `-f` is specified, `name` refers to a shell function.

`-v` If option `-v` is specified, `name` refers to a shell variable. Read-only shell variables cannot be deleted.

If neither `-f` nor `-v` is specified, `name` refers to a shell variable. If there is no variable with a corresponding name then it is possible that any function having the same name may be deleted.

`name` Name of the shell variable or function to be removed from the current environment. Several blank-separated names may be specified in a single command line.

Exit status

0 All shell variables or shell functions have been successfully deleted.

>0 At least one shell variable or shell function could not be deleted.

Locale

The following environment variables affect the execution of `unset`:

`LANG`  Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

`LC_ALL`  If set to a non-empty string value, override the values of all the other internationalization variables.

`LC_CTYPE`  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

`LC_MESSAGES`  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

`NLSPATH`  Determine the location of message catalogs for the processing of `LC_MESSAGES`.

794
Example  Remove the shell function `ll`:

```bash
$ type ll
ll is a function
ll(){
   ls -al $* • pg
}
$ unset ll
$ ll
ll not found
```

See also  `set`
**usp**

**set POSIX control parameters dynamically**

This command can only be issued by the superuser. The `usp` command has the following functions:

- Displays the current values of all POSIX control parameters (`-s` option).
- Modifies the current values of the following POSIX control parameters: `DBLPOOL`, `FORCEDTERM`, `HDSTNI`, `HDPTNI`, `HEAPSZ`, `MAXTIMERC`, `MAXUP`, `NOSTTY`, `NOTTY` and `NPROC`.
  Depending on the option specified (`-p` or `-P`), the modification is either only valid until the POSIX subsystem is terminated or also the next time POSIX is started.
- Checks whether a value specification is permitted for a selected control parameter (`-c` option).

**Syntax**

```
usp...-h
usp...-s[...parameter]
usp...-c...parameter ...v...value
usp...-p...parameter ...v...value
usp...-P...parameter ...v...value
```

**Options**

- **-h** (`h - help`) The syntax of the `usp` command and a list of all POSIX control parameters are output. In addition to the names and the meanings of the POSIX control parameters, this list also specifies whether the modification of a parameter is supported by the `usp` command (SUPPORTED) or not (unsupported).

- **-s** (`s - show`) The current value of the control parameters specified with `parameter` is output. If `parameter` is not specified, the current values of all (supported and unsupported) control parameters are output.

  **parameter**
  Name of the control parameter. Only the control parameters `DBLPOOL`, `FORCEDTERM`, `HDSTNI`, `HDPTNI`, `HEAPSZ`, `MAXTIMERC`, `MAXUP`, `NOSTTY`, `NOTTY` and `NPROC` which can be modified with the support of the `usp` command can be specified using the `-c`, `-p` and `-P` options. **All** control parameters can be specified with the `-s` option. The specification can be in uppercase or lowercase characters. You can also obtain the information regarding whether or not the modification of a control parameter is supported by the `usp` command using `usp -h`.

- **-c** (`c - check`) Checks whether the value specified using the `-v` option is within the limits permitted for `parameter` (minimum value ≤ value ≤ maximum value).
The current value of the control parameter specified using `parameter` is replaced by the value specified using the `-v` option provided this value is legal (see the `-c` option). The condition `value > current value` must also be fulfilled for the following control parameters: `HDSTNI`, `HDPTNI`, `HEAPSZ`, `MAXUP`, `NOSTTY`, `NOTTY` and `NPROC`. The modification only applies for the current POSIX subsystem session. After the POSIX subsystem has been restarted, the value from the POSIX information file `SYSSSI.POSIX-BC.<version>` applies again.

A modification to the control parameter `DBLPOOL` only becomes effective the next time the `posdbl` command is issued. The contents of the global program cache are not saved. A combination of the `usp` and `posdbl` commands is therefore required to modify the size of the global program cache; for details on this, see the POSIX manual "Basics for Users and System Administrators" [1].

(like `-p` option). The new value is also entered in the POSIX information file `SYSSSI.POSIX-BC.<version>` and consequently also applies after the POSIX subsystem has been restarted.

Before the modification is made in the POSIX information file, a backup copy is created with the name `SYSSSI.POSIX-BC.<version>.SAVE.<date><time>`.

(-v `value`)

`(v -value)` Specifies the value which is to be set or checked.

**Hint**

You are recommended to enter the modified values in the SYSSSI file (e.g. with the `-P` option) and to use them the next time the system is started. This makes it unnecessary to adjust them dynamically again during the system session.

When the maximum value `HDPTNI` (number of local file systems) is increased and used, the new value must be contained in the POSIX information file `SYSSSI.POSIX-BC.<version>` the next time POSIX is started. If more local file systems are entered in the management tables (e.g. `/etc/vfstab`) than the value in the POSIX information file permits, the start of the POSIX subsystem is aborted with an error.

All dynamic modifications to control parameters are documented by means of logging in POSIX. Each modification that is made to a value is documented on the console. In addition, all modifications are recorded in the POSIX file `/var/adm/messages`. 
**uudecode**

**uudecode**  **decode a binary file**

*uudecode* and *uuencode* are used in combination to send a text or binary file via *mailx*. You can send the file either directly or via a *mailx* chain linking a series of directly linked systems.

*uudecode* reads an encoded file and recreates the original file with the access permissions (file modes) and name specified in the encoding information (see section “*uuencode* encode a binary file” on page 800).

You must have write permission for the encoded file.

*uudecode* runs with the user ID assigned to *uucp*. Thus permission to use *uudecode* may be denied in a directory where there is no write permission for "others".

Refer to *uuencode* for further information.

**Syntax**

```
uudecode[<file>]
```

*file*

File to be decoded.

*file* not specified:

*uudecode* reads from standard input.

**Locale**

The following environment variables affect the execution of *uudecode*:

- **LANG**
  
  Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  
  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**
  
  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

- **LC_MESSAGES**
  
  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- **NLSPATH**
  
  Determine the location of message catalogs for the processing of *LC_MESSAGES*. 
Example User bill has been mailed an encoded file:

```
$ mailx
From john Wed Mar 11 14:42 MET 2009
Content_Length
begin 744 scrpt
M9F]R(&YA;64@*:6@*@ID:R @:6:@6R M9" B)&YA;64B(%T*(' @('!H)&X@M)&-H;R B*1V>BD@)&YA;64B" @("!;'-E(&%C:&\@B!@':.\+7,)&YA
1:6&@(' @)FD*9&\]N90HB
end

? q
```

The header line of the encoded message contains the word begin followed by the file mode (744) and the name of the file (scrpt). The file is now to be restored to its original condition and have the specified name scrpt:

```
$ mailx | uudecode
$ cat scrpt
for name in *
do  if [ -d "$name" ]
   then echo "(dir) $name"
   else echo " `ls -s $name`"
  fi
done
```

See also mailx, uuencode
uuencode

encode a binary file

**uuencode** and **uudecode** are used in combination to send a text or binary file via **mailx**. You can send the file either directly or via a **mailx** chain linking a series of directly linked systems.

**uuencode** takes the named source file or the data from standard input and produces an encoded version of it on the standard output. Only printing ASCII characters are used for encoding, which among other things enables the transfer of 8-bit data over systems that are not 8-bit transparent. The encoding includes the file mode (permissions) and destination file name for recreating the file on the remote system.

**Syntax**

```
uuencode [source-file] destination-file
```

- **source-file**
  - File to be encoded.
  - *source-file* not specified: **uuencode** reads from standard input.

- **destination-file**
  - Name of the destination file. Here you specify a path name which refers to the remote system.

**Hint**

**uuencode** and **uudecode** should be used as follows:

You call **uuencode**:

```
uuencode [source-file] destination-file | mailx system1!system2! ..!user
```

The encoded file is sent to the specified **user** on the remote system, who can then decode it with **uudecode**. The user on the remote system must have write permission for the file.

The encoded file produced by **uuencode** is a normal text file and can be edited with any text editor to change the file mode or the name of the destination file.

Encoding the source file expands it by 35% (3 bytes become 4 plus control information) and thus increases the time required to transmit it.

**Locale**

The following environment variables affect the execution of **uuencode**:

- **LANG**
  - Provide a default value for the internationalization variables that are unset or null. If **LANG** is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**
  - If set to a non-empty string value, override the values of all the other internationalization variables.
uuencode

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of LC_MESSAGES.

**Example**
A file called `letter` is to be sent in encoded form to user `bill` working on system `roland`. Once decoded the file is to be called `scrpt`

```
$ uuencode letter scrpt | mailx roland!bill
```

**See also**
`mailx`, `uudecode`
uuname

uuname list names of known systems

uuname lists the names of the local system.

Syntax

uuname[-l]

-1 Returns the local system name in the format "%s\n", <system name>
   This name may differ from the system name output by the uname -n command.
   -l not specified:
   uuname returns the exit status 0.

Locale

The following environment variables affect the execution of uuname:

LANG Provide a default value for the internationalization variables that are unset
       or null. If LANG is unset of null, the corresponding value from the implement-
       ation-specific default locale will be used. If any of the internationalization
       variables contains an invalid setting, the utility will behave as if none of the
       variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other inter-
        nationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data
        as characters (for example, single- as opposed to multi-byte characters in
        arguments).

LC_MESSAGES Determine the locale that should be used to affect the format and contents
       of diagnostic messages written to standard error and informative messages
       written to standard output.

NLSPATH Determine the location of message catalogs for the processing of
       LC_MESSAGES.

Example

Querying the exit status and displaying the local system name

$ uuname
$ echo $?
0
$ uuname -l
D016ZE07

See also

uname
vi screen oriented (visual) display editor

\textit{vi} is a display-oriented text editor.

Only users who have accessed the POSIX shell via \texttt{rlogin} can use \textit{vi} (except in line mode).

If you call \textit{vi} on a block-mode terminal, the \texttt{ex} editor will automatically be started instead.

Layout of this description

The description of \textit{vi} is divided into eight main parts:

- Command overview (extract) (on page 804ff)
- Introduction (on page 807)
- \textit{vi} modes (on page 808)
  - Command mode
  - Input mode
  - Last line mode
  - \texttt{ex} command mode
  - \texttt{ex} input mode
- Program invocation (on page 811)
- Screen layout (on page 814)
- Functionality (on page 815)
  - Saving the editor buffer and quitting \textit{vi}
  - Text buffers
  - Movement commands
- Commands (on page 820)
  - Definitions
  - Control commands
  - Commands of the \textit{vi} command mode
- Adapting \textit{vi} to the terminal (on page 839)
  - Customizing the \textit{vi} environment
  - Environment variables
Command overview

The following overview summarizes the most common \textit{vi} commands, grouping them together by function. In the section entitled \textit{Commands} you will find a detailed description of the commands in alphabetical order.

<table>
<thead>
<tr>
<th>Invoking \textit{vi}</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{vi}</td>
<td>Call \textit{vi} with empty editor buffer</td>
</tr>
<tr>
<td>\texttt{vi file}</td>
<td>Call \textit{vi} and load \textit{file}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quitting \textit{vi}</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ZZ}</td>
<td>Quit \textit{vi}, writing editor buffer out to current file; as \texttt{:wq}</td>
</tr>
<tr>
<td>\texttt{:w}</td>
<td>Write editor buffer to current file</td>
</tr>
<tr>
<td>\texttt{:w file}</td>
<td>Write editor buffer to \textit{file}</td>
</tr>
<tr>
<td>\texttt{:q}</td>
<td>Quit \textit{vi} if changes saved</td>
</tr>
<tr>
<td>\texttt{:q!}</td>
<td>Quit \textit{vi}, discarding unsaved changes</td>
</tr>
<tr>
<td>\texttt{:wq}</td>
<td>Quit \textit{vi}, writing editor buffer out to current file; as \texttt{ZZ}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Positioning the cursor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{h}</td>
<td>One character to left</td>
</tr>
<tr>
<td>\texttt{j}</td>
<td>One line down in same column</td>
</tr>
<tr>
<td>\texttt{+}</td>
<td>One line down and to start of line</td>
</tr>
<tr>
<td>\texttt{d}</td>
<td>One line down and to start of line</td>
</tr>
<tr>
<td>\texttt{k}</td>
<td>One line up in same column</td>
</tr>
<tr>
<td>\texttt{-}</td>
<td>One line up and to start of line</td>
</tr>
<tr>
<td>\texttt{l}</td>
<td>One character to right</td>
</tr>
<tr>
<td>\texttt{H}</td>
<td>Top of screen</td>
</tr>
<tr>
<td>\texttt{M}</td>
<td>Middle of screen</td>
</tr>
<tr>
<td>\texttt{L}</td>
<td>Bottom of screen</td>
</tr>
<tr>
<td>\texttt{0}</td>
<td>Left edge of screen</td>
</tr>
<tr>
<td>\texttt{l}</td>
<td>Left edge of screen</td>
</tr>
<tr>
<td>\texttt{S}</td>
<td>Last character in line</td>
</tr>
<tr>
<td>\texttt{^}</td>
<td>First non-whitespace character in line</td>
</tr>
<tr>
<td>\texttt{w}</td>
<td>Start of next word</td>
</tr>
</tbody>
</table>
### Positioning the cursor

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>e</code></td>
<td>End of next word</td>
</tr>
<tr>
<td><code>b</code></td>
<td>Start of previous word</td>
</tr>
<tr>
<td><code>W</code></td>
<td>Start of next word (extended word)</td>
</tr>
<tr>
<td><code>E</code></td>
<td>End of next word (extended word)</td>
</tr>
<tr>
<td><code>B</code></td>
<td>Start of previous word (extended word)</td>
</tr>
</tbody>
</table>

### Positioning the window

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CTRL</code>+<code>d</code></td>
<td>Half a screen down</td>
</tr>
<tr>
<td><code>CTRL</code>+<code>u</code></td>
<td>Half a screen up</td>
</tr>
<tr>
<td><code>CTRL</code>+<code>f</code></td>
<td>One screen down</td>
</tr>
<tr>
<td><code>CTRL</code>+<code>b</code></td>
<td>One screen up</td>
</tr>
<tr>
<td><code>CTRL</code>+<code>e</code></td>
<td>One line down</td>
</tr>
<tr>
<td><code>CTRL</code>+<code>y</code></td>
<td>One line up</td>
</tr>
<tr>
<td><code>z</code>+<code>J</code></td>
<td>Current line to top of screen</td>
</tr>
<tr>
<td><code>z</code>+<code>.</code></td>
<td>Current line at middle of screen</td>
</tr>
<tr>
<td><code>z</code>+<code>-</code></td>
<td>Current line to bottom of screen</td>
</tr>
<tr>
<td><code>1G</code></td>
<td>Start of file</td>
</tr>
<tr>
<td><code>nG</code></td>
<td>Line n to middle of screen</td>
</tr>
<tr>
<td><code>G</code></td>
<td>End of file</td>
</tr>
</tbody>
</table>

### Deleting and restoring text

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>Delete character at cursor position</td>
</tr>
<tr>
<td><code>X</code></td>
<td>Delete character to left of cursor</td>
</tr>
<tr>
<td><code>dd</code></td>
<td>Delete current line</td>
</tr>
<tr>
<td><code>dG</code></td>
<td>Delete from current line to end of file</td>
</tr>
<tr>
<td><code>dH</code></td>
<td>Delete from start of file to current line inclusive</td>
</tr>
<tr>
<td><code>D</code></td>
<td>Delete rest of line from cursor position</td>
</tr>
<tr>
<td><code>u</code></td>
<td>Restore deleted text from standard buffer</td>
</tr>
<tr>
<td><code>U</code></td>
<td>Restore current line</td>
</tr>
<tr>
<td><code>p</code></td>
<td>Write buffer contents after current cursor position</td>
</tr>
<tr>
<td><code>P</code></td>
<td>Write buffer contents before current cursor position</td>
</tr>
</tbody>
</table>
## vi, Command overview

### Changing text

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Replace current character with next character entered</td>
</tr>
<tr>
<td>R</td>
<td>Overwrite text starting at current character</td>
</tr>
<tr>
<td>o</td>
<td>Insert blank line below current line</td>
</tr>
<tr>
<td>O</td>
<td>Insert blank line above current line</td>
</tr>
<tr>
<td>a</td>
<td>Append text after current character</td>
</tr>
<tr>
<td>A</td>
<td>Append text to end of line</td>
</tr>
<tr>
<td>i</td>
<td>Insert text before current cursor position</td>
</tr>
<tr>
<td>I</td>
<td>Insert text at start of line</td>
</tr>
</tbody>
</table>

### Copying

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Yank current line</td>
</tr>
<tr>
<td>yy</td>
<td>Yank current line</td>
</tr>
<tr>
<td>Yp</td>
<td>Duplicate current line</td>
</tr>
<tr>
<td>yyp</td>
<td>Duplicate current line</td>
</tr>
<tr>
<td>:10,13t 20</td>
<td>Copy lines 10 through 13 below line 21</td>
</tr>
<tr>
<td>:t.t.</td>
<td>Duplicate current line</td>
</tr>
<tr>
<td>:t.</td>
<td>Duplicate current line</td>
</tr>
<tr>
<td>:r file</td>
<td>Copy contents of file below current line</td>
</tr>
</tbody>
</table>

### Cut and paste

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3dd</td>
<td>Delete 3 lines including current line</td>
</tr>
<tr>
<td>p</td>
<td>Insert saved lines</td>
</tr>
<tr>
<td>^a3dd</td>
<td>Delete 3 lines and save in buffer a</td>
</tr>
<tr>
<td>^ap</td>
<td>Insert saved line below current line</td>
</tr>
</tbody>
</table>

### Search and replace

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/xyz</td>
<td>Search forward for xyz</td>
</tr>
<tr>
<td>/xyz.../</td>
<td>Search for xyz and 2 blanks</td>
</tr>
<tr>
<td>:s/xyz/abc/</td>
<td>Replace xyz with abc in current line</td>
</tr>
<tr>
<td>:old/a/old/new</td>
<td>Find old, then replace old with new</td>
</tr>
<tr>
<td>:1,$/old/new/g</td>
<td>Globally replace old with new</td>
</tr>
</tbody>
</table>
Introduction

`vi` displays a maximum of 23 lines of the file being edited. You can look at different parts of the file by moving this window. `vi` offers a wide range of text editing options. You can

- create, modify, copy and delete text
- position the cursor using simple commands (e.g. at the beginning or end of a word, line, sentence, paragraph, or the file)
- search and replace text patterns using regular expressions
- call a subshell
- process a range of the current file with a POSIX command
- edit multiple files during the same session and copy text from one file to another
- recover a file after an interrupted session.

`vi` is an enhanced version of the `ex` line editor (see `ex`). You can switch between the two editors while editing and you can execute `ex` commands from within `vi`. 
vi, Modes

vi modes

The vi editor provides you with a number of modes in which you can operate:

– vi command mode
– vi input mode
– vi last line mode
– ex command mode
– ex input mode.

When invoked, the editor always comes up in the vi command mode, from which you can switch to other modes if required. The mode of the editor determines how keystrokes are interpreted.

vi command mode

When vi is invoked, it comes up in the vi command mode. No text can be entered in this mode. Keystrokes are immediately interpreted as vi commands, without being displayed on the screen. If the entered command is legal, it is executed, and the result is immediately displayed on the screen.

vi input mode

The input mode allows you to enter or modify text in the editor buffer. The editor switches to this mode when you enter one of the vi commands A, a, C, c, I, i, O, o, S, s or R (see the section entitled “Commands of the vi command mode” on page 826). All following input characters, including various non-printing characters, are written to the buffer and the screen.

Most vi commands cannot be used in input mode. There are, however, a few vi commands that have a special significance in input mode. These include:

- **ESC**  
  Exit input mode; return to vi command mode.

- **DEL**  
  Terminate input mode; return to vi command mode.

- **J**  
  Newline.

- **CTRL@**  
  Repeat last text input. You enter this command immediately after switching to input mode. vi inserts the last text entered in input mode provided this text consists of no more than 128 characters. Control is immediately switched back to command mode.
vi, Modes

CTRL\text{n}\text{d}
If you have defined automatic indentation, you can use \text{CTRL}\text{n}\text{d} to move the margin one tab stop to the left. You can set the shift width with the \textit{shiftwidth} option. This command is only relevant at the beginning of a new input line and with the \textit{autoindent} option set.

CTRL\text{h}
Go back one character.

CTRL\text{t}
One tab to the right in accordance with the value of \textit{shiftwidth}. This command may only be entered at the start of a new input line.

CTRL\text{v}
Insert a non-printing character.

CTRL\text{w}
Go back one word.
\textbackslash \quad \text{Escape for rubout character \textcircled{G}.}

\textit{vi last line mode}

In the \textit{vi} last line mode, you can enter \textit{vi}, \textit{ex}, and POSIX commands in the status line. The last line mode is activated by the input of a colon (:), slash (/), or question mark (?). Another method is to enter an exclamation mark (!), followed by a cursor movement command to a line. Commands entered in the last line mode must be terminated with \textbackslash \text{n} or \text{ESC}. The input of a command can be aborted by pressing \text{DEL}.

\textbf{Caution!}
Entering \text{ESC} in the \textit{vi} last line mode does not delete a command just entered, but causes it to be executed instead. In \textit{vi} last line mode you must use \text{DEL} to delete an entered command.

\textbf{:}
Entering a colon in the \textit{vi} command mode causes all following input up to the next \text{ESC} or \text{n} to be interpreted as an \textit{ex} command. (All \textit{ex} commands are described under \textit{ex}, \textit{ex commands}.) After the colon is entered, the cursor moves to the status line at the bottom of the screen, and the colon as well as the input command are displayed there. Almost all \textit{ex} commands can now be used, except for those that would switch \textit{ex} to the \textit{ex} input mode.
The percent character (%) represents the name of the file currently in the buffer; the hash character (#) stands for the name of the file last processed during the same session. (see section “Current and alternate file” on page 327).
Example

```bash
:! diff # %
```

`vi` writes the differences between the previously edited file and the current file on standard output. Neither of the two files is changed.

/ or ?

When a slash or question mark is entered, the cursor moves to the status line, allowing you to specify a regular expression for a forward search (with `/`) or a backward search (with `?`). The `vi` command `n` repeats the search in the same direction; `N` repeats the search in the reverse direction.

!

When an exclamation point is entered in the `vi` command mode and a movement command specifying a line is given to indicate the scope, the cursor moves to the status line. The exclamation mark is then displayed at the start of the status line. If you enter a second exclamation mark as the movement command, the position is the current line. The movement command is not displayed. You can now enter a POSIX command, which must be terminated with `]` or `[ESC]`. The scope, i.e. the range from the current line to the specified position, becomes the input of the POSIX command (if it reads from standard input) and is replaced by its output (standard output and standard error). If no output is provided by the command (e.g. `true`), the specified range is deleted, i.e. replaced by a "null" string.

ex command mode

You switch to `ex` command mode by entering the command `Q` in `vi` command mode. You thus quit the display-oriented `vi` and switch to the line-oriented `ex` editor. Entering the command `vi[a]` in `ex` command mode returns you to `vi`.

ex input mode

You switch to the `ex` input mode by entering the command `a`, `i` or `c` in the `ex` command mode. All input in this mode is written to the editor buffer. To terminate this mode you enter a single period in the first column of a line.

Most `ex` commands can be entered in one of two ways:

- from within `vi` command mode, by entering the `vi` command colon `:` followed by the `ex` command required.
- from within `ex` command mode.
Program invocation

When invoked, the vi editor comes up in the vi command mode. If you have switched to vi input mode, pressing [ESC] (or [DEL]) will take the editor back to the vi command mode.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi[...option][...file].</td>
<td></td>
</tr>
<tr>
<td>option</td>
<td></td>
</tr>
<tr>
<td>-t...tag</td>
<td>The file containing the tag is loaded for editing. The editor positions the line containing the definition of the tag in the center of the screen. A tags file containing the search strings for the definition must be in the same directory. This option is typically used by for C programmers who want the editor to be positioned on the definition of a function or macro as soon as the file is opened. The tags file needed for this purpose must previously have been created with the ctags command.</td>
</tr>
<tr>
<td>-l</td>
<td>The lisp mode is activated, and the text is correspondingly indented to comply with typical Lisp programming conventions. The following vi commands have special meanings in lisp mode: (, ), /, /], [[ and ]].</td>
</tr>
<tr>
<td>-r[...file]</td>
<td>Restores a vi session if the vi editor or the system crashed during your previous session. Changes that are only in the editor buffer are not copied to the edited file after an editor or system crash. The POSIX operating system does, however, try to back up the buffer contents by creating a copy of the buffer, if possible. The file containing changes made prior to the crash is recovered and retrieved into the vi buffer. You can now continue editing the file or write the changes to another file.</td>
</tr>
<tr>
<td>-L</td>
<td>Lists the names of all the files saved following a system or editor crash.</td>
</tr>
<tr>
<td>-R</td>
<td>Opens the named file in read-only mode. This mode prevents the original file from being accidentally overwritten. The file can still be overwritten with the ex command w!</td>
</tr>
<tr>
<td>-w...n</td>
<td>Sets the window size to n text lines.</td>
</tr>
</tbody>
</table>

Caution!

