Preface

The complete XENIX Reference Manual is actually divided into six parts and distributed as individual reference sections in the various volumes of the XENIX Operating, Text Processing, and Development Systems. The following table lists the name, content, and location of each reference section.

Section	Description	XENIX Volume
C	Commands – used with the XENIX Operating System.	User's Reference
CP	Programming Commands – used with the Development System.	Programmer's Reference
CT	Text Processing Commands – used with the Text Processing System.	Text Processing Guide
DOS	Routines – used with the Development System	Programmer's Reference
F	File Formats – description of various system files not defined in section M.	User's Reference
HW	Hardware specific manual pages – information about XENIX procedures specific to your computer.	Run Time Environment
M	Miscellaneous – information used for access to devices, system maintenance, and communications.	User's Reference
S	System Calls and Library Routines – available for C and assembly language programming.	Programmer's Reference

In the manual pages, a given command, routine, or file is referred to by name and section. For example, the programming command "cc", which is described in the Programming Commands (CP) section, is listed as *cc*(CP).

The alphabetized table of contents given on the following pages is a complete listing of all XENIX commands, system calls, library routines, and file formats. The permuted index, found at the end of the XENIX *User's Reference*, and the the end of the XENIX *Programmer's Reference*, is useful in matching a desired task with the manual page that describes it.

Alphabetized List

Commands, Systems Calls, Library Routines and File Formats

Contents

Commands (C)

date	Prints and sets the date.
dc	Invokes an arbitrary precision calculator.
dd	Converts and copies a file.
devnm	Identifies device name.
df	Report number of free disk blocks.
diff3	Compares three files.
diff	Compares two text files.
dircmp	Compares directories.
dirname	Delivers directory part of pathname.
disable	Turns off terminals and printers.
diskcp, diskcmp	Copies or compares floppy disks.
divvy	Divides disk partitions.
dmesg	Displays the system messages on the console.
dos, doscat, doscp, dosdir, dosformat, dosls, dosrm, dosrmdir	Access DOS files.
dparam	Displays/changes hard disk characteristics.
dtype	Determines disk type.
du	Summarizes disk usage.
dump	Performs incremental file system backup.
dumpdir	Prints the names of files on a backup archive.
echo	Echoes arguments.
ed	Invokes the text editor.
euable	Turns on terminals and line printers.
env	Sets environment for command execution.
ex	Invokes a text editor.
expr	Evaluates arguments as an expression.
factor	Factor a number.
false	Returns with a nonzero exit value.
fdisk	Maintain disk partitions.
file	Determines file type.
find	Finds files.
finger	Finds information about users.
fixhdr	Changes executable binary file headers.
format	Format floppy disks.
fsck	Checks and repairs file systems.
getopt	Parses command options.
grep, egrep, fgrep	Searches a file for a pattern.
grpcheck	Checks group file.
haltsys, reboot	Closes out the file systems and shuts down the system
hd	Displays files in hexadecimal format.
head	Prints the first few lines of a stream.
id	Prints user and group IDs and names.
imacct	Generate an IMAGEN accounting report.
imprint	Prints text files on an IMAGEN printer.
ipcrm	Removes a message queue, semaphore set or shared memory ID.

ipcs	Reports the status of inter-process communication.
ipr, oldipr	Put files into the IMAGEN printer queue.
ips, ipbs	Imagen protocol handler.
iprint	Converts text files to DVI format.
join	Joins two relations.
kill	Terminates a process.
l	Lists information about contents of directory.
lc	Lists directory contents in columns.
line	Reads one line.
ln	Makes a link to a file.
logname	Gets login name.
lp, lpr, cancel	Send/cancel requests to lineprinter.
lpadmin	Configures the lineprinter spooling system.
lpinit	Adds, reconfigures and maintains lineprinters.
lpr	Sends files to the lineprinter queue for printing.
lpsched, lpshut, lpmove	Starts/stops the lineprinter.
lpstat	Prints lineprinter status information.
ls	Gives information about contents of directories.
mail	Sends, reads, or disposes of mail.
mesg	Permits or denies messages sent to a terminal.
mkdev	Calls scripts to add peripheral devices.
mkdir	Makes a directory.
mkfs	Constructs a file system.
mknod	Builds special files.
mkuser	Adds a login ID to the system.
more	Views a file one screen full at a time.
mount	Mounts a file structure.
mtape	Magnetic tape maintenance.
mv	Moves or renemes files.
mvdir	Moves a directory.
ncheck	Generates names from inode numbers.
netutil	Administers the XENINX network.
newform	Changes the format of a text file.
newgrp	Logs users into a new group.
news	Print news items.
nice	Runs a command at a different priority.
nl	Adds linenumbers to a file.
nohup	Runs a command immune to hangups and quits.
od	Displays files in octal format.
pack, pcat, unpack	Compresses and expands files.
passwd	Changes login password.
pg	File perusal filter for soft-copy terminals.
pr	Prints files on the standard output.
ps	Reports process status.
pstat	Reports system information.

pwadmin	Performs password aging administration.
pwcheck	Checks password file.
pwd	Prints working directory name.
quot	Summarizes file system ownership.
random	Generates a random number.
rcp	Copies files across XENIX systems.
red	Invokes a restricted version of *ed*.
remote	Executes
restore, restor	Invokes incremental file system restorer.
rm	Removes files.
rmdir	Removes directories.
rmuser	Removes a user account from the system.
rsh	Invokes a restricted shell (command interpreter).
runbig	Runs a command that may require more memory than normal.
sddate	Prints and sets backup dates.
sdiff	Compares files side-by-side.
sed	Invokes the stream editor.
setcolor	Set screen color.
setkey	Assigns the function keys.
setmnt	Establishes /etc/mnttab table.
settime	Changes the access and modification dates of files.
sh	Invokes the shell command interpreter.
shl	Shell layer manager
shV	Invokes the shell command interpreter.
shl	Shell
shutdown	Terminates all processing.
sleep	Suspends execution for an interval.
sort	Sorts and merges files.
split	Splits a file into pieces.
stty	Sets the options for a terminal.
su	Makes the user a super-user or another user.
sum	Calculates checksum and counts blocks in a file.
swapctl	Adds swap area.
sync	Updates the super-block.
sysadmin	Performs file system backups and restores files.
sysadmsh	Menu driven system administration utility.
tail	Delivers the last part of a file.
tape	Magnetic tape maintenance program.
tar	Archives files.
tee	Creates a tee in a pipe.
telinit, mkinittab	Alternative method of turning terminals on and off.
test	Tests conditions.
tic	Terminfo compiler.
tid	Terminfo decompiler.
touch	Updates access and modification times of a file.
tput	Queries the terminfo database.

tr	Translates characters.
true	Returns with a zero exit value.
tset	Sets terminal modes.
tty	Gets the terminal's name.
umask	Sets file-creation mode mask.
umount	Dismounts a file structure.
uname	Prints the name of the current XENIX system.
uniq	Reports repeated lines in a file.
units	Converts units.
uucico	Executes work files for uucp data transmission.
uuclean	Clean-up the uucp spool directory.
uucp, uulog, uuname	Copies files from XENIX to XENIX.
uuinstall	Administers UUCP control files.
uustat	uucp status inquiry and job control.
uusub	Monitor uucp network.
uuto, uupick	Public XENIX-to-XENIX file copy.
uux	Executes command on remote XENIX.
vi, view, vedit	Invokes a screen-oriented display editor.
vmstat	Reports virtual memory statistics.
vsh	Menu driven visual shell.
wait	Awaits completion of background processes.
wall	Writes to all users.
wc	Counts lines, words and characters.
what	Identifies files.
who	Lists who is on the system.
whodo	Determines who is doing what.
write	Writes to another user.
xargs	Constructs and executes commands.
yes	Prints string repeatedly.

Name

intro – Introduces XENIX commands.

Description

This section describes use of the individual commands available in the XENIX Operating System. Each individual command is labeled with either a C, a CP, or a CT for easy reference from other volumes. The letter "C" stands for "command". The letters "P" and "T" stand for commands that come with the optional XENIX Development System (Programming) and the XENIX Text Processing System, respectively. For example, the reference *date*(C) indicates a reference to a discussion of the **date** command in the C section; the reference *cc*(CP) indicates a reference to a discussion of the **cc** command in the XENIX Development System; and the reference *spell*(CT) indicates a reference to a discussion of the **spell** command in the XENIX Text Processing System. The Text Processing and Development Systems are optional supplemental packages to the standard Operating System.

The "M" Miscellaneous section contains miscellaneous information including a great deal of system maintenance information. Other reference sections include the "S" System Services section, the "DOS" Routines section and the "F" File Format section.

Syntax

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(*s*)] [*cmdarg*(*s*)]

where:

name	Is the name of an executable file.
option	– *noargletter*(*s*) or, – *argletter*<>*optarg* where <> is optional whitespace.
noargletter	Is a single letter representing an option without an argument.
argletter	Is a single letter representing an option requiring an argument.

optarg Is an argument (character string) satisfying preceding *argletter*.

cmdarg Is a pathname (or other command argument) *not* beginning with **-**. **-** by itself indicates the standard input.

See Also

getopt(C), getopt(S)

Diagnostics

Upon termination, each command returns 2 bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(S) and *exit*(S)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

Notes

Not all commands adhere to the syntax described here.

Name

accept, reject - Allows/prevents print requests to a lineprinter or class of printers.

Syntax

/usr/lib/accept destinations
/usr/lib/reject [**-r**[reason]] destinations

Description

accept allows *lp*(C) to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(C) to find the status of *destinations.*

reject prevents *lp*(C) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(C) to find the status of *destinations*. The following option is useful with *reject*:

-r[*reason*] Associates a *reason* with disabling (using *disable (C)) the printer. The reason* applies to all printers listed up to the next **-r** option. If the **-r** option is not present or the **-r** option is given without a *reason,* then a default *reason* is used. *Reason* is reported by *lpstat(C).* Please see *disable(C)* for an example of *reason* syntax.

Files

/usr/spool/lp/*

See Also

enable(C), lp(C), lpadmin(C), lpinit(C), lpsched(C), lpstat(C), disable(C).

Name

acctcom – Searches for and prints process accounting files.

Syntax

acctcom [[options][file]] . . .

Description

acctcom reads *file*, the standard input, or **/usr/adm/pacct**, in the form described by *acct*(F) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE(K), and optionally, F (the *fork/exec* flag: **1** for *fork* without *exec*) and STAT (the system exit status).

The command name is prepended with a # if it was executed with super-user privileges. If a process is not associated with a known terminal, a **?** is printed in the TTYNAME field.

If no *files* are specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using & in the shell), **/usr/adm/pacct** is read, otherwise the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file **/usr/adm/pacct** is usually the current file to be examined; a busy system may need several files, in which case all but the current file will be found in **/usr/adm/pacct**?. The *options* are:

–b Reads backwards, showing latest commands first.

–f Prints the *fork/exec* flag and system exit status columns in the output.

–h Instead of showing mean memory size, it shows the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:

(total CPU time)/(elapsed time).

–i Prints columns containing the I/O counts in the output.

–k Instead of memory size, shows total kcore-minutes.

-m Shows mean core size (the default).

-r Shows CPU factor (user time/(system-time + user-time).)

-t Shows separate system and user CPU times.

-v Excludes column headings from the output.

-l *line* Shows only processes belonging to terminal **/dev/***line*.

-u *user* Shows only processes belonging to *user* that may be specified by a user ID, a login name that is then converted to a user ID, a **#** which designates only those processes executed with super-user privileges, or **?** which designates only those processes associated with unknown user IDs.

-g *group* Shows only processes belonging to *group*. The *group* may be designated by either the group ID or group name.

-d *mm/dd*
Any *time* arguments following this flag are assumed to occur on the given month and day, rather than during the last 24 hours. This is needed for looking at old files.

-s *time* Shows only those processes that existed on or after *time*, given in the form *hr:min:sec*. The *:sec* or *:min:sec* may be omitted.

-e *time* Shows only those processes that existed on or before *time*. Using the same *time* for both **-s** and **-e** shows the processes that existed at *time*.

-n *pattern*
Shows only commands matching *pattern* that may be a regular expression as in *ed* (C) except that **+** means one or more occurrences.

-H *factor* Shows only processes that exceed *factor*, where factor is the "hog factor" as explained in option **-h** above.

-I *number*
Shows driver processes transferring more characters than the cutoff *number*.

-O *time* Shows only those processes with operating system CPU time that exceeds *time*.

−C *time* Shows only those processes that exceed *time* (the total CPU time).

Multiple options have the effect of a logical AND.

Files

/etc/passwd

/usr/adm/pacct

/etc/group

See Also

accton(C), ps(C), su(C), acct(S), acct(F), utmp(M)

Notes

acctcom only reports on processes that have terminated; use *ps*(C) for active processes.

Name

accton – Turns on accounting.

Syntax

accton [file]

Description

accton turns on and off process accounting. If no *file* is given then accounting is turned off. If *file* is given, the kernel appends process accounting records. (See *acct* (S) and *acct* (F)).

Files

/etc/passwd	Used for login name to user ID conversions
/usr/adm/pacct	Current process accounting file
/usr/adm/sulogin	Super-user login history file
/etc/wtmp	Login/logout history file

See Also

acctcom(C), acct(S), acct(F), su(C), utmp(M)

Name

asktime – Prompts for the correct time of day.

Syntax

/etc/asktime

Description

This command prompts for the time of day. You must enter a legal time according to the proper format as defined below:

[[*yy*]*mmdd*]*hhmm*

Here the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24-hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. The current year is the default if no year is mentioned.

Examples

This example sets the new time, date, and year to "11:29 April 20, 1985".

```
Current system time is Wed Nov  3 14:36:23 PST 1985
Enter time ([yymmdd]hhmm): 8504201129
```

Diagnostics

If you enter an illegal time, *asktime* prompts with:

Try again:

Notes

asktime is normally performed automatically by the system startup file **/etc/rc** immediately after the system is booted; however, it may be executed at any time. The command is privileged, and can only be executed by the super-user.

Systems which autoboot will invoke *asktime* automatically on reboot. On these systems, if you don't enter a new time or press return within 1 minute of invoking *asktime* , the system will use the time value it has. If RETURN alone is entered, the time is unchanged.

Name

assign, deassign − Assigns and deassigns devices.

Syntax

assign [**−u**] [**−v**] [**−d**] [device] ...

deassign [**−u**] [**−v**] [device] ...

Description

assign attempts to assign *device* to the current user. The *device* argument must be an assignable device that is not currently assigned. An *assign* command without an argument prints a list of assignable devices along with the name of the user to whom they are assigned.

deassign is used to "deassign" devices. Without any arguments, *deassign* will deassign all devices assigned to the user. When arguments are given, an attempt is made to deassign each *device* given as an argument.

With these commands you can exclusively use a device, such as a tape drive or floppy drive. This keeps other users from using the device. They have a similar effect as *chown*(C) and *chmod*(C), although they only act on devices in **/dev**. Other aspects are discussed further on.