The file can still be overwritten with the ex command w!

-w...n not specified:

The window size is set to the value of the LINES environment variable. If this variable has not been defined, the default value is as set in the terminfo definition for the terminal.
vi, Invocation

-c..command

This option enables you to invoke vi and directly execute an ex command or use a vi movement command to position the editor at a specific line of the file being edited. If you use a movement command, the line required will be displayed in the center of the screen. By contrast, if you execute an ex command, the editor will be positioned at the last line of the file after the command is executed, unless the ex command you specify (e.g. search) positions the editor itself.

command not specified:
vi issues a usage message and aborts.

command specified:
command may be specified as a line number (n), a search command (/pattern, see the section entitled Commands of the vi command mode), or any other ex command ("ex command" or "ex command"). The specified command is executed when vi is loaded.

n
n is an integer. vi is positioned to the nth line of the file.

/pattern
vi is positioned to the line containing pattern. pattern is a simple regular expression (see Tables and directories, Regular expressions). If the specified pattern includes metacharacters, you must escape such characters with a backslash \\ or enclose pattern in single quotes 'pattern', so that the shell does not interpret the metacharacters.

'ex-command' or "ex-command"
ex-command can be any arbitrary ex command. The ex command must be quoted (using single or double quotes) in order to prevent it from being interpreted by the shell. If the editor is not already positioned as a result of an ex command, vi will position to the last line of the file.

Example

When vi is called with:

vi -c "set number showmode" appointments

the file appointments is opened, the cursor is position on the last line of the file, and the line numbers and the vi mode are displayed on the screen (see example in ex description).

file

Name of the file you want to edit. If you give a number of files, vi begins with the first file specified. The ex command n can be used to switch to the next file (see also ex commands rew and ar).

file not specified:
If you invoke vi without a file name, an empty buffer is initially provided. If required, the contents of this buffer can be later written to a named file by using the ex command w file.
**Acoustic signal (not for block-mode terminals)**

An acoustic signal is sent to the terminal when

- you press the **ESC** or **DEL** keys in command mode
- you use an illegal command
- `vi` receives a SIGINT signal.

If the `vi` editor receives a SIGINT signal (**DEL**) during text input, or during the input of a command on the bottom line, the input is terminated (or the command cancelled) and the editor returns to the command mode. The receipt of a SIGINT signal in the command mode produces an acoustic signal.
Screen layout

A character screen on a terminal is generally made up of 24 lines and 80 columns. By default, the first 23 screen lines are used to display the text of the file being edited. In xterm sessions more than 24 lines can be displayed. In this case the number of screen lines that \texttt{vi} uses is determined automatically.

The last line on the screen is a status and command line. The status line is used to enter commands and to display information and messages from the \texttt{vi} editor. Text lines that do not fit on the screen are continued in the next line.

\texttt{vi} uses two special characters as markers in column 1 to identify the line status.

\texttt{~} The tilde (\texttt{~}) identifies lines that are present on the screen but not in the editor buffer. These are lines past the end of the file. If the last text line of a file is located in the middle of the screen, for example, all lines thereafter will be marked with \texttt{~}. The tilde disappears as soon as you enter text into the first such marked line. When you invoke \texttt{vi} with a new or empty file, all screen lines except for the first and last line (status line) are marked with this character.

\texttt{@} The commercial at (\texttt{@}) marker indicates lines which \texttt{vi} cannot display correctly on the screen. There are two possible causes:

\begin{itemize}
\item A text line may extend over more than one line on the screen. This means that text lines may be present in the editor buffer, but the space on the screen after the last fully displayed text line does not suffice for another full line of text.
\item Deleting individual lines from a file can mean that what appears on the screen no longer matches the contents of the editor buffer. In this case, the \texttt{vi} command \texttt{CTRL}\texttt{L} can be used to redraw the screen.
\end{itemize}

The last line on the screen is the status line. It is used:

\begin{itemize}
\item to enter \texttt{vi} commands that begin with \texttt{/}, \texttt{?}, \texttt{!} or \texttt{:}
\item to display messages
\item for the output of \texttt{ex} commands
\item to display the mode (the command mode is not displayed).
\end{itemize}
Functionality

`vi` always works on a buffer, not on files. When you start a `vi` session, a copy of the file you are editing is read into the editor buffer. If you omit the file name or specify a name that does not exist, you will begin with an empty buffer. All your editing changes will initially be recorded in this buffer; the original file is not changed until you copy the buffer contents back to it. This is done by explicitly saving the buffer during the session (with the `ex` command `w`) or while quitting the `vi` editor (with the `ex` commands `wq`, `x`, or the `vi` command `ZZ`). Use the `ex` command `q!` if you prefer to quit `vi` without modifying the original file. If you are creating a new file, it will not be created until you explicitly save the buffer contents to this file.

Saving the editor buffer and quitting `vi`

`vi` must be in `vi` command mode in order to save the editor buffer or quit `vi`.

You save the contents of your editor buffer to the edited file or to a specified file by means of the following `ex` command (see Functionality of `vi`, Entering `ex` commands):

```
:w[<file>]
```

*(write, `ex` command)* The contents of the editor buffer are saved to the specified file.

*file* not specified:

The contents of the editor buffer are saved to the edited file.

You have the following ways of quitting `vi`:

```
:q
```

*(quit, `ex` command)* Quit `vi`. This command functions only if no changes have yet been made to the editor buffer or if the modified editor buffer was saved to a file.

```
:q!
```

*(quit, `ex` command)* Quit `vi`. Changes made to the editor buffer are lost.

```
:x
```

*(exit, `ex` command)* Quit `vi` and write the modified editor buffer to the current file.

```
ZZ
```

*(`vi` command)* Quit `vi` and write the modified editor buffer to the current file.

```
:wq[<file>]
```

*(write and quit, `ex` command)* Quit `vi` and write the editor buffer to the specified file.

*file* not specified:

The contents of the editor buffer are written to the current file.
Text buffers

The vi editor makes use of
- one temporary buffer
- 9 numbered buffers
- 26 named buffers

The contents of these buffers can be accessed by using vi commands in the vi command mode.

Temporary buffer

The last change that you make in the editor buffer is stored by vi in the temporary buffer. If, for example, you have deleted a line, the buffer will contain this line. It is used by the vi commands U and u (undo). The u command therefore allows you to reverse the effect of the last command that changed the editor buffer. u also reverses the effect of u.

If the last vi command was a copy (y), delete (d), or replace (except R and r) command, the temporary buffer can also be accessed by using the vi commands P and p (put).

P and p copy the contents of the temporary buffer before (P) or after (p) the current line. Unlike U (u), P (p) does not modify the temporary buffer. This allows you to use P (p) several times in order to insert the same text at multiple locations in the file.

Caution!
The contents of the temporary buffer are overwritten by each command that changes the editor buffer. The temporary buffer cannot therefore be used to copy text from one file to another.

The only commands that may be used between a copy, delete, or replace command and the vi command P (p) are cursor movement commands (see “Movement commands” on page 818). This is because other commands would change the contents of the temporary buffer.

If vi is in input mode and you use one of the arrow keys, the vi command U (u) cannot be subsequently used to undo editing changes.

Numbered buffers

When you delete one or more lines (vi command d), the text you delete is copied to the first of nine numbered buffers. If you now delete some more lines, the contents of buffer 1 are copied to buffer 2, and the most recently deleted lines are again preserved in buffer 1. Thus, buffer 1 always contains the most recently deleted material, buffer 2, the text from the previous delete, and so on, until buffer 9, which contains the material deleted nine steps ago.
The contents of numbered buffers can be retrieved using the \textit{vi} commands \texttt{P} and \texttt{p}. Numbered buffers are referenced by a double quote and the number of the buffer in question. For example: \texttt{"4P}

This \textit{vi} command inserts the contents of buffer number 4 before the current line (or current position of the cursor).

The \textit{vi} dot (.) command recovers the contents of the buffer from the next numbered buffer, provided your last \textit{vi} command also retrieved text from a numbered buffer. In the above example, you would thus retrieve the contents of buffer number 5. The contents of numbered buffers are not lost when you switch files (with \textit{ex} command \texttt{n} or \texttt{e}).

\textit{Named buffers}

A copy (\texttt{y}) or delete (\texttt{d}) command can be used to write text into one of the twenty-six named buffers. These buffers are designated in lowercase from \texttt{a-z} or uppercase from \texttt{A-Z}.

Lowercase and uppercase letters refer to the same buffer, however, with the following difference.

When you use a lowercase letter, the old buffer contents are overwritten by the new text, i.e. are deleted. By contrast, when uppercase letters are used, the old buffer contents are retained, and the new text is appended to the old. The \textit{vi} editor does not change the contents of a named buffer during an editor session unless you explicitly overwrite its contents. \textit{vi} writes text into a named buffer when you enter a double quote, followed by the buffer name and one of the \textit{vi} commands \texttt{d} or \texttt{y}.

\textit{Examples}

\texttt{'A10dd}

Delete (\texttt{ad}) the current line and 9 additional lines (10) and append them to the contents of buffer \texttt{A}.

\texttt{'by4w}

Copy (\texttt{y}) the current and the next three words (\texttt{4w}) into buffer \texttt{b}.

\texttt{'cCstring[ESC]}

Overwrite (\texttt{C}) the current line from the cursor position to the end of the line with \texttt{string}, and save the overwritten text in buffer \texttt{c}.

\texttt{'Ap}

Recover the text from buffer \texttt{A} and insert (\texttt{p}) into the file after the current line or position.
Movement commands

`vi` provides you with a large number of commands, including search commands, which control the position of the cursor on the screen. These “movement” commands are entered in the `vi` command mode. The following overview lists the most important movement commands. The definitions of paragraph, section, sentence, word and extended word are given in the section “Definitions” on page 820.

Character-oriented positioning

`[n]CTRL|h`  
$n$ characters to the left

`[n]h`  
$n$ characters to the left

`[n]l`  
$n$ characters to the right

`0`  
to the start of a line

`^`  
to the first non-whitespace character in the line

`[n]l`  
to the first or $n$th column of the line

`$`  
to the last character in the line

`[n]f`char  
forward to the $n$th instance of `char`

`[n]f`char  
back to the $n$th instance of `char`

`[n]t`char  
forward to the position to the left of the $n$th instance of `char`

`[n]t`char  
back to the position to the right of the $n$th instance of `char`

Word-oriented positioning

`[n]b`  
back to the beginning of the $n$th previous word

`[n]B`  
back to the beginning of the $n$th previous extended word

`[n]e`  
forward to the end of the $n$th word

`[n]E`  
forward to the end of the $n$th extended word

`[n]w`  
forward to the beginning of the $n$th word

`[n]W`  
forward to the beginning of the $n$th extended word

Line-oriented positioning

`[n]k`  
up $n$ lines in the same column

`[n]j`  
down $n$ lines in the same column

`[n]CTRL|J`  
down $n$ lines in the same column
vi, Functionality

[n]J  down n lines to the first non-whitespace character
[n]+  down n lines to the first non-whitespace character
[n]-  up n lines to the first non-whitespace character
[line]G to the specified line number or end of file (if no line is given)
[n]$ to the last character of the n'th line

Positioning by sentences, paragraphs, and sections

[n]( back to beginning of n'th previous sentence
[n]) forward to beginning of n'th sentence
[n]{ back to beginning of current paragraph
[n}] forward to beginning of next paragraph
[n][[ back to n'th previous section boundary
[n][]] forward to n'th section boundary
% to the matching parenthesis or brace in C source programs.

Screen-oriented positioning

[n]H to the first character in the n'th text line on the screen
M to the middle of the screen
[n]L to the first character in the last text line on the screen
[line]z+ current line or specified line to top of screen
[line]z. current line or specified line to middle of screen
[line]z- current line or specified line to bottom of screen

Searching for patterns and markers

/pattern forward to pattern
?qpattern back to pattern
'marker to marked line (beginning of line)
`marker to marker.
vi, Commands

Commands

When invoked, the vi editor comes up in the vi command mode. If you have switched to vi input mode, pressing the [ESC] (or [DEL]) key will take the editor back to the vi command mode.

As described above, you can enter ex commands either in vi command mode or in ex command mode. Only the vi commands are described below. ex commands are described under ex.

Hitting [ESC] or [DEL] in the vi command mode cancels a partial command. [ESC] sounds an acoustic signal for character terminals if the editor is not in input mode or if there is no partially entered command.

Caution! Entering [ESC] in the vi last line mode does not delete a command just entered, but causes it to be executed instead. In vi last line mode you must use the [DEL] key to delete an entered command.

Unless otherwise specified, the commands are valid only in vi command mode and have no special effect in input mode.

Definitions

Extended word

The term "extended word" refers to any sequence of characters which does not include any blanks, tabs, or newline characters. See Word.

Paragraph

A paragraph is defined by the value of the ex option paragraphs. Blank lines and lines in which a section begins are also treated as paragraph boundaries.

Section

A section is defined by the value of the ex option sections. Lines that begin with the formfeed character ([CTRL] L) or a left brace { are also treated as section boundaries.

If the ex option lisp has been set, any left parenthesis at the start of a line is also treated as a section boundary.

Sentence

A sentence end is defined by

a period (.),
an exclamation point (!), or
a question mark (?)
followed by at least two spaces (to avoid confusion with abbreviations) or a newline character. To allow for quotations, any right parentheses, right square brackets, or single or double quotes between the punctuation and the newline character or spaces are ignored.

Word
A word is either any sequence of alphanumeric characters or a string of special characters. A word is terminated by a blank, tab, newline character or the following word.
See Extended word on page 820.

Examples
((!!++;-. is one word
me&you2 are 3 words (me & you2), but one extended word.

String
Any sequence of characters.
Control commands

To enter a control command, you hold down [CTRL] and press the key for the command required. Many control commands can be prefixed by a positive integer which specifies a count. The square brackets in the description signify that the specification of a count is optional. You must not press [CTRL] whilst entering count. The specified count is not displayed, but influences the effect of the subsequent command. If you specify an incorrect value, you can cancel your input by pressing [ESC]. The count is used to:

- indicate the range of text to be affected in a movement command, e.g.

  5[CTRL]U
  scrolls back the screen by five lines.

- specify a repetition factor, e.g.

  4[CTRL]B
  scrolls four screen pages back.

Unless explicitly stated otherwise, the default value for count is one.

[CTRL] @

In vi input mode:
If you enter this and nothing else, the last text input is repeated at the position at which you enter [CTRL]@. Afterwards, control is automatically returned to command mode.

[CTRL] @ only functions for a maximum of 127 characters.

[count] [CTRL] b

(backward) Scrolls backward by 21 screen lines to display the window above the current one. A count can be used to indicate the number of windows the editor should go back. If possible, two overlapping lines from the previous screen are displayed in the new window.

[count] [CTRL] d

In vi command mode:

- count not specified:
  count = 11, or last count used with [CTRL]d or [CTRL]u
  (down) Causes the screen to scroll forward by half a window. The count specifies the number of text lines to be scrolled, and will be remembered during your vi session for future [CTRL]D and [CTRL]U commands.

- In vi input mode:
  count not specified:
  In input mode, this command causes the cursor to back up by shiftwidth spaces over the indentation provided by the ex option autoindent or [CTRL]t (see ex, ex options). [CTRL]d only functions in a line in which no characters have been entered as yet.

  count can also be: ^ or 0
vi, Commands

\[^{\text{CTRL}}\text{d}\] Cancels automatic indentation for this line.

\[0^{\text{CTRL}}\text{d}\] Terminates the automatic indentation feature.

\[\text{count}\] \[^{\text{CTRL}}\text{e}\]
Scrolls forward by the number of lines given in \text{count}, leaving the cursor where it is if possible.

\[\text{count}\] \[^{\text{CTRL}}\text{f}\]
(forward) Scrolls forward by 21 screen lines. A \text{count} may be used to indicate how many screen pages to scroll forward.

\[^{\text{CTRL}}\text{g}\]
Displays the following information in the status line:
– the current file name
– possibly a status entry such as [Modified] if the file has been changed since the last time it was saved with the \text{ex} command \text{w}
– the current line number
– the total number of all text lines in the editor buffer
– the percentage of the current line to the total length of the file in lines.

This command is equivalent to the \text{ex} command \text{f}.

\[\text{count}\] \[^{\text{CTRL}}\text{h}\]
(same as \[\text{CTRL}\text{h}\])
In \text{vi} command mode:
The cursor is moved by \text{count} characters to the left but no further than the left margin (see also the \text{vi} command \text{h}).

In \text{vi} input mode:
A \text{count} cannot be specified.

The cursor is then moved by one character to the left but stops at the beginning of the text just entered.

\[\text{count}\] \[^{\text{CTRL}}\text{j}\]
(same as \[\text{CTRL}\text{n}\] and \[\text{CTRL}\text{j}\])
Moves the cursor \text{count} lines down in the same column.

This command is equivalent to \[\text{CTRL}\text{n}\] and \[\text{CTRL}\text{j}\].

\[^{\text{CTRL}}\text{l}\]
Clears and redraws the current screen. This command can be used when messages or reports are displayed on the screen (e.g. as a result of a \text{write}).

\[\text{count}\] \[^{\text{CTRL}}\text{m}\]
(same as \[\text{CTRL}\text{J}\])
Moves the cursor to the first character in the next line that is not a blank or tab. A \text{count} specifies the number of lines to go forward.
[count] \texttt{CTRL}n
This command is equivalent to \texttt{CTRL}j and \texttt{j}.

[count] \texttt{CTRL}p
Moves the cursor \texttt{count} lines up in the same column.
This command is equivalent to the \texttt{vi} command \texttt{k}.

\texttt{CTRL}r
(refresh) The current screen is cleared and redrawn. Previously deleted lines marked with \texttt{@} are removed. See \texttt{CTRL}l.

\texttt{CTRL}t
In \texttt{vi} input mode:
(tabulator) Inserts white space, defined by the value specified in the \texttt{ex} option \texttt{shiftwidth}
(see section “\texttt{ex options}” on page 345). The white space will, however, not be "visible" until the \texttt{vi} input mode is terminated. The inserted space can only be backed over by using the \texttt{CTRL}d command.

[count] \texttt{CTRL}u
\texttt{count} not specified:
\texttt{count} = 11, or last \texttt{count} used with \texttt{CTRL}d or \texttt{CTRL}u
(up) Scrolls up a half-window of text. A \texttt{count} can be specified to scroll a specific number of lines. The \texttt{count} is remembered during the \texttt{vi} session for future \texttt{CTRL}d and \texttt{CTRL}u commands.

\texttt{CTRL}v
In \texttt{vi} input mode:
“Quotes” or “escapes” a special character so that it can be placed in the text (e.g. escape or control characters). For instance, the sequence \texttt{CTRL} v \texttt{ESC} places the ESC character into the file.

\texttt{CTRL}w
In \texttt{vi} input mode:
(word) Backs up to the start of the previous word, but cannot be used to move beyond the text just entered. The backed over words remain displayed on the screen but are deleted from the buffer and must be re-entered, if required.

[count] \texttt{CTRL}y
Scrolls backward by \texttt{count} lines (towards the beginning of the file), leaving the cursor where it is, if possible.
vi, Commands

\texttt{CTRL][}
(equivalent to \texttt{ESC})

In \texttt{vi} command mode:
Cancels a partially entered command; sounds an acoustic signal if there is none.

In \texttt{vi} last line mode:
Terminates input of a command just entered and executes it.

In \texttt{vi} input mode:
Terminates \texttt{vi} input mode.

\texttt{ESC}

See \texttt{CTRL}[.}
vi, Commands

Commands of the vi command mode

Most vi commands accept a preceding number as an argument. Unless explicitly stated otherwise, the default value is one. This number, or count, is shown within square brackets in the description, which indicates that the entry is optional. The specified count is not displayed, but influences the effect of the following command. If you specify an incorrect count, you can cancel your input by pressing [ESC].

The following cases must be differentiated:

[count]command
  Here, count specifies how many times the command is to be executed. count must be a positive integer. count cannot be zero since 0 is also a vi command.

  count not specified:
  vi assumes a value of 1 for count.

[line]z
  vi places the line with the specified line number at the top, bottom, or center of the screen (see detailed description of the vi command z).

  line not specified:
  vi places the current line at the top, center, or bottom of the screen.

[column]
  Specifies the column to which you position the cursor.

The following vi commands can be followed by a movement command movecmd to indicate the range to be affected by the command:

c, d, y, <, >, !, and =.

The <, >, !, and = commands, which only process whole lines, affect all lines that are fully or partially included in the specified range.

The range processed by the c, d, and y commands extends from (and including) the current cursor position to the position specified by the movement command. It is thus significant whether the cursor position is set by line or by word. For example, if the cursor position is set by line, all lines (including the current line) are processed as complete lines. If the cursor position is set by word, the processed range extends till the position before or after the word, depending on whether the cursor was set to the beginning or end of the word.

[count]a
  (append) Switches to vi input mode, appending the entered text after the current cursor position. A count can be used to specify how many times the inserted text is to be replicated, but only if the inserted text is limited to one line.
vi, Commands

[count]A
(append) Switches to vi input mode, appending the entered text at the end of the current text line. If the inserted text is all on one line, a count can be used to specify how many times the inserted text is to be replicated, but only if the inserted text is limited to one line. This command is equivalent to $a.

[count]b
(back) Backs up to the beginning of a previous word, where count specifies how many words to back up.

[count]B
(back) Backs up to the beginning of a previous "extended" word, where count specifies how many words to move.

[count]movecmd
(change) Overwrites, i.e. changes a range of text. This command must be followed by a movement command.
- If the position specified by movecmd is in the current line, this position is marked by a $ character, and the editor enters the vi input mode to replace the range between the cursor and the $ by the entered replacement text.
- If the position specified by movecmd is in a new text line, the behavior of c depends on whether the scope of the change is defined by line or not. If the scope is defined by line, all fully or partially affected lines are deleted. However, if movecmd defines the scope by word, only the affected range is deleted. In both cases, vi switches to input mode to replace the deleted text by the new text entered. vi saves the deleted text in the temporary buffer, which means that it can be recovered with P or "lp, for example.

count
A count can be passed through to the movement command.

[count]cc
The command countcc causes the whole of the current line (or count lines) to be changed, regardless of the cursor position in the current line.

[count]C
(change) Changes the rest of the current line. This command is equivalent to count$.
**vi, Commands**

[count]dmovecmd
[count]dd
   (delete) Deletes a range of text.
   This command must be followed by a movement command. *d* deletes the range from
   the current cursor position to the end of the indicated range. If more than part of a single
   line is affected, the deleted text is saved in the numbered buffers 1 - 9, which means
   that it can be recovered with *p*.
   count   A count can be passed through to the movement command which indicates
           the range.
   d       The *countdd* deletes the whole of the current line (or *count* lines).

"buffer[count]dmovecmd
"buffer[count]dd
   Deletes and stores in named buffers.
   The text from the current cursor position to the end of the indicated range is deleted
   from the file and stored in a named buffer. *buffer* is a single uppercase or lowercase
   letter.

D   (delete) Deletes the rest of the current line. This command is equivalent to *d$*.
[count]e
   (end) Moves the cursor forward to the end of the *count*th word.
[count]E
   (end) Moves the cursor forward to the end of the *count*th extended word.
[count]fchar
   (find) *f* must be followed by a single character. *vi* scans the rest of the current line (from
   left to right) for that character and moves the cursor to it if found. A *count* is equivalent
   to repeating the search that many times.
   A semicolon repeats the search in the same direction, while a comma repeats it in the
   reverse direction.
[count]Fchar
   (find) *F* must be followed by a single character. *vi* scans backward in the current line (i.e.
   from right to left) for that character and moves the cursor to it if found. A *count* is equiva-
   lent to repeating the search that many times.
   A semicolon repeats the search in the same direction, while a comma repeats it in the
   reverse direction.
[line]G
   (go to) *vi* moves the cursor to the beginning of the line specified by line number *line*.
   *line* not specified:
   The cursor position is set to the last line of the file.
vi, Commands

[count]h
(home) Moves the cursor by count characters to the left, but only in the current line.

[count]H
(home) Moves the cursor to the top line of the screen. If a count is given, the cursor is moved to that line on the screen, counting from the top. In both cases, the cursor is placed on the first non-whitespace character on the line.

[count]i
(insert) Switches to vi input mode, inserting the entered text before the current cursor position. If the new text is limited to one line, a count can be specified to indicate how many times the text is to be replicated.

[count]I
(insert) Switches to vi input mode, inserting the entered text before the first character of the current line. If specified, a count repeats the effect (see above).

[count]j
Moves the cursor down by count text lines in the same column.
This command is equivalent to [CTRL]j and [CTRL]n.

[count]J
Joins the current line with the next one, supplying appropriate white space: one space between words, two spaces after a period, and no spaces at all if the first character of the next line is a right parenthesis ). A count can be used to specify the number of lines to be joined.

[count]k
Moves the cursor up by count text lines in the same column.
This command is equivalent to [CTRL]P.

[count]l
Moves the cursor by count characters to the right, but only in the current line.
This command is equivalent to space.

[count]L
(last) Moves the cursor to the first non-whitespace character of the last text line on the screen. If specified, a count moves to that line counting from the bottom.

Example

d3L deletes all lines from (and including) the current line up to (and including) the third text line from the bottom.
**vi, Commands**

**m**marker
(mark) Marks the current cursor position. *m* must be followed by a *marker*, which can be any single lowercase letter.

**Addressing**

`marker` A backquote moves the cursor to the exact position of *marker*.

`'marker` A single quote moves the cursor to the first character in the line containing *marker*.

**M** (middle) Moves the cursor to the middle line on the screen, at the first non-whitespace position on the line.

**n** Repeats the last / or ? scanning command.

**N** Repeats the last / or ? scanning command, but in reverse direction.

**o** (open) Switches to *vi* input mode and opens (i.e. inserts) a line below the current line.

**O** (open) Switches to *vi* input mode and opens a new line above the current line.

**p** Fetch text from temporary buffer

(put) Writes text from the temporary buffer after the cursor. If the text was saved by *vi* commands as a whole line (e.g. *dd*, *y*), it is inserted in a line below the current cursor position.

"bufferp
(put) Fetch text from numbered or named buffer

You can specify the *buffer* as:

- a digit from 1-9 to retrieve text from the buffer with the corresponding number
- a letter, to retrieve text from the named buffer with the corresponding name

**P**

"bufferP
(put) Similar to *p*, but putting the text before (or above) the current cursor position.

**Q** (quit) Quits *vi* mode and enters *ex* command mode. You must enter the *vi* [Esc] command to return to the *vi* command mode.

[count]rchar
(replace) Must be followed by a single character *char*; the character under the cursor is replaced by the specified one. If a count is specified, the next *count* characters are replaced with the single character given.

**R** (replace) Replaces several characters; *vi* enters the replace (or typeover) mode at the current cursor position. The characters on the screen are overwritten by the characters entered, until the input is terminated with [Esc].
vi, Commands

[count]s
(substitute) Deletes the single character under the cursor and enters vi input mode; the entered text replaces the deleted character. A count specifies how many characters from the current line are changed. The last character to be changed is marked with a $, as in c.

Example
5sblah blah blah[ESC]
replaces the character under the cursor and the next four characters to its right by the string blah blah blah. The same effect can be achieved with:
c5sblah blah blah[ESC]

[count]S
(substitute) Switches to vi command mode. Whole lines are substituted (same as cc). A count can be used to indicate how many lines to replace.

[count]t
(to) t must be followed by a single character. vi scans forward through the remainder of the current line for the specified character. If it finds the character, it places the cursor on the character which precedes it. A count is equivalent to searching for the count occurrence of the character. Using , or ; to repeat the search is not meaningful.

[count]Tchar
(to) T must be followed by a single character. vi scans backwards through the current line for the specified character. If it finds the character, it places the cursor on the character which comes after it. A count is equivalent to searching for the count occurrence of the character. Using , or ; to repeat the search is not meaningful.

u (undo) Reverses the last change made to the temporary buffer. If repeated, will alternate between these two states, thus is its own inverse.

Example
d3w deletes three words from the current cursor position onward
u puts back the three words
u deletes the three words again.

The contents of the temporary buffer can be accessed by using the p and P commands. If u is used after an insert command that inserted at least one new text line, the text deleted with u is saved in numbered buffer 1.

U (undo) Restores the current line to its state before the cursor was last moved to it. U only functions as long as you have not left the current line.
vi, Commands

[count]w
(word) Moves forward to the beginning of the next word; a count specifies how many words to move.

[count]W
(word) Moves forward to the beginning of the next extended word; a count specifies how many words to move.

[count]x
Deletes the character at the current cursor position. With a count, deletes count characters forward from the current cursor position, but only on the current line.

[count]X
Deletes the character before the cursor. A count, if specified, deletes that many characters before the cursor on the current line.

[count]ymovecmd
[count]yy
Saves text in the temporary buffer
(yank) This command must be followed by a movement command to indicate the range. The specified text range is copied (yanked) into the temporary temporary buffer.

Examples
yw yanks the text from the cursor position to the end of the next word into the temporary buffer.
4yw yanks 12 words into the temporary buffer.
4yy yanks the current text line and the three lines below it into the temporary buffer.

"buffer[count]ymovecmd
"buffer[count]yy
Yank text into named buffers
(yank) If the y command is preceded by the name of an named buffer ("buffer), the text will also be copied into this buffer. buffer must be specified as an uppercase or lowercase letter.

Examples
"a4yy yank the current text line and the three lines below it into named buffer a.
"ap put back the contents of a after the current cursor position.
vi, Commands

[count]Y
Yank line(s) into temporary buffer
(yank) Yanks a copy of the current line or count lines into the temporary buffer (same as yy).

"buffer[count]Y
Yank line(s) into named buffers
(Y - yank) Yanks a copy of the current line (or count lines) into the named buffer given in buffer, where buffer is an uppercase or lowercase letter. This format corresponds to "bufferyy.

[line]z[count]char
/pattern/z[count]char
Moves the text line specified by line or pattern to the screen position defined by char.
char must follow z and may assume the following values:
  † or plus sign (+)
     Specifies the top of the screen
dot (.)    Specifies the center of the screen
minus sign (-)
     Specifies the bottom of the screen

line indicates the line number of the text line to be positioned.
line not specified:
The current line is used by default.
pattern identifies the line in which pattern first occurs.
count is an integer between 1 and 23 that may be given after z to specify the lines to be redrawn.
count not specified: count = 23.

ZZ  Exits the vi editor, writing out the buffer contents to the file with the current file name
(see ex command, f) if the buffer was changed since the last save (with the ex command w). This command is equivalent to the ex command x.

[count]space
The cursor is moved by count characters to the right (stops at the end of the line).
This command is equivalent to l.
vi, Commands

[count]movecmd

An ! with a movement command allows you to switch to the vi last line mode to process text lines with a POSIX command. The ! must be followed by a movement command to indicate the scope, which must go beyond the current line. The specified line range (from the current position to the one given by the movement command) is then passed as the standard input of the POSIX command and is replaced by the output (standard output and standard error) of the system command. Lines are always passed as complete text lines.

The cursor does not move to the status line until the movement command has been entered. The POSIX command to be executed must be terminated with the Ú key.

movecmd
Movement command to a particular line. Here are some examples:

3j move down 3 lines
lineG go to the line number given in line
L move to the last screen line.

A second ! can also be specified for movecmd; this passes the current line as standard input to the POSIX command. The second exclamation mark can be preceded by a count to indicate the number of lines to be passed to the POSIX command, starting with the current line.

count
Serves as a repetition factor.

Examples

!!wc Ú
Passes the current line to wc as input and replaces it by the output of wc.

!1Gsort Ú
Sorts the range from the current line to the start of the file.

3!$tr a b Ú
Replaces all occurrences of a by b in the entire current line and the two following lines.

[count]$ Moves the cursor to the end of the current line. A count specifies the number of lines to go forward (e.g. 2$ moves the cursor to the end of the next line).

" Precedes the name of a named or numbered buffer. See the vi commands d, p, and y.

% Moves the cursor to the parenthesis, brace, or bracket that matches the one at the current position. Left and right parentheses ( ), square brackets [ ], and braces { } represent matching pairs.
&    Repeats the last ex command s; is equivalent to the ex command \&.

'marker
(single quote)
Move cursor to marked line

'marker moves the cursor to first non-whitespace character in the marked line. marker is a lowercase letter that was set as a marker with the vi command m.

If the ' command is used as a movement command for some other command, all lines in the specified range are treated as whole lines, including the current and the marked line.

Example

d'a
Deletes all lines from the current line up to the marked line, including both extremes.

'' (two single quotes)
Move cursor to previous current line

The '' command moves the cursor to the beginning of the line in which it was located prior to the execution of one of the vi commands below.

H, L, M, n, N, %, ', (, ), [, ], ', {, }.

/pattern #]
?qpattern #]

If the '' command is used as a movement command for some other command, all lines in the specified range are treated as whole lines, including the current and the marked line.

Example

d''
Deletes the range from the current line to the previous current line. You should first use '' to verify the location of the previous current line. If none exists, vi deletes the range from the current line to the beginning of the file. The deleted text can be retrieved with u.

[count](
Moves the cursor back to the beginning of a sentence; a count moves back that many sentences.

If the ex option lisp is set, moves to the beginning of a lisp s-expression (see section “ex options” on page 345). Sentences also begin at paragraph and section boundaries (see { and [}).
vi, Commands

[count])
Moves forward to the beginning of the countth sentence.

[count]+
Moves the cursor to the first non-whitespace character in the next line. Use a count to move more than a line (as in CTRL m).

[count],
Reverses the last vi search command (for a single character):
  f, F, t, or T,
searching in the opposite direction in the current line. A count can be used as a repetition factor.

[count]-
(hyphen) Moves the cursor backward to the first non-whitespace character in the preceding line. A count specifies how many lines to move back.

[count].
(dot) Repeats count times the last vi command which changed the buffer.

If the last command given was used to output the contents of a numbered buffer, a following dot command retrieves the contents of the next numbered buffer in sequence.

/pattern [a]
The / immediately takes you to the command line (vi last line mode) where you can specify a regular expression for pattern. The search begins when you terminate pattern with a. vi then scans forward for the next occurrence of a matching string and moves the cursor to it. If no string matching pattern is found, the cursor remains in the current position. The interrupt signal SIGINT can be given to terminate the search.

Since the ex option wrapscan (abbreviated to ws) is set by default, the search wraps around at the end of the file and continues downward from the start of the file to the current cursor position. The ex command setNOWS causes the search to terminate at the end of the file (see the ex command set and the section on "ex options" under ex).

The vi commands n and N repeat the search in the same and reverse directions, respectively.

If the / is used as a movement command with some other command to specify an extent of text, the defined region is from (and including) the current cursor position to just before the matched string.
Example

If a line contains the following text:
Larry Speakes, the Speaker of the House, spoke well.
And the cursor is positioned at the comma after Speakes, entering the command:
\texttt{d/ spoke} \[\text{\texttt{d}}\]
will leave:
Larry Speakes spoke well.

? Scans backwards; the reverse of / (see /).

0 Moves the cursor to the first character on the current line. The zero is not interpreted as a \texttt{vi} command when preceded by a non-zero digit.

: The \texttt{vi} command colon (:\texttt{)} switches to \texttt{vi} last line mode, allowing you to enter an \texttt{ex} command. The \texttt{ex} command is terminated by entering \[\text{\texttt{}}\].
If the \texttt{ex} command you wish to enter is longer than one input line, you will need to switch to the \texttt{ex} command mode with \texttt{Q}.

[count];
Repeats the previous \texttt{vi} search command (for a single character):
\texttt{f}, \texttt{F}, \texttt{i}, or \texttt{T}.
A \texttt{count} can be used for a number of repetitions.

[count]<movecmd
Must be followed by a movement command to another line; shifts the inclusive range from the current to the specified line by \texttt{shiftwidth} columns to the left. See section "\texttt{ex} options" on page 345. A \texttt{count} is passed through to the movement command.

A second < character can also be specified as \texttt{movecmd}; << causes the current line to be shifted to the left (or \texttt{count} lines, including the current line).

[count]=movecmd
Must be followed by a movement command to another line. If the \texttt{ex} option \texttt{lisp} is set, the specified lines are re-indented as if they had been typed in with the \texttt{autoindent} option set. A \texttt{count} can be used to specify how many lines are to be processed. \texttt{movecmd} may also be a second equals sign, which causes only the current line (or \texttt{count} lines) to be indented.

[count]>movecmd
This \texttt{vi} command shifts specified lines by \texttt{shiftwidth} characters to the right (see <).
` marker
(backquote)

Move cursor to set marker

The ` marker is a lowercase letter which identifies a marker that was set with the `vi` command `m`. `marker places the cursor at the exact character position marked.

If the ` command is used as a movement command with some other command to specify its scope, the defined region extends from (and including) the current cursor position up to just before the previous current position.

Example

```
d ` a
```

Deletes the range from the current cursor position up to the marker `a`.

```
` ` (double backquotes)

Move marker to previous current position

The ` ` command moves the cursor to the position it occupied before one of the `vi` commands listed below was executed.


If the ` ` command is used as a movement command with some other command to specify its scope, the defined region extends from (and including) the current cursor position up to just before the previous current position.

[count][[}

Backs up to the section boundary given by `count`.

[count][]

Moves forward to the section boundary given by `count`.

^ Moves the cursor to the first non-whitespace character on the current line.

[count]{}

Moves back to the beginning of the current paragraph. You can use a `count` to indicate how many paragraphs to move.

[count]]

Moves forward to the beginning of the next paragraph. A `count` can be used to indicate how many paragraphs to move.
Moves the cursor to the specified column, if possible.

*column* not specified: $column = 1$

~ Case shifting; converts the character at the current cursor position from uppercase to lowercase or vice versa as appropriate, and moves the cursor one character to the right.

**Adapting vi to the terminal**

The output of the *vi* program depends on the type of terminal used. The *vi* editor need not be adapted in the POSIX environment.

*vi* uses the value of the environment variable `TERM` to locate the required terminal information in a database (`/usr/lib/terminfo/*/*`). If `TERM` has not been defined or if you are working at some other type of terminal, you must reset the value of `TERM`. You can do this in a number of ways:

– set and export the shell variable `TERM`.
– define and export `TERM` in your `.profile`.
– use the *ex* command `set term` (in *ex* command mode only).
– enter the *ex* command `set term` in the `.exrc` file.
– define and export the `EXINIT` environment variable in your `.profile`.
– if all else fails, consult your system administrator.

⚠️ **Caution!**

Make sure you only specify the actual name of your terminal for `TERM`. The use of other names may cause your terminal to malfunction.

**Customizing the vi environment**

Default values are set using *ex* options. The available options are described under “*ex options*” on page 345.

The *ex* command `set all` displays a list of all the current values. For example:

- `directory=/tmp` Directory for the editor buffer
- `nonumber` Line numbers are not displayed
- `report=5` A message is issued when more than 5 lines are changed
- `scroll=11` The screen is scrolled by 11 lines when `CTRL` D is entered
- `noshowmode` The *vi* mode is not displayed
vi, Customizing vi

term=97801  Terminal type
window=23   Number of screen lines for the text window
wrapmargin=0 No automatic wraparound.

There are two types of option, those with Boolean values and those with non-Boolean values. showmode is an example of a Boolean type option. If it is set, set all displays the string showmode. If it is not set, set all displays noshowmode. With non-Boolean types, set all displays the option name and the value to which it is set. scroll is an example of a non-Boolean option.

If you have trouble with the arrow keys in vi (for example, if pressing an arrow key switches you from input to command mode), you should switch off the timeout option (set notimeout).

The vi environment can be customized to suit your requirements and working habits. For example, vi can be made to display line numbers of the current mode of operation in the status line.

There are three ways of changing the default values for vi:
– during a vi session
– by using the EXINIT environment variable
– by using the .exrc file

Let us assume, for example, that you want to have the line numbers and the vi mode displayed.

– During a vi session:
  To display the line numbers, use the ex command set number during a vi session. The ex command set showmode causes vi to indicate when it is in input mode. The customized environment will only be effective for the duration of the session.

– By using the EXINIT environment variable:
  You must write the following lines into the .profile file in your HOME directory:
  EXINIT='set number showmode'
  export EXINIT
  Using the command .profile you make known the new variable to your shell (see . (dot) command). If you now call vi, line numbers will be displayed, and vi will inform you when you are in the input mode. This default setting is always effective.

– By using the .exrc file:
  You must create this file yourself. The .exrc file can be used to define permanent vi settings. To ensure that vi displays line numbers and the mode, you must enter the following line into this file:
  set number showmode

  The .exrc file has to be in your HOME directory. For security reasons, vi does not evaluate .exrc files in subdirectories. The .exrc in the HOME directory is always read.
The default values defined for vi are also valid for ex.

It is possible to combine the various methods, but on certain conditions. Values set within a vi session have the highest priority. You may therefore change the current values at any time during a session by means of the ex command set. If EXINIT is set, the $HOME/.exrc file is not executed.

File

$HOME/exrc
File containing default values for ex and vi. These default values are overridden by values set using EXINIT or during a vi session.

Variable

COLUMNS
Number of columns per screen.

LINES
Number of lines per screen.

SHELL
Determine the preferred command-line interpreter for use in !, shell, read and other commands with an operand of the form !string. For the shell command the program will be invoked with the single argument -i, for all others it will be invoked with the two arguments -c and string. If this variable is null or not set, sh will be used.

TERM
The type of terminal used.

EXINIT
Environment variable with default values for ex and vi.

PATH
Determine the search path for the shell command specified in the editor commands shell, read and write and the visual-mode command !

Locale

The following environment variables affect the execution of vi:

LANG
Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL
If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements within regular expressions.
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

Determine the location of message catalogs for the processing of `LC_MESSAGES`.

See also `edit`, `ex`

wait await process completion

The POSIX shell built-in wait causes the shell to wait
– until a previously started background process is complete (if you call wait with the
  required process ID), or
– until all previously started background processes terminate (if you call wait without
  arguments).

When a command is run in the background (with &), the interactive shell does not normally
wait for it to finish before prompting you for a new command. By contrast, if a command is
started as a foreground process, the shell will always wait for it.

This also applies with respect to shell scripts. Thus, if a script contains a command that is
to process the output of a previously started background command, wait can be used to
ensure that the background command terminates first.

If the error message fork failed - too many processes appears when you try to run a
process in the background, you may be able to lessen the system load by running wait. If
this has no effect, it may be that the system table is full or that there are too many active
foreground processes.

Hint

If a pipeline has 3 or more stages, the shell groups processes into pairs and spawns a
process to control each pair. Hence the original processes are not child processes of the
shell. wait cannot wait for processes which are not children of the shell.

Syntax

\begin{verbatim}
wait[,...process_ID,...]
\end{verbatim}

process_ID
Process identification number of the background process whose completion is to be
waited for by the shell.

Only one process ID may be specified. If more than one process ID is given in the
command line, wait will only consider the first one and will ignore the rest without issuing
an error message.

The exit status returned by wait is that of the specified background process. If the
process you specify is not active, wait behaves as it does if you do not specify a PID
(see below).

The shell always assigns the process ID of the last command run in the background to
the $! (exclamation point) variable. The contents of this variable can be accessed with $!

process_ID not specified:
wait waits for all your shell’s currently active background processes to complete and
returns an exit status of 0.
**wait**

**Exit status**

If you have called `wait` without an argument and all known process IDs have terminated, the exit status is always 0.

- **1-126** Error
- **127** The command for the last process ID is unknown.

**Locale**

The following environment variables affect the execution of `wait`:

- `LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `LC_ALL` If set to a non-empty string value, override the values of all the other internationalization variables.

- `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- `LC_MESSAGES` Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

**Example 1**

The shell is to wait for an active background process:

```bash
$ find / -name tst -print 2>/dev/null &
3456
$ wait 3456
home/martin/PASCAL/tst:
```
Example 2  The shell script *patience* illustrates how to wait for a specific background command to terminate. The contents of this shell script are as follows:

```
: Invocation with sh patience
sort list > sorted &
pid1=$!
tar cvf /dev/dsk/f0t /home/anne &
pid2=$!
:
wait $pid1
pg sorted
```

The process IDs of the two background commands are stored in the variables `pid1` and `pid2`, respectively. These process IDs are thus available for future use, as the `!` variable always holds the process ID of the last background command started.

The *wait* command causes the script to wait for the `sort` command to complete before displaying the contents of `sorted` with the `pg` command.

See also  *waitpid()* [4]
**wc**

**word, line and byte or character count**

*wc* counts the number of lines, words and characters contained in files and writes the results to standard output.

**Syntax**

```bash
wc[-cmd][lw][file...]
```

No option specified

*wc* outputs three numeric values for the number of lines, words, and characters.

**Option**

-**c**  *wc* outputs the number of bytes. Spaces, tabs and newline characters are counted. The option `-c` acts exactly like the option `-m` since one byte corresponds to one character.

-**m**  *wc* outputs the number of characters. Spaces, tabs and newline characters are counted. The option `-m` acts exactly like the option `-c` since one character corresponds to one byte.

-**l**  Reports the number of lines, based on the number of newline characters counted by *wc*.

-**w**  Reports the number of words. A word is defined as a non-empty string delimited by whitespace characters. Whitespace characters are blanks, tabs and newline characters.

**File**

Name of the file whose lines, words, and characters are to be counted. The file name will be printed along with the counts.

More than one file may be specified. If several files are named, *wc* prints an additional line with a grand total of the individual values for all specified files.

**file** not specified:

*wc* reads from standard input.

**Locale**

The following environment variables affect the execution of *wc*:

- **LANG**  Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- **LC_ALL**  If set to a non-empty string value, override the values of all the other internationalization variables.

- **LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments). *LC_CTYPE* defines which characters are treated as whitespace characters.
**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

**NLSPATH**
Determine the location of message catalogs for the processing of _LC_MESSAGES_.

**Example 1**
Listing line, word, and character counts for the files _logic_, _plan_ and _rem_.

```bash
$ wc logic plan rem
 27  139  1077 logic
  5   15   140 plan
  3    6    51  rem
 35  160  1268 total
```

**Example 2**
Checking the number of files in the current directory.

```bash
$ ls | wc -l
 31
```

**Example 3**
Checking the number of users working on the system.

```bash
$ who | wc -l
   6
```

**Example 4**
Counting the number of unique words in a file.

```bash
$ cat file | sed 's/ \*\*\*/\n> /g' | sort -u | wc -l
```

Explanation:

*sed* creates a list of all words in _file_ by replacing one or more blanks by newline characters. *sort -u* sorts this list, removing all repetitions. *wc -l* then counts the lines in this list and prints the total.
The built-in `whence` command in the POSIX shell `sh` specifies how each name is to be specified as a command name.

### Syntax

```
whence[\[-pv\]..name.....
```

- `-p` produces a more detailed report.
- `-v` does a path search for `name` even if `name` is an alias, a function, or a reserved word.

**name**

Name of the command to be interpreted.

### Exit status

- `0` The command was executed correctly.
- `>0` No command named `name` was found.

### Example

Various `whence` output possibilities

```
$ whence echo
  echo
$ whence -pv echo
  echo is /usr/bin/echo
$ whence -v echo
  echo is a shell builtin
$ whence -v cat
  cat is a tracked alias for /usr/bin/cat
```

### See also

`type`
**who**

**display who is on the system**

*who* provides information on:
- the login and terminal name under which you are currently working
- the login name, login time, and terminal name for each current system user
- the process ID of the command interpreter (shell) being used
- the last time a terminal was used
- when logins, logouts, and system breakdowns have taken place since the
  /var/adm/wtmp file was cleared by the system administrator.
- the last time the system clock was changed
- the processes spawned by the *init* process

**Hint**

*who* normally gets its information from the /var/adm/utmp file. The information in this file is brought up to date every time a login is performed. In single-user mode no logins are performed. Thus after a shutdown to single-user mode *who* may not be able to supply accurate information about the current login status. Use *who am i* instead.

**Syntax**

**Format 1:** `who[...-mu][-s[-bHlprt][...file]]`

**Format 2:** `who[...-mTu][-abdHlprt][...file]`

**Format 3:** `who...q[...n_number][...file]`

**Format 4:** `who...am...i`

**Format 5:** `who...am...l`

**List detailed information**

**Format 1**

`who[...-mu][-s[-bHlprt][...file]]`

**Format 2**

`who[...-mTu][-abdHlprt][...file]`

No option specified

*who* lists the following for every system user currently logged in:
- login name under which the user logged in
- name of the terminal on which the user logged in
- login time.

The columns in the output are described below in *Output*.

**options**

- `-a` (all) Turns on options `-b`, `-d`, `-l`, `-p`, `-r`, `-t`, `-T` and `-u`.
- `-b` (boot) Shows the date and time of the last reboot.
who

- **d** (dead) Shows all processes that have terminated and not been respawned by *init*. The terminated processes are output together with their exit status (*EXIT* field) and the number of the signal which terminated the process. This may be useful in determining why a process died.

- **H** (headings) Prints headings above the output columns.

- **l** (login) Lists the processes where the system is waiting for someone to log in. In such cases the entry in the *NAME* field is the name of the program (or *LOGIN*) and the *STATE* field does not appear. The other fields have their usual meanings.

- **m** who only outputs information on the active terminal.

- **p** (process) Shows all processes spawned by the *init* process. The *NAME* field shows the name of the program executed by *init*. The *LINE* field is irrelevant in conjunction with the -p option, so only a dot is output in this field.

- **r** (run level) Shows the current run level of the *init* process. The *LINE* field indicates the current run level, the *TIME* field the date it was entered. *IDLE* shows the current run level in numeric form, *PID* shows how often the system has been in this state before, and *COMMENT* indicates the system's previous run level. The *NAME* field has no meaning with the -r option.

- **s** (standard) Lists the users who are currently logged in. The *NAME* field gives the login name, *LINE* is the device name (without the /dev/) of the terminal on which the user logged in. *TIME* shows when the user logged in. This option is the default. -s can not be combined with -a, -d or -T.

- **T** (terminal) The terminal state is output in addition to the default values. The *STATE* field shows whether other users can write to the terminal: a plus sign + indicates that they can, a minus sign – means that they cannot. The POSIX administrator can write to any terminal. If it is not possible for the system to obtain this information, a query ? appears in this field.

- **t** (time) Shows the last time the system administrator changed the system clock (using the date command).

- **u** (user) Lists those users who are currently logged in. The *NAME* field is the user's login name. *LINE* is the device name (without the /dev/) of the terminal on which the user logged in. *TIME* shows when the user logged in. The *IDLE* field provides information on the last time there was activity on the terminal in question: a dot (.) indicates activity within the last minute. If the terminal has not been used for over 24 hours or since the system was last booted, the entry is marked *old*. The *PID* field indicates the process ID of the command interpreter (shell) the user is working with.
who

Name of the file from which who obtains its information. file, if given, will typically be /var/adm/wtmp. who then provides information about logins, logouts and system breakdowns that have taken place since this file was last cleared by the system administrator.

file not specified:
who obtains its information from the file /var/adm/utmp.

Output
The following section lists the headings of the columns output by who and discusses the information displayed in each column. The special information produced by the -b, -d, -p, -r and -t options is explained above in the detailed option description.

NAME
The NAME field indicates the user's login name.

STATE
The STATE indicates whether other users can write to the terminal: a plus sign + indicates that they can, a minus sign – means that they cannot. The system administrator can write to any terminal. A query in this field indicates that the line is defective.

LINE
The LINE field gives the device name (without the /dev/) of the terminal on which the user logged in.

TIME
The TIME field shows when the user logged in.

IDLE
The IDLE field provides information on the last time there was activity on the terminal in question: a dot (.) indicates activity within the last minute. If the terminal has not been used for over 24 hours or since the system was last booted, the entry is marked old.

PID
The PID field indicates the process ID of the command interpreter (shell) the user is working with.

COMMENT
The COMMENT field indicates the system's previous run level.

EXIT
The EXIT column (exit status) is a list of dead processes together with the number of the signal which terminated each process. This may be useful in determining why a process died.

The NAME, LINE and TIME information is produced by options -a, -l, -s, -T and -u. STATE is shown only by -T, IDLE, PID by -a, -l and -u. The -a option also provides information on the exit status (EXIT).
who

Format 3  **List concise information**

     who -q [ -n number][ -f file]

- **q** (quick) who only outputs the names and number of users currently logged on. Additional options, with the exception of **-n**, are ignored.

- **n number**
  who displays number users per line. number must be at least 1.

- **-n number** not specified:
  by default who displays users per line.

- **f file**
  see Format 1

**List information on the invoking user**

Format 4  **who .am .i**

Format 5  **who .am .I**

The formats 4 and 5 are identical to **who -m**.

who lists:
- the login name under which you logged in
- the device name (without the /dev/) of the terminal on which you logged in
- the login time.

**File**

/var/adm/utmp
File from which who obtains its information by default. This file records the latest status.

/var/adm/wtmp
This file can be used as an alternative to /var/adm/utmp. This file is regularly cleared by the system administrator.

**Locale**

The following environment variables affect the execution of **who**:

**LANG**
Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.
who

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**LC_TIME**  Determine the format and contents of date and time strings.

**NLSPATH**  Determine the location of message catalogs for the processing of **LC_MESSAGES**.

Example 1  Listing general information on all users currently logged in by calling `who` without options.