Available options include:

−d
Performs the action of *deassign*. The **−d** option may be embedded in *device* names to assign some devices and deassign others.

−v
Gives verbose output.

−u
Suppresses assignment or deassignment, but performs error checking.

The *assign* command will not assign any assignable devices if it cannot assign all of them. *deassign* gives no diagnostic if the *device* cannot be deassigned. Devices may be automatically deassigned at logout, but this is not guaranteed. *Device* names may be just the beginning of the device required. For example,

assign fd

should be used to assign all floppy disk devices. Raw versions of *device* will also be assigned, e.g., the raw floppy disk devices **/dev/rfd?** would be assigned in the above example.

Note that in many installations the assignable devices such as floppy disks have general read and write access, so the *assign* command may not be necessary. This is particularly true on single-user systems. Devices supposed to be assignable with this command should be owned by the user *asg*. The directory **/dev** should be owned by **bin** and have mode 755. The *assign* command (after checking for use by someone else) will then make the device owned by whoever invokes the command, without changing the access permissions. This allows the system administrator to set up individual devices that are freely available, assignable (owned by *asg*), or nonassignable and restricted (not owned by *asg* and with some restricted mode).

Note that the first time *assign* is invoked, it builds the assignable devices table **/etc/atab** . This table is used in subsequent invocations to save repeated searches of the **/dev** directory. If one of the devices in **/dev** is changed to be assignable (i.e., owned by *asg*), then **/etc/atab** should be removed (by the super-user) so that a correct list will be built the next time the command is invoked.

Return Values

Exit code 0 returned if successful, 1 if problems, 2 if *device* cannot be assigned.

Name

at, batch – Executes commands at a later time.

Syntax

at time [date] [+ increment]

at -r job ...

at -l[job ...]

at -q[letter] time [date] [job ...]

Description

at and *batch* read commands from the standard input to be executed at a later time. *at* allows you to specify a time when the commands should be executed, while *batch* executes jobs when the system load level permits.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, *umask*, and *ulimit* are retained when the commands are executed. Open file descriptors, traps, and priorities are lost.

A user is permitted to use *at* if his name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, the file **/usr/lib/cron/at.deny** is checked to determine if the user should be denied access to *at*. If neither file exists, only root is allowed to submit a job. If only the **at.deny** file exists, global usage is permitted. The allow/deny files consist of one user name per line.

The options are:

time The *time* may be specified as 1, 2, or 4 digits. One- and two-digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour:minute*. A suffix **am** or **pm** may be appended; otherwise a 24-hour clock time is understood. The suffix **zulu** may be used to indicate GMT. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

date An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days", **today** and **tomorrow**, are recognized. If no *date* is given, **today** is assumed if the given hour is greater than the current hour and **tomorrow** is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

increment

The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.) Thus, legitimate commands include:

at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday

−r Removes jobs previously scheduled by the *at* or *batch* command. Unless you are the super-user, you can only remove your own jobs.

−l Lists all the jobs currently scheduled for the invoking user.

−q*letter*

Places the specified job in a queue denoted by *letter*, where *letter* is any letter from "a" to "z" (not uppercase). The queue letter is appended to the job number. The following letters have special significance:

a	*at* queue
b	*batch* queue
c	*cron* queue

at and *batch* write the job number and schedule time to standard error. *batch* submits a batch job. It is almost equivalent to "at now," but with a difference: **batch** goes into a different queue; **at now** will respond with the error message too late.

Examples

The *at* and *batch* commands read the commands to be executed at a later time from the standard input. *sh*(C) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

The following sequence can be used at a terminal:

batch
nroff *filename* > *outfile*
<Ctrl-D> (press "Ctrl" and press "D")

This sequence, which demonstrates redirecting standard error to a pipe (|), is useful in a shell procedure (the sequence of output redirection specifications is significant):

batch <<!
nroff *filename* 2>&1 >*outfile* |mail
loginid
!

To have a job reschedule itself, invoke *at* from within the shell procedure by including code similar to the following within the shell file:

echo "sh *shellfile*" |at 1900 thursday next week

Files

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

See Also

cron(C), kill(C), mail(C), nice(C), ps(C), sh(C)

Diagnostics

Complains about syntax errors and times out of range.

Name

awk – Searches for and processes a pattern in a file.

Syntax

awk [**-Fc**] [**-f** programfile | 'program'] [parameters] [files]

Description

awk scans each input *file* for lines that match patterns specified in *program* or in *programfile*. When a line of *files* matches a pattern, an associated action may be performed. *awk* is useful for compiling information, performing arithmetic on input data, and for doing iterative or conditional processing.

The options are:

-Fc	Sets the field separator variable (FS) to the letter "c". The default field separators are tab and space.
-f	Causes *awk* to take its program from *programfile*.

The arguments are:

programfile	A file containing an *awk* program.
program	An *awk* program. Programs given on the command line must be enclosed in single quotation marks to prevent interpretation by the shell.
parameters	May be passed to *awk* in the form x=..., y=..., etc.
files	The name(s) of the file or files to be processed. If no filename is given, the standard input is used.

An *awk* program consists of statements in the form:

```
pattern { action }
```

Pattern-action statements may appear on the *awk* command line or in an *awk* program file.

If no *pattern* is given, all lines in the input file are matched. If no *action* is given, each matched line is displayed on the standard output.

A pattern may be a literal string or a regular expression, or a combination of a regular expression and a field or variable separated by operators.

awk also provides two patterns, BEGIN and END, that can be used to perform actions before the first line is read and after the last line is read, respectively.

To select a range of lines, use two patterns on a single program line, separated by a comma.

An action is a sequence of statements separated by a semicolon, newline, or right brace. See *Statements* later in this section.

Variables

In addition to variables declared and initialized by the user, *awk* has the following program variables:

NR Number of records.

NF Number of fields in a record.

FS Input field separator.

OFS Output field separator.

RS Input record separator.

ORS Output record separator.

$0 The current record.

$1, $*n* Fields in the current record.

OFM The output format for numbers. The default is % .6g.

FILENAME
The name of the input file currently being read.

Arrays may be used to store data. Arrays do not need to be dimensioned before use. For example, "w[i]" denotes the *i*th item of array *w*.

Expressions

A pattern match with a field or variable may be tested with the following operators:

~ Matches the regular expression.

!~ Does not match the regular expression.

awk processes relational expressions using the following operators:

<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
>=	Greater than or equal to
>	Greater than

Patterns can be combined using the operators:

&&	And
||	Or
!	Not

An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the following operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo

Concatenation is indicated by a blank.

The following C operators are also available in expressions:

++	Increment
--	Decrement
+=	Add and assign
-=	Subtract and assign

*= Multiply and assign

/= Divide and assign

%= Modulo and assign

Statements

if (*conditional*) *statement* [else *statement*]
while (*conditional*) *statement*
for (*expression* ; *conditional* ; *expression*) *statement*
break
continue
{ [*statement*] ... }
variable = *expression*
print [*expression-list*] [>*expression*]
printf *format* [, *expression-list*] [>*expression*]
next #skip remaining patterns on input line

while Used the same as in C.

for The iterative construction. It can be used the same as in the C language, or as an array iterator.

break Similar to its C counterpart.

continue Similar to its C counterpart.

print Prints its arguments on the standard output, or in a file if redirected.

printf Prints *expression-list* in the format specified in *format*. See *printf*(S).

next Stops processing the current record and moves to the next record, if any.

Comments are preceded by a number sign (#).

Functions

awk has the following built-in functions:

exit(x) Terminates the *awk* program. If x is given, this value is *awk*'s return value. If x is not given, 0 is returned. If the program has an END section, it is invoked before termination.

exp(x) Exponentiation of the value of x.

index(*s*, *t*) Returns the starting position of the leftmost occurrence of *t* in *s*. If *t* is not a substring of *s*, then index(*s*, *t*) is 0.

int(*x*) Returns the largest integer less than or equal to *x*. If *x* is negative, its value is the smallest integer greater than or equal to *x*.

length(*x*) A function whose value is the number of characters in the string (*x*). With no arguments, *length* is equivalent to $0.

log(*x*) Natural logarithm of *x*.

split(*x*, *y*) Assigns the fields of string *x* to successive elements of array *y*.

sqrt(*x*) Square root of *x*.

substr(*string*, *index*, *length*)
Returns the substring of *string* that begins at *index* and is *length* characters long.

Examples

The following displays lines in *file* longer than 72 characters:

```
awk '{length > 72}' file
```

The following prints the first two fields in opposite order:

```
awk '{ print $2, $1 }' file
```

The following adds up the first columns and prints their sum and average:

```
        { s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

The following prints the fields in *file* in reverse order:

```
awk { for (i = NF; i > 0; --i) print $i } file
```

The following prints all lines between start/stop pairs:

```
awk '{/start/, /stop/}' file
```

The following *awk* program file will print all lines in the object file whose first field is different from the first field in the previous line:

```
$1 != prev { print; prev = $1 }
```

The following program prints a file, filling in page numbers starting at 5:

```
/Page/  {$2 = n++;}
        {print}
```

The command line has the form: awk -f program n=5 input

See Also

grep(C), lex(CP), malloc(S), sed(C)
XENIX *Text Processing Guide*

Notes

Input whitespace is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number, add 0 to it; to force it to be treated as a string, concatenate the null string ("") to it.

This command is explained in detail in the XENIX *Text Processing Guide.*

Name

backup – Performs incremental file system backup.

Syntax

backup [key [arguments] filesystem]

Description

backup copies all files changed after a certain date in the date in the *filesystem*. The *key* specifies the date and other options about the backup, where a *key* consists of characters from the set **0123456789kfusd.** The meanings of these characters are described below:

f Places the backup on the next *argument* file instead of the default device.

u If the backup completes successfully, writes the date of the beginning of the backup to the file **/etc/ddate.** This file records a separate date for each file system and each backup level.

0-9 This number is the "backup level". Backs up all files modified since the last date stored in the file **/etc/ddate** for the same file system at lesser levels. If no date is determined by the level, the beginning of time is assumed; thus the option **0** causes the entire file system to be backed up.

s For backups to magnetic tape, the size of the tape is specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, *backup* will wait for reels to be changed. The default size is 2,300 feet.

d For backups to magnetic tape, the density of the tape, expressed in BPI, is taken from the next *argument*. This is used in calculating the amount of tape used per write. The default is 1600.

k This option is used when backing up to a block-structured device, such as a floppy disk. The size (in K-bytes) of the volume being written is taken from the next *argument*. If the **k** argument is specified, any **s** and **d** arguments are ignored. The default is to use s and **d.**

If no arguments are given, the *key* is assumed to be **9u** and a default file system is backed up to the default device.

The first backup should be a full level-0 backup:

backup 0u

Next, periodic level 9 backups should be made on an exponential progression of tapes or floppies:

backup 9u

This progression is shown as follows:

1 2 1 3 1 2 1 4 ...

where backup 1 is used every other time, backup 2 every fourth, backup 3 every eighth, etc.) When the level-9 incremental backup becomes unmanageable because a tape is full or too many floppies are required, a level-1 backup should be made:

backup 1u

After this, the exponential series should progress as if uninterrupted. These level-9 backups are based on the level-1 backup, which is based on the level-0 full backup. This progression of levels of backups can be carried as far as desired.

The default file system and the backup device depend on the settings of the variables DISK and TAPE, respectively, in the file **/etc/default/backup**.

Files

/etc/ddate	Records backup dates of file system/level
etc/default/backup	Default backup information

See Also

XENIX *Operations Guide*
cpio(C), default(M), dumpdir(C), restore(C), sddate(C), backup(F)

Diagnostics

If the backup requires more than one volume (where a volume is likely to be a floppy disk or tape), you will be asked to change volumes. Press RETURN after changing volumes.

Notes

Sizes are based on 1600 BPI for blocked tape; the raw magnetic tape device has to be used to approach these densities. Write errors to the backup device are usually fatal. Read errors on the file system are ignored.

It is not possible to successfully *restore* an entire active root file system.

Warning

When backing up to floppy disks, be sure to have enough *formatted* floppies ready before starting a backup.

Name

banner – Prints large letters.

Syntax

banner strings

Description

banner prints its arguments (each up to 10 characters long) in large letters on the standard output. This is useful for printing names at the front of printouts.

See Also

echo(C)

Name

basename – Removes directory names from pathnames.

Syntax

basename string [suffix]

Description

basename deletes any prefix ending in **/** and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. The result is the "base" name of the file, i.e., the filename without any preceding directory path and without an extension. It is used inside substitution marks (\` \`) in shell procedures to construct new filenames.

The related command *dirname* deletes the last level from *string* and prints the resulting path on the standard output.

Examples

The following command displays the filename **memos** on the standard output:

```
basename /usr/johnh/memos.old .old
```

The following shell procedure, when invoked with the argument **/usr/src/cmd/cat.c**, compiles the named file and moves the output to a file named **cat** in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

See Also

dirname(C), sh(C)

Name

bc – Invokes a calculator.

Syntax

bc [**–c**] [**–l**] [file ...]

Description

bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The **–l** argument stands for the name of an arbitrary precision math library. The syntax for *bc* programs is as follows: L means the letters a–z, E means expression, S means statement.

Comments:

Enclosed in **/*** and ***/**

Names:

Simple variables: L
Array elements: L [E]
The words "ibase", "obase", and "scale"

Other operands:

Arbitrarily long numbers with optional sign and decimal point
(E)
sqrt (E)
length (E) Number of significant decimal digits
scale (E) Number of digits right of decimal point
L (E , ... , E)

Additive operators:

+
–

Multiplicative operators:

*
/
% (remainder)
^ (exponentiation)

Unary operators:

```
++
--      (prefix and postfix; apply to names)
```

Relational operators:

```
==
<=
>=
!=
<
>
```

Assignment operators:

```
=
=+
=-
=*
=/
=%
=^
```

Statements:

```
E
{ S ; ... ; S }
if ( E ) S
while ( E ) S
for ( E ; E ; E ) S
null statement
break
quit
```

Function definitions:

```
define L ( L ,..., L ) {
        auto L, ... , L
        S; ... S
        return ( E )
}
```

Functions in **−l** math library:

s(x)	Sine
c(x)	Cosine
e(x)	Exponential
l(x)	Log
a(x)	Arctangent
j(n,x)	Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc*(C). Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

bc is actually a preprocessor for *dc*(C), which it invokes automatically, unless the **−c** (compile only) option is present. If the **−c** option is present, the *dc* input is sent to the standard output instead.

Example

The following defines 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; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

The following prints the approximate values of the exponential function of the first ten integers:

```
for(i=1; i<=10; i++) e(i)
```

Files

/usr/lib/lib.bc   Mathematical library

/usr/bin/dc   Desk calculator proper

See Also

dc(C)
The XENIX User's Guide

Notes

A *For* statement must have all three E's.

Quit is interpreted when read, not when executed.

Trigonometric values should be given in radians.

Name

bdiff – Compares files too large for *diff*.

Syntax

bdiff file1 file2 [n] [–s]

Description

bdiff compares two files, finds lines that are different, and prints them on the standard output. It allows processing of files that are too large for *diff*. *bdiff* splits each file into *n*-line segments, beginning with the first nonmatching lines, and invokes *diff* upon the corresponding segments. The arguments are:

n The number of lines *bdiff* splits each file into for processing. The default value is 3500. This is useful when 3500-line segments are too large for *diff*.

–s
Suppresses printing of *bdiff* diagnostics. Note that this does not suppress printing of diagnostics from *diff*.

If *file1* (or *file2*) is a dash (–), the standard input is read.

The output of *bdiff* is exactly that of *diff*. Line numbers are adjusted to account for the segmenting of the files, and the output looks as if the files had been processed whole.

Files

/tmp/bd?????

See Also

diff(C)

Notes

Because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

Specify the maximum number of lines if the first difference is too far down in the file for *diff* and an error is received.

Name

bfs – Scans big files.

Syntax

bfs [-] name

Description

bfs is like *ed* (C) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 255 characters per line. *bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit* (C) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional dash (**-**) suppresses printing of sizes. Input is prompted for with an asterisk (*) when "P" and RETURN are typed. The "P" acts as a toggle, so prompting can be turned off again by entering another "P" and a RETURN. Note that messages are given in response to errors only if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols other than the standard slash (**/**) and (**?**): A greater-than sign (**>**) indicates downward search without wraparound, and a less-than sign (**<**) indicates upward search without wraparound. Note that parentheses and curly braces are special and need to be escaped with a backslash (****). Since *bfs* uses a different regular expression-matching routine from *ed*, the regular expressions accepted are slightly wider in scope (see *regex* (S)). Differences between *ed* and *bfs* are listed below:

+ A regular expression followed by **+** means *one or more times*. For example, **[0-9]+** is equivalent to **[0-9][0-9]***.

\{m\} \{m,\} \{m,u\}
Integer values enclosed in **\{ \}** indicate the number of times the preceding regular expression is to be applied. *m* is the minimum number and *u* is a number, less than 256, which is the maximum. If only *m* is present (e.g., **\{m\}**), it indicates the exact number of times the regular expression is to be applied. **\{m,\}** is analogous to **\{m,infinity\}**. The plus (**+**) and star (*) operations are equivalent to **\{1,\}** and **\{0,\}** respectively.

(. . .)$*n* The value of the enclosed regular expression is to be returned. The value will be stored in the *(n+1)*th argument following the subject argument. At most ten enclosed regular expressions are allowed. *regex* makes its assignments unconditionally.

(. . .) Parentheses are used for grouping. An operator, e.g. *, +, \{\}, can work on a single character or a regular expression enclosed in parenthesis. For example, \(a*\(cb+\)*\)$0.

There is also a slight difference in mark names: only the letters "a" through "z" may be used, and all 26 marks are remembered.

The **e**, **g**, **v**, **k**, **p**, **q**, **w**, **=**, **!** and null commands operate as described under *ed* except that **e** doesn't remember filenames and **g** and **v** when given no arguments return the line after the line you were on. Commands such as **---**, **+++-**, **+++=**, **-12**, and **+4p** are accepted. Note that **1,10p** and **1,10** will both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no remembered filename. The **w** command is independent of output diversion, truncation, or crunching (see the **xo**, **xt** and **xc** commands, below). The following additional commands are available:

xf *file*

Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received, or an error occurs, reading resumes with the file containing the **xf**. **Xf** commands may be nested to a depth of 10.

xo [*file*]

Further output from the **p** and null commands is diverted to the named *file*. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label*

This positions a *label* in a command file. The *label* is terminated by a newline, and blanks between the **:** and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(. , .)**xb**/*regular expression*/*label*

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between **1** and **$**.

2. The second address is less than the first.

3. The regular expression doesn't match at least one line in the specified range, including the first and last lines.

On success, dot (.) is set to the line matched and a jump is made to *label*. This command is the only one that doesn't issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.

The **xb** command is allowed only if it is read from somewhere other than a terminal. If it is read from a pipe only a downward jump is possible.

xt *number*
Output from the **p** and null commands is truncated to a maximum of *number* characters. The initial number is 255.

xv[*digit*] [*spaces*] [*value*]
The variable name is the specified *digit* following the **xv**. **Xv5100** or **xv5 100** both assign the value **100** to the variable **5**. **Xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables **5** and **6**:

```
1,%5p
1,%5
%6
```

prints the first 100 lines.

```
g/%5/p
```

globally searches for the characters **100** and prints each line containing a match. To escape the special meaning of **%**, a **** must precede it. For example,

```
g/".*\%[cds]/p
```

could be used to match and list lines containing *printf* characters, decimal integers, or strings.

Another feature of the **xv** command is that the first line of output from a XENIX command can be stored into a variable.

The only requirement is that the first character of *value* be a !. For example,

```
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

puts the current line in variable 5, prints it, and increments the variable 6 by one. To escape the special meaning of ! as the first character of *value*, precede it with a \. For example,

```
xv7\!date
```

stores the value **!date** into variable 7.

xbz *label*

xbn *label*
These two commands test the last saved *return code* from the execution of a XENIX command (!*command*) or nonzero value, respectively, and jump to the specified label. The two examples below search for the next five lines containing the string size:

```
xv55
```

Name

cal – Prints a calendar.

Syntax

cal [[month] year]

Description

cal prints a calendar for the specified year. If a month is also specified, a calendar for that month only is printed. If no arguments are specified, the current, previous, and following months are printed, along with the current date and time. The *year* must be a number between 1 and 9999; *month* must be a number between 1 and 12 or enough characters to specify a particular month. For example, May must be given to distinguish it from March, but S is sufficient to specify September. If only a month string is given, only that month of the current year is printed.

Notes

Beware that "cal 84" refers to the year 84, not 1984.

The calendar produced is that for England and her colonies. Note that England switched from the Julian to the Gregorian calendar in September of 1752, at which time eleven days were excised from the year. To see the result of this switch, try "cal 9 1752".

Name

calendar – Invokes a reminder service.

Syntax

calendar [–]

Description

calendar consults the file **calendar** in the user's current directory and mails him lines that contain today's or tomorrow's date. Most reasonable month-day dates, such as "Sep. 7," "september 7", and "9/7", are recognized, but not "7 September", "7/12" or "07/12".

On weekends "tomorrow" extends through Monday. Lines that contain the date of a Monday will be sent to the user on the previous Friday. This is not true for holidays.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in his login directory and sends the user the results by *mail* (C). Normally this is done daily, in the early morning, under the control of *cron* (C).

Files

calendar

/usr/lib/calprog   To figure out today's and tomorrow's dates

/etc/passwd

/tmp/cal*

See Also

cron(C), mail(C)

Notes

To get reminder service, a user's **calendar** file must have read permission for all.

Name

capinfo – convert termcap descriptions into terminfo descriptions.

Syntax

capinfo capfile infofile

DESCRIPTION

capinfo invokes an *ex*(C) script to begin the conversion of a termcap terminal description into the equivalent terminfo description. The conversion needs to be completed by hand. The following areas should be given special attention:

– Padding specifications in *termcap*(M) have this syntax:

n.m*

while those in *terminfo*(M) are:

$<n.m*>

These are not converted by *capinfo*.

– Many *terminfo* capabilities do not have *termcap* equivalents. The XENIX extensions to termcap do not have terminfo equivalents.

– The *termcap* capabilities *cr*, *nl*, and *ht* are noted in the *ex* script as being problematical.

See Also

termcap(M), terminfo(M), terminfo(F), tic(C)

Notes

This command will be improved to handle the above exceptions.

Name

cat – Concatenates and displays files.

Syntax

cat [**-u**] [**-s**] [**-v**] [**-t**] [**-e**] file ...

Description

cat reads each *file* in sequence and writes it on the standard output. If no input file is given, or if a single dash (–) is given, *cat* reads from the standard input. The options are:

-s
Suppresses warnings about nonexistent files.

-u
Causes the output to be unbuffered.

-v
Causes non-printing characters (with the exception of tabs, new-lines, and form feeds) to be displayed. Control characters are displayed as "^X" (Ctrl-*X*); the DEL character (octal 0177) is printed as "^?." Non-ASCII characters (with the high bit set) are printed as "M -x," where *x* is the character specified by the seven low order bits.

-t Causes a tab to be printed as "^I." This option is ignored if the **-v** option is not specified.

-e
Causes a "$" character to be printed at the end of each line (prior to the new-line). This option is ignored if the **-v** option is not set.

No input file may have the same name as the output file unless it is a special file.

Examples

The following example displays *file* on the standard output:

```
cat file
```

The following example concatenates *file1* and *file2* and places the result in *file3*:

```
cat file1 file2 >file3
```

The following example concatenates *file1* and appends it to *file2*:

```
cat file1 >> file2
```

See Also

cp(C), pr(C)

Warning

Command lines such as:

```
cat file 1 file2 > file1
```

will cause the original data in *file1* to be lost; therefore, you must be careful when using special shell characters.

Name

cd – Changes working directory.

Syntax

cd [directory]

Description

If specified, *directory* becomes the new working directory; otherwise the value of the shell parameter $HOME is used. The process must have search (execute) permission in all directories (components) specified in the full pathname of *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and executed by the shell.

If the shell is reading its commands from a terminal, and the specified directory does not exist (or some component cannot be searched), spelling correction is applied to each component of *directory*, in a search for the "correct" name. The shell then asks whether or not to try and change directory to the corrected directory name; an answer of *n* means "no", and anything else is taken as "yes".

Notes

Wildcard designators do not work with the **cd** command.

See Also

pwd(C), sh(C), chdir(S)

Name

chgrp – Changes group ID.

Syntax

chgrp *group file* ...

Description

chgrp changes the group ID of each *file* to *group*. The group may be either a decimal group ID or a group name found in the file **/etc/group.**

Files

/etc/passwd

/etc/group

See Also

chown(C), chown(S), passwd(M), group(M)

Notes

Only the owner or the super-user can change the group ID of a file.

Name

chmod – Changes the access permissions of a file or directory.

Syntax

chmod *mode file* ...

chmod [*who*] +−= [permission ...] *file* ...

Description

The *chmod* command changes the access permissions (or *mode*) of a specified file or directory. It is used to control file and directory access by users other than the owner and super-user. The *mode* may be an expression composed of letters and operators (called *symbolic mode*), or a number (called *absolute mode*).

A *chmod* command using *symbolic mode* has the form:

chmod [*who*] +−= [permission ...] *filename*

In place of *who* you can use one or any combination of the following letters:

a Stands for "all users". If *who* is not indicated on the command line, **a** is the default. The definition of "all users" depends on the user's *umask*. See *umask*(C).

g Stands for "group", all users who have the same group ID as the owner of the file or directory.

o Stands for "others", all users on the system.

u Stands for "user", the owner of the file or directory.

The operators are:

+ Adds permission

− Removes permission

= Assigns the indicated permission and removes all other permissions (if any) for that *who*. If no permission is assigned, existing permissions are removed.

Permissions can be any combination of the following letters:

x Execute (search permission for directories)

r Read

w Write

s Sets owner or group ID on execution of the file to that of the owner of the file. The mode "u+s" sets the user ID bit for the file. The mode "g+s" sets the group ID bit. Other combinations have no effect.

t Saves text in memory upon execution. ("Sticky bit", see *chmod*(S)). Only the mode "u+t" sets the sticky bit. All other combinations have no effect. This mode can only be set by the super-user.

Multiple symbolic modes may be given, separated by commas, on a single command line. See the following Examples section for sample permission settings.

A *chmod* command using *absolute mode* has the form:

chmod *mode filename*

where *mode* is an octal number constructed by performing logical OR on the following:

4000	Set user ID on execution
2000	Set group ID on execution
1000	Sets the sticky bit (see *chmod*(S))
0400	Read by owner
0200	Write by owner
0100	Execute (search in directory) by owner
0040	Read by group
0020	Write by group
0010	Execute (search in directory) by group
0004	Read by others
0002	Write by others
0001	Execute (search in directory) by others
0000	No permissions

Examples

Symbolic Mode

The following command gives all users execute permission for *file*:

chmod +x *file*

The following command removes read and write permission for group and others from *file*:

chmod go-rw *file*

The following command gives other users read and write permission for *file*:

chmod o+rw *file*

The following command gives read permission to group and other:

chmod g+r,o+r *file*

Absolute Mode

The following command gives all users read, write and execute permission for *file*:

chmod 0777 *file*

The following command gives read and write permission to all users for *file*:

chmod 0666 *file*

The following command gives read and write permission to the owner of *file* only:

chmod 0600 *file*

See Also

ls(C), chmod(S)

Notes

The user ID, group ID and sticky bit settings are only useful for binary executable files. They have no effect on shell scripts.

Name

chown – Changes owner ID.

Syntax

chown owner file ...

Description

chown changes the owner ID of the *files* to *owner*. The owner may be either a decimal user ID or a login name found in the file **/etc/passwd.**

Files

/etc/passwd

/etc/group

See Also

chgrp(C), chown(S), group(M), passwd(M)

Notes

Only the owner or the super-user can change a file's owner or group ID.

Name

chroot – Changes root directory for command.

Syntax

chroot newroot command

Description

The given command is executed relative to the new root. The meaning of any initial slashes (**/**) in pathnames is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Notice that:

```
chroot newroot command >x
```

creates the file **x** relative to the original root, not the new one.

This command is restricted to the super-user.

The new root pathname is always relative to the current root even if a *chroot* is currently in effect. The *newroot* argument is relative to the current root of the running process. Note that it is not possible to change directories to what was formerly the parent of the new root directory; i.e., the *chroot* command supports the new root as an absolute root for the duration of the *command*. This means that "/.." is always equivalent to "/".

See Also

chdir(S)

Notes

Exercise extreme caution when referencing special files in the new root file system.

command must be under *newroot* or *command* is reported:
command: not found

Name

clear – Clears a terminal screen.

Syntax

clear

Description

The *clear* command clears a terminal screen. It examines the environment for a terminal type ($TERM) and termcap ($TERMCAP). The program then uses either a clear screen sequence (:cl=*string:*) found in $TERMCAP, or looks for a clear screen sequence in a file given in $TERMCAP.

Files

/etc/termcap

See Also

termcap(M)

Notes

If *clear* cannot find a clear screen sequence, it does nothing.

Name

clri – Clears inode.

Syntax

/etc/clri file-system i-number ...

Description

clri writes zeros on the 64 bytes occupied by the inode numbered *i-number*. *File-system* must be a special filename referring to a device containing a file system. After *clri* is executed, any blocks in the affected file will show up as "missing" if the file system is checked with *fsck*(C). Use *clri* only in emergencies and exercise extreme care.

Read and write permission is required on the specified *file-system* device. The inode becomes allocatable.