```
$ who
QM212JNA   term/003     Mon Aug 18 17:24:54 MSZ 2008
```

Example 2  Listing detailed information with column headings on all users currently logged in by calling `who` with the `-H` and `-u` options.

```
$ who -Hu
NAME       LINE         TIME          IDLE    PID
CMPQD004   tty001       Mar  7 10:53   .     6252
QM212JNA   tty002       Mar  7 11:10  0:22   6289
```

Example 3  Listing concise information on all users currently logged in by calling `who` with the `-q` option.

```
$ who -q
CMPQD004   QM212JNA
# users=2
```

Example 4  Listing information on the invoking user with `who am i`.

```
$ who am i
QM212JNA   term/003     Mon Aug 18 17:24:54 MSZ 2008
```

See also  `date`, `mesg` `wait()` [4]
write

write to another user

To receive messages you must be working on a character-mode terminal; in other words, you must be accessing the POSIX shell via `rlogin`.

In contrast, all users can send messages, including those working on block-mode terminals.

`write` is used to send messages to another user. `write` reads one line at a time from standard input and echoes the lines read as messages on the specified user’s terminal. When `write` is invoked, a message header initially appears on the recipient’s terminal indicating the name of the sender, the terminal, and the time at which the message is sent. This is followed by the actual messages.

Users may communicate interactively by sending messages to each other with `write` (see “Two-way conversations” on page 855).

Before the call

Users who have denied other users permission to write to their terminals (with `mesg -n`) or who are working on block-mode terminals cannot be written to with `write`. A user with POSIX administrator privileges can write to any terminal.

Syntax

```
write recipient[...ttyname]

  text
  ...
  ...

  [CTRL][D]
```

`recipient`
Login name of a user logged in to a terminal. You can also send yourself messages. If a user is logged in to more than one terminal at the same time, the appropriate terminal can be specified as well.

The command `who` lists all currently logged-in users and their terminals.

`ttyname`
Number of the terminal on which the recipient is logged in.

`ttyname` not specified:

`write` searches the file `/var/adm/utmp` for the terminal. If several entries exist because a user is logged in more than once, `write` uses the first entry recorded in the file. The following message is displayed:

```
user is logged on more than one place.
You are connected to "ttyname".
Other locations are:
  ttyname
```
write

text
Text to be sent as a message. write reads from standard input one line at a time up to an end-of-file signal:

- A line that begins with an exclamation mark (!) is interpreted as a command; write passes everything that follows the ! in the line to the shell. The command is then executed, while write remains in an active state. Any output that the command writes to standard output will be included in the text to be sent as a message.
- All other normal lines are sent to the recipient as a message.
- When write encounters the end-of-transmission character ("EOT \n"), it displays this character on the other terminal and then exits.
- Non-printing characters (with the exception of \d, \v, \n, \r and \a) are converted before they are sent. Control characters will appear as caret ^<uppercase-letter>. <uppercase-letter> is the letter which results when the 7th bit is set in the ASCII character set. For example, \003 is displayed as "^C".

Two-way conversations
Users may communicate interactively by sending messages to each other with write.
Conversation between two users proceeds as follows:

1. User A invokes write with the login name of user B. User B receives an announcement (message header) informing him or her that user A wishes to communicate something.

   Message from sender (terminal) [time]

   User A hears two bell tones to indicate that the connection has been set up and that user B is able to receive.

2. User B therefore calls write with the login name of user A.

   write sender [terminal]

   The message header received by user A serves as an acknowledgement.

3. Both users can now communicate with one another. Each user should end a message with a distinctive signal so that the other knows when to reply. There should also be a distinctive signal to indicate when the conversation is to be terminated.

4. The conversation can be terminated by pressing \[CTRL\] [D] or \[DEL\]. The command mesg -n can then be used if other users are to be denied permission to write further messages.
write

Error user is not logged on. or user is not at 'tty'
The addressed user has not logged in.
Permission denied.
The addressed user has denied other users permission to write to his/her terminal (see mesg) or his/her terminal is a blockterminal.
Warning: You have your terminal set to "mesg -n". No reply possible.
Your terminal cannot receive messages from other users (see mesg).
Warning: You are on a Block terminal. No reply possible.
Your terminal cannot receive messages from other users (see mesg).
Can no longer write to ttyname
The user of the other terminal has withdrawn write permission since transmission started (see mesg).

File /var/adm/utmp
The file in which all logged in users are registered.

File /usr/bin/sh
Command interpreter for lines beginning with !.

Exit status 0 Successful completion
>0 The addressed user is not logged on or the addressed user denies permission.

Locale The following environment variables affect the execution of write:

LANG Provide a default value for the internationalization variables that are unset or null. If LANG is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

LC_MESSAGES Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.
Example

Send a message to user karen:

```
$ write karen
!date
Mon Oct 13 19:00:13 MET 2008
Today, at 15.10,
I decided to quit
smoking!  [CTRL] [D]
```

See also  mails, msg, pr, sh, talk, who
**xargs** construct argument list(s) and execute command

*xargs* combines command-line arguments with arguments that it reads from standard input and executes the specified command one or more times. The number of arguments read for each command invocation and the manner in which they are combined can be determined by the options specified.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or newline characters; empty lines are discarded. Blanks and tabs may be embedded as part of an argument only if they are either escaped with a backslash \ or if the argument is enclosed in single or double quotes ('...' or "..."). Otherwise, they are interpreted as separators between arguments. The usual quoting mechanisms apply as well, i.e. metacharacters are also escaped as mentioned above.

**Syntax**

```bash
xargs[...option][...command][...initial_argument][...]
```

Options `-I` (or `-i`), `-L` (or `-l`) and `-n` determine how the command-line arguments and the arguments read from standard input are to be selected for each invocation of *command*.

If none of the `-I` (or `-i`), `-L` (or `-l`) or `-n` options are used:

The initial arguments specified in the *xargs* command line are followed by arguments read continuously from standard input until an internal buffer is full. *command* is then executed with the accumulated arguments. This process is repeated until there are no more arguments.

If combinations of the `-I` (or `-i`), `-L` (or `-l`) or `-n` options are used:

When there are conflicts (e.g. `-l` and `-n`), the last specified option takes effect.

**-L** repstr (or `-l[repstr]`)

Insert mode; *command* is executed for each line read from standard input, taking each line as an argument and inserting it for each occurrence of *repstr* in the list of initial arguments specified when *xargs* was called.

A maximum of 5 arguments in the list of initial arguments may each contain one or more instances of *repstr*. Blanks and tabs at the beginning of each line are ignored. Constructed arguments may not grow larger than 255 characters.

The `-L` option automatically also sets the `-x` option.

This corresponds to the old option `-i`, which is still supported. For *repstring*, braces `{}` are assumed in the case of `-i` if not explicitly specified.
xargs

-L number (or -l[number])

command is executed for every number non-empty argument lines that xargs reads from standard input. If fewer than number lines remain when command is invoked for the last time, command will be executed with these lines.

A line is considered to end with the first newline character in it unless the last character in the line is a blank or a tab. A trailing blank or tab signals continuation to the next non-empty line.

The -L option automatically also sets the -x option.

This corresponds to the old option -l, which is still supported. If number is not specified for -l, the value 1 is assumed.

-n number

Executes command using as many standard input arguments as possible (up to a maximum of number). Fewer arguments will be used if their total size is greater than maxsize characters (see -s option). If fewer than number arguments remain for the last invocation, these arguments will be used. If option -x is also set, each number arguments must not exceed the maximum limitation specified with maxsize (see option -s), as otherwise xargs will terminate execution.

-E eofstr (or -e[eofstr])

Defines eofstr as the logical end-of-file string. xargs reads from standard input until either the actual end-of-file or the logical eofstr is encountered.

Specify a string of eofstr. This string is interpreted as the logical end-of-file string when xargs is executed.

This corresponds to the old option -e, which is still supported. If -e is specified without eofstr, a logical end-of-file string is no longer specified. The underscore _ has no special meaning and is processed as a normal character.

Neither -E nor -e specified:
The underscore (_) is interpreted as the logical end-of-file.

-p Prompt mode; the user is asked whether to execute command at each invocation. In addition, the trace mode (option -t) is turned on to print the invoked command, followed by a ?... prompt.

-s maxsize

Sets the maximum number of characters in each argument list to maxsize (an argument list is a combination of arguments formed on the basis of rules defined by the -I (or -i), -L (or -l), or -n options). Note that the character count for maxsize includes one extra character for each argument and the count of characters in the command name.

-t Trace mode; the command and each constructed argument list are echoed to standard error just prior to their execution.
xargs

-x  Causes xargs to terminate if any argument list would exceed the specified maximum of maxsize characters (see option -s).

The -x option is forced when options -I (or -i) and -L (or -l) are set. When none of the -I (or -i), -L (or -l) or -n options are coded, the execution of xargs is terminated if the total length of all arguments exceeds maxsize characters (see option s).

command
Any command may be specified for command. xargs will terminate if it receives an exit status of 255 from command or cannot execute it. If command is a shell script, it should include an exit command which explicitly returns a suitable exit status so as to avoid accidentally returning with 255 (e.g. exit -1).

command not specified:
The command echo is assumed for command.

initial_argument
Argument lists are constructed (as described above) by combining initial arguments specified on the xargs command line with arguments read from the standard input (see options -I (or -i), -L (or -l) and -n). The specified command is then executed with these argument lists.

The initial arguments always appear at the beginning of an argument list, except when the -I or -i option is set.

initial_argument not specified:
Argument lists are only constructed from arguments read from the standard input.

Exit status
0      Each call of command returned the exit status 0.
1-125  The arguments could not be combined as requested, or at least one call of command returned a non-zero exit status or a further error occurred.
126    The specified command exists, but cannot be executed.
127    The specified command cannot be found.

Locale
The following environment variables affect the execution of xargs:

LANG   Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.
**xargs**

**LC_COLLATE**  Determine the locale for the behavior of ranges, equivalence classes and multicharacter collating elements used in extended regular expressions defined for the yes/no queries.

**LC_CTYPE**  Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments) and the behavior of character classes used in extended regular expressions defined for the yes/no queries.

**LC_MESSAGES**  Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**  Determine the location of message catalogs for the processing of **LC_MESSAGES**.

**Example 1**  The following shell script named *relocate* moves files with names that do not begin with a period (.) from one directory to another:

```
$ cat relocate
ls $1 | xargs -I {} -t mv $1/{} $2/{}
$ relocate dir1 dir2
mv dir1/file1 dir2/file1
mv dir1/file2 dir2/file2
mv dir1/file3 dir2/file3
```

The two positional parameters $1 and $2 are assigned the values of the arguments specified (dir1 and dir2) when *relocate* is invoked. The command $ls $1 displays the contents of the dir1 directory with one file name in each line. These file names are sequentially inserted for {} (option -I {}). Each mv command is echoed with its argument list (option -t) just before it is invoked.

**Example 2**  The following shell script named *who_and_when* collects the output of the parenthesized commands (...) in one line and appends this line to the end of the file log:

```
$ cat who_and_when
(logname; date; echo $0 $*) | xargs >>log
$ cat log
michael Mon Mar 9 14:21:06 MEZ 2009 who_and_when
```

The two positional parameters $1 and $2 are assigned the values of the arguments specified (dir1 and dir2) when *relocate* is invoked. The command $ls $1 displays the contents of the dir1 directory with one file name in each line. These file names are sequentially inserted for {} (option -I {}). Each mv command is echoed with its argument list (option -t) just before it is invoked.
Example 3  The following shell script named `archive` places files in the current directory whose names do not begin with a period (.) in an archive named `archive.a` (see `ar`):

```
$ cat archive
ls | xargs -p -L 1 ar r archive.a
$ archive
ar r archive.a file1 ?... y
ar: creating archive.a
ar r archive.a file2 ?... n
ar r archive.a file3 ?... y
ar r archive.a file4 ?... n
$ ar t archive.a
file1
file3
```

`ls` displays the contents of the current directory on standard output with one file name in each line. `xargs` then calls `ar` with the arguments `r, archive.a`, and each file name returned by `ls`. Since the `-p` option has been set, you are asked whether or not the corresponding `ar` command is to be executed in each case. If you respond with a "y" the first time, `ar` creates the archive file `archive.a`, issues a corresponding message, and saves the current file in it. Your subsequent responses to each prompt will then determine which files are to be stored in `archive.a`. If required, the contents of `archive.a` can be listed at the end with the `ar t` command.

See also  `echo`
**yacc**

yet another compiler-compiler

The command **yacc** (yet another compiler-compiler) converts decontextualized grammar into a set of tables for a simple automaton ###, which runs an LALR(1) syntax analysis algorithm. The grammar may be ambiguous. Certain rules are applied to resolve these ambiguities.

The C compiler must compile the output file `y.tab.c` (or `file_prefix.tab.c`, if option `-b` was specified) in order to create a program `yyparse`. This program must be linked to a lexical analysis program `yylex` as well as to `main` and `yyerror`, an error handling routine. These routines must be provided by the user. The command `lex` is used to create lexical analyzers which can be used by `yacc`.

You may specify the following options when calling **yacc**:

- **-d** Generates the file `y.tab.h` with the `#define` statements which associate the token codes assigned by `yacc` with the token names allocated by the user. This means that source files other than `y.tab.c` can access the token codes.

- **-l** Specifies that the code generated in `y.tab.c` (or `file_prefix.tab.c`, if option `-b` was specified) contains no `#line` statements. Use this option only if the grammar and associated actions have been fully tested and are error-free.

- **-t** By default this option compiles code to support runtime error detection. Although the error detection code is always generated in `y.tab.c` (or `file_prefix.tab.c`, if option `-b` was specified), it is subject to conditional compiler control. By default, this is not included when `y.tab.c` is compiled. Irrespective of whether or not the option `-t` is set, provision of the code for runtime error detection is controlled by the preprocessor symbol `YYDEBUG`. If `YYDEBUG` has a value other than zero the error detection code is linked. If the value is zero, the code is not linked. Programs created without error detection code are smaller and execute more quickly than programs with error detection code.

- **-v** Prepares the `y.output` (or `file_prefix.output`, if option `-b` was specified) file. This contains a description of the syntax analysis tables and a message about conflicts which have resulted from ambiguities in the grammar.

- **-b** `file_prefix` The output files begin with the prefix `file_prefix` instead of `y`. The other files required by `yacc`, such as `y.tab.c`, `y.tab.h` and `y.output` are then also assigned the file names `file_prefix.tab.c`, `file_prefix.tab.h` and `file_prefix.output`.

Syntax

```
yacc [-dltv][-b file_prefix][-p sym_prefix][-y driver_file][-V][-Q[y|n]][... file ...]
```
yacc

-p sym_prefix
All external names generated by yacc start with the name sym_prefix instead of yy. This also applies to the functions yyparse(), yylex() and yyerror() and the variables yylval, yychar and yydebug. Internal names may also be affected by the option -p. However, the -p option does not affect #define symbols which are generated by yacc.

-Q[y/n]
Defines whether version information concerning the generated yacc version is to be written to y.tab.c (or file_prefix.tab.c, if option -b was specified) (y) or not (n).

yn stands for a yes/no argument in whatever language environment is set. In an English-language environment you enter -Qy to have version information written to y.tab.c and -Qn to suppress the version information. In a German-language environment, for example, you would use -Qj or -Qn (for ja or nein).
By default, no version information is output.

-V Prints the version information for yacc to the standard error output.

-y driver_file
Defines a personal yaccpar file.

File
y.output, y.tab.c
y.tab.h Definitions of token names
(file_prefix. is used instead of y, if option -b was specified)
yacc.tmp, yacc.debug, yacc.acts Temporary files
/usr/lib/yaccpar Analysis algorithm prototype for C programs
/usr/ccs/lib/liby.a for linking the required modules

Exit status The number of reduce/reduce and read/reduce conflicts is reported at the standard error output; a detailed report can be found in the y.output (or file_prefix.output, if option -b was specified) file. A message is also output if certain rules cannot be accessed from the start symbol.

Hint As the file names are usually determined by default (i.e. option -b is not specified), no more than one yacc process can be active at any one time in a given directory.

If c89 [5] is used to link a yacc program, -ly must be specified as the library parameter.

Locale The following environment variables affect the execution of yacc:

LANG Provide a default value for the internationalization variables that are unset or null. If LANG is unset of null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.
yacc

**LC_ALL**
If set to a non-empty string value, override the values of all the other internationalization variables.

**LC_CTYPE**
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments and input files).

**LC_MESSAGES**
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH**
Determine the location of message catalogs for the processing of \texttt{LC_MESSAGES}.

See also \texttt{lex}
**zcat**

**expand and concatenate compressed data**

*zcat* takes files that have been compressed with *compress* and copies them to standard output in their original form. The compressed file remains compressed. Files compressed with *compress* can be restored to their original form with the *uncompress* command.

*zcat*, together with *compress* and *uncompress*, forms a group of commands used to compress and expand files and to display files in expanded form. *zcat* is identical to *uncompress -c*.

**Syntax**

```
zcat [...file...]
```

file

Name of the compressed file to be output in its original form. You can name any number of files.

You may enter the name of the file with or without the `.Z` extension; if you omit the extension, *zcat* automatically searches for the appropriate `.Z` file.

**Error**

The following errors result in the failure of the *zcat* command.

*filename*: no such file or directory

The file you specified does not exist.

*filename*: not in compressed format

The file you specified was not compressed using *compress*.

*filename*: compressed with xxbits, can only handle yybits

The file was compressed by a program which could handle more bits than the compression code on this machine. You may be able to recompress the file with a smaller number of bits.

uncompress: corrupt input

There has been a SIGSEGV signal (addressing error due to segmentation violation). This usually means that the input file is corrupted.

**Locale**

The following environment variables affect the execution of *zcat*:

*LANG*  
Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

*LC_ALL*  
If set to a non-empty string value, override the values of all the other internationalization variables.

*LC_CTYPE*  
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).
LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH

Determine the location of message catalogs for the processing of LC_MESSAGES.

Example

The file \texttt{zcat\_ex} is printed with \texttt{cat}, compressed with \texttt{compress}, and then output again with \texttt{zcat}.

\begin{verbatim}
$ cat zcat_ex
I have still not been compressed!

$ compress -fv zcat_ex
zcat_ex: Compression: -16.12\% -- replaced with zcat_ex.Z

$ zcat zcat_ex.Z
I have still not been compressed!
\end{verbatim}

See also \texttt{cat}, \texttt{compress}, \texttt{uncompress}
:: return true value

The POSIX shell built-in `:` (colon) returns zero exit status and does nothing else. It is used in shell scripts in the following ways:

- Without command-line arguments, it does exactly the same as the `true` command. This means that you can also create the condition `true` using the shell built-in `:`.
- If you include arguments, the shell interprets all metacharacters that appear in them. This method of using the shell built-in `:` enables you to assign a default value to free shell variables without initiating any other action.
- If you do not use any shell metacharacters in the command-line arguments, or if you escape them, the shell built-in `:` has the same function as the hash character `#`, i.e. it introduces comments. With `:`, though, unlike `#`, the next unescaped command separator terminates the comment.

Syntax

```
: [argument]...
```

argument
Any string delimited by blanks or tabs. The last argument must be terminated by a command separator.

You can specify any number of arguments, provided they are separated by at least one tab or blank.

As with every other command, the string is first interpreted by the shell (see section “sh shell, the standard command language interpreter” on page 679). The shell built-in `:` returns only a zero exit status.

argument not specified:
The shell built-in `:` returns only a zero exit status and does nothing else.

Exit status

0 in every case

Example 1
You can use the shell built-in `:` to fill in a branch of an `if` or `case` construct if you do not want anything to happen in that branch. Suppose the shell script `not_x` contains the following:

```
if test -x $1
then :
else echo $1 is not executable!
fi
```

This shell script tests the file that you name as your first command-line argument. If the file is executable, the shell script does nothing; otherwise, the message “file is not executable!” is issued.
Example 2  The following shell script assigns the process ID of the current shell process to the shell variable `name` if this variable is undefined or contains the null string:

```bash
: ${name:=$$}
echo $name
```

See also  `true`
execute commands in current environment

The POSIX shell built-in `. (dot)` executes the specified shell script in the current shell.

If you define new shell variables in the shell script or modify the values of existing shell variables, these variables take effect in the current shell environment.

If you set or unset shell options in the shell script with the shell built-in `set`, these options are set/unset in the current shell.

When you call the shell script, you can pass keyword parameters to it, but you cannot redefine the positional parameters. However, within the script you can access the positional parameters of the current shell.

If you use `set` to redefine the positional parameters within the script, the new values apply in the current shell.

**Syntax**

```
...file
```

**file**

Name of the shell script to be executed in the current shell. The shell searches for `file` in all directories whose path names have been assigned to the `PATH` variable.

Read permission is required for the specified file.

**Variable**

`PATH`

Search path of the shell

**Locale**

The following environment variables affect the execution of `.:

- `LANG` Provide a default value for the internationalization variables that are unset or null. If `LANG` is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility will behave as if none of the variables had been defined.

- `LC_ALL` If set to a non-empty string value, override the values of all the other internationalization variables.

- `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single- as opposed to multi-byte characters in arguments).

- `LC_MESSAGES` Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

- `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.


Built-in sh command

Example When you create or modify your $HOME/.profile file, the commands and assignments contained in it are normally not executed until you next log in (open a login shell). The dot command enables you to make these changes effective in the current shell itself, provided that the current directory is assigned to the PATH variable:

$ . .profile

See also sh
The POSIX shell built-in `[ ... ]` is used to check whether specific conditions are satisfied. Such conditions may be:
- file attributes,
- characteristics and comparisons of strings, and
- algebraic comparisons of integers.

Conditions can be both combined and negated.

`[ ... ]` returns the following results::
- Exit status 0 (true) if the condition is satisfied.
- Exit status 1 (false), if the condition is not satisfied or was not fully defined. A false exit status is also returned if you do not specify any condition.

Depending on the exit status, you can execute various commands, terminate loops, etc.

The shell built-in `[ ... ]` has two forms (see syntax). The effect is the same in both cases.

**Syntax**

```
[<expression>]
[[<expression>]]
```

See the shell built-in `test` for the syntax description.

**Example**

The shell script `dr` produces a customized listing of the contents of the current directory. It tests each entry in the directory with the command `[ -d "$name" ]` to find out whether it is a subdirectory. Depending on the result of this test, it lists the name preceded either by the label "(dir)" or by the size of the file in blocks:

```
$ cat dr
for name in *
  do if [ -d "$name" ]
    then echo "(dir) $name"
    else echo " `ls -s $name`"
  fi
done
$ dr
(dir) letters
  2 dr
(dir) cards
(dir) documents
```

For further information and examples refer to the `test` command.

**See also**

`test`
5 Tables and directories

5.1 Summary of command XPG4 conformity

Commands which conform fully to the XPG4 standards

asa, at, awk, basename, batch, bc, cal, cancel, cmp, comm, compress, cp, csplit, cut, dd, dirname, du, env, expand, expr, fold, getconf, head, join, locale, localedef, logger, logname, mesg, mkdir, mkfifo, mv, nice, nm, nohup, paste, patch, pathchk, pax, pr, printf, renice, rm, rmdir, sed, sleep, split, sum, tabs, talk, tee, time, tput, tr, tsort, tty, touch, unpack, unexpand, uniq, uudecode, uuencode, wc, who, write, xargs, zcat

Commands which differ from the XPG4 standard

Most of the differences result from additional options.

<table>
<thead>
<tr>
<th>POSIX command</th>
<th>Additional option(s)</th>
<th>Other differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar</td>
<td>-S, -V</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>-s</td>
<td></td>
</tr>
<tr>
<td>chgrp</td>
<td>-h</td>
<td></td>
</tr>
<tr>
<td>chmod</td>
<td></td>
<td>When &lt;perm&gt; is specified then the entry of l (lock) and t (t bit) as well as u, g and o is supported in addition to the XPG4 standard.</td>
</tr>
<tr>
<td>chown</td>
<td>-h</td>
<td></td>
</tr>
<tr>
<td>cksum</td>
<td>-C</td>
<td></td>
</tr>
<tr>
<td>crontab</td>
<td>-ll-rl-e [user]</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>-a</td>
<td>The system clock can only be adjusted by seconds or fractions of seconds</td>
</tr>
<tr>
<td>df</td>
<td>-F, -n, -b, -c, -e, -q, -l, -v, -V, -o</td>
<td></td>
</tr>
<tr>
<td>diff</td>
<td>-a, -h, -i, -l, -n, -s, -t, -w, -D, -S</td>
<td></td>
</tr>
<tr>
<td>ed</td>
<td>-W</td>
<td></td>
</tr>
</tbody>
</table>
### Summary of command XPG4 conformity

<table>
<thead>
<tr>
<th>POSIX command</th>
<th>Additional option(s)</th>
<th>Other differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>egrep</td>
<td>-b, -h</td>
<td></td>
</tr>
<tr>
<td>ex</td>
<td>-r, -L</td>
<td></td>
</tr>
<tr>
<td>fc</td>
<td>-s</td>
<td></td>
</tr>
<tr>
<td>fgrep</td>
<td>-b, -h</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>-c, -f, -h, -m</td>
<td></td>
</tr>
</tbody>
</table>
| find          | -follow, -fstype     | +/-<num> (instead of <num> alone) is permitted for the options -links, -size, -atime, -ctime, -mtime and -size. A symbolic reference is also permitted for -type <char>.
|              | -local, -mount       |                   |
| gencat        | -l, -m               |                   |
| grep          | -b, -h, -r           |                   |
| iccov         |                      | Additional conversion between ISO646 and EDF03 (from ASCII 7-bit to EBCDIC). |
| id            | -a[user>]]           |                   |
| lex           | -V, -Q(y/n)          | Additional syntax compared to XPG4 standard. |
| ln            | -n                   | ln -s <name><reference> ln -s <name>...<directory> |
| lp            |                      | The -m and -w options of the XPG4 standard are not supported. |
| lpstat        |                      | The -v option of the XPG4 standard is not supported. For the -u option you can only specify your own login name. |
| ls            | -L, -b               |                   |
| mailx         | -F, -i, -n, -V       | In send mode In read mode |
| make          | -b, -d               |                   |
| man           |                      | A special database is scanned with the -k option. |
| more          | -d, -f, +<char>, +<line_number>, +/-<pattern> | The -t option of the XPG4 standard is not supported. |
| nice          |                      | Has no effect on BS2000 task priorities |
| nl            | -f,.type             |                   |
| od            | -f, -D, -F, -O, -S, -X |                   |
| ps            | -c, -j, -s, -T       |                   |
| renice        |                      | Has no effect on BS2000 task priorities |
### Summary of command XPG4 conformity

**Commands which are not (or no longer) defined in the XPG4 standard**

<table>
<thead>
<tr>
<th>POSIX command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>adduser</td>
<td>add individual user number</td>
</tr>
<tr>
<td>bs2cmd</td>
<td>execute BS2000 command</td>
</tr>
<tr>
<td>bs2cp</td>
<td>copy BS2000 files</td>
</tr>
<tr>
<td>bs2do</td>
<td>call BS2000 procedures from the POSIX shell</td>
</tr>
<tr>
<td>bs2file</td>
<td>set BS2000 file attributes</td>
</tr>
<tr>
<td>bs2lp</td>
<td>send files to a printer</td>
</tr>
<tr>
<td>bs2pkey</td>
<td>set P keys</td>
</tr>
<tr>
<td>cpio</td>
<td>copy in and out</td>
</tr>
<tr>
<td>debug</td>
<td>debug POSIX programs</td>
</tr>
<tr>
<td>dumpfs</td>
<td>dump file system</td>
</tr>
<tr>
<td>edt</td>
<td>process file with EDT (BS2000)</td>
</tr>
<tr>
<td>fsck</td>
<td>check consistency of a file system and make corrections interactively</td>
</tr>
<tr>
<td>fsexexpand</td>
<td>expand existing file systems</td>
</tr>
<tr>
<td>ftyp</td>
<td>define processing mode for BS2000 files</td>
</tr>
<tr>
<td>POSIX command</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>hd</td>
<td>hex dump</td>
</tr>
<tr>
<td>info</td>
<td>online diagnostic tool</td>
</tr>
<tr>
<td>ipcrm</td>
<td>remove inter-process communication facilities</td>
</tr>
<tr>
<td>ipcs</td>
<td>inter-process communication status</td>
</tr>
<tr>
<td>mkfs</td>
<td>make a filesystem</td>
</tr>
<tr>
<td>mknod</td>
<td>make an inode</td>
</tr>
<tr>
<td>mkpart</td>
<td>hard disk maintenance utility</td>
</tr>
<tr>
<td>mount</td>
<td>mount a file system</td>
</tr>
<tr>
<td>mountall</td>
<td>mount file systems</td>
</tr>
<tr>
<td>pdbl</td>
<td>set up and manage user-specific program cache</td>
</tr>
<tr>
<td>pkginfo</td>
<td>display information on software packages in POSIX</td>
</tr>
<tr>
<td>posdbl</td>
<td>manage POSIX loader</td>
</tr>
<tr>
<td>rcp</td>
<td>remote file copy</td>
</tr>
<tr>
<td>rsh</td>
<td>remote shell</td>
</tr>
<tr>
<td>rmpart</td>
<td>remove partition</td>
</tr>
<tr>
<td>show_pubset_export</td>
<td>show file systems affected by EXPORT-PUBSET</td>
</tr>
<tr>
<td>start_bs2fsd</td>
<td>start copy daemons</td>
</tr>
<tr>
<td>sync</td>
<td>flush system buffers</td>
</tr>
<tr>
<td>tar</td>
<td>file archiver</td>
</tr>
<tr>
<td>umount</td>
<td>unmount a file system</td>
</tr>
<tr>
<td>umountall</td>
<td>unmount file systems</td>
</tr>
</tbody>
</table>
5.2 Regular POSIX shell expressions

Regular expressions are a tool for scanning a text for strings which match a defined pattern. A regular expression stands for a set of character strings. A member of this set of strings is said to be matched by the regular expression. A pattern is constructed from one or more regular expressions.

Regular expressions comprise a string of characters, which can be further classified into:

- ordinary characters, and
- metacharacters.

All alphanumeric characters (all letters and digits) and most other characters are ordinary characters. Within a pattern, ordinary characters match themselves, i.e. the pattern *abc* will match only those strings that contain the character sequence *abc* anywhere in them.

There is, however, a small set of characters, known as metacharacters, which have special meanings when encountered in patterns. These characters are described below.

There are two forms of regular expression:

- simple regular expressions
- extended regular expressions

The syntax of these forms of regular expression is described in the following sections.

The following table shows which commands support regular expressions:

<table>
<thead>
<tr>
<th>Command</th>
<th>Regular expression form</th>
</tr>
</thead>
<tbody>
<tr>
<td>awk</td>
<td>extended</td>
</tr>
<tr>
<td>ed</td>
<td>simple</td>
</tr>
<tr>
<td>egrep</td>
<td>extended</td>
</tr>
<tr>
<td>ex</td>
<td>*)</td>
</tr>
<tr>
<td>expr</td>
<td>simple</td>
</tr>
<tr>
<td>grep</td>
<td>simple</td>
</tr>
<tr>
<td>lex</td>
<td>extended</td>
</tr>
<tr>
<td>nl</td>
<td>simple</td>
</tr>
<tr>
<td>sed</td>
<td>simple</td>
</tr>
<tr>
<td>vi</td>
<td>*)</td>
</tr>
</tbody>
</table>

*) The *ex* and *vi* commands process regular expressions which differ in certain respects from simple regular expressions. These differences are described under *ex* and *vi*. 


## Simple regular expressions

Simple regular expressions are constructed as follows

<table>
<thead>
<tr>
<th>No.</th>
<th>Regular expression</th>
<th>Stands for</th>
<th>Example</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c</td>
<td>The character $c$, where $c$ is not a special character (metacharacter).</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>\c</td>
<td>The character $c$, where $c$ can be any character other than ( ) { } 1 2 3 4 5 6 7 8 9. Regular expressions in this form are meaningful if $c$ is a metacharacter. \c then stands for character $c$ itself, as the backslash escapes its special meaning as a metacharacter.</td>
<td>\a</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Any single character . a, x, *, ...</td>
<td>.</td>
<td>a, x, *, ...</td>
</tr>
<tr>
<td>4</td>
<td>[s]</td>
<td>Any character from s, where s is a set of characters. If a right square bracket ] is to be one of the characters in the set, it has to be placed first in the set. If a hyphen - is to be one of the characters in the set, it has to be placed first or last. If a caret ^ is to be one the characters in the set, it can be placed anywhere but first.</td>
<td>[mz]</td>
<td>m, z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[c1-c2] Any character in the range $c1$ to $c2$, in accordance with the EBCDIC sort sequence (inclusive of limits $c1$ and $c2$). $c1$ must come before $c2$ in the EBCDIC collating sequence. If it does not, $c1$-$c2$ does not denote a range but simply stands for the characters $c1$ and $c2$. The two forms can be combined: [s1c1-c2s2]</td>
<td>[a-m]</td>
<td>a, m and any character in between in the EBCDIC collating sequence m, a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ ]a</td>
<td>[a]</td>
<td>. a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[a-]</td>
<td>[-a]</td>
<td>-. a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[a^]</td>
<td>[a^]</td>
<td>a, ^</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ado-qxz]</td>
<td>[ado-qxz]</td>
<td>a, d, o, q, x, z and any character coming between o and q in the EBCDIC collating sequence</td>
</tr>
</tbody>
</table>
### Regular POSIX shell expressions

<table>
<thead>
<tr>
<th>No.</th>
<th>Regular expression</th>
<th>Stands for</th>
<th>Example</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>![s]</td>
<td>Any character not included in set s.</td>
<td>![xy2]</td>
<td>any character except x, y, z</td>
</tr>
<tr>
<td></td>
<td>![c1-c2]</td>
<td>Any character not in the range between c1 and c2 inclusive. Refer also to [c1-c2].</td>
<td>![0-9]</td>
<td>any character except 0, 9 and all characters coming between 0 and 9 in the EBCDIC collating sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The two forms can be combined: ![s1-c1-c2s2]</td>
<td>![a0-9b]</td>
<td>any character except a, b, 0, 9 and all characters coming between 0 and 9 in the EBCDIC collating sequence</td>
</tr>
<tr>
<td>6</td>
<td>r*</td>
<td>Zero, one or more occurrences of regular expression r. r has to be of form 1 - 5, 12, 15 or 16.</td>
<td>a*</td>
<td>nothing, a, aa, aaa, ...</td>
</tr>
<tr>
<td>7</td>
<td>![m,n]</td>
<td>At least m and at most n occurrences of regular expression r. r has to be of form 1 - 5, 12, 15 or 16.</td>
<td>![1,2]</td>
<td>a or aa</td>
</tr>
<tr>
<td></td>
<td>![m]</td>
<td>Precisely m occurrences of regular expression r. r has to be of form 1 - 5, 12, 15 or 16.</td>
<td>![3]</td>
<td>aaa</td>
</tr>
<tr>
<td></td>
<td>![m]</td>
<td>At least m occurrences of regular expression r. r has to be of form 1 - 5, 12, 15 or 16.</td>
<td>![3]</td>
<td>aaa, aaaa, aaaa, ...</td>
</tr>
<tr>
<td>8</td>
<td>rx</td>
<td>(Concatenation) An occurrence of regular expression r followed by an occurrence of regular expression x. r and x can be any regular expressions.</td>
<td>![ab]</td>
<td>ax, a3, a*, bz, ...</td>
</tr>
<tr>
<td>9</td>
<td>![r]</td>
<td>An occurrence of regular expression r appearing at the start of a line, i.e. straight after a newline character or at the start of the file. r can be a regular expression in any form other than number 9.</td>
<td>![aA]pple</td>
<td>apple or Apple at the start of a line</td>
</tr>
</tbody>
</table>
Regular POSIX shell expressions

Precedence

The precedence of operators in regular expressions is as shown in the following table.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>[. ] [= ] [: :]</td>
<td>high precedence</td>
</tr>
<tr>
<td>&lt;char&gt;</td>
<td>.</td>
</tr>
<tr>
<td>[ ]</td>
<td>.</td>
</tr>
<tr>
<td>( )</td>
<td>.</td>
</tr>
<tr>
<td>* ? + {m,n}</td>
<td>.</td>
</tr>
<tr>
<td>Concatenation</td>
<td>.</td>
</tr>
<tr>
<td>^ $</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>low precedence</td>
</tr>
</tbody>
</table>

Tables and directories

<table>
<thead>
<tr>
<th>No.</th>
<th>Regular expression</th>
<th>Stands for</th>
<th>Example</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>r$</td>
<td>An occurrence of regular expression r at the end of a line, i.e. directly before a newline character. r can be a regular expression in any form other than number 10.</td>
<td>[bB]arge$</td>
<td>barge or Barge at the end of a line</td>
</tr>
<tr>
<td>11</td>
<td>(()</td>
<td>Occurrences of regular expression r. r can be any regular expression. Only useful together with number 12</td>
<td>(aA)pple\ )</td>
<td>apple, Apple</td>
</tr>
<tr>
<td>12</td>
<td>\n</td>
<td>n is an integer in the range from 1 to 9. \n appearing in a concatenated regular expression stands for regular expression x, where x is the n-th regular expression enclosed in ( and ) sequences that appeared earlier in the concatenated regular expression.</td>
<td>(a\ (b\ )\ )\ 2</td>
<td>abb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s\ (illy\ )b\ 1</td>
<td>sillybilly</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>\ (ab\ )x\ 1*</td>
<td>abx, abxab, abxabab, ...</td>
</tr>
</tbody>
</table>
Metacharacters

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>The character to the left has a special meaning if</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>– it is not preceded by a backslash \</td>
</tr>
<tr>
<td>.</td>
<td>– it is not preceded by a backslash \ and</td>
</tr>
<tr>
<td>[</td>
<td>– it does not appear between [ and ]</td>
</tr>
<tr>
<td>^</td>
<td>– it is not the first character in a pattern and</td>
</tr>
<tr>
<td></td>
<td>– it does not come after \</td>
</tr>
<tr>
<td></td>
<td>– it is the last character in a pattern</td>
</tr>
<tr>
<td></td>
<td>– it is the first character in square brackets [ ... ]</td>
</tr>
<tr>
<td></td>
<td>– it is in square brackets but not placed first or last</td>
</tr>
<tr>
<td></td>
<td>– it is not preceded by a backslash \</td>
</tr>
<tr>
<td>[.   ]</td>
<td>Character pairs to the left are special characters if they occur within a bracket expression (in square brackets). They will need to be closed by the corresponding character pair [.], =] or :].</td>
</tr>
<tr>
<td></td>
<td>Example: [:upper:] indicates all uppercase letters.</td>
</tr>
</tbody>
</table>
### Extended regular expressions

Extended regular expressions include the regular expressions with the following exception:

The construction used for simple regular expressions \(`\ldots\)` has *no* special significance for extended regular expressions, for example the extended regular expression `\(`\(ab\)`\)` represents the string \((ab)`\).

Moreover, extended regular expressions provide the following syntax elements for pattern creation:

<table>
<thead>
<tr>
<th>No.</th>
<th>Regular expression</th>
<th>Stands for</th>
<th>Example</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><code>r {m,n }</code></td>
<td>At least (m) and at most (n) occurrences of regular expression (r). (r) has to be of form 1 - 5, 12, 15 or 16.</td>
<td><code>a {1,2 }</code></td>
<td>a or aa</td>
</tr>
<tr>
<td></td>
<td><code>r {m}</code></td>
<td>Precisely (m) occurrences of regular expression (r). (r) has to be of form 1 - 5, 12, 15 or 16.</td>
<td><code>a {3}</code></td>
<td>aaaa</td>
</tr>
<tr>
<td></td>
<td><code>r {m, }</code></td>
<td>At least (m) occurrences of regular expression (r). (r) has to be of form 1 - 5, 12, 15 or 16.</td>
<td><code>a {3, }</code></td>
<td>aaaa, aaaa, aaaa, ...</td>
</tr>
<tr>
<td>13</td>
<td><code>r+</code></td>
<td>One or more occurrences of regular expression (r). (r) has to be of form 1 - 5, 15 or 16.</td>
<td><code>u+</code></td>
<td>u, uu, uuu, ...</td>
</tr>
<tr>
<td>14</td>
<td><code>r?</code></td>
<td>Zero or one occurrence of regular expression (r). (r) has to be of form 1 - 5, 15 or 16.</td>
<td><code>u?</code></td>
<td>nothing or u</td>
</tr>
<tr>
<td>15</td>
<td><code>(\)</code></td>
<td>Strings matching regular expression (r). (r) can be any regular expression.</td>
<td><code>(ok(abc))</code></td>
<td>okabc</td>
</tr>
<tr>
<td></td>
<td><code>(au)*</code></td>
<td></td>
<td><code>(au)*</code></td>
<td>nothing or au, auau, ...</td>
</tr>
<tr>
<td>16</td>
<td><code>(r1/r2)</code></td>
<td>Strings matching regular expression (r1) or regular expression (r2).</td>
<td><code>(ok?ko)</code></td>
<td>ok or ko</td>
</tr>
</tbody>
</table>
Tables and directories

Regular POSIX shell expressions

Precedence

The precedence of operators in extended regular expressions is as shown in the following table.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ .. ] [ != ] [ : : ]</td>
<td>high precedence</td>
</tr>
<tr>
<td>&lt;char&gt;</td>
<td>.</td>
</tr>
<tr>
<td>[ ]</td>
<td>.</td>
</tr>
<tr>
<td>()</td>
<td>.</td>
</tr>
<tr>
<td>* ? + {m,n}</td>
<td>.</td>
</tr>
<tr>
<td>Concatenation</td>
<td>.</td>
</tr>
<tr>
<td>^ $</td>
<td>.</td>
</tr>
</tbody>
</table>

Examples

1. Simple regular expressions

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab.d</td>
<td>a - b - any one character - d</td>
<td>abcd, abXd, ab*d, ...</td>
</tr>
<tr>
<td>ab.&quot;d</td>
<td>a - b - any string (including the null string) - d</td>
<td>abd, abxd, abX*Yd, ...</td>
</tr>
<tr>
<td>ab[xyz]d</td>
<td>a - b - either x or y or z - d</td>
<td>abxd, abyd, abzd</td>
</tr>
<tr>
<td>ab[^c]d</td>
<td>a - b - any character other than c - d</td>
<td>abbd, abXd, ab*d, ...</td>
</tr>
<tr>
<td>^abcd$</td>
<td>a line containing only the string abcd</td>
<td></td>
</tr>
</tbody>
</table>

2. Extended regular expressions

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab.+d</td>
<td>a - b - any sequence of one or more characters - d</td>
<td>abjd, abX*Yd, ...</td>
</tr>
<tr>
<td>abc?d</td>
<td>a - b - c or nothing - d</td>
<td>abd, abcd</td>
</tr>
<tr>
<td>(abc</td>
<td>xyz)</td>
<td>abc or xyz</td>
</tr>
</tbody>
</table>
## 5.3 Metacharacters for the POSIX shell

### Argument and command separators

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank, Newline, Tab</td>
<td>Argument separators, depending on the value of the IFS variable</td>
</tr>
<tr>
<td>Newline</td>
<td>End of command</td>
</tr>
<tr>
<td></td>
<td>Pipe symbol</td>
</tr>
<tr>
<td>;</td>
<td>End of command</td>
</tr>
<tr>
<td>&amp;</td>
<td>End of command; a command terminated with this symbol is run as a background process</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>ANDIF; the next command is executed only if the preceding command returns an exit status of zero</td>
</tr>
</tbody>
</table>

### Command grouping

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(command_list)</td>
<td>Execute command_list in a subshell</td>
</tr>
<tr>
<td>{ command_list;}</td>
<td>Collect the output of all commands from command_list</td>
</tr>
</tbody>
</table>

### Execute command and replace with output

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>command</code></td>
<td>Replace command with its output</td>
</tr>
<tr>
<td>$(command)</td>
<td>Replace command with its output</td>
</tr>
</tbody>
</table>
### File name generation

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>As a pattern on its own: replaced by the list of all file names in the current directory that do not begin with a period (.). As part of a pattern: replaced by no, one or more character(s) depending on the file names which match the pattern.</td>
</tr>
<tr>
<td>?</td>
<td>As a pattern on its own: replaced by the list of all file names in the current directory that consist of exactly one character, except for a period (.). As part of a pattern: replaced by no, one or more character(s) depending on the file names which match the pattern.</td>
</tr>
<tr>
<td>[s]</td>
<td>Replaced by exactly one of the characters which are not contained in the string s depending on the file names which match the pattern.</td>
</tr>
<tr>
<td>[c1-c2]</td>
<td>Replaced by any one character from the range between c1 and c2 inclusive to match file names in the current directory. c1 and c2 must be ordinary characters. The characters which collate between c1 and c2 are determined by the EBCDIC collating sequence. Expressions of type [s] and type [c1-c2] can be combined: [s</td>
</tr>
<tr>
<td>[!s]</td>
<td>Replaced by exactly one of the characters which are not contained in the string s depending on the file names which match the pattern.</td>
</tr>
<tr>
<td>[!c1-c2]</td>
<td>Replaced by exactly one of the characters which do not lie in the range c1 to c2 depending on the file names which match the pattern (see [c1-c2]). Expressions of type [!s] and type [!c1-c2s2] can be combined: [!s</td>
</tr>
</tbody>
</table>

### Redirection of standard output

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[n] &gt; file</td>
<td>Redirect standard output [or file descriptor n] to file, deleting original contents</td>
</tr>
<tr>
<td>[n] &gt;&gt; file</td>
<td>Redirect standard output [or file descriptor n] to file, retaining original contents (append)</td>
</tr>
<tr>
<td>[n] &amp;digit</td>
<td>Redirect standard output [or file descriptor n] to the file associated with file descriptor digit</td>
</tr>
<tr>
<td>[n] &amp;-</td>
<td>Close standard output [or file descriptor n]</td>
</tr>
</tbody>
</table>
### Redirection of standard input

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[n]&lt;file</code></td>
<td>Redirect standard input [or file descriptor ( n )] to ( file )</td>
</tr>
<tr>
<td><code>[n]&lt;&lt;&lt;argument</code></td>
<td>Start a &quot;here&quot; document or redirect to file descriptor ( n )</td>
</tr>
<tr>
<td><code>[n] &lt;&lt;=argument</code></td>
<td>Start a &quot;here&quot; document or redirect to file descriptor ( n ); stripping leading tabs</td>
</tr>
<tr>
<td><code>[n]&lt;&amp;digit</code></td>
<td>Redirect standard input to the file [or file descriptor ( n )] associated with file descriptor ( digit )</td>
</tr>
<tr>
<td><code>[n]&lt;&amp;-</code></td>
<td>Close standard input [or file descriptor ( n )]</td>
</tr>
<tr>
<td><code>[n]&lt;=&gt;file</code></td>