The primary purpose of this routine is to remove a file which, for some reason, does not appear in a directory. If you use *clri* to destroy an inode which does appear in a directory, track down the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to this file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated inode, so the whole cycle is likely to be repeated again and again.

See Also

fsck(C), ncheck(C)

Notes

If the file is open, *clri* is likely to be ineffective.

Name

cmchk – Reports hard disk block size.

Syntax

cmchk

Description

Reports the hard disk block size (BSIZE) in bytes.

Name

cmp – Compares two files.

Syntax

cmp [**–l**] [**–s**] file1 file2

Description

cmp compares two files and, if they are different, displays the byte and line number of the differences. If *file1* is **–**, the standard input is used.

The options are:

–l Prints the byte number (decimal) and the differing bytes (octal) for each difference.

–s Returns an exit code only, 0 for identical files, 1 for different files and 2 for an inaccessible or missing file.

This command should be used to compare binary files; use *diff* (C) or *diff3* (C) to compare text files.

See Also

comm(C), diff(C), diff3(C)

Diagnostics

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

Name

comm – Selects or rejects lines common to two sorted files.

Syntax

comm [– [123]] file1 file2

Description

comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort* (C)), and produces a three-column output: lines only in *file1*; lines only in *file2* ; and lines in both files. The filename **–** means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm –12** prints only the lines common to the two files; **comm –23** prints only lines in the first file but not in the second; **comm –123** is a no-op.

See Also

cmp(C), diff(C), sort(C), uniq(C)

Name

config – Configures a XENIX system.

Syntax

/usr/sys/conf/config [–i] [–c *file*] **–m** *master dfile*

Description

config takes a description of a XENIX system and generates compilable files that define the configuration tables for the various devices on the system.

Options include:

–m Specifies the name of the file that contains all the information regarding supported devices; **/usr/sys/conf/master** is the standard name. This file is supplied with the XENIX system and should *not* be modified by the user. The *configure*(C) utility should be used to update **/usr/sys/conf/master** and *dfile.*

–i Requests assembly-language output, instead of the default C language output.

–c Specifies the names of the configuration table files. **c.c** and **space.c** are the default names unless the **-i** option is given, in which case the default names are **c.asm** and **space.inc.**

dfile contains system device information and is divided into two parts. The first contains physical device specifications. The second contains system-dependent information. Any line with an asterisk (*) in column 1 is a comment. A standard *dfile* is provided as **/usr/sys/conf/xenixconf.** The *configure*(C) utility should also be used to update.

All configurations are assumed to have a set of required devices, such as the system clock, which must be present to run XENIX. These devices *must not* be specified in *dfile*.

First Part of *dfile*

Each line contains two fields, delimited by spaces and/or tabs in the following format:

    devname number

where *devname* is the name of the device, and *number* is the number (decimal) of devices associated with the corresponding controller. The device name can be any name given in part 1 of the

/usr/sys/conf/master file, or any alias given in part 3 of the same file; *number* is optional, and if omitted, a default value which is the maximum value for that controller is used.

There are certain drivers that may be provided with the system that are actually *pseudo-device* drivers; that is, there is no real hardware associated with the driver. If the system has such drivers, they are described in section M of the XENIX *User's Reference Manual.*

Second Part of *dfile*

The second part contains three different types of lines. Note that *all* specifications of this part *are required*, although their order is arbitrary.

1. **root/pipe** device specification

Two lines, each having three fields:

root	devname	minor
pipe	devname	minor

where *devname* is the name of the device, and *minor* is the minor device number (in octal). The device name can be any name given in part 1 of the **/usr/sys/conf/master** file, or any alias given in part 3 of the same file.

2. **swap** device specification

One line that contains five fields as follows:

swap devname minor swplo nswap

where *devname* is the name of the device, *minor* is the minor device number (in octal), *swplo* is the lowest disk block (decimal) in the swap area, and *nswap* is the number of disk blocks (decimal) in the swap area. The device name can be any name given in part 1 of the **/usr/sys/conf/master** file, or any alias given in part 3 of the same file.

3. *Parameter* specification

One or more lines, each having two fields as follows:

name number

where *name* is a tunable parameter name, and *number* is the desired value (in decimal) for the given parameter. Only names that have been defined in part 4 of the **/usr/sys/conf/master** file can be used; *number* overrides the default value for the given parameter. The following is a list of the available parameters:

buffers	Maximum number of external (mapped-out) buffers available to the kernel. If set to 0, *config* computes the optimum number for the system.
sabufs	Maximum number of internal (non-mapped) buffers available.
hashbuf	Maximum number of hash buffers.
inodes	Maximum number of inodes per file system.
files	Maximum number of open files per file system.
mounts	Maximum number of mounted file systems.
coremap	Maximum number of core map elements.
swapmap	Maximum number of swap map elements.
pages	Number of memory pages. On segmented systems such as the 286, this value should be 0.
calls	Maximum number of entries in the system timeout table.
procs	Maximum number of processes per system.
maxproc	Maximum number of processes per user.
texts	Maximum number of text segments per system.
clists	Maximum number of clists per system.
locks	Maximum number of file locks per system.
shdata	Maximum number of shared data segments per system.
timezone	Number of minutes difference between the local timezone and Greenwich Mean Time.
daylight	Daylight savings time in effect (1) or not in effect (0).
msgmap	Number of entries in message map.
msgmax	Maximum message size.
msgmnb	Maximum number of bytes in a message queue.
msgmni	Number of message queue identifiers.
msgtql	Number of message headers in the system.
msgssz	Number of bytes in message segments.
msgseg	Number of message segments.
semmap	Number of entries in semaphore map.
semmni	Number of semaphore identifiers.
semmnu	Number of undo structures in the system.
semmsl	Maximum number of semaphores per identifier.
semopm	Maximum number of operations per *semop*(S) call.

semume	Maximum number of undo entries per process.
semvmx	Maximum semaphore value.
semaem	Maximum value for "adjust on exit".
semmns	Number of semaphores in the system.
cmask	Default file creation mask for process 0.
maxprocmem	Maximum amount of memory available per process. This value cannot be greater than 75% of total user memory. If set to 0, *config* computes the optimum value.
screens	Number of Multiscreens for the systems. If set to 0, *config* computes the optimum value.
emaps	Maximum number of distinct eight-bit channel maps in the system.
nodename	The nodename of the system (as used by **uucp**(C) and other programs).

Examples

Suppose you wish to configure a system with the following devices:

One HD disk drive controller with 1 drive
One FD fioppy disk drive controller with 1 drive

You must also specify the following parameter information:

root device is an HD (pseudo disk 3)
pipe device is an HD (pseudo disk 3)
swap device is an HD (pseudo disk 2)
    with a swplo of 0 and a nswap of 2300
number of buffers is 50
number of processes is 50
maximum number of processes per user ID is 15
number of mounts is 8
number of inodes is 120
number of files is 120
number of calls is 30
number of texts is 35
number of character buffers is 150
number of swapmap entries is 50
number of memory pages is 512
number of file locks is 100
timezone is pacific time
daylight time is in effect
number of entries in message map is 513
maximum message size is 8192
maximum number of bytes in a message queue is 16384
number of message queue identifiers is 10
number of message headers in the system is 40
message segment size is 8

number of message segments is 1024
number of entries in semaphore map is 21
number of semaphore identifiers is 10
number of undo structures in the system is 60
maximum number of semaphores per identifiers is 10
maximum number of operations per *semop* call is 5
maximum number of undo entries per process is 5
maximum semaphore value is 32767
maximum value for "adjust on exit" is 16384
number of semaphores in the system is 40

The actual system configuration would be specified as follows:

```
hd      1
fd      1
root    hd      3
pipe    hd      3
swap    hd      2       0       2300
* Comments may be inserted in this manner
buffers 50
procs   150
maxproc         15
mounts 8
inodes  120
files   120
calls   30
texts   35
clists  150
swapmap         50
pages   (1024/2);
locks   100
timezone        (8*60)
daylight1
msgmap          (MSGSEG/2+1)
msgmax          8192
msgmnb          8192
msgmni10
msgtql  40
msgssz  8
msgseg  1024
semmap          (SEMMNS/2+1)
semmni10
semmnu          20
semmsl 10
semopm          5
semume          5
semvmx          32767
semaem          16384
semmns          40
```

Files

/usr/sys/conf/master	default input master device table
c.c	default output driver configuration table file
space.c	default output resource configuration table file
c.asm	default driver configuration in assembly language
space.inc	default resource configuration in assembly language

See Also

configure(C), master(F)

Diagnostics

Diagnostics are routed to the standard output and are self-explanatory.

Notes

The value on the right-hand side of a a parameter specification must be a double-quoted character string, an integer, the name of another parameter defined within the **master**(F) file, or some arithmetical combination of integers and defined parameter names. Only the "+", "-", "*", and "/" operators can be used in an arithmetical expression. Expressions are interpreted left-to-right: if operator precedence is in doubt, parenthesize.

Name

configure - **xenix** configuration program.

Syntax

configure [*options*] [*parm*=*val* ...]

Description

The *configure* program determines and alters different kernel resources. For end users, *configure* is easier than modifying the system configuration files directly. For device driver writers, *configure* avoids the difficulties of editing configuration files that have already been edited by an earlier driver configuration script.

Resources are modified interactively or with command-line arguments. Adding or deleting device driver components requires the command line options.

The next paragraphs discuss how to use *configure* interactively. Command line options are discussed in the "Options" section.

Interactive Usage

configure functions interactively when no options are given, or when **-f** is the only option specified on the command line.

When you invoke *configure* interactively, you first see a category menu that looks something like this:

```
1.     Disk Buffers
2.     Character Buffers
3.     Files, Inodes, and Filesystems
4.     Processes, Memory Management & Swapping
5.     Clock
6.     MultiScreens
7.     Message Queues
8.     Semaphores
9.     Shared Data
10.    System Name

Select a parameter category to reconfigure by typing a number
from 1 to 10, or type 'q' to quit:
```

To choose a category, enter its number, (e.g. "1" for "Disk Buffers") then press RETURN.

Each category contains a number of configurable resources. Each resource is presented by displaying its true name, a short description, and its current value. For example, for the "Disk Buffers" category you might see:

NBUF: total disk buffers. Currently determined at system start up:
NSABUF: system-addressable (near) disk buffers. Currently 10:
NHBUF: hash buffers (for disk block sorting). Currently 128:

To keep the current value, simply press RETURN. Otherwise, enter an appropriate value for the resource, then press RETURN. *configure* checks each value to make sure that it is within an appropriate range. If not, it continues to prompt for a value until you enter one that is appropriate.

To exit from *configure* enter 'q' at the category menu prompt. If any changes are made, *configure* asks if it should update the configuration files with the changes. To keep the old configuration values, enter 'n' at this prompt, and no changes are made. Otherwise, enter 'y' and *configure* updates the required system configuration files. After *configure* has completed, the kernel is ready for linking.

Options

The command line options are designed for writers of driver-installation shell scripts. You can configure drivers, remove driver definitions from the configuration files, and modify some driver attributes, all from the command line. There are also options for querying the current driver configuration.

configure uses the following options:

-a [*func1 func2* ...]
-d [*func1 func2* ...]
-b
-c
-d [*func1 func2* ...]
-f *master_file* [*dfile*]
-l *priority_level*
-m *major*
-n
-q
-t
-v *interrupt_vector* [*interrupt_vector2...*]
-w
-x
-y *resource*

-m, **-b**, and **-c**
These options are used to define which driver is being referenced. Following **-m** must be the major device number of the driver. If you are configuring a block driver, **-b** must appear; if you are configuring a character driver, **-c** must appear. Both are used when configuring a driver with both kinds of interfaces.

-a and **-d**
Each option is followed by a list of functions to add or delete, respectively. These are the names of the functions that appear within *bdevsw*[ ] or *cdevsw*[ ], as appropriate, plus the names of the initialization, clock poll, halt and interrupt routines, if present, plus the names of the tty, stream, and tab structure pointers. *configure* enforces the rules that all of a driver's routines must have a common prefix, and that the prefix be 2-4 characters long.

-v This option modifies the system notion of the vectors on which this device can interrupt. A device may interrupt on up to 4 vectors.

-l This sets the interrupt priority level of the device, which is almost always the same as the type of *spl*() call used: a driver that interlocks using *spl5*() almost always has an interrupt priority level of 5.

-q If the **-q** option is given, no *qswtch*( ) is possible after returning from the device interrupt.

-f The configuration is maintained in two data files, whose default names are *master* and *xenixconf*. The **-f** option can be used to specify alternate names. Note that if **-f** is the only option present, the program is still interactive.

-n If **-n** is present, the two configuration data files are modified, but no '.o' files are produced. This option is useful when configuring a driver package containing multiple drivers.

-w This option suppresses warning messages.

-x This dumps all the resource prompts known to *configure*. These reveal the name, description and current value of each parameter capable of being reconfigured. Category prompts are not dumped.

-y The **-y** option prints out the current value of the requested resource.

-t This option prints out nothing (except possibly error messages). However, it has a return value of 1 if a driver corresponding to the given combination of **-m**, **-b**, **-c** and options is already configured, and returns 0 if no such driver is present.

Setting Command-line Parameters

Any number of arguments can be given on the command line of the form *resource=value.* These arguments can be given at the same time as an add or delete driver request, but must follow all the driver-configuration arguments on the command line.

Some resources have values that are character strings. In this case their values must be enclosed within the characters \" . The quotes are syntactically necessary for them to be used as C-language strings, and the backslashes protect the quotes from being removed by the shell.

Examples

Print out the current value of NCLIST:

```
configure -y NCLIST
```

Return 1 if character major device 7 and vector 3 are available:

```
configure -t -v 7 -m 3 -c
```

Add a clock-time polling and initialization routine to the already configured "foo" driver, a hypothetical character driver at major device #17:

```
configure -a foopoll fooinit -c -m 17
```

Delete the "foo" driver:

```
configure -m 17 -d -c
```

Add a new "hypo" driver, a block driver with a character interface. It absorbs 3 different interrupt vectors, at priority 6:

```
configure -a hypoopen hypoclose hyporead hypowrite hypoioctl\
 hypostrategy hypotab hypointr -b -c -l 6 -v 17 42 49
```

Notes

Kernel Data Space Restrictions (286 only)

If the total size of all the allocated resources grows too large, the group will not fit within the kernel's 64k near data segment. You will not see messages about excessive size from *configure*, but you may see them from the linker when you attempt to link the kernel.

Files

/usr/sys/conf/master
/usr/sys/conf/xenixconf
/usr/sys/conf/config

See Also

master(F), config(C)

Name

copy – Copies groups of files.

Syntax

copy [option] ... source ... dest

Description

The *copy* command copies the contents of directories to another directory. It is possible to copy whole file systems since directories are made when needed.

If files, directories, or special files do not exist at the destination, then they are created with the same modes and flags as the source. In addition, the super–user may set the user and group ID. The owner and mode are not changed if the destination file exists.

Note that there may be more than one source directory. If so, the effect is the same as if the *copy* command had been issued for each source directory with the same destination directory for each copy.

Options do not have to be given as separate arguments, and may appear in any order, even after the other arguments. The options are:

–a Asks the user before attempting a copy. If the response does not begin with a "y", then a copy is not done. option.

–l Uses links instead whenever they can be used. Otherwise a copy is done. Note that links are never done for special files or directories.

–n Requires the destination file to be new. If not, then the *copy* command does not change the destination file. The **–n** flag is meaningless for directories. For special files an **–n** flag is assumed (i.e., the destination of a special file must not exist).

–o If set then every file copied has its owner and group set to those of the source. If not set, then the file's owner is the user who invoked the program.

–m If set, then every file copied has its modification time and access time set to that of the source. If not set, then the modification time is set to the time of the copy.

-r If set, then every directory is recursively examined as it is encountered. If not set then any directories that are found are ignored.

-ad Asks the user whether a **-r** flag applies when a directory is discovered. If the answer does not begin with a "y", then the directory is ignored.

-v If the verbose option is set messages are printed that reveal what the program is doing.

Arguments to *copy* are:

source This may be a file, directory or special file. It must exist. If it is not a directory, then the results of the command are the same as for the *cp* command.

dest The destination must be either a file or directory that is different from the source.

If the source and destination are anything but directories, then *copy* acts just like a *cp* command. If both are directories, then *copy* copies each file into the destination directory according to the flags that have been set.

Examples

This command line verbosely copies all files in the current directory to **/tmp/food**:

```
copy -v . /tmp/food
```

The next command line copies all files, except for those that begin with a period (.), and copies the immediate contents of any child directories:

```
copy * /tmp/logic
```

This command is the same as the previous one, except that it recursively examines all subdirectories, and it sets group and ownership permissions on the destination files to be the same as the source files:

```
copy -ro * /tmp/logic
```

Notes

Special device files can be copied. When they are copied, any data associated with the specified device is *not* copied.

Name

cp – Copies files.

Syntax

cp file1 file2

cp files directory

Description

There are two ways to use the *cp* command. With the first way, *file1* is copied to *file2*. Under no circumstance can *file1* and *file2* be identical. With the second way, *directory* is the location of a directory into which one or more *files* are copied.

See Also

copy(C), cpio(C), ln(C), mv(C), rm(C), chmod(S)

Notes

Special device files can be copied. If the file is a named pipe, then the data in the pipe is copied to a regular file. Similarly, if the file is a device, then the file is read until the end-of-file is reached, and that data is copied to a regular file. It is illegal to copy a directory to a file.

Name

cpio – Copies file archives in and out.

Syntax

cpio –o [**acBv**]

cpio –i [**Bcdmrtuv**] [patterns]

cpio –p [**adlmruv**] directory

Description

cpio –o (copy out) reads the standard input to obtain a list of pathnames and copies those files onto the standard output together with pathname and status information.

cpio –i (copy in) extracts from the standard input (which is assumed to be the product of a previous *cpio –o*) the names of files selected by zero or more *patterns* given in the name-generating notation of *sh* (C). In *patterns*, the special characters **?**, *****, and [. . .] match the slash (**/**) character. The default for *patterns* is ***** (i.e., select all files).

Remember to escape special characters to prevent expansion by the shell.

cpio –p (pass) copies out and in during a single operation. Destination pathnames are interpreted relative to the named *directory*.

The meanings of the available options are:

–a Resets access times of input files after they have been copied.

–B Blocks input/output 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from raw devices).

–d Directories are created as needed.

–c Writes header information in ASCII character form for portability.

–r Interactively renames files. If the user types a null line, the file is skipped.

-t Prints a table of contents of the input. No files are created.

-u Copies unconditionally (normally an older file will not replace a newer file with the same name).

-v Verbose: causes a list of filenames to be printed. When used with the **-t** option, the table of contents looks like the output of an **ls -l** command (see *ls* (C)).

-l Whenever possible, links files rather than copying them. Usable only with the **-p** option.

-m Retains previous file modification time. This option is ineffective on directories that are being copied.

Examples

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/fd
```

```
cd olddir
find . -print | cpio -pdl newdir
```

Or:

```
find . -print | cpio -oB >/dev/rfd
```

See Also

ar(CP), find(C), cpio(F)

Notes

Pathnames are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and thereafter linking information is lost. Only the superuser can copy special files.

Name

cron – Executes commands at specified times.

Syntax

/etc/cron
crontab [file]
crontab –r
crontab –l

Description

cron is the clock daemon that executes commands at specified dates and times according to the instructions in the files located in **/usr/spool/cron/crontabs**. Regularly scheduled commands can be specified according to instructions found in crontab files; users can submit their own crontab file via the *crontab* command. Commands which are to be executed only once may be submitted via the *at* command. Because *cron* never exits, it should be executed only once. This is best done by running *cron* from the initialization process through the file **/etc/rc**.

crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The - *crontab* file in the **crontabs** directory is given the user's login name. The **–r** option removes a user's crontab from the crontab directory. *crontab* **–l will list the crontab file for the invoking user.**

A user is permitted to use *crontab* if their name appears in the file **/usr/lib/cron/cron.allow**. If that file does not exist, the file **/usr/lib/cron/cron.deny** is checked to determine if the user should be denied access to *crontab*. If neither file exists, only root is allowed to submit a job. Global usage is permitted by the existence of an empty **cron.deny** file. **cron.deny** is checked only if **cron.allow** does not exist. The allow/deny files consist of one user name per line.

The **crontabs** files consist of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (0-6, with 0=Sunday). Each of these patterns may contain:

- A number in the (respective) range indicated above
- Two numbers separated by a minus (indicating an inclusive range)

- A list of numbers separated by commas (meaning all of these numbers)

- An asterisk (meaning all legal values)

Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field is a string that is executed by the shell at the specified time(s). A % in this field is translated into a newline character. Only the first line (up to a % or end-of-line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your **$HOME** directory with an **arg0** of **sh.** Users who desire to have their *.profile* executed must explicitly do so in the crontab file. *cron* supplies a default environment for every shell, defining **HOME, LOGNAME, SHELL** (**=/bin/sh**), and **PATH** (**=:/bin:/usr/bin:/usr/lbin**).

cron examines the **crontabs** directory periodically to see if it has changed; if it has, *cron* reads it. Thus it takes only a short while for entries to become effective.

Examples

An example **crontabs** file follows:

```
30 4 * * *      /etc/sa -s > /dev/null
0 4 * * *       calendar -
15 4 * * *      find /usr/preserve -mtime +7 -a -exec rm -f {} ;
30 4 1 1 1      /usr/lib/uucp/uuclean
40 4 * * *      find / -name '#*' -atime +3 -exec rm -f {} ;
1,21,41 * * * * (echo -n ' '; date; echo ) >/dev/console
```

A history of all actions by *cron* can be recorded in **/usr/lib/cron/log**. This logging occurs only if the variable CRONLOG in **/etc/default/cron** is set to YES. By default this value is set to NO and no logging occurs. If logging should be turned on, be sure to monitor the size of **/usr/lib/cron/log** so that it doesn't unreasonably consume disk space.

Files

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs/*	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users
/usr/lib/cron/.proto	cron environment information
/etc/default/cron	cron logging default information

See Also

at(C), sh(C)

Notes

cron reads the files in the **crontabs** directory only when there is a change, but it reads the in-core version of the tables periodically.

Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

Name

csh – Invokes a shell command interpreter with C-like syntax.

Syntax

csh [**-cefinstvVxX**] [arg ...]

Description

csh is a command language interpreter. It begins by executing commands from the file **.cshrc** in the home directory of the invoker. If this is a login shell, it also executes commands from the file **.login** there. In the normal case, the shell begins reading commands from the terminal, prompting with % . Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally, each command in the current line is executed.

When a login shell terminates, it executes commands from the file **.logout** in the user's home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters &, |, ;, <, >, (,), form separate words. If doubled in &&, ||, <<, or >>, these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with \. A newline preceded by a \ is equivalent to a blank.

In addition, strings enclosed in matched pairs of quotations, ´, ` or ", form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of \ or " characters, a newline preceded by a \ gives a true newline character.

When the shell's input is not a terminal, the character # introduces a comment which continues to the end of the input line. It does not have this special meaning when preceded by \ and placed inside the quotation marks `, ´, and ".

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by | characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ;, and are then executed sequentially. A sequence of pipelines may be executed without waiting for it to terminate by following it with a &. Such a sequence is automatically prevented from being terminated by a hangup signal; the *nohup* command need not be used.

Any of the above may be placed in parentheses to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with || or && indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions.*)

Substitutions

The following sections describe the various transformations the shell performs on the input in the order in which they occur.

History Substitutions

History substitutions can be used to reintroduce sequences of words from previous commands, possibly performing modifications on these words. Thus, history substitutions provide a generalization of a *redo* function.

History substitutions begin with the character ! and may begin anywhere in the input stream if a history substitution is not already in progress. The ! may be preceded by a \ to prevent its special meaning; a ! is passed unchanged when it is followed by a blank, tab, newline, =, or (. History substitutions may also occur when an input line begins with ^. This special abbreviation will be described later.

Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been entered without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list, the size of which is controlled by the *history* variable. The previous command is always retained. Commands are numbered sequentially from 1.

For example, enter the command:

history

Now, consider the following output from the history command:

```
 9 write michael
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the prompt by placing a ! in the prompt string.

With the current event 13 we can refer to previous events by event number !11, relatively as in !-2 (referring to the same event), by a prefix of a command word as in !d for event 12 or !w for event 9, or by a string contained in a word in the command as in !?mic? also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case !! refers to the previous command; thus !! alone is essentially a *redo*. The form !# references the current command (the one being entered). It allows a word to be selected from further left in the line, to avoid retyping a long name, as in !#:1.

To select words from an event, we can follow the event specification by a : and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, and so on. The basic word designators are:

0 First (command) word

n *n*th argument

^ First argument, i.e. 1

$ Last argument

% Word matched by (immediately preceding) ?*s*? search

x-*y*
Range of words

-*y* Abbreviates 0-*y*

* Abbreviates ^-$, or nothing if only 1 word in event

x * Abbreviates *x* -$

x - Like *x* * but omitting word $

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ^, $, *, - or %. After the optional word designator, a sequence of modifiers can be placed, each preceded by a :. The following modifiers are defined:

h Removes a trailing pathname component

r Removes a trailing .xxx component

s/*l* /*r* /
Substitutes *l* for *r*

t Removes all leading pathname components

& Repeats the previous substitution

g Applies the change globally, prefixing the above

p Prints the new command but does not execute it

q Quotes the substituted words, preventing substitutions

x Like q, but breaks into words at blanks, tabs, and newlines

Unless preceded by a g, the modification is applied only to the first modifiable word. In any case it is an error for no word to be applicable.

The left sides of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of /; a \ quotes the delimiter within the *l* and *r* strings. The character & in the right side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in !?*s* ?. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing ? in a contextual scan.

A history reference may be given without an event specification, e.g., !$. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus !?foo?^!$ gives the first and last arguments from the command matching ?foo?.

A special abbreviation of a history reference occurs when the first nonblank character of an input line is a ^. This is equivalent to !:s^, providing a convenient shorthand for substitutions on the text of the previous line. Thus ^lb^lib fixes the spelling of lib in the previous command. Finally, a history substitution may be surrounded with { and } if necessary to insulate it from the characters that follow. Thus, after ls -ld ~paul we might do !{l}a to do ls -ld ~paula, while !la would look for a command starting la.

Quotations With ´ and "

The quotation of strings by ´ and " can be used to prevent all or some of the remaining substitutions. Strings enclosed in ´ are prevented any further interpretation. Strings enclosed in " are variable and command expansion may occur.

In both cases, the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitution* below) does a " quoted string yield parts of more than one word; ´ quoted strings never do.

Alias Substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for ls is ls -l the command "ls /usr" would map to "ls -l /usr". Similarly if the alias for lookup was "grep \!^ /etc/passwd" then "lookup bill" would map to "grep bill /etc/passwd".

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can alias print "´pr \!* | lpr´" to make a command that paginates its arguments to the lineprinter.

There are four *csh* aliases distributed with the XENIX System V *csh*. These are **pushd** , **popd** , **swapd** , and **flipd** . These aliases maintain a directory stack.

pushd *dir*
Pushes the current directory onto the top of the directory stack, changes to the directory **dir** .

popd
Changes to the directory at the top of the stack, then removes (pops) the top directory from the stack, and announces the current directory.

swapd
Swaps the top two directories on the stack. The directory on the top becomes the second to the top, and the second to the top directory becomes the top directory.

flipd
Flips between two directories, the current directory and the top directory on the stack. If you are currently in **dir1** , and **dir2** is on the top of the stack, when **flipd** is invoked, you change to **dir2** and **dir1** is replaced as the top directory on the stack. When **flipd** is again invoked, you change to **dir1** and **dir2** is again the top directory on the stack.

Variable Substitution

The shell maintains a set of variables, each of which has a list of zero or more words as its value. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the **-v** command line option.

Other operations treat variables numerically. The at-sign (@) command permits numeric calculations to be performed and the result assigned to a variable. However, variable values are always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed, keyed by dollar sign ($) characters. This expansion can be prevented by preceding the dollar sign with a backslash (\) except within double quotation marks (") where it *always* occurs, and within single quotation marks (') where it *never* occurs. Strings quoted by back quotation marks (`) are interpreted later (see *Command substitution* below) so dollar sign substitution does not occur there until later, if at all. A dollar sign is passed unchanged if followed by a blank, tab, or end-of-line.

Input and output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in double quotation marks or given the :q modifier, the results of variable substitution may eventually be command and filename substituted. Within double quotation marks (") a variable whose value consists of multiple words expands to a portion of a single word, with the words of the variable's value separated by blanks. When the :q modifier is applied to a substitution, the variable expands to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following sequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}
: Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters, digits, and underscores.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}
: May be used to select only some of the words from the value of *name*. The selector is subjected to $ substitution and may consist of a single number or two numbers separated by a -. The first word of a variables value is numbered 1. If the first number of a range is omitted it defaults to 1. If the last member of a range is omitted it defaults to $#name. The selector * selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
: Gives the number of words in the variable. This is useful for later use in a [selector].

$0 Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}
Equivalent to $argv[number].

$* Equivalent to $argv[*].

The modifiers :h, :t, :r, :q and :x may be applied to the substitutions above as may :gh, :gt and :gr. If braces { } appear in the command form then the modifiers must appear within the braces. Only one : modifier is allowed on each $ expansion.

The following substitutions may not be modified with : modifiers.

$?name
${?name}
Substitutes the string 1 if name is set, 0 if it is not.

$?0 Substitutes 1 if the current input filename is known, 0 if it is not.

$$ Substitutes the (decimal) process number of the (parent) shell.

Command and Filename Substitution

Command and filename substitution are applied selectively to the arguments of built-in commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command Substitution

Command substitution is indicated by a command enclosed in back quotation marks. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within double quotation marks, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename Substitution

If a word contains any of the characters *, ?, [or { or begins with the character ~, then that word is a candidate for filename substitution, also known as globbing. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of filenames which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing filename, but it is not required for each pattern to match. Only the metacharacters *, ?, and [imply pattern matching, the characters ~ and { being more akin to abbreviations.

In matching filenames, the character . at the beginning of a filename or immediately following a /, as well as the character / must be matched explicitly. The character * matches any string of characters, including the null string. The character ? matches any single character. The sequence within square brackets [] matches any one of the characters enclosed. Within square brackets [], a pair of characters separated by - matches any character lexically between the two.

The character ~ at the beginning of a filename is used to refer to home directories. Standing alone, it expands to the invoker's home directory as reflected in the value of the variable ***home***. When followed by a name consisting of letters, digits and - characters the shell searches for a user with that name and substitutes their home directory; thus ~ken might expand to /usr/ken and ~ken/chmach to /usr/ken/chmach. If the character ~ is followed by a character other than a letter or / or appears not at the beginning of a word, it is left unchanged.

The metanotation a{b,c,d}e is a shorthand for abe ace ade. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus ~source/s1/{oldls,ls}.c expands to /usr/source/s1/oldls.c /usr/source/s1/ls.c, whether or not these files exist, assuming that the home directory for source is /usr/source. Similarly ../{memo,*box} might expand to ../memo ../box ../mbox. (Note that memo was not sorted with the results of matching *box.) As a special case {, } and {} are passed unchanged.

Input/Output

The standard input and standard output of a command may be redirected with the following syntax:

< name
: Opens file *name* (which is first variable, command and filename expanded) as the standard input.

<< word
: Reads the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting backslash, double, or single quotation mark, or a back quotation mark appears in *word*, variable and command substitution is performed on the intervening lines, allowing \ to quote $, \ and `. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resulting text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name
: The file *name* is used as standard output. If the file does not exist, then it is created; if the file exists, it is truncated, and its previous contents are lost.

  If the variable *noclobber* is set, then the file must not already exist or it must be a character special file (e.g., a terminal or /dev/null) or an error results. This helps prevent accidental destruction of files. In this case, the ! forms can be used and suppress this check.

  The forms involving & route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as < input filenames are.

>> name
>>& name
>>! name
>>&! name
: Uses file *name* as standard output like > but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the ! forms is given. Otherwise similar to >.

If a command is run detached (followed by &) then the default standard input for the command is the empty file /dev/null. Otherwise, the command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form |& rather than just |.

Expressions

A number of the built-in commands (to be described later) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, *exit*, *if*, and *while* commands. The following operators are available:

```
|| && | ^ & == != <= >= < > << >>
+ - * / % ! ~ ( )
```

Here the precedence increases to the right, == and !=, <=, >=, <, and >, << and >>, + and -, * / and % being, in groups, at the same level. The == and != operators compare their arguments as strings, all others operate on numbers. Strings which begin with 0 are considered octal numbers. Null or missing arguments are considered 0. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser (& | < > ( )) they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in { and } and file enquiries of the form -*l* name where *l* is one of:

r	Read access
w	Write access
x	Execute access
e	Existence
o	Ownership
z	Zero size
f	Plain file
d	Directory

The specified name is command and filename expanded, then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. 0. Command executions succeed, returning true, i.e. 1, if the command exits with status 0, otherwise they fail, returning false, i.e. 0. If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

Control Flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach, switch,* and *while* statements, as well as the *if-then—else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto commands will succeed on non-seekable inputs.)

Built-In Commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, then it is executed in a subshell.

alias
alias name
alias name wordlist

The first form prints all aliases. The second form prints the alias for *name* . The final form assigns the specified *wordlist* as the alias of *name; wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*

break

Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while* statement. The remaining commands on the current line are executed. Multilevel breaks are thus possible by writing them all on one line.

breaksw

Causes a break from a *switch,* resuming after the *endsw.*

case label:

A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name

Changes the shell's working directory to directory *name*. If no argument is given, it then changes to the home directory of the user. If *name* is not found as a subdirectory of the current directory (and does not begin with **/**, **./**, or **../**), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with **/**, then this is tried to see if it is a directory.

continue

Continues execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:

Labels the default case in a *switch* statement. The default should come after all *case* labels.

echo wordlist

The specified words are written to the shell's standard output. A \c causes the echo to complete without printing a newline. A \n in *wordlist* causes a newline to be printed. Otherwise the words are echoed, separated by spaces.

else
end
endif
endsw

See the description of the *foreach, if, switch,* and *while* statements below.

exec command

The specified command is executed in place of the current shell.

exit
exit(expr)

The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)

...

end

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The built-in command *continue* may be used to continue the loop prematurely and the built-in command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with ? before any statements in the loop are executed.

glob wordlist
Like *echo* but no \ escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word
The specified *word* is filename and command expanded to yield a string of the form label. The shell rewinds its input as much as possible and searches for a line of the form label: possibly preceded by blanks or tabs. Execution continues after the specified line.

history
Displays the history event list.

if (expr) command
If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed.

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif
If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else*.)

logout
Terminates a login shell. The only way to log out if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command

The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user may specify negative niceness by using "nice -number ...." The command is always executed in a subshell, and the restrictions placed on commands in simple *if* statements apply.

nohup
nohup command

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. Unless the shell is running detached, *nohup* has no effect. All processes detached with & are automatically *nohup*ed. (Thus, *nohup* is not really needed.)

onintr
onintr -
onintr label

Controls the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form onintr - causes all interrupts to be ignored. The final form causes the shell to execute a goto label when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirection occurs exactly once, even if *count* is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word.* The fourth form sets the *indexth* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist.* In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv name value

Sets the value of the environment variable *name* to be *value,* a single string. Useful environment variables are TERM, the type of your terminal and SHELL, the shell you are using.

shift
shift variable

The members of *argv* are shifted to the left, discarding *argv[1]*. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name

The shell reads commands from *name. Source* commands may be nested; if they are nested too deeply, the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Input during *source* commands is **never** placed on the history list.

switch (string)
case str1:
  ...
  breaksw
...
default:
  ...
  breaksw
endsw

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters *, ?, and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a default label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels, as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command

With no argument, a summary of time used by this shell and its children is printed. If arguments are given, the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask
umask value

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others, or 022 giving all access except no write access for users in the group or others.

unalias pattern

All aliases whose names match the specified pattern are discarded. Thus, all aliases are removed by unalias *. It is not an error for nothing to be *unaliased*.

unhash

Use of the internal hash table to speed location of executed programs is disabled.

unset pattern

All variables whose names match the specified pattern are removed. Thus, all variables are removed by unset *; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset*.

wait

All child processes are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and process numbers of all children known to be outstanding.

while (expr)
...
end

While the specified expression evaluates nonzero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

@
@ name = expr
@ name[index] = expr

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains <, >, & or | then at least this part of the expression must be placed within ( ). The third form assigns the value of *expr* to the *index*th argument of *name*. Both *name* and its *index*th component must already exist.

The operators *=, +=, etc. are available as in C. The space separating the name from the assignment operator is optional. Spaces are mandatory in separating components of *expr* which would otherwise be single words.

Special postfix ++ and -- operators increment and decrement *name* respectively, i.e. @ i++.

Predefined Variables

The following variables have special meaning to the shell. Of these, *argv, child, home, path, prompt, shell* and *status* are always set by the shell. Except for *child* and *status* this setting occurs only at initialization; these variables will not be modified unless done explicitly by the user.

The shell copies the environment variable PATH into the variable *path*, and copies the value back into the environment whenever *path* is set. Thus is is not necessary to worry about its setting other than in the file *.cshrc* as inferior *csh* processes will import the definition of *path* from the environment.

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e., $1 is replaced by $argv[1], etc.

cdpath Gives a list of alternate directories searched to find subdirectories in *cd* commands.

child The process number printed when the last command was forked with &. This variable is *unset* when this process terminates.

echo Set when the **-x** command line option is given. Causes each command and its arguments to be echoed just before it is executed. For nonbuilt-in commands all expansions occur before echoing.

Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.

histchars Can be assigned a two-character string. The first character is used as a history character in place of !, the second character is used in place of the ^ substitution mechanism. For example, set histchars=",;" will cause the history characters to be comma and semicolon.

history Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. A *history* that is too large may run the shell out of memory. The last executed command is always saved on the history list.

home The home directory of the invoker, initialized from the environment. The filename expansion of ~ refers to this variable.

ignoreeof If set, the shell ignores end-of-file from input devices that are terminals. This prevents a shell from accidentally being terminated by pressing Ctrl-D.

mail The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell responds with, "You have new mail" if the file exists with an access time not greater than its modify time.

If the first word of the value of *mail* is numeric, it specifies a different mail checking interval: in seconds, rather than the default, which is 10 minutes.

If multiple mail files are specified, then the shell responds with "New mail in *name*", when there is mail in the file *name*.

noclobber As described in the section *Input/Output*, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that >> redirections refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a filename expansion to not match any existing files; rather, the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e., echo [still gives an error.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable, then only full pathnames will execute. The usual search path is /bin, /usr/bin, and ., but this may vary from system to system. For the super-user, the default search path is /etc, /bin and /usr/bin. A shell which is given neither the **-c** nor the **-t** option will normally hash the contents of the directories in the *path* variable after reading *.cshrc*, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* command, or the commands may not be found.

prompt The string which is printed before each command is read from an interactive terminal input. If a ! appears in the string, it will be replaced by the current event number unless a preceding \ is given. Default is % , or # for the super-user.

shell The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Nonbuilt-In Command Execution* below.) Initialized to the (system-dependent) home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Built-in commands which fail return exit status 1, all other built-in commands set status 0.

time Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, real time, and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose Set by the -v command line option, causes the words of each command to be printed after history substitution.

Nonbuilt-In Command Execution

When a command to be executed is found to not be a built-in command, the shell attempts to execute the command via *exec*(S). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a **-c** nor a **-t** option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash)*, or if the shell was given a **-c** or **-t** argument, and in any case for each directory component of *path* which does not begin with a /, the shell concatenates with the given command name to form a pathname of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus (cd ; pwd) ; pwd prints the *home* directory; leaving you where you were (printing this after the home directory), while cd ; pwd leaves you in the home directory. Parenthesized commands are most often used to prevent *cd* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias are prepended to the argument list to form the shell command. The first word of the *alias* should be the full pathname of the shell (e.g. $shell). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument List Processing

If argument 0 to the shell is - then this is a login shell. The flag arguments are interpreted as follows:

-c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

-e The shell exits if any invoked command terminates abnormally or yields a nonzero exit status.

-f The shell will start faster, because it will neither search for nor execute commands from the file .cshrc in the invoker's home directory.

-i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their input and output are terminals.

-n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.

-s Command input is taken from the standard input.

-t A single line of input is read and executed. A \ may be used to escape the newline at the end of this line and continue onto another line.

-v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

-x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

-V Causes the *verbose* variable to be set even before .cshrc is executed.

-X Causes the *echo* variable to be set even before .cshrc is executed.

After processing of flag arguments, if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options were given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by $0. On a typical system, most shell scripts are written for the standard shell (see *sh*(C)), the C shell will execute such a standard shell if the first character of a script is not a # (i.e. if the script does not start with a comment). Remaining arguments initialize the variable *argv*.

Signal Handling

The shell normally ignores *quit* signals. The *interrupt* and *quit* signals are ignored for an invoked command if the command is followed by &; otherwise the signals have the values which the shell inherited from its parent. The shells handling of interrupts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file .logout.

Files

~/.cshrc	Read at by each shell at the beginning of execution
/etc/cshrc	Systemwide default *cshrc* file if none is present
~/.login	Read by login shell, after .cshrc at login
~/.logout	Read by login shell, at logout
/bin/sh	Shell for scripts not starting with a #
/tmp/sh*	Temporary file for <<
/dev/null	Source of empty file
/etc/passwd	Source of home directories for ~name

Limitations

Words can be no longer than 512 characters. The number of arguments to a command which involves filename expansion is limited to 1/6 number of characters allowed in an argument list, which is 5120, less the characters in the environment. Also, command substitutions may substitute no more characters than are allowed in an argument list.

To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

See Also

access(S), exec(S), fork(S), pipe(S), signal(S), umask(S), wait(S), a.out(F), environ(M)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Built-in control structure commands like **foreach** and **while** cannot be used with |, & or ;.

Commands within loops, prompted for by ?, are not placed in the *history* list.

It is not possible to use the colon (:) modifiers on the output of command substitutions.

csh attempts to import and export the PATH variable for use with regular shell scripts. This only works for simple cases, where the PATH contains no command characters.

This version of *csh* does not support or use the process control features of the 4th Berkeley Distribution.

Name

csplit – Splits files according to context.

Syntax

csplit [−s] [−k] [**−f** prefix] file arg1 [. . . argn]

Description

csplit reads *file* and separates it into n+1 sections, defined by the arguments *arg1*. . . *argn*. By default the sections are placed in xx00 . . . xx*n* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00: From the start of *file* up to (but not including) the line referenced by *arg1*.

01: From the line referenced by *arg1* up to the line referenced by *arg2*.

⋮

n+1: From the line referenced by *argn* to the end of *file*.

The options to *csplit* are:

−s *csplit* normally prints the character counts for each file created. If the **−s** option is present, *csplit* suppresses the printing of all character counts.

−k *csplit* normally removes created files if an error occurs. If the **−k** option is present, *csplit* leaves previously created files intact.

−f *prefix* If the **−f** option is used, the created files are named *prefix*00 . . . *prefixn*. The default is **xx00** . . . **xx***n*.

The arguments (*arg1* . . . *argn*) to *csplit* can be a combination of the following:

/rexp/ A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional **+**or **−** some number of lines (e.g., **/Page/−5**).

%rexp%
This argument is the same as */rexp/*, except that no file is created for the section.

lnno A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotation marks. Regular expressions may not contain embedded newlines. *csplit* does not affect the original file; it is the users responsibility to remove it.

Examples

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, **cobol00 . . . cobol03.** After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The **-k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/^}/+1' {20}
```

Assuming that **prog.c** follows the normal **C** coding convention of ending routines with a **}** at the beginning of the line, this example will create a file containing each separate **C** routine (up to 21) in **prog.c**.

See Also

ed(C), sh(C), regex(S)

Diagnostics

Self-explanatory except for:

arg – out of range

which means that the given argument did not reference a line between the current position and the end of the file.

Name

cu – Calls another XENIX system.

Syntax

cu [**-** sspeed] [**-** aacu] [**-** lline] [**- h**] [**- o** | **- e**] telno

cu [**-** sspeed] [**-***lline*] [**- h**] [**- o** | **- e**] dir

Description

cu "calls up" another XENIX system through a modem or a direct serial connection. It also controls the transmission and reception of data and programs during the call. *cu* looks at each line in the file **/usr/lib/uucp/L-devices** until it finds a line that matches the options given in the command line. If it finds an appropriate line, it will attempt to make a connection. If it cannot find the proper line, *cu* quits.

The options are:

–s*speed*
Specifies the transmission speed. 1200 baud is the default value. Other speeds available are 110, 150, 300, 1200, 2400, 4800 and 9600 baud. Directly connected lines may be set to other speeds. Most modems are restricted to 300 and 1200 baud.

–a*acu*
Specifies the device name of the ACU (automatic calling unit) device. If not specified, *cu* will use the first available *acu* with the right speed.

–l*line*
Specifies the device name of the communications line. If not specified, *cu* will use the first available direct line (if *dir* is specified) or *acu* (if a *telno* is specified) with the right speed.

–h
Emulates local echo. This feature supports calls to systems that expect half-duplex mode terminals.

–e
Specifies that even-parity data is to be generated for data sent to the remote system.

–o
Specifies that odd-parity data is to be generated for data sent to the remote system.

Telno is the telephone number of the remote system.

For *acu* connections, *cu* invokes **/usr/lib/uucp/dial** to dial the modem. Consult your modem manual to determine the correct sequences to include in the phone number for pauses, pulse dialing, etc.

For directly connected lines, the string "dir" is used instead of *telno*. See the Examples later in this section for sample command lines.

After making the connection, *cu* runs as two processes: *transmit* and *receive*. The *transmit* process reads data from the standard input and, except for lines beginning with a tilde (~), passes it to the remote system. The *receive* process accepts data from the remote system and, except for lines beginning with a tilde, passes it to the standard output. Normally, an automatic XON/XOFF (DC3/DC1) protocol controls input from the remote system so the buffer is not overrun. Lines beginning with a tilde have special meanings.

The *transmit* process interprets lines beginning with a tilde as follows:

~. Terminates the conversation.

~! Escapes to an interactive shell on the local system.

~!*cmd*... Runs *cmd* on the local system (via **sh -c**).

~$*cmd*... Runs *cmd* locally and sends its output to the remote system.

~%**take** *remote* [*local*]
Copies file *remote* (on the remote system) to file *local* on the local system. If *local* is omitted, the *remote* filename is used in both places. Use of this line requires the existence of *echo*(C) and *cat*(C) on the remote system. If tabs are to be copied without expansion, **stty tabs** mode should be set on the remote system.

~%**put** *local* [*remote*]
Copies file *local* (on the local system) to file *remote* on the remote system. If *local* is omitted, the *remote* filename is used in both places. Use of this line requires the existence of *stty* (C) and *cat* (C) on the remote system. It also requires that the current erase and kill characters on the remote system be identical

	to the current ones on the local system. Backslashes are inserted at appropriate places.
~%b or ~% **break**	Sends a break char to the remote system.
~~...	Sends the line ~... to the remote system.
~%**nostop**	Turns off the XON/XOFF input control protocol for the remainder of the session. This is useful if the remote system is one which does not respond properly to the XON/XOFF characters.

The *receive* process normally copies data from the remote system to its standard output. A line from the remote system that begins with ~> diverts the output to a file. Data is appended to a file if ~>> is used. The diversion is terminated by a trailing ~>. The complete sequence is:

~>[>]:*file*
zero or more lines to be written to *file*
~>

Examples

A sample command for a dialup connection is:

cu 5559801

cu selects the first available *acu* at the default speed of 1200 baud.

A sample command for a direct connection is:

cu dir

cu will select the first available direct line at the default speed of 1200 baud.

You can force *cu* to use a specific *acu* device, *line* device or *speed* with the command line options **-a**, **-l** and **-s**. This is useful if you wish to use the same modem for dialup connections at both 300 and 1200 baud, or if you have more than one directly connected computer. For example:

cu -a tty12 -s 300 5559801

will force *cu* to place the call through **/dev/tty12** at 300 baud.

cu -l tty12 dir

will cause **/dev/tty12** to be used for a direct connection at 1200 baud.

Files

/usr/lib/uucp/L-devices	Device information
/usr/lib/uucp/dial	Dialer program

See Also

cat(C), echo(C), stty(C), tty(M)

Diagnostics

Exit code is zero for normal exit, nonzero (various values) otherwise.

Device busy: Someone else is using the desired line.

Notes

There is an artificial slowing of transmission by *cu* during the **~%put** operation so that loss of data is unlikely.

ASCII files only can be transferred using **~%take** or **~%put**; binary files cannot be transferred.

cu opens devices for exclusive use. If *cu* terminates abnormally, the device may remain locked.

Name

custom - Installs specific portions of the XENIX System

Syntax

custom [-odt] [-irl [package]] [-f [file]]

Description

With *custom* you can create a custom installation by selectively installing or deleting portions of the XENIX system. *custom* is executable only by the super-user and is either interactive or can be invoked from the command line with several options.

Files are extracted or deleted in *packages.* A package is a collection of individual files. Packages are grouped together in *sets.*

Three default *sets* are always available:

Operating System

Development System

Text Processing System

You can also install additional *sets*. You can list the available *packages* by using the *custom* command as described next.

Usage

To use *custom* interactively, enter:

custom

You see a list of sets. For example:

1. Operating System
2. Development System
3. Text Processing System
4. Add a Supported Product

The program prompts you to choose a set from which to work. If the data files for that set are not already installed on the hard disk, *custom* prompts you for the floppy which contains these data files and installs them. You may also see menu items for each product that has been previously added using the "Add a Supported Product" option. If you are adding a new product, you will be prompted for volume 1 of the new product distribution and *custom* will extract the product information necessary to support it.

When you select a valid set, you see a menu like this:

1. Install one or more packages
2. Remove one or more packages
3. List the files in a package
4. Install a single file
5. Select a new set to customize
6. Display current disk usage
7. Help

When you enter a menu option, you are prompted for further information. This is what the options prompt, and what action occurs:

1. Install
   Prompts for one or more package names.

   Calculates which installation volumes (distribution media) are needed, then prompts for the correct volume numbers. If multiple packages are specified, the names should be separated by spaces on the command line.

This option, as well as "2" and "3," displays a list of all available packages in the currently selected set. Each line describes the package name, whether the package is fully installed, not installed or partially installed, the size of the package (in 512 byte blocks), and a one line description of the package contents.

2. Remove
   Prompts for one or more package names.

   Deletes the correct files in the specified package. If multiple packages are specified the names should be separated by spaces on the command line.

Displays available packages (see option "1").

3. List files in a package
   Lists all files in the specified package.

   Prompts for one or more package names. Enter the name of the desired package(s).

Displays available packages (see option "1").

4. Install a single file

   Extract the specified file from the distribution set.

   Filename should be a full pathname relative to the root directory "/".

5. Select a new set

   Allows you to work from a different set than the current one.

6. Display current disk usage

   Tells you your current disk usage.

7. Help

   Prints a page of instructions to help you use *custom*.

Options

Three arguments are required for a completely non-interactive use of *custom*:

> A set identifier
> **(-o, -d,** or **-t),**
>
> A command
> **(-i, -r, -l,** or **-f),**
>
> And either one or more package names, or a file name

If any information is missing from the command line, *custom* prompts for the missing data.

Only one of **-o, -d,** or **-t** may be specified. These stand for:

> **-o**
> Operating System
> **-d**
> Development System
> **-t**
> Text Processing System

Only one of **-i, -r, -l,** or **-f** may be specified, followed by an argument of the appropriate type (one or more package names, or a file name). These options perform the following:

-i Install the specified package(s)

-r Remove the specified package(s)

-l List the files in the specified package(s).

-f Install the specified file.

Files

/etc/base.perms
/etc/soft.perms
/etc/text.perms
/etc/perms/*

See Also

fixperm(M), df(C), du(C), install(C)

Notes

If you upgrade any part of your system, *custom* detects if you have a different release and prompts you to insert the floppy volume that updates the custom data files. Likewise, if you insert an invalid product or a volume out of order, you will be promted to reinsert the correct volume.

Name

date – Prints and sets the date.

Syntax

date [mmddhhmm[yy]] [**+**format]

Description

If no argument is given, or if the argument begins with **+**, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24-hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *date* takes care of the conversion to and from local standard and daylight time.

If the argument begins with **+**, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf* (S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by a percent sign (**%**) and will be replaced in the output by its corresponding value. A single percent sign is encoded by doubling the percent sign, i.e., by specifying "%%". All other characters are copied to the output without change. The string is always terminated with a newline character.

Field Descriptors:

n Inserts a newline character

t Inserts a tab character

m Month of year – 01 to 12

d Day of month – 01 to 31

y Last 2 digits of year – 00 to 99

D Date as mm/dd/yy

H Hour – 00 to 23

M Minute – 00 to 59

S Second – 00 to 59

T Time as HH:MM:SS

j Julian date – 001 to 366

w Day of the week – Sunday = 0

a Abbreviated weekday – Sun to Sat

h Abbreviated month – Jan to Dec

r Time in AM/PM notation

Example

The line

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

generates as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

Diagnostics

no permission	You aren't the super-user and you are trying to change the date.
bad conversion	The date set is syntactically incorrect.
bad format character	The field descriptor is not recognizable.

Name

dc – Invokes an arbitrary precision calculator.

Syntax

dc [file]

Description

dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but you may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number
: The value of the number is pushed on the stack. A number is an unbroken string of the digits 0–9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

+ – / * % ^
: The top two values on the stack are added (**+**), subtracted (**–**), multiplied (*****), divided (**/**), remaindered (**%**), or exponentiated (**^**). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

s*x*
: The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the s is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x*
: The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d
: The top value on the stack is duplicated.

p
: The top value on the stack is printed. The top value remains unchanged. **p** interprets the top of the stack as an ASCII string, removes it, and prints it.

f
: All values on the stack are printed.

q Exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x Treats the top element of the stack as a character string and executes it as a string of *dc* commands.

X Replaces the number on the top of the stack with its scale factor.

[...] Puts the bracketed ASCII string onto the top of the stack.

<*x* >*x* =*x*
The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

v Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! Interprets the rest of the line as a XENIX command.

c All values on the stack are popped.

i The top value on the stack is popped and used as the number radix for further input.

I Pushes the input base on the top of the stack.

o The top value on the stack is popped and used as the number radix for further output.

O Pushes the output base on the top of the stack.

k The top of the stack is popped, and that value is used as a nonnegative scale factor; the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

Z Replaces the number on the top of the stack with its length.

? A line of input is taken from the input source (usually the terminal) and executed.

; : Used by *bc* for array operations.

Example

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

See Also

bc(C)

Diagnostics

x is unimplemented	The octal number *x* corresponds to a character that is not implemented as a command
stack empty	Not enough elements on the stack to do what was asked
Out of space	The free list is exhausted (too many digits)
Out of headers	Too many numbers being kept around
Out of pushdown	Too many items on the stack
Nesting Depth	Too many levels of nested execution

Notes

bc is a preprocessor for *dc*, providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs. For interactive use, *bc* is preferred to *dc* .

Name

dd – Converts and copies a file.

Syntax

dd [option=value] ...

Description

dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

Option	*Value*
if=*file*	Input filename; standard input is default
of=*file*	Output filename; standard output is default
ibs=*n*	Input block size *n* bytes (default is BSIZE block size)
obs=*n*	Output block size (default is BSIZE block size)
bs=*n*	Sets both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy needs to be done
cbs=*n*	Conversion buffer size
skip=*n*	Skips *n* input records before starting copy
seek=*n*	Seeks *n* records from beginning of output file before copying
count=*n*	Copies only *n* input records
conv=ascii	Converts EBCDIC to ASCII
conv=ebcdic	Converts ASCII to EBCDIC
conv=ibm	Slightly different map of ASCII to EBCDIC
conv=lcase	Maps alphabetics to lowercase

Option	*Value*
conv=ucase	Maps alphabetics to uppercase
conv=swab	Swaps every pair of bytes
conv=sync	Pads every input record to *ibs*
conv="... , ..."	Several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and newline added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output record of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

Examples

This command reads an EBCDIC tape, blocked ten 80-byte EBCDIC card images per record, into the ASCII file **outfile**:

```
dd if=/dev/rmt0 of=outfile ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. *dd* is especially suited to I/O on raw physical devices because it allows reading and writing in arbitrary record sizes.

See Also

copy(C), cp(C), tar(C)

Diagnostics

f+p records in(out)	Numbers of full and partial records read(written)

Notes

The ASCII/EBCDIC conversion tables are taken from the 256-character standard in the CACM Nov, 1968. The *ibm* conversion corresponds better to certain IBM print train conventions. There is no universal solution.

Newlines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC.

Name

devnm – Identifies device name.

Syntax

/etc/devnm [names]

Description

Devnm identifies the special file associated with the mounted file system where the argument *name* resides.

This command is most commonly used by **/etc/rc** to construct a mount table entry for the **root** device.

Examples

Be sure to type full pathnames in this example:

/etc/devnm /usr

If **/dev/hd1** is mounted on **/usr**, this produces:

hd1 /usr

Files

/dev/* Device names

/etc/rc Xenix startup commands

See Also

setmnt(C)

Name

df – Report number of free disk blocks.

Syntax

df [**−t**] [**−f**] [**−v −i**] [file-systems]

Description

df prints out the number of free blocks and free inodes available for on-line file systems by examining the counts kept in the super-blocks; *file–systems* may be specified by device name (e.g., **/dev/root**). If the *file–systems* argument is unspecified, the free space on all of the mounted file systems is sent to the standard output. The list of mounted file systems is given in **/etc/mnttab.**

Options include:

−t Causes total allocated block figures to be reported as well as number of free blocks.

−f Reports only an actual count of the blocks in the free list (free inodes are not reported). With this option, df reports on raw devices.

−v Reports the percent of blocks used as well as the number of blocks used and free.

−i Reports the percent of inodes used as well as the number of inodes used and free. Use the **−i** option with the **−v** option to display counts of blocks and inodes free as well as the percentage of inodes and blocks used.

The **−v** and **−i** options can not be used with other *df* options.

Files

/dev/*
/etc/mnttab

See Also

fsck(C), mount(C), mnttab(F)

Notes

See *Notes* under *mount*(C).

This utility reports sizes in 512 byte blocks. On systems which use 1024 byte blocks, this means a file of 500 bytes uses 2 blocks. *df* will report 2 blocks less free space, rather than 1 block, since the file uses one system block of 1024 bytes. Refer to the **machine(HW)** manual page for the block size used by your system.

Name

diff – Compares two text files.

Syntax

diff [**–efbh**] file1 file2

Description

diff tells what lines must be changed in two files to bring them into agreement. If *file1* or *file2* is a dash (–), the standard input is used. If *file1* or *file2* is a directory, *diff* uses the file in that directory that has the same name as file (*file2* or *file1* respectively) it is compared to. For example:

diff */tmp dog*

compares the file named *dog*, that is in the */tmp* directory, with the file *dog* in the current directory. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward, one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

The **–b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **–e** option produces a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **–f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **–e**, the following shell procedure helps maintain multiple versions of a file:

(shift; cat $*; echo '1,$p') | ed – $1

This works by performing a set of editing operations on an original ancestral file. This is done by combining the sequence of *ed* scripts given as all command line arguments except the first. These scripts are presumed to have been created with *diff* in the order given on the command line. The set of editing operations is then piped as an editing script to *ed* where all editing operations are performed on the ancestral file given as the first argument on the command line. The final version of the file is then printed on the standard output. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand.

Except in rare circumstances, *diff* finds the smallest sufficient set of file differences.

The −h option does a fast, less-rigorous job. It works only when changed stretches are short and well separated, but also works on files of unlimited length. The **−e** and **−f** cannot be used with the −h option.

Files

/tmp/d?????

/usr/lib/diffh for −h

See Also

cmp(C), comm(C), ed(C)

Diagnostics

Exit status is 0 for no differences, 1 for some differences, 2 for errors.

Notes

Editing scripts produced under the **−e** or **−f** option do not always work correctly on lines consisting of a single period (.).

Name

diff3 – Compares three files.

Syntax

diff3 [**–ex3**] file1 file2 file3

Description

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	All three files differ
====1	*File1* is different
====2	*File2* is different
====3	*File3* is different

The type of change suffered in converting a given range of a given file to some other range is indicated in one of these ways:

f : *n1* **a**	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f : *n1* , *n2* **c**	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the **–e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ==== and ====3. The **–x** option produces a script to incorporate changes flagged with "====". Similarly, the **–3** option produces a script to incorporate changes flagged with "====3". The following command applies a resulting editing script to *file1*:

```
(cat script; echo '1,$p') |ed – file1
```

Files

/tmp/d3*

/usr/lib/diff3prog

See Also

diff(C)

Notes

The **−e** option does not work properly for lines consisting of a single period.

The input file size limit is 64K bytes.

Name

dircmp – Compares directories.

Syntax

dircmp [−d] [−s] [−w*n*] dir1 dir2

Description

dircmp examines *dir1* and *dir2* and generates tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated in addition to a list that indicates whether the files common to both directories have the same contents.

There are three options available:

−d Performs a full *diff* on each pair of like-named files if the contents of the files are not identical.

−s Reports whether the files are different.

−w*n* Changes the width of the output line to *n* characters. The default width is 72.

See Also

cmp(C), diff(C).

Name

dirname – Delivers directory part of pathname.

Syntax

dirname string

Description

dirname delivers all but the last component of the pathname in *string* and prints the result on the standard output. If there is only one component in the pathname, only a "dot" is printed. It is normally used inside substitution marks (\` \`) within shell procedures.

The companion command *basename* deletes any prefix ending in a slash (**/**) and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output.

Examples

The following example sets the shell variable NAME to **/usr/src/cmd**:

NAME=\`dirname **/**usr**/**src**/**cmd**/**cat.c\`

This example prints **/a/b/c** on the standard output:

dirname **/**a**/**b**/**c**/**d

This example prints a "dot" on the standard output:

dirname file.ext

See Also

basename(C), sh(C)

Name

disable – Turns off terminals and printers.

Syntax

disable tty ...
disable [-c][-r[reason]] printers

Description

For terminals, this program manipulates the **/etc/ttys** file and signals *init* to disallow logins on a particular terminal. For printers, *disable* stops print requests from being sent to the named printer. The following options can be used:

−c Cancels any requests that are currently printing.

−r[*reason*] Associates a *reason* with disabling the printer. The *reason* applies to all printers listed up to the next **−r** option. If the **−r** option is not present or the **−r** option is given without a *reason*, then a default *reason* is used. *Reason* is reported by *lpstat*(C).

Examples

In this example, a printer named *linepr* is disabled because of a paper jam:

disable -r"paper jam" linepr

Files

/dev/tty*

/etc/ttys

/usr/spool/lp/*

See Also

login(M), enable(C), ttys(M), getty(M), init(M), lp(C), lpinit(C), lpstat(C), ungetty(M)

Name

diskcp, diskcmp – Copies or compares floppy disks.

Syntax

diskcp [-f] [-d] [-s] [-96]
diskcmp [-d] [-s] [-96]

Description

These commands provide easy copying of a source floppy disk. *dd*(C) is used to make an image of the source floppy (the one you wish to copy). On machines with one floppy drive *diskcp* temporarily transfers the image to the hard disk until a blank "target" floppy is inserted into the floppy drive. On machines with two floppy drives *dd* immediately places the image of the source floppy directly on the target floppy.

The options are:

-f Format the target floppy disk before the image is copied.

-d The computer has dual floppy drives. *diskcp* copies the image directly onto the target floppy.

-s Uses *sum*(C) to compare the contents of the source and target floppies; gives an error message if the two do not match.

-96
Instructs *diskcp* and *diskcmp* to process 96tpi floppies. The default (no flag) is 48tpi.

diskcmp functions similarly to *diskcp*. It compares the contents of one floppy disk with the contents of a second floppy disk using the *cmp* utility.

Examples

To make a copy of a floppy, place the source floppy in the drive and type:

```
diskcp
```

When *diskcp* is finished copying to the hard disk, it prompts you to insert the target floppy in the drive. If you specify the **-f** flag when you invoke *diskcp* , the program formats the target floppy. When the copy is finished, *diskcp* prompts if you would like to make another copy of the same source disk. If you enter 'n', it prompts if you would like to copy another source disk.

Specify the **-d** flag on the command line if you have two floppy drives:

```
diskcp -d
```

Notes

If *diskcp* encounters a write error while copying the source image to the target disk, it formats the disk and tries to write the source image again. This happens most often when an unformatted floppy is used and the **-f** flag is not specified.

Files

/usr/bin/diskcp
/usr/bin/diskcmp
/tmp/disk$$

See Also

cmp(C), dd(C), sum(C)

Name

divvy - Disk dividing utility

Syntax

divvy −b block_device −c character_device [−v virtual_drive] [−p physical_drive] [−i] [−m]

Description

divvy divides an *fdisk*(C) partition into a number of separate areas known as "divisions". A division is identified by unique major and minor device numbers and can be used for a file system, swap area, or for isolating bad spots on the device.

With *divvy* you can:

- Divide a disk or *fdisk* partition into separate devices.
- Create new file systems.
- Change the device names of file systems.
- Change the size of file systems.
- Remove file systems.

Options

Options to *divvy* are:

−b *block_device*
Major device number of block interface.

−c *character_device*
Major device number of character interface.

−v *virtual_device*
For dividing a virtual drive.

−p *physical_drive*
For dividing one of several physical disks that share the same controller.

−i Disk being divided will contain a **root** file system on division 0.

−m
Disk being divided should be made into a number of mountable file systems.

Usage

The device being divided must be a block device with a character interface. For example, to use *divvy* on a device with a block-interface major number 1 and character interface number of 1, enter:

divvy −b 1 −c 1

The **−v** option specifies which virtual drive to divide. The default is the active drive. Here, "virtual drive" is the same as an MS-DOS partition. Virtual drive numbers are determined with the *fdisk*(C) utility.

The **−p** option allows division of one of several physical disks sharing a controller. *divvy* defaults to the first physical device numbered "0." To access a second physical disk, use the **−p** 1 option.

The **−i** option specifies the device being divided will contain a **root** file system. With this option, device nodes are created relative to the new **root**, generally a hard disk, instead of the current **root**, often an installation floppy. A root filesystem and a recover area are created. *divvy* prompts for the size of the swap area. If the disk is large enough, then *divvy* prompts for a separate **/u** (user) filesystem. *divvy* also prompts for block-by-block control over the layout of the filesystem(s). If the root filesystem is large enough to require a scratch filesystem, (more than 40,000 blocks) then *divvy* will prompt for whether one should be created. *divvy* is invoked with the **−i** option during XENIX installation.

The **−m** option is used for initial installation on devices that will not be used as the root. It causes the user to be prompted for a number of file systems.

When *divvy* is invoked from the command line, you see a main menu:

n[ame]	Name or rename a division.
c[reate]	create a new file system on this division.
p[revent]	Prevent a new file system from being created on this division.
s[tart]	Start a division on a different block.
e[nd]	End a division on a different block.
r[estore]	Restore the original partition table.

Please enter your choice or 'q' to quit:

To choose a command, enter the first letter of the command, then press RETURN.

The *divvy* division table might look something like this:

Name	New File System?	#	First Block	Last Block
root	no, exists	0	0	13754
swap	no, exists	1	13755	15135
u	no, exists	2	15136	25135
	no	3	–	–
	no	4	–	–
	no	5	–	–
recover	no, exists	6	25136	25145
d1057all	no	7	0	25546

When pertinent, *divvy* also displays information about block allocation for bad tracks.

If you select option 'n', you can change the name of the device. *divvy* prompts you for the division number (from the *divvy* table displayed above), then for a new name.

Option 'c' causes a given division to become a new, empty file system when you exit from *divvy*. After using the 'c' option, you will see a 'yes' in the 'New File System?' column. If you use option 'p,' the 'yes' in the 'New File System?' column will change to a 'no', and the contents of the division will not change.

With the 's' or 'start' command, you can start a division on a different block number. With the 'e' or 'end' command, you can end a division on a different block number.

You can use these two commands to change the size of a partition. For example, if your disk is similar to the one in the sample *divvy* table above, and you want to make the **root** file system smaller and the **swap** area larger, do this:

> Make the **root** division smaller with the 'e' command.
>
> Use the 's' command to make the **swap** division bigger.

Note that if any of the divisions overlap, *divvy* will complain when you try to exit and put you back in the menus to correct the situation.

The 'r' or 'restore' command restores the original partition table. This is useful if you make a serious mistake and want to return to where you started.

When you exit from *divvy*, you are prompted whether you want to save any changes you made, or exit without saving the changes. At this time, you can also go back to the *divvy* menu, and may also have the option to reinstall the original, default partition table.

See Also

badtrk(M), fdisk(C), fsck(C), hd(M), mkdev(C), mkfs(C), mknod(C)

Notes

divvy requires kernel level support from the device driver. If *divvy* lists the size of a disk as "0" blocks, or displays the following error messages, the device may not support dividing:

    cannot read division table

or:

    cannot get drive parameters

These errors may also occur if the prerequisite programs *fdisk* and *badtrk* are not run correctly.

If you change the size of filesystems (such as **/u**) after you have installed a XENIX filesystem, you will have to run *mkfs* on the filesystem and reinstall the files that are kept there. This is because the free list for that filesystem has changed. Be sure to backup the files in any filesystem you intend to change, using *backup*(C), *tar*(C), or *cpio*(C), before you run *divvy* . After XENIX is installed, the bounds of the **root** file system must not be changed.

During installation, if the filesystem on division 0 (generally root) becomes or remains large enough to require a scratch area during *fsck*, and one does not already exist, *divvy* prompts for whether one should be created. (The resulting filesystem, **/dev/scratch**, is used by *autoboot* if it runs *fsck*. **/dev/scratch** should also be entered when *fsck* prompts for a scratch file name, provided that the filesystem being checked is not larger than the root filesystem.) If all disk divisions have been used up, *divvy* will not prompt for a scratch filesystem, even if the root filesystem is large enough to require one.

Name

dmesg - Displays the system messages on the console.

Syntax

dmesg [-]

Description

The *dmesg* command displays all the system messages that have been generated since the last time the system was booted. If the option – is specified, it displays only those messages that have been generated since the last time the *dmesg* command was performed.

dmesg can be invoked periodically by placing instructions in the file **/usr/lib/crontab** . It can also be invoked automatically by **/etc/rc** whenever the system is booted. See "Notes", below.

dmesg logs all error messages it prints in **/usr/adm/messages.** If *dmesg* is invoked automatically, the **messages** file continues to grow and can become very large. The system administrator should occasionally erase its contents.

Files

/etc/dmesg
/usr/adm/messages
/usr/adm/msgbuf

Notes

dmesg is included in this release for backwards compatibility only. The device **/dev/error** provides a more flexible means of logging error messages, and is recommended over *dmesg*. See *error*(M) for more information.

See Also

cron(C), error(M), messages(M)

Name

dos, doscat, doscp, dosdir, dosformat, dosls, dosrm, dosrmdir – Access DOS files.

Syntax

doscat [–r |–m] file ...

doscp [–r |–m] file1 file2

doscp [–r |–m] file ... directory

dosdir directory ...

dosformat [–fqv] drive

dosls directory ...

dosmkdir directory ...

dosrm file ...

dosrmdir directory ...

Description

The *dos* commands provide access to the files and directories on MS-DOS disks and on a DOS partition of the hard disk. The commands perform the following actions:

doscat Copies one or more DOS files to the standard output. If **–r** is given, the files are copied without newline conversions. If **–m** is given, the files are copied with newline conversions (see "Conversions" below).

doscp Copies files between a DOS disk and a XENIX filesystem. If *file1* and *file2* are given, *file1* is copied to *file2*. If a *directory* is given, one or more *files* are copied to that directory. If **–r** is given, the files are copied without newline conversions. If **–m** is given, the files are copied with newline conversions (see "Conversions" below).

dosdir Lists DOS files in the standard DOS style directory format.

dosformat Creates a DOS 2.0 formatted diskette. The drive may be specified in either DOS drive convention, using the default file **/etc/default/msdos**, or using the XENIX special file name. The **−f** option suppresses the interactive feature. The **−q** (quiet) option is used to suppress information normally displayed during *dosformat* . The **−q** option does not suppress the interactive feature. The **−v** option prompts the user for a volume label after the diskette has been formatted. The maximum size of the volume label is 11 characters.

dosls Lists DOS directories and files in a XENIX style (see *ls*(C)).

dosrm Removes files from a DOS disk.

dosmkdir Creates a directory on a DOS disk.

dosrmdir Deletes directories from a DOS disk.

The *file* and *directory* arguments for DOS files and directories have the form:

device:name

where *device* is a XENIX pathname for the special device file containing the DOS disk, and *name* is a pathname to a file or directory on the DOS disk. The two components are separated by a colon (:). For example, the argument:

/dev/fd0:/src/file.asm

specifies the DOS file, **file.asm**, in the directory, **/src**, on the disk in the device file **/dev/fd0.** Note that slashes (and not backslashes) are used as filename separators for DOS pathnames. Arguments without a *device:* are assumed to be XENIX files.

For convenience, the user configurable default file, **/etc/default/msdos**, can define DOS drive names to be used in place of the special device file pathnames. It may contain the following lines:

A=/dev/fd0
C=/dev/hd0d
D=/dev/hd1d

The drive letter "A" may be used in place of special device file pathname **/dev/fd0** when referencing DOS files (see "Examples" below). The drive letter "C" or