<td>( file ) [or file descriptor ( n )] is opened as the standard input for reading and writing</td>
</tr>
</tbody>
</table>

### Shell variables and parameters

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name=value</code></td>
<td>Assign a value to the variable ( name )</td>
</tr>
<tr>
<td><code>${name}</code></td>
<td>Value of the variable ( name ); keyword parameter</td>
</tr>
<tr>
<td><code>${(name)}</code></td>
<td>Like ( $name ); the braces separate the variable name from following figures or characters</td>
</tr>
<tr>
<td><code>${(name-default_value)</code></td>
<td>Replace with ( default_value ) if name is undefined</td>
</tr>
<tr>
<td><code>${(name=default_value)</code></td>
<td>Assign ( default_value ) if name is undefined</td>
</tr>
<tr>
<td><code>${(name?default_value)</code></td>
<td>Shell terminates process execution with error message parameter : ( default_value ) if name is undefined</td>
</tr>
<tr>
<td><code>${(name+default_value)</code></td>
<td>Replace with the null string if name is undefined; replace with ( default_value ) if name is defined</td>
</tr>
<tr>
<td><code>${(name:-default_value)</code></td>
<td>Replace with ( default_value ) if name is undefined or if its value is the null string</td>
</tr>
<tr>
<td><code>${(name:=default_value)</code></td>
<td>Assign ( default_value ) if name is undefined or if its value is the null string</td>
</tr>
<tr>
<td><code>${(name:?default_value)</code></td>
<td>Shell terminates process execution with error message parameter : ( default_value ) if name is undefined or if its value is the null string</td>
</tr>
<tr>
<td><code>${(name:+default_value)</code></td>
<td>Replace with the null string if name is undefined or if its value is the null string; replace with ( default_value ) if name is undefined or if its value is the null string</td>
</tr>
<tr>
<td><code>${(#name)</code></td>
<td>Replace with the length of the string formed by ( name ). If ( name ) is one of the characters * or commercial at @ then the substitution is undefined</td>
</tr>
</tbody>
</table>
### Metacharacters for the POSIX shell

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>${name%pattern}$</td>
<td>If the POSIX shell pattern <code>pattern</code> matches the end of the value of <code>name</code>, then the substitute text is formed from the value of <code>name</code> without the deleted segment which corresponds to <code>pattern</code>. The shortest matching pattern is deleted.</td>
</tr>
<tr>
<td>${name%%pattern}$</td>
<td>If the POSIX shell pattern <code>pattern</code> matches the end of the value of <code>name</code>, then the substitute text is formed from the value of <code>name</code> without the deleted segment which corresponds to <code>pattern</code>. The shortest matching pattern is deleted.</td>
</tr>
<tr>
<td>${name#pattern}$</td>
<td>If the POSIX shell pattern <code>pattern</code> is to match the start of the value of <code>name</code>, then the substitute text is formed from the value of <code>name</code> from which the segment which corresponds to <code>pattern</code> has been deleted. The shortest matching pattern is deleted.</td>
</tr>
<tr>
<td>${name##pattern}$</td>
<td>If the POSIX shell pattern <code>pattern</code> is to match the start of the value of <code>name</code>, then the substitute text is formed from the value of <code>name</code> from which the segment which corresponds to <code>pattern</code> has been deleted. The shortest matching pattern is deleted.</td>
</tr>
<tr>
<td>$0$</td>
<td>0th command-line argument, i.e. the name of the command, the shell script or the current shell</td>
</tr>
<tr>
<td>$1$, $2$, …, $9$</td>
<td>Positional parameters</td>
</tr>
<tr>
<td>$*$</td>
<td>All command-line arguments</td>
</tr>
<tr>
<td>&quot;$&quot;</td>
<td>All command-line arguments as a single argument</td>
</tr>
<tr>
<td>&quot;$@&quot;</td>
<td>All command-line arguments</td>
</tr>
<tr>
<td>&quot;$@*&quot;</td>
<td>All command-line arguments as separate arguments</td>
</tr>
<tr>
<td>$#$</td>
<td>Number of command-line arguments, excluding $0</td>
</tr>
<tr>
<td>$$$&quot;</td>
<td>Process ID (PID) of the current shell</td>
</tr>
<tr>
<td>$!$</td>
<td>PID of the last command run in the background</td>
</tr>
<tr>
<td>$?$</td>
<td>Exit status of the last executed command that was not run in the background</td>
</tr>
<tr>
<td>$-$</td>
<td>All options set in the current shell</td>
</tr>
</tbody>
</table>
## Metacharacters for the POSIX shell

### Shell functions

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name() { command_list;}</code></td>
<td>Shell function; when <code>name</code> is invoked, the commands from <code>command_list</code> are executed</td>
</tr>
</tbody>
</table>

### Quoting metacharacters

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>Escapes the following metacharacter</td>
</tr>
<tr>
<td><code>' ...'</code></td>
<td>Escapes all metacharacters between quotes; treat as a single argument</td>
</tr>
</tbody>
</table>
| `" ..."` | Does not escape the metacharacters `$ `...` \\
Treat as a single argument |

### Miscellaneous

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#</code></td>
<td>Comment character in shell scripts</td>
</tr>
<tr>
<td><code>;;</code></td>
<td>Command list terminator within <code>case</code> constructs</td>
</tr>
<tr>
<td><code>~</code></td>
<td>Tilde substitution</td>
</tr>
<tr>
<td><code>~=</code></td>
<td>Tilde substitution</td>
</tr>
<tr>
<td><code>~+</code></td>
<td>Tilde substitution</td>
</tr>
<tr>
<td><code>(( ))</code></td>
<td>Arithmetic evaluation</td>
</tr>
<tr>
<td><code>$(expression)</code></td>
<td>Arithmetic evaluation</td>
</tr>
<tr>
<td><code>[[]]</code></td>
<td>Conditional expressions</td>
</tr>
</tbody>
</table>
### 5.4 ASCII character set (ISO 646)

The following table combines the international version of the ISO 646 character set, the US version (equivalent to the 7-bit US ASCII coded character set), the British version and the German version.

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
<th>hex</th>
<th>Meaning</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>NUL</td>
<td>@</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>01</td>
<td>SOH</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>02</td>
<td>STX</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>03</td>
<td>ETX</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>04</td>
<td>EOT</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>05</td>
<td>ENQ</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>06</td>
<td>ACK</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>07</td>
<td>BEL</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>08</td>
<td>BS</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>09</td>
<td>HT</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>0A</td>
<td>LF</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>0B</td>
<td>VT</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>0C</td>
<td>FF</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>0D</td>
<td>CR</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>0E</td>
<td>SO</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>0F</td>
<td>SI</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>10</td>
<td>DLE</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>21</td>
<td>11</td>
<td>DC1</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>22</td>
<td>12</td>
<td>DC2</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>23</td>
<td>13</td>
<td>DC3</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>24</td>
<td>14</td>
<td>DC4</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>15</td>
<td>NAK</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>26</td>
<td>16</td>
<td>SYN</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>27</td>
<td>17</td>
<td>ETB</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>30</td>
<td>18</td>
<td>CAN</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>31</td>
<td>19</td>
<td>EM</td>
<td>\ or /</td>
</tr>
<tr>
<td>26</td>
<td>32</td>
<td>1A</td>
<td>SUB</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>33</td>
<td>1B</td>
<td>ESC</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>34</td>
<td>1C</td>
<td>FS</td>
<td></td>
</tr>
<tr>
<td>decimal</td>
<td>octal</td>
<td>hex</td>
<td>Meaning</td>
<td>Control</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----</td>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>29</td>
<td>35</td>
<td>1D</td>
<td>GS</td>
<td>]</td>
</tr>
<tr>
<td>30</td>
<td>36</td>
<td>1E</td>
<td>RS</td>
<td>^</td>
</tr>
<tr>
<td>32</td>
<td>40</td>
<td>20</td>
<td>SP</td>
<td>SPACE</td>
</tr>
<tr>
<td>33</td>
<td>41</td>
<td>21</td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>42</td>
<td>22</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>43</td>
<td>23</td>
<td>#</td>
<td>see 1)</td>
</tr>
<tr>
<td>36</td>
<td>44</td>
<td>24</td>
<td>$</td>
<td>see 2)</td>
</tr>
<tr>
<td>37</td>
<td>45</td>
<td>25</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>46</td>
<td>26</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>47</td>
<td>27</td>
<td>'</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td>28</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>51</td>
<td>29</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>52</td>
<td>2A</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>53</td>
<td>2B</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>54</td>
<td>2C</td>
<td>,</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>55</td>
<td>2D</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>56</td>
<td>2E</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>57</td>
<td>2F</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>60</td>
<td>30</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>61</td>
<td>31</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>62</td>
<td>32</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>63</td>
<td>33</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>64</td>
<td>34</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>65</td>
<td>35</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>66</td>
<td>36</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>67</td>
<td>37</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>70</td>
<td>38</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>71</td>
<td>39</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>72</td>
<td>3A</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>73</td>
<td>3B</td>
<td>;</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>74</td>
<td>3C</td>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>75</td>
<td>3D</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>76</td>
<td>3E</td>
<td>&gt;</td>
<td></td>
</tr>
</tbody>
</table>
### ASCII character set (ISO 646)

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
<th>hex</th>
<th>Meaning</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>77</td>
<td>3F</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>40</td>
<td>@</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>101</td>
<td>41</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>102</td>
<td>42</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>103</td>
<td>43</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>104</td>
<td>44</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>105</td>
<td>45</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>106</td>
<td>46</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>107</td>
<td>47</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>110</td>
<td>48</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>111</td>
<td>49</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>112</td>
<td>4A</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>113</td>
<td>4B</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>114</td>
<td>4C</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>115</td>
<td>4D</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>116</td>
<td>4E</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>117</td>
<td>4F</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>120</td>
<td>50</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>121</td>
<td>51</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>122</td>
<td>52</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>123</td>
<td>53</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>124</td>
<td>54</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>125</td>
<td>55</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>126</td>
<td>56</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>127</td>
<td>57</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>130</td>
<td>58</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>131</td>
<td>59</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>132</td>
<td>5A</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>133</td>
<td>5B</td>
<td>[ see 1); German: Å</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>134</td>
<td>5C</td>
<td>\ see 1); German: Ö</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>135</td>
<td>5D</td>
<td>] see 1); German: Ü</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>136</td>
<td>5E</td>
<td>^</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>137</td>
<td>5F</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
### ASCII character set (ISO 646)

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
<th>hex</th>
<th>Meaning</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>140</td>
<td>60</td>
<td>`</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>141</td>
<td>61</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>142</td>
<td>62</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>143</td>
<td>63</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>144</td>
<td>64</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>145</td>
<td>65</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>146</td>
<td>66</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>147</td>
<td>67</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>150</td>
<td>68</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>151</td>
<td>69</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>152</td>
<td>6A</td>
<td>j</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>153</td>
<td>6B</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>154</td>
<td>6C</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>155</td>
<td>6D</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>156</td>
<td>6E</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>157</td>
<td>6F</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>160</td>
<td>70</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>161</td>
<td>71</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>162</td>
<td>72</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>163</td>
<td>73</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>164</td>
<td>74</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>165</td>
<td>75</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>166</td>
<td>76</td>
<td>v</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>167</td>
<td>77</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>170</td>
<td>78</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>171</td>
<td>79</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>172</td>
<td>7A</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>173</td>
<td>7B</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>174</td>
<td>7C</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>175</td>
<td>7D</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>176</td>
<td>7E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>177</td>
<td>7F</td>
<td>DEL</td>
<td>Delete / Interrupt (Signal2)</td>
</tr>
</tbody>
</table>
5.5 EDF04 character set

**EBCDIC.DF.04 code table**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>&amp;</td>
<td>-</td>
<td>ø</td>
<td>Ø</td>
<td>°</td>
<td>μ</td>
<td>ç</td>
<td>ü</td>
<td>i</td>
<td>Ü</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBSP</td>
<td>é</td>
<td>/</td>
<td>É</td>
<td>a</td>
<td>j</td>
<td>£</td>
<td>A</td>
<td>J</td>
<td>÷</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>å</td>
<td>é</td>
<td>Ä</td>
<td>Å</td>
<td>™</td>
<td>¥</td>
<td>B</td>
<td>K</td>
<td>S</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ä</td>
<td>é</td>
<td>Ä</td>
<td>Å</td>
<td>™</td>
<td>C</td>
<td>L</td>
<td>T</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>à</td>
<td>é</td>
<td>Ä</td>
<td>Å</td>
<td>™</td>
<td>D</td>
<td>M</td>
<td>U</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>á</td>
<td>í</td>
<td>Á</td>
<td>Í</td>
<td>™</td>
<td>E</td>
<td>N</td>
<td>V</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ä</td>
<td>í</td>
<td>Ä</td>
<td>Í</td>
<td>™</td>
<td>F</td>
<td>O</td>
<td>W</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>á</td>
<td>í</td>
<td>Á</td>
<td>Í</td>
<td>™</td>
<td>G</td>
<td>P</td>
<td>X</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ç</td>
<td>i</td>
<td>Ç</td>
<td>Ì</td>
<td>™</td>
<td>H</td>
<td>Q</td>
<td>Y</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ñ</td>
<td>ß</td>
<td>Ñ</td>
<td>Ñ</td>
<td>™</td>
<td>I</td>
<td>R</td>
<td>Z</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For EDF04 character set:

1) ASCII: as international variant (see table)

2) ASCII: ~

British: as international variant (see table)

German: umlauts (see table)
### EBCDIC.DF.03 code table (international variant)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>&amp;</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>/</td>
<td>a</td>
<td>i</td>
<td>A</td>
<td>J</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>SYN</td>
<td>b</td>
<td>k</td>
<td>s</td>
<td>B</td>
<td>K</td>
<td>S</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>c</td>
<td>l</td>
<td>t</td>
<td>C</td>
<td>L</td>
<td>T</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>d</td>
<td>m</td>
<td>u</td>
<td>D</td>
<td>M</td>
<td>U</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>HT</td>
<td>LF</td>
<td>e</td>
<td>n</td>
<td>v</td>
<td>E</td>
<td>N</td>
<td>V</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>BS</td>
<td>ETB</td>
<td>f</td>
<td>o</td>
<td>w</td>
<td>F</td>
<td>O</td>
<td>W</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>DEL</td>
<td>ESC</td>
<td>EOT</td>
<td>g</td>
<td>p</td>
<td>x</td>
<td>G</td>
<td>P</td>
<td>X</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>CAN</td>
<td></td>
<td>h</td>
<td>q</td>
<td>y</td>
<td>H</td>
<td>Q</td>
<td>Y</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>EM</td>
<td></td>
<td>i</td>
<td>r</td>
<td>z</td>
<td>I</td>
<td>R</td>
<td>Z</td>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>'</td>
<td>!</td>
<td>^</td>
<td>:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td></td>
<td>-</td>
<td>$</td>
<td>,</td>
<td>#</td>
<td>[</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>&lt;</td>
<td>*</td>
<td>%</td>
<td>@</td>
<td>\</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>IS3</td>
<td>ENQ</td>
<td>(</td>
<td>)</td>
<td>_</td>
<td>'</td>
<td>]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>ACK</td>
<td>+</td>
<td>;</td>
<td>&gt;</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>IS1</td>
<td>BEL</td>
<td>?</td>
<td>&quot;</td>
<td>~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

894  U22794-J-Z125-6-76
Related publications

The manuals are available as online manuals, see http://manuals.ts.fujitsu.com, or in printed form which must be paid and ordered separately at http://manualshop.ts.fujitsu.com.

[1] POSIX (BS2000/OSD)
Basics for Users and System Administrators
User Guide

[2] POSIX (BS2000/OSD)
BS2000 filesystem bs2fs
User Guide

[3] POSIX (BS2000/OSD)
SOCKETS/XTI for POSIX
User Guide

for POSIX Applications
Reference Manual

POSIX Commands of the C/C++ Compiler
User Guide

Common Runtime Environment
User Guide

[7] EDT (BS2000/OSD)
Statements
User Guide

[8] NFS (BS2000/OSD)
Network File System
User Guide
Related publications

[9] JV (BS2000(OSD))
   Job Variables
   User Guide

    Commands
    User Guide

    Reference manual (only online)

[12] Reliant UNIX
    System administrator's reference manual
    Reference manual (only online)

[13] SINIX
    Programmer's Guide: Internationalization - Localization
    Programmer's Guide
Related publications

Other publications

[14]  XBD, Issue 4  
X/Open CAE Specification, August 1994  
System Interfaces Definitions, Issue 4, Version 2  
X/Open Company Limited

X/Open CAE Specification, August 1994  
System Interfaces and Headers, Issue 4, Version 2  
X/Open Company Limited

[16]  XSH, Issue 4  
X/Open CAE Specification, August 1994  
Commands and Utilities, Issue 4, Version 2  
X/Open Company Limited

[17]  el Lozy, M.:  
Editing in a UNIX Environment  
The vi/ex Editor  
Prentice-Hall, 1985
Index

. (POSIX command) 870
/etc/hosts.equiv 639, 655
: (POSIX command) 868
[ ] (POSIX command) 872

A
absolute mode syntax (chmod) 206
access time
  update of last (touch) 742
accessing the POSIX shell
  from a UNIX or SINIX computer 25
  via BS2000 terminal 24
via emulation 26
acoustic signal (vi) 813
action (awk) 126
adduser (POSIX command) 91
affixes, file name (basename) 149
alias (mailx) 496
alias (POSIX command) 92
alias variable
  defining (alias) 92
  delete (unalias) 783
  exporting (alias) 92
aliasing 45
allow file (at) 100
allow file (batch) 151
allow file (crontab) 238
append text (ed) 295
ar (POSIX command) 94
arc tangent (awk) 135
archive
  maintaining (ar) 94
  structure (ar) 97
argument
  assign to shell variable (read) 643
  combine command-line arguments (xargs) 858
  evaluate (eval) 320
  execute as command (command) 220
  interpret as expression (expr) 366
  read from standard input (read) 643
  replace by file name 885
  scan for options (getopts) 408
  separators 884
  write to standard output (echo) 284
argument list
  construct and execute command (xargs) 858
  parse (getopts) 408
arithmetic evaluation 56, 448
arithmetic functions
  commands 79
  POSIX shell 56
arithmetic language (bc) 154
array
  associative (awk) 121
  dynamic (awk) 121
  for statement (awk) 130
  index (awk) 121
  looping (awk) 130
asa (POSIX command) 99
ASCII file, converting 28
ASCII file format 28
ASCII-EBCDIC conversion 29
at (POSIX command) 100
atan2 function (awk) 135
awk (POSIX command) 107
awk language, basic elements (awk) 117
Index

B
background job (bg) 166
background process
  wait for termination (wait) 843
basename (POSIX command) 149
basic language elements (awk) 117
batch (POSIX command) 151
bc (POSIX command) 154
bg (POSIX command) 166
binary file, see file
binary text files 400
blank interpretation 53
block-mode terminal 23, 27
  operation 27
break (built-in POSIX command) 43
break statement (awk) 127
BS2000 file
  copy (edt) 308
    defining attributes (bs2file) 189
    determining type (ftyp) 399
BS2000 software interfaces 11
bs2cp (POSIX command) 169
bs2file (POSIX command) 189
bs2lp (POSIX command) 190
bs2pkey (POSIX command) 192
buffer
  flush (sync) 712
  of ed 290
  of ex 326
  of vi 816
  save contents (ed) 290
  write to disk (sync) 712
C
C library 11
cal (POSIX command) 193
calculation function (bc) 154
calendar
  print (cal) 193
  scheduling commands 78
cancel
  print jobs (cancel) 195
cancel (POSIX command) 195
case conversion (NLS) 86
cat (POSIX command) 197
c 66
cd (POSIX command) 199
change group ID for file (chgrp) 202
change owner (chown) 210
change working directory (cd) 199
change, terminal options (stty) 701
character classification (NLS) 86
character count (wc) 846
character set
  7-bit 889
  EDF04 893
  internationalized program (NLS) 85
  ISO 646 889
character terminal 26
character, delete (tr) 752
character-mode terminal 23, 27
  operation 27
characters
  commands for reading, converting and outputting 78
  transliterate (tr) 752
check file system (fsck) 395
checksum
  calculate (sum) 710
  polynomial 212
chgrp (POSIX command) 202
child process 39
  check value (ulimit) 775
  output runtime 741
chmod (POSIX command) 204
chown (POSIX command) 210
cksum (POSIX command) 212
close, file or pipe (awk) 135
cmp (POSIX command) 215
code conversion (iconv) 425
collation (NLS) 86
combining conditions 736
  precedence 736
comm (POSIX command) 217
command
  command substitution 47
  evaluate command line (eval) 320
  execute (command) 220
execute later (batch) 151
execute permission (crontab) 238
executing and replacing with output 884
execution 64
execution (awk) 144
ignore signals (nohup) 582
mailx commands (mailx) 510
mailx commands in command files (mailx) 495
measure runtime (time) 739
measure runtime (times) 741
order of redirections 38
processing via the POSIX shell 40
query type (type) 771
query type (whence) 848
redefine (alias) 92
redirecting input and output 36
re-entry 65
regular execution (crontab) 238
separators 884
set environment (env) 316
substitution 47
time (time) 739
XPG4 conformity 873
command (POSIX command) 220
command interpreter 73
command invocation 33
command mode (ed) 290
command mode (ex) 326
command mode (vi) 808
command overview
ed (ed) 293
ex (ex) 334
mailx, alphabetical (mailx) 512
mailx, functional (mailx) 492, 510
sed (sed) 663
vi (vi) 804
command strings 38
command structure (ed) 291
command-line arguments
echo (echo) 284
commands
  grouping 884
comment 45
common lines
  select/reject (comm) 217
communication
  with another user (talk) 721
  write to another user (write) 854
compare
  files (diff) 274
  strings (test) 732
compiler commands 79
compress (POSIX command) 224
compress, files (compress) 224
compressed file
  display uncompressed (zcat) 866
  expand (uncompress) 787
concatenate, files (cat) 197
condition
  combining 736
  negating (test) 735
  test ([ ]) 872
  test (test) 732
conditional expression 57
configuration values, get (getconf) 404
constant
  alphanumeric (awk) 117
  numeric (awk) 117
continue (built-in POSIX command) 43
continue statement (awk) 128
control-flow statement
  break 41
  case 42
  for 62
  if 42
  while 42
control-flow statement (awk) 127
break 127
continue 128
do 128
exit 129
for 129, 130
if 131
next 131
while 132
# Index

<table>
<thead>
<tr>
<th>Conversion</th>
<th>ASCII-EBCDIC (edt)</th>
<th>308</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>code set (iconv)</td>
<td>425</td>
</tr>
<tr>
<td>Convert</td>
<td>ASCII-EBCDIC</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>file (dd)</td>
<td>260</td>
</tr>
<tr>
<td>Copy</td>
<td>BS2000 file (edt)</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>file (cp)</td>
<td>189, 227</td>
</tr>
<tr>
<td></td>
<td>file (dd)</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>remote file (rcp)</td>
<td>639</td>
</tr>
<tr>
<td>Corrections in file systems (fsck)</td>
<td>395</td>
<td></td>
</tr>
<tr>
<td>Cosine function (awk)</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>Count</td>
<td>characters (wc)</td>
<td>846</td>
</tr>
<tr>
<td></td>
<td>lines (wc)</td>
<td>846</td>
</tr>
<tr>
<td></td>
<td>words (wc)</td>
<td>846</td>
</tr>
<tr>
<td>Cdp (POSIX command)</td>
<td>227</td>
<td></td>
</tr>
<tr>
<td>CRC</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>Create parser</td>
<td>863</td>
<td></td>
</tr>
<tr>
<td>Crontab (POSIX command)</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>CrtE</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Csplit (POSIX command)</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>Cursor, positioning (vi)</td>
<td>809</td>
<td></td>
</tr>
<tr>
<td>Cut (POSIX command)</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>Cyclic redundancy check (cksum)</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>Cylinder groups</td>
<td>display information (dumpfs)</td>
<td>282</td>
</tr>
</tbody>
</table>

| Description format  | POSIX commands     | 15  |
| Description structure| POSIX commands     | 15  |
| Device class        | 544                |
| Df (POSIX command)  | 267                |
| Diagnostics, online (info) | 430             |
| Diff (POSIX command) | 274               |
| Differential file comparator (diff) | 274 |
| Directory           | change (cd)        | 199 |
|                     | change mode (chmod) | 204 |
|                     | change owner (chown)| 210 |
|                     | change user group (chgrp) | 202 |
|                     | copy (cp)          | 227 |
|                     | find               | 385 |
|                     | list contents (ls) | 480 |
|                     | make (mkdir)       | 539 |
|                     | print working (pwd)| 638 |
|                     | remove (rm)        | 650 |
|                     | remove/delete (rmdir) | 652 |
|                     | rename (mv)        | 566 |
|                     | scan (find)        | 385 |
|                     | search for (find)  | 385 |
|                     | dirname (POSIX command) | 279 |
| Disk blocks, output (df) | 267       |
| Disk space          | display used (du)  | 280 |
|                     | output (df)        | 267 |
|                     | disk usage, summarize (du) | 280 |
## Index

<table>
<thead>
<tr>
<th>Display</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>files</td>
<td>197</td>
</tr>
<tr>
<td>files (cat)</td>
<td>197</td>
</tr>
<tr>
<td>inodes (df)</td>
<td>267</td>
</tr>
<tr>
<td>last part of file (tail)</td>
<td>717</td>
</tr>
<tr>
<td>do statement (awk)</td>
<td>128</td>
</tr>
<tr>
<td>du (POSIX command)</td>
<td>280</td>
</tr>
<tr>
<td>dump</td>
<td></td>
</tr>
<tr>
<td>file contents (od)</td>
<td>584</td>
</tr>
<tr>
<td>hexadecimal (hd)</td>
<td>420</td>
</tr>
<tr>
<td>octal (hd)</td>
<td>420</td>
</tr>
<tr>
<td>dumpfs (POSIX command)</td>
<td>282</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ed</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBCDIC format</td>
<td>28</td>
</tr>
<tr>
<td>echo (POSIX command)</td>
<td>284</td>
</tr>
<tr>
<td>ed (POSIX command)</td>
<td>289</td>
</tr>
<tr>
<td>edit</td>
<td></td>
</tr>
<tr>
<td>cut field from line (cut)</td>
<td>250</td>
</tr>
<tr>
<td>mail (mailx)</td>
<td>493</td>
</tr>
<tr>
<td>mail messages (mailx)</td>
<td>498</td>
</tr>
<tr>
<td>editing, fold lines (fold)</td>
<td>392</td>
</tr>
<tr>
<td>editor</td>
<td></td>
</tr>
<tr>
<td>acoustic signal (vi)</td>
<td>813</td>
</tr>
<tr>
<td>addresses (ed)</td>
<td>292</td>
</tr>
<tr>
<td>addresses (vi)</td>
<td>830</td>
</tr>
<tr>
<td>addressing (ex)</td>
<td>331</td>
</tr>
<tr>
<td>alternate file (ex)</td>
<td>327</td>
</tr>
<tr>
<td>append text (ed)</td>
<td>295</td>
</tr>
<tr>
<td>buffer (ed)</td>
<td>290</td>
</tr>
<tr>
<td>buffer (ex)</td>
<td>330</td>
</tr>
<tr>
<td>buffer (vi)</td>
<td>815, 816</td>
</tr>
<tr>
<td>character-oriented positioning (vi)</td>
<td>818</td>
</tr>
<tr>
<td>command address (ex)</td>
<td>331</td>
</tr>
<tr>
<td>command mode (ed)</td>
<td>290</td>
</tr>
<tr>
<td>command mode (ex)</td>
<td>326</td>
</tr>
<tr>
<td>command mode (vi)</td>
<td>808, 820</td>
</tr>
<tr>
<td>command overview (ed)</td>
<td>293</td>
</tr>
<tr>
<td>command overview (ex)</td>
<td>334</td>
</tr>
<tr>
<td>command overview (vi)</td>
<td>804</td>
</tr>
<tr>
<td>command structure (ed)</td>
<td>291</td>
</tr>
<tr>
<td>current file (ex)</td>
<td>327</td>
</tr>
<tr>
<td>cursor positioning (vi)</td>
<td>818</td>
</tr>
<tr>
<td>customizing (vi)</td>
<td>839</td>
</tr>
</tbody>
</table>

| Display-oriented (vi)                | 803  |
| ed script (ed)                       | 303  |
| edit messages (mailx)                | 493  |
| ex command mode (vi)                 | 808  |
| ex input mode (vi)                   | 808  |
| input mode (ed)                      | 290  |
| input mode (ex)                      | 326  |
| input mode (vi)                      | 808  |
| insert text (ed)                     | 290  |
| insert text (ex)                     | 335, 336 |
| insert text (vi)                     | 826  |
| last line mode (vi)                  | 808  |
| line (ed)                            | 289  |
| line (ex)                            | 323  |
| line (sed)                           | 659  |
| line-oriented positioning (vi)       | 818  |
| lisp mode (vi)                       | 811  |
| mode selection (vi)                  | 808, 810 |
| presetting (ex)                      | 327  |
| quit (ed)                            | 290  |
| quit (ex)                            | 326  |
| quit (sed)                           | 664  |
| quit (vi)                            | 815  |
| range (sed)                          | 662  |
| read file (sed)                      | 664  |
| read in file (ed)                    | 295  |
| restore ex session (ex)              | 324  |
| save file (ed)                       | 290  |
| save file (ex)                       | 326  |
| save file (sed)                      | 665  |
| save file (vi)                       | 815  |
| screen layout (vi)                   | 814  |
| screen-oriented positioning (vi)     | 819  |
| search string (sed)                  | 664  |
| sed script (sed)                     | 661  |
| select mode (ed)                     | 290  |
| select mode (ex)                     | 326  |
| shell command execution (vi)         | 809  |
| status line (vi)                     | 823  |
| stream (sed)                         | 659  |
| vi session recovery (vi)             | 811  |
| window positioning (vi)              | 819  |
| word-oriented positioning (vi)       | 818  |

Editors 77
edt (POSIX command) 308
egrep (POSIX command) 310
encode
  files for mailing (uuencode) 800
env (POSIX command) 316
environment
  POSIX shell 60
  query or modify (command summary) 73
environment variable (env) 316
environment variable, see shell variable
environment, see user environment
EOF (trap) 759
eval (POSIX command) 320
evaluate expression (expr) 366
ex (POSIX command) 323
ex, options 345
exec (POSIX command) 353
execute commands
  at later time (at) 100
  at later time (batch) 151
  regularly (crontab) 238
execute permission
  at 100
  batch 151
  crontab 238
exit
  input mode (ed) 290
  exit (POSIX command) 358
  exit statement (awk) 129
exit status
  defining 358
  querying 18
  return not equal to 0 (false) 371
  return zero (: 868
  zero (true) 766
exp function (awk) 135
expand
  compressed files (uncompress) 787
expand (POSIX command) 361
exponential function (awk) 135
export (POSIX command) 363
export shell variable (export) 363
export, alias variable (alias) 92
exported alias variable 46
expr (POSIX command) 366
expression
  arithmetic 56
  awk 122
  bc expression (bc) 156
  conditional 57
  evaluate (expr) 366
  regular, see regular expression
  extended regular expression, see regular expression
F
false (POSIX command) 371
fc (POSIX command) 65, 372
fg (POSIX command) 376
fgrep (POSIX command) 377
field (awk) 114
field delimiters 251
field descriptor (date) 255
  modified 256
field separation
  default (awk) 114
  rules (awk) 115
field separator (awk) 114
field separators, rules (awk) 115
FIFO
  search for (find) 385
  FIFO making (mkfifo) 541
file
  calculate checksum (sum) 710
  change group ownership (chgrp) 202
  change mode (chmod) 204
  change owner (chown) 210
  check type (file) 382
  close (awk) 135
  commands for compressing and expanding 75
  commands for saving and archiving 75
  compress (compress) 224
  concatenate and print (cat) 197
  convert (dd) 260
  copy (cp) 189, 227
  copy (dd) 260
  cut column from line (cut) 250
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cut field from line (cut)</td>
<td>250</td>
</tr>
<tr>
<td>decode after mailing (uudecode)</td>
<td>798</td>
</tr>
<tr>
<td>decompress (uncompress)</td>
<td>787</td>
</tr>
<tr>
<td>deliver last part (tail)</td>
<td>717</td>
</tr>
<tr>
<td>display non-printing characters in octal (sed)</td>
<td>664</td>
</tr>
<tr>
<td>display opening lines (head)</td>
<td>423</td>
</tr>
<tr>
<td>display uncompressed (zcat)</td>
<td>866</td>
</tr>
<tr>
<td>display user file size limits (ulimit)</td>
<td>775</td>
</tr>
<tr>
<td>encode for mailing (uuencode)</td>
<td>800</td>
</tr>
<tr>
<td>find</td>
<td>385</td>
</tr>
<tr>
<td>find in directory (find)</td>
<td>385</td>
</tr>
<tr>
<td>find printable strings in binary files (strings)</td>
<td>699</td>
</tr>
<tr>
<td>format (pr)</td>
<td>618</td>
</tr>
<tr>
<td>join files on identical-valued field (join)</td>
<td>440</td>
</tr>
<tr>
<td>link (ln)</td>
<td>453</td>
</tr>
<tr>
<td>manage (ar)</td>
<td>94</td>
</tr>
<tr>
<td>manage and process (command summary)</td>
<td>74</td>
</tr>
<tr>
<td>mandatory locking (ls)</td>
<td>482</td>
</tr>
<tr>
<td>merge (sort)</td>
<td>688</td>
</tr>
<tr>
<td>merge lines (paste)</td>
<td>588</td>
</tr>
<tr>
<td>move (mv)</td>
<td>564</td>
</tr>
<tr>
<td>multi-column output (pr)</td>
<td>618</td>
</tr>
<tr>
<td>number lines (nl)</td>
<td>572</td>
</tr>
<tr>
<td>output (command summary)</td>
<td>74</td>
</tr>
<tr>
<td>output contents (command summary)</td>
<td>74</td>
</tr>
<tr>
<td>page through (more, page)</td>
<td>546</td>
</tr>
<tr>
<td>print (bs2lp)</td>
<td>190</td>
</tr>
<tr>
<td>print (Ip)</td>
<td>472</td>
</tr>
<tr>
<td>process (command summary)</td>
<td>74</td>
</tr>
<tr>
<td>processing (command summary)</td>
<td>74</td>
</tr>
<tr>
<td>query and modify properties (command summary)</td>
<td>75</td>
</tr>
<tr>
<td>r bit (chmod)</td>
<td>205</td>
</tr>
<tr>
<td>read in editor (sed)</td>
<td>664</td>
</tr>
<tr>
<td>remove/delete (rm)</td>
<td>650</td>
</tr>
<tr>
<td>rename (mv)</td>
<td>564</td>
</tr>
<tr>
<td>s bit (chgrp)</td>
<td>202</td>
</tr>
<tr>
<td>s bit (chmod)</td>
<td>205</td>
</tr>
<tr>
<td>save in editor (ed)</td>
<td>290</td>
</tr>
<tr>
<td>save in editor (ex)</td>
<td>326</td>
</tr>
<tr>
<td>save in editor (sed)</td>
<td>665</td>
</tr>
<tr>
<td>save in editor (vi)</td>
<td>815</td>
</tr>
<tr>
<td>search (egrep)</td>
<td>310</td>
</tr>
<tr>
<td>search (fgrep)</td>
<td>377</td>
</tr>
<tr>
<td>search (grep)</td>
<td>411</td>
</tr>
<tr>
<td>search for (find)</td>
<td>385</td>
</tr>
<tr>
<td>search for common lines in files (comm)</td>
<td>217</td>
</tr>
<tr>
<td>search for repeated lines (uniq)</td>
<td>791</td>
</tr>
<tr>
<td>set mode mask (umask)</td>
<td>777</td>
</tr>
<tr>
<td>set user file size limits (ulimit)</td>
<td>775</td>
</tr>
<tr>
<td>set-group-ID bit (chmod)</td>
<td>204</td>
</tr>
<tr>
<td>set-user-ID/set-group-ID bit (chmod)</td>
<td>205</td>
</tr>
<tr>
<td>set-user-ID/set-group-ID bits (chmod)</td>
<td>207</td>
</tr>
<tr>
<td>show as character strings (hd)</td>
<td>420</td>
</tr>
<tr>
<td>show in decimal (hd)</td>
<td>420</td>
</tr>
<tr>
<td>show in decimal (od)</td>
<td>584</td>
</tr>
<tr>
<td>show in hexadecimal (hd)</td>
<td>420</td>
</tr>
<tr>
<td>show in hexadecimal (od)</td>
<td>584</td>
</tr>
<tr>
<td>show in octal (hd)</td>
<td>420</td>
</tr>
<tr>
<td>show in octal (od)</td>
<td>584</td>
</tr>
<tr>
<td>sort (sort)</td>
<td>688</td>
</tr>
<tr>
<td>split (csplit)</td>
<td>245</td>
</tr>
<tr>
<td>split (split)</td>
<td>695</td>
</tr>
<tr>
<td>sticky bit (chmod)</td>
<td>205, 208</td>
</tr>
<tr>
<td>sticky bit (ls)</td>
<td>482</td>
</tr>
<tr>
<td>t bit (chmod)</td>
<td>205, 208</td>
</tr>
<tr>
<td>t bit (ls)</td>
<td>482</td>
</tr>
<tr>
<td>test attributes</td>
<td>57</td>
</tr>
<tr>
<td>update access time (touch)</td>
<td>742</td>
</tr>
<tr>
<td>update modification time (touch)</td>
<td>742</td>
</tr>
<tr>
<td>w bit (chmod)</td>
<td>205</td>
</tr>
<tr>
<td>write checksum (cksum)</td>
<td>212</td>
</tr>
<tr>
<td>x bit (chmod)</td>
<td>205</td>
</tr>
<tr>
<td>file (POSIX command)</td>
<td>382</td>
</tr>
<tr>
<td>file attributes, test (test)</td>
<td>732</td>
</tr>
<tr>
<td>file comparison (diff)</td>
<td>274</td>
</tr>
<tr>
<td>file descriptor 0</td>
<td>36</td>
</tr>
<tr>
<td>file descriptor 1</td>
<td>36</td>
</tr>
<tr>
<td>file descriptor 2</td>
<td>38</td>
</tr>
<tr>
<td>file editing</td>
<td>659</td>
</tr>
</tbody>
</table>

Index
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>merge (sort)</td>
<td>688</td>
</tr>
<tr>
<td>sort (sort)</td>
<td>688</td>
</tr>
<tr>
<td>transliterate characters (tr)</td>
<td>752</td>
</tr>
<tr>
<td>file name</td>
<td></td>
</tr>
<tr>
<td>generate</td>
<td>54</td>
</tr>
<tr>
<td>strip from full access path (dirname)</td>
<td>279</td>
</tr>
<tr>
<td>strip suffix (basename)</td>
<td>149</td>
</tr>
<tr>
<td>file processing</td>
<td></td>
</tr>
<tr>
<td>command summary</td>
<td>74</td>
</tr>
<tr>
<td>cut column from line (cut)</td>
<td>250</td>
</tr>
<tr>
<td>cut field from line (cut)</td>
<td>250</td>
</tr>
<tr>
<td>search for common lines (comm)</td>
<td>217</td>
</tr>
<tr>
<td>file processing (awk)</td>
<td>113</td>
</tr>
<tr>
<td>file size limit</td>
<td></td>
</tr>
<tr>
<td>check for current shell (ulimit)</td>
<td>775</td>
</tr>
<tr>
<td>file size, write (cksum)</td>
<td>212</td>
</tr>
<tr>
<td>file system</td>
<td></td>
</tr>
<tr>
<td>display information (dumpfs)</td>
<td>282</td>
</tr>
<tr>
<td>manage (command summary)</td>
<td>76</td>
</tr>
<tr>
<td>modify (command summary)</td>
<td>76</td>
</tr>
<tr>
<td>mount (mount)</td>
<td>551</td>
</tr>
<tr>
<td>mount (mountall)</td>
<td>562</td>
</tr>
<tr>
<td>unmount (umount)</td>
<td>781</td>
</tr>
<tr>
<td>unmount (umountall)</td>
<td>782</td>
</tr>
<tr>
<td>file system check (fsck)</td>
<td>395</td>
</tr>
<tr>
<td>file-creation mode mask</td>
<td>778</td>
</tr>
<tr>
<td>set (umask)</td>
<td>778</td>
</tr>
<tr>
<td>files, prepare for printing (pr)</td>
<td>618</td>
</tr>
<tr>
<td>find (POSIX command)</td>
<td>385</td>
</tr>
<tr>
<td>flush system buffers</td>
<td>712</td>
</tr>
<tr>
<td>fold (POSIX command)</td>
<td>392</td>
</tr>
<tr>
<td>for statement (awk)</td>
<td>129, 130</td>
</tr>
<tr>
<td>format</td>
<td></td>
</tr>
<tr>
<td>files</td>
<td>618</td>
</tr>
<tr>
<td>hash table (hash)</td>
<td>417</td>
</tr>
<tr>
<td>line (fold)</td>
<td>392</td>
</tr>
<tr>
<td>merge lines (paste)</td>
<td>588</td>
</tr>
<tr>
<td>number lines (nl)</td>
<td>572</td>
</tr>
<tr>
<td>of a sed script</td>
<td>661</td>
</tr>
<tr>
<td>output (printf)</td>
<td>626</td>
</tr>
<tr>
<td>format elements, for printf</td>
<td>627</td>
</tr>
<tr>
<td>formatted output (awk)</td>
<td>140</td>
</tr>
<tr>
<td>formatted output as string (awk)</td>
<td>142</td>
</tr>
<tr>
<td>free and reserved disk blocks</td>
<td></td>
</tr>
<tr>
<td>displaying (df)</td>
<td>267</td>
</tr>
<tr>
<td>fsck (POSIX command)</td>
<td>395</td>
</tr>
<tr>
<td>fsexpand (POSIX command)</td>
<td>397</td>
</tr>
<tr>
<td>ftyp (POSIX command)</td>
<td>399</td>
</tr>
<tr>
<td>function</td>
<td></td>
</tr>
<tr>
<td>arc tangent (awk)</td>
<td>135</td>
</tr>
<tr>
<td>arithmetic (awk)</td>
<td>134</td>
</tr>
<tr>
<td>atan2 (awk)</td>
<td>135</td>
</tr>
<tr>
<td>bc function (bc)</td>
<td>156</td>
</tr>
<tr>
<td>built-in (awk)</td>
<td>134</td>
</tr>
<tr>
<td>calculate square root (awk)</td>
<td>143</td>
</tr>
<tr>
<td>calculator (bc)</td>
<td>154</td>
</tr>
<tr>
<td>call shell command (awk)</td>
<td>144</td>
</tr>
<tr>
<td>cosine (awk)</td>
<td>135</td>
</tr>
<tr>
<td>define substring (awk)</td>
<td>144</td>
</tr>
<tr>
<td>exp (awk)</td>
<td>135</td>
</tr>
<tr>
<td>formatted output (awk)</td>
<td>140</td>
</tr>
<tr>
<td>formatted output as a string (awk)</td>
<td>142</td>
</tr>
<tr>
<td>general (awk)</td>
<td>134</td>
</tr>
<tr>
<td>getline (awk)</td>
<td>136</td>
</tr>
<tr>
<td>index (awk)</td>
<td>137</td>
</tr>
<tr>
<td>int (awk)</td>
<td>138</td>
</tr>
<tr>
<td>length (awk)</td>
<td>138</td>
</tr>
<tr>
<td>log (awk)</td>
<td>138</td>
</tr>
<tr>
<td>logarithm (awk)</td>
<td>138</td>
</tr>
<tr>
<td>pattern matching (awk)</td>
<td>138</td>
</tr>
<tr>
<td>POSIX shell</td>
<td>61</td>
</tr>
<tr>
<td>print (awk)</td>
<td>139</td>
</tr>
<tr>
<td>printf (awk)</td>
<td>140</td>
</tr>
<tr>
<td>rand (awk)</td>
<td>141</td>
</tr>
<tr>
<td>return length (awk)</td>
<td>138</td>
</tr>
<tr>
<td>search for substrings (awk)</td>
<td>137</td>
</tr>
<tr>
<td>set seed for rand (awk)</td>
<td>143</td>
</tr>
<tr>
<td>shell function</td>
<td>888</td>
</tr>
<tr>
<td>shell function delete (unset)</td>
<td>794</td>
</tr>
<tr>
<td>sine (awk)</td>
<td>142</td>
</tr>
<tr>
<td>split (awk)</td>
<td>142</td>
</tr>
<tr>
<td>sprintf function (awk)</td>
<td>142</td>
</tr>
<tr>
<td>sqrt (awk)</td>
<td>143</td>
</tr>
<tr>
<td>srand (awk)</td>
<td>143</td>
</tr>
<tr>
<td>standard output function (awk)</td>
<td>139</td>
</tr>
<tr>
<td>string function (awk)</td>
<td>134</td>
</tr>
<tr>
<td>sub (awk)</td>
<td>143</td>
</tr>
</tbody>
</table>
Index

subdivide strings (awk) 142
substitution (awk) 143
substr (awk) 144
system (awk) 144

G
gencat (POSIX command) 401
general awk functions 134
generation, file name 54
getconf (POSIX command) 404
getline function (awk) 136
getopts (POSIX command) 408
GID, change (chgrp) 202
global program cache
setting up/administering 611, 614
global substitution function (awk) 137
grep (POSIX command) 411
group ID
change (chgrp) 202
print (id) 427
group name, print (id) 427
group, change (newgrp) 568

H
hard link (ln) 453
hash (POSIX command) 416
hash table
delete (hash) 417
format (hash) 417
process (hash) 416
hd (POSIX command) 420
head (POSIX command) 423
header (mailx) 489
help, on-line documentation 80
here document 37
heterogeneous network 11
hexadecimal dump (hd) 420
hexadecimal dump (od) 584
history file 65
access (fc) 372

I
iconv (POSIX command) 425
id (POSIX command) 427
IEEE 11
include files (make) 529
index function (awk) 137
info (POSIX command) 430
information, locale-specific (locale) 462
inode
create (mknod) 544
displaying (df) 267
number (ln) 457
inode (ln) 457
input
after command invocation 33
copy (tee) 730
join pipes (tee) 730
options 34
read (read) 643
redirecting (exec) 354
redirection 36
input file (awk) 113
input function (awk) 134
input mode
exit (ed) 290
quit (ed) 290
input mode (ed) 290
input mode (ex) 326
input mode (vi) 808
insert
text (ex) 335, 336
int function (awk) 138
integer
algebraic comparison (test) 732
perform calculations with (expr) 366
truncate to (awk) 138
integer arithmetic 56
internationalization (NLS) 85
interpret carriage-control characters 99
inter-process communication
remove facilities 432
show status 434
invocation, command 33
ipcrm (POSIX command) 432
IPv6 address 640
iteration (awk) 129
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>J</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete (crontab)</td>
<td>240</td>
</tr>
<tr>
<td>display status (jobs)</td>
<td>439</td>
</tr>
<tr>
<td>run in foreground (fg)</td>
<td>376</td>
</tr>
<tr>
<td>scheduled (crontab)</td>
<td>238</td>
</tr>
<tr>
<td>specify for background processing (bg)</td>
<td>166</td>
</tr>
</tbody>
</table>

| J | job management, command summary | 79 |

| J | jobs (POSIX command) | 439 |

| J | jobs, referencing | 62 |

| J | join (POSIX command) | 440 |

| J | join files on identical-valued field (join) | 440 |

| K | kill (POSIX command) | 444 |

| K | kill process (kill) | 444 |

| L | I bit (chmod) | 208 |

| L | large file aware | 66 |

| L | large file safe | 66 |

| L | Lempel-Ziv coding (compress) | 224 |

| L | length function (awk) | 138 |

| L | let (POSIX command) | 448 |

| L | lex (POSIX command) | 450 |

<table>
<thead>
<tr>
<th>L</th>
<th>library</th>
</tr>
</thead>
<tbody>
<tr>
<td>archive library (make)</td>
<td>532</td>
</tr>
<tr>
<td>library functions</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>change (ulimit)</td>
<td>775</td>
</tr>
<tr>
<td>set (ulimit)</td>
<td>775</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>line</th>
</tr>
</thead>
<tbody>
<tr>
<td>count (wc)</td>
<td>846</td>
</tr>
<tr>
<td>cut field (cut)</td>
<td>250</td>
</tr>
<tr>
<td>fold</td>
<td>392</td>
</tr>
<tr>
<td>merge (paste)</td>
<td>588</td>
</tr>
<tr>
<td>merge lines (paste)</td>
<td>588</td>
</tr>
<tr>
<td>numbering (nl)</td>
<td>572</td>
</tr>
<tr>
<td>paste</td>
<td>588</td>
</tr>
<tr>
<td>report repeated (uniq)</td>
<td>791</td>
</tr>
<tr>
<td>search for (egrep)</td>
<td>310</td>
</tr>
<tr>
<td>search for (fgrep)</td>
<td>377</td>
</tr>
<tr>
<td>search for (grep)</td>
<td>411</td>
</tr>
</tbody>
</table>

| L | show (egrep) | 310 |
|---|---|
| show (fgrep) | 377 |
| show (grep) | 411 |

<table>
<thead>
<tr>
<th>L</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>hard link (ln)</td>
<td>453</td>
</tr>
<tr>
<td>inode (ln)</td>
<td>457</td>
</tr>
<tr>
<td>inode number (ln)</td>
<td>457</td>
</tr>
<tr>
<td>make (ln)</td>
<td>453</td>
</tr>
<tr>
<td>symbolic link (ln)</td>
<td>453</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>names of UUCP systems (uuname)</td>
<td>802</td>
</tr>
<tr>
<td>list contents of directory (ls)</td>
<td>480</td>
</tr>
<tr>
<td>ln (POSIX command)</td>
<td>453</td>
</tr>
<tr>
<td>load file (editor)</td>
<td>295</td>
</tr>
<tr>
<td>local environment, define (localedef)</td>
<td>466</td>
</tr>
<tr>
<td>locale (POSIX command)</td>
<td>462</td>
</tr>
<tr>
<td>localedef (POSIX command)</td>
<td>466</td>
</tr>
<tr>
<td>localization (NLS)</td>
<td>85</td>
</tr>
<tr>
<td>log function (awk)</td>
<td>138</td>
</tr>
<tr>
<td>log messages</td>
<td>470</td>
</tr>
<tr>
<td>logarithm (awk)</td>
<td>138</td>
</tr>
<tr>
<td>logger (POSIX command)</td>
<td>470</td>
</tr>
<tr>
<td>login name</td>
<td></td>
</tr>
<tr>
<td>list active users (who)</td>
<td>849</td>
</tr>
<tr>
<td>print (id)</td>
<td>427</td>
</tr>
<tr>
<td>return (logname)</td>
<td>471</td>
</tr>
<tr>
<td>login shell</td>
<td>679</td>
</tr>
<tr>
<td>logname (POSIX command)</td>
<td>471</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>continue statement (awk)</td>
<td>128</td>
</tr>
<tr>
<td>do statement (awk)</td>
<td>128</td>
</tr>
<tr>
<td>POSIX shell</td>
<td>41</td>
</tr>
<tr>
<td>skip remainder (awk)</td>
<td>128</td>
</tr>
<tr>
<td>while statement (awk)</td>
<td>132</td>
</tr>
<tr>
<td>loop counter (awk)</td>
<td>129</td>
</tr>
<tr>
<td>loop termination (awk)</td>
<td>127</td>
</tr>
<tr>
<td>Ip (POSIX command)</td>
<td>472</td>
</tr>
<tr>
<td>lpsstat (POSIX command)</td>
<td>477</td>
</tr>
<tr>
<td>ls (POSIX command)</td>
<td>480</td>
</tr>
</tbody>
</table>

| M | machine type, output name (uname) | 785 |

| M | macro definitions (make) | 530 |
|---|---|
| macro substitution mechanism (make) | 533 |
Index

mail
  alias (mailx) 496
  default values of variables (mailx) 516
  define editor (mailx) 523
  delete (mailx) 494, 497
  diagnostics (mailx) 518
  display (mailx) 493, 501
  edit (mailx) 493, 498, 506
  edit message (mailx) 498
  invoke shell (mailx) 494
  processing status (mailx) 508
  read (mailx) 489, 490
  read mode (mailx) 496
  read-mode functionality (mailx) 507
  reply to (mailx) 494, 499, 503
  save (mailx) 493, 497, 503, 506
  send (mailx) 489, 494
  send mode (mailx) 501, 509
  variables (mailx) 504

mailx (POSIX command) 488
make (POSIX command) 526
makefile 526
  create 527
  default rules 531
  directives 528
  environment variables 529
  execute 528
  include files 529
  internal macros 530
  macro assignment 529
  macro definitions 530

man (POSIX command) 535
match function (awk) 138
merge
  files (sort) 688
  lines (paste) 588
mesg (POSIX command) 537
message
  delete (mailx) 494
  deny (mesg) 537
  display (mailx) 501
  edit (mailx) 493, 506
  generate message catalog (gencat) 401
  log (logger) 470
  permit (mesg) 537
  reply to (mailx) 494, 499, 503
  save (mailx) 493, 503
  send (mailx) 494
  send to another user (write) 854
message catalog
  generate (gencat) 401
message catalog (NLS) 85
message header (mailx) 489
message queue 434
message queues 432
metacharacters
  for printf 626
  for regular expressions 881
  for the POSIX shell 884
  quoting 55, 888
mkdir (POSIX command) 539
mkfifo (POSIX command) 541
mkfs (POSIX command) 543
mknod (POSIX command) 544
mode mask
  default setting (umask) 778
mode syntax
  absolute (chmod) 206
  symbolic (chmod) 204
  mode, change (chmod) 204
  more (POSIX command) 546
mount
  file system (mount) 551
  file system (mountall) 562
  remote resource (mount) 551
mount (POSIX command) 551
mountall (POSIX command) 562
move, file (mv) 564
mv (POSIX command) 564

N
Native Language System (NLS) 85
network
  communicate with other users (talk) 721
  decode files (uudecode) 798
  encode files (uuencode) 800
  heterogeneous 11
  inter-user communication (write) 854
list UUCP names (uname)  802
output node name (uname)  785
newgrp (POSIX command)  568
next statement (awk)  131
nice (POSIX command)  571
nl (POSIX command)  572
nm (POSIX command)  579
nohup (POSIX command)  582
number, lines (nl)  572

doct
object file, name list (nm)  579
octal
dump (hd)  420
dump (od)  584
represent non-printing characters (sed)  664
od (POSIX command)  584
on-line documentation  80
operating system
output of basic data (uname)  785
operation
block-mode terminal  27
character-mode terminal  27
operator (awk)  123
operator, logical (awk)  125
option, input  34
options, get and parse (getopts)  408
output
alias variable (alias)  92
data display terminal options (stty)  701
formatted (awk)  140
formatted (printf)  626
left-adjusted (printf)  627
operating system data  785
redirecting  36
redirecting (exec)  354
redirection (awk)  139
right-adjusted (printf)  627
shell variable (set)  670
output function (awk)  134
overlay
current shell (exec)  353
current shell (newgrp)  568
owner, change (chown)  210

P
P keys, setting  28
packages
information on  611
page
text file (more, page)  546
parameter
set positional parameters (set)  670
substitution  48
parameters, positional (shift)  682
parent process  39
parent-child process hierarchy  39
partition, remove (rpmart)  654
password to access shell  25
paste (POSIX command)  588
patch (POSIX command)  593
path name of working directory (pwd)  638
path prefixes
strip (basename)  149
pathchk (POSIX command)  597
pathname check (pathchk)  597
pattern  877
compound (awk)  125
regular expression (awk)  124
relational expression (awk)  124
search for (egrep)  310
search for (grep)  411
pattern (awk)  124
pattern matching (awk)  124, 138
pattern range (awk)  125
pax (POSIX command)  600
pdbl  607
permissions
change (chmod)  204
set (umask)  778
pipe
close (awk)  135
copy input (tee)  730
join pipes (tee)  730
two-way  38
pipeline  32
POSIX shell  44
pkginfo (POSIX command)  611
posdbl  611, 614
Index

positional parameters
  shift values (shift) 682

POSIX
  definition 11
  process structure 39
  shell 11
  software interfaces 11
  software product 11
  standard 11
  subsystem 11

POSIX command
  pdbl 607
  posdbl 614

POSIX commands
  posdbl 611
  summary 72
  XPG4 conformity 873

POSIX commands, see commands

POSIX packages
  information on 611

POSIX shell
  alias 45
  aliasing 45
  arithmetic evaluation 56
  array 48
  background process 38
  blank interpretation 53
  break 41, 43
  calling 24
  case 42
  character substitution 54
  command execution 64
  command re-entry 65
  command substitution 47
  command summary 72
  compound commands 41
  compound expression 59
  continue 43
  environment 60
  export 60
  expression 57
  fg 62
  file name generation 54
  for 41, 62
  function 61
  function definition 44
  HISTFILE 374
  HISTSIZE 374
  HOME 52
  if 42
  job 62
  jobs 63
  loop 41
  monitor 62
  noclobber 36
  parameter substitution 48
  pipeline 44
  processing commands 40
  redirecting input 36
  redirecting output 36
  return 43, 61
  select 41
  shift positional parameters (shift) 682
  signal handling 63
  starting 24
  stopped 62
  string 58
  subscript 48
  terminating 25
  terminating (exit) 358
  trap 63
  typeset 56, 60, 61, 448
  unalias 46
  unset 61
  until 43
  variable substitution 48
  variable, see shell variable
  while 42

pr (POSIX command) 618

precedence
  when combining conditions 736

presettings 27

print (POSIX command) 625

print checksum and block count of a file
  (sum) 710

print function (awk) 139
Index

print job
  cancel (cancel) 195
  display status information (lpstat) 477
  sending (bs2lp) 190
  submit (lp) 472
printable strings
  find in binary files (strings) 699
printer
  display status information (lpstat) 477
printf (POSIX command) 626
printf function (awk) 140
printing
  prepare files for (pr) 618
priority
  for extended regular expressions 883
  for regular expressions 880
  running processes (renice) 649
PRNT macro (BS2000) 190
process
  child process 39
  control (command summary) 79
  kill (kill) 444
  output total runtime (times) 741
  outputting information (command summary) 79
  parent-child process hierarchy 39
  report status (ps) 630
  send signal (kill) 444
  spawning 39
  suspend (sleep) 686
  wait for background processes to terminate (wait) 843
process structure 39
processor, output type (uname) 785
profile file, executing 27
program
  flow control (awk) 127
  internationalization (NLS) 85
  localization (NLS) 85
  message catalog (NLS) 85
  regular execution (crontab) 238
  terminate (awk) 129
  update (make) 526
program structure
  BEGIN section (awk) 112
  END section (awk) 112
  main section (awk) 112
programming language
  arithmetic function (awk) 134
  input functions (awk) 134
  output function (awk) 134
  string function (awk) 134
prompt, POSIX shell 40, 53
protecting environment variables (readonly) 647
ps (POSIX command) 630
pwd (POSIX command) 638
Q
  querying and modifying user properties 78
queue (batch) 151
quoting 55
  arithmetic operators 56
  metacharacter 55
R
  r bit (chmod) 205
  rand function (awk) 141
random number
  return (awk) 141
  return (rand) 141
range
  inclusive (sed) 662
rcp (POSIX command) 639
read (POSIX command) 643
readonly (POSIX command) 647
record (awk) 113
record processing (awk) 131
record separation, rules (awk) 115
record separator (awk) 113
record separators, rules (awk) 115
redirecting
  standard input 36, 353, 886
  standard output 36, 353, 885
Index

regular expression 877
  extended 882
  extended (awk) 124
  extended, precedence 883
metacharacters 881
overview 877
precedence 880
search for (egrep) 310
search for (fgrep) 377
search for (grep) 411
simple 878
regular expression (ed) 292
regular expression (ex) 328
regular expression (more, page) 547
regular expressions (awk) 138
remove
directory (rm) 650
files (rm) 652
rename
directory (mv) 566
file (mv) 564
renice (POSIX command) 649
repeated lines, report (uniq) 791
resource
  mount 551
  unmount (umount) 781
restrictive shell 679
return (built-in POSIX command) 43
rlogin 25
rm (POSIX command) 650
rmdir (POSIX command) 652
rsh (POSIX command) 655
runtime
  measure a shell procedure (time) 739
  measure command (times) 741
S
  s bit (chgrp) 202
  s bit (chmod) 205, 207
  save
    buffer (vi) 815
    buffer contents (ed) 290
    buffer contents (ex) 326
    file (ed) 290
  file (ex) 326
  file (sed) 665
  file (vi) 815
  mail (mailx) 493, 497, 506
scan
directories (find) 385
  directory (find) 385
  scheduling command execution (at) 100
screen layout (vi) 814
script
  ed script (ed) 303
  here script (ed) 305
screen layout (more, page) 546
SDF-P variable SYSPOSIX 27
search
  file (egrep) 310
  file (fgrep) 377
  file (grep) 411
  for pattern (egrep) 310
  for pattern (fgrep) 377
  for pattern (grep) 411
  for printable strings in binary files
    (strings) 699
  for repeated lines (uniq) 791
  for string (ed) 300
  for string (egrep) 310
  for string (ex) 329
  for string (fgrep) 377
  for string (grep) 411
  for string (sed) 664
SECOS 25
sed
  commands 663
  functionality 661
  sed (POSIX command) 659
  sed script
    format 661
    selection conditions, compare (awk) 124
    selection criterion (awk) 124
    semaphores 432, 434
    send, signal (kill) 444
    separators for commands and arguments 884
    set (POSIX command) 670
    set-group-ID bit (chmod) 204, 205, 207
Index

set-user-ID bit (chmod) 205, 207
sh (POSIX command) 679
shared memory 432, 434
shell
  auxiliary commands (summary) 77
  check limit (ulimit) 775
  command execution (mailx) 504
  delete shell function (unset) 794
  delete shell variable (unset) 794
  execute script in current shell (ZH) 870
  overlay (exec) 353
  overlay (newgrp) 568
  process hash table (hash) 416
  protect shell variables (readonly) 647
  read from standard input (read) 643
  restrictive 679
  see also POSIX shell
  set option (set) 670
  set positional parameters (set) 670
  start (sh) 679
shell command
  call (awk) 144
shell function, see function
shell procedure 22, 39
  auxiliary commands (summary) 77
  execute in current shell (.) 870
  here document 37
  measure runtime (time) 741
shell procedures, passing variables 39
shell script 22
  signal handling (trap) 758
  terminate (exit) 358
shell variable 48
  assign value (read) 643
  CD-PATH 51
  COLUMNS 51
  default values 53
  delete (unset) 794
  display (set) 670
  EDITOR 51
  ENV 51
  ERRNO 50
  examine values (echo) 284
  export (export) 363
  FCEDIT 51, 65, 374
  FDPATH 51
  HISTFILE 51
  HISTSIZE 52, 65
  IFS 52
  in shell procedure 39
  initializing 27
  IO_CONVERSION 29, 52
  LINENO 50
  LINES 52
  MAIL 52
  MAILCHECK 52
  MAILPATH 52
  OLDPWD 50
  OPTARG 50
  OPTIND 50
  PATH 52
  PPID 50
  protect (readonly) 647
  PS1 53
  PS2 53
  PS3 53
  PS4 53
  PWD 50
  RANDOM 50
  REPLY 51
  SECONDS 51
  set attributes for (typeset) 773
  SHELL 53
  TMOUT 53
  VISUAL 53
  shift (POSIX command) 682
  shifting (NLS) 86
  show_pubset_export (POSIX command) 684
  SIGHUP (kill) 444
  SIGHUP (trap) 759
  SIGINT (kill) 444
  SIGINT (mailx) 509, 519
  SIGINT (trap) 759
  SIGKILL (kill) 444
  signal
    EOF (trap) 759
    ignore (nohup) 582
    send to process (kill) 444
Index

shell script (trap) 758
SIGHUP (kill) 444
SIGHUP (trap) 759
SIGINT (kill) 444
SIGINT (mailx) 509, 519
SIGINT (trap) 759
SIGKILL (kill) 444
signal handling (trap) 758
SIGQUIT (kill) 444
SIGQUIT (trap) 759
SIGTERM (kill) 444
SIGTERM (trap) 759
signal handling
  shell script (trap) 758
SIGQUIT (kill) 444
SIGQUIT (trap) 759
SIGTERM (kill) 444
SIGTERM (trap) 759
sine (awk) 142
sleep (POSIX command) 686
software interfaces
  BS2000 11
  POSIX 11
software project
  update (make) 529
sort (POSIX command) 688
sort files (sort) 688
sorted files
  common lines in (comm) 217
special file
  search for (find) 385
special variable (awk) 119
split
  file (csplit) 245
  file (split) 695
split (POSIX command) 695
split function (awk) 142
sqrt function (awk) 143
square root calculation (awk) 143
rand function (awk) 143
SRPM 25

standard
  POSIX 11
  UNIX95 11
  XPG4 11
  XPG4.2 11
standard input
  read (read) 643
  redirect 36, 886
  redirecting (exec) 354
standard output
  redirect 885
  redirecting 36
  redirecting (exec) 354
standard output function (awk) 139
start of a file
  display (head) 423
start_bs2fsd (POSIX command) 698
START-POSIX-SHELL (BS2000 command) 24
statement
  awk statement (awk) 126
  bc statement (bc) 155
status report for processes (ps) 630
status, message (mailx) 508
stderr 38
stdin 38
stdout 38
sticky bit (chmod) 205, 208
sticky bit (ls) 482
storage, checking availability (command summary) 80
stream editor (sed) 659
string
  attribute 58
  comparing (expt) 366
  comparison 58
  comparison (test) 732
  find in binary files (strings) 699
  search for (ed) 300
  search for (egrep) 310
  search for (ex) 329
  search for (fgrep) 377
  search for (grep) 411
  search for (sed) 664
  search for (vi) 836
Index

strings (POSIX command) 699
strip suffix from file name (basename) 149
structure of descriptions
POSIX commands 15
stty (POSIX command) 701
sub function (awk) 143
subdivide strings (awk) 142
subshell
call (sh) 679
terminating (exit) 358
substitution
alias 45
command 47
parameter 48
tilde 46
substitution function (awk) 143
substitution function, global (awk) 137
substr function (awk) 144
substring definition (awk) 144
substring search (awk) 137
suffixes of file name (basename) 149
sum (POSIX command) 710
super block
display information (dumpfs) 282
suspend process execution (sleep) 686
symbolic link (ln) 453
symbolic mode syntax (chmod) 204
sync (POSIX command) 712
SYSSRPM file 25
system
show time of last reboot (who) 849
system buffer flushing 80
system data, displaying information (ps) 80
system function (awk) 144
system names, output (uname) 785
T
t bit (chmod) 205, 208
t bit (ls) 482
tab, set tab stops (tabs) 713
table of all defined file systems 558
tabs (POSIX command) 713	ail (POSIX command) 717
talk (POSIX command) 721
talk, with another user (talk) 721
tee (POSIX command) 730
telnet 26
terminal
block-mode 27
check options (stty) 701
display capabilities (tput) 746
display name (tput) 746
initialize (tput) 746
modify attributes 80
output path name of terminal (tty) 769
query terminfo database (tput) 746
reset (tput) 746
setting P keys 80
terminal file 544
terminating
POSIX shell 25
POSIX shell (exit) 358
shell script (exit) 358
subshell (exit) 358
terminfo database
query (tput) 746
test
conditions ([ ]) 872
conditions (test) 732
file attributes (test) 732
test (POSIX command) 732
text editing
display editor (vi) 803
line editor (ed) 289
line editor (ex) 323
line editor (sed) 659
merge lines (paste) 588
text manipulation (awk) 107
text processing
cut column from line (cut) 250
text, insert (ex) 335, 336
tilde
mailx commands (mailx) 510
substitution 46
<table>
<thead>
<tr>
<th>time</th>
<th>commands/shell scripts (time)</th>
<th>739</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>print (date)</td>
<td>254</td>
</tr>
<tr>
<td>time</td>
<td>(POSIX command)</td>
<td>739</td>
</tr>
<tr>
<td></td>
<td>command execution (crontab)</td>
<td>238</td>
</tr>
<tr>
<td>times</td>
<td>(POSIX command)</td>
<td>741</td>
</tr>
<tr>
<td>touch</td>
<td>(POSIX command)</td>
<td>742</td>
</tr>
<tr>
<td>tput</td>
<td>(POSIX command)</td>
<td>746</td>
</tr>
<tr>
<td>tr</td>
<td>(POSIX command)</td>
<td>752</td>
</tr>
<tr>
<td>trap</td>
<td>(POSIX command)</td>
<td>758</td>
</tr>
<tr>
<td>true</td>
<td>(POSIX command)</td>
<td>766</td>
</tr>
<tr>
<td>tsort</td>
<td>(POSIX command)</td>
<td>767</td>
</tr>
<tr>
<td>tty</td>
<td>(POSIX command)</td>
<td>769</td>
</tr>
<tr>
<td></td>
<td>two-way pipe</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>type, check file type (file)</td>
<td>382</td>
</tr>
<tr>
<td></td>
<td>typeset (POSIX command)</td>
<td>773</td>
</tr>
<tr>
<td>U</td>
<td>ulimit (POSIX command)</td>
<td>775</td>
</tr>
<tr>
<td></td>
<td>umask (POSIX command)</td>
<td>778</td>
</tr>
<tr>
<td></td>
<td>umount (POSIX command)</td>
<td>781</td>
</tr>
<tr>
<td></td>
<td>umountall (POSIX command)</td>
<td>782</td>
</tr>
<tr>
<td></td>
<td>unalias (POSIX command)</td>
<td>783</td>
</tr>
<tr>
<td></td>
<td>uname (POSIX command)</td>
<td>785</td>
</tr>
<tr>
<td></td>
<td>uncompress (POSIX command)</td>
<td>787</td>
</tr>
<tr>
<td></td>
<td>unexpand (POSIX command)</td>
<td>789</td>
</tr>
<tr>
<td></td>
<td>uniq (POSIX command)</td>
<td>791</td>
</tr>
<tr>
<td></td>
<td>UNIX95 standard</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>unmount</td>
<td></td>
</tr>
<tr>
<td></td>
<td>file system (umount)</td>
<td>781</td>
</tr>
<tr>
<td></td>
<td>remote resource (umountall)</td>
<td>782</td>
</tr>
<tr>
<td></td>
<td>several file systems (umountall)</td>
<td>782</td>
</tr>
<tr>
<td></td>
<td>unset (POSIX command)</td>
<td>794</td>
</tr>
<tr>
<td></td>
<td>update</td>
<td></td>
</tr>
<tr>
<td></td>
<td>access and modification</td>
<td>742</td>
</tr>
<tr>
<td></td>
<td>times (touch)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list active users (who)</td>
<td>849</td>
</tr>
<tr>
<td></td>
<td>send messages to other</td>
<td>854</td>
</tr>
<tr>
<td></td>
<td>users (write)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>talk to another user (talk)</td>
<td>721</td>
</tr>
<tr>
<td></td>
<td>user environment, current</td>
<td>316</td>
</tr>
<tr>
<td></td>
<td>(env)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user group</td>
<td></td>
</tr>
<tr>
<td></td>
<td>change for directory</td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>(chgrp)</td>
<td></td>
</tr>
<tr>
<td>user ID</td>
<td>effective (chmod)</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>entering (adduser)</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>print (id)</td>
<td>427</td>
</tr>
<tr>
<td></td>
<td>real (chmod)</td>
<td>207</td>
</tr>
<tr>
<td>user number</td>
<td>add (adduser)</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>user-specific program</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td>setting up/administering</td>
<td>607</td>
</tr>
<tr>
<td></td>
<td>usp (POSIX command)</td>
<td>796</td>
</tr>
<tr>
<td></td>
<td>uudecode (POSIX command)</td>
<td>798</td>
</tr>
<tr>
<td></td>
<td>uuencode (POSIX command)</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td>uuname (POSIX command)</td>
<td>802</td>
</tr>
<tr>
<td>V</td>
<td>variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>assign value (read)</td>
<td>643</td>
</tr>
<tr>
<td></td>
<td>bc variable (bc)</td>
<td>156</td>
</tr>
<tr>
<td></td>
<td>declaration (awk)</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>default value (mailx)</td>
<td>516</td>
</tr>
<tr>
<td></td>
<td>export (export)</td>
<td>363</td>
</tr>
<tr>
<td></td>
<td>initialization (awk)</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>input file (awk)</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>POSIX shell</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>protect shell variables</td>
<td>647</td>
</tr>
<tr>
<td></td>
<td>(readonly)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>special (awk)</td>
<td>116, 118, 119</td>
</tr>
<tr>
<td></td>
<td>user-defined (awk)</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>version number of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>operating system</td>
<td></td>
</tr>
<tr>
<td></td>
<td>output (uname)</td>
<td>785</td>
</tr>
<tr>
<td></td>
<td>Verwaltungdateien</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/etc/vfstab</td>
<td>558</td>
</tr>
<tr>
<td></td>
<td>vi (POSIX command)</td>
<td>803</td>
</tr>
<tr>
<td>W</td>
<td>w bit (chmod)</td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>wait (POSIX command)</td>
<td>843</td>
</tr>
<tr>
<td></td>
<td>wait for background</td>
<td>843</td>
</tr>
<tr>
<td></td>
<td>processes to terminate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(wait)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wc (POSIX command)</td>
<td>846</td>
</tr>
<tr>
<td></td>
<td>whence (POSIX command)</td>
<td>848</td>
</tr>
<tr>
<td></td>
<td>while statement (awk)</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td>who (POSIX command)</td>
<td>849</td>
</tr>
<tr>
<td></td>
<td>word count (wc)</td>
<td>846</td>
</tr>
</tbody>
</table>
Index

working directory
  print path name of (pwd)  638
write (POSIX command)  854
write arguments to standard output (print)  625
write checksum of a file (cksum)  212

X
  x bit (chmod)  205
X/Open  11
  xargs (POSIX command)  858
  XPG4 conformity of commands  873
  XPG4 standard  11
  XPG4.2 standard  11

Y
  yacc (POSIX command)  863

Z
  zcat (POSIX command)  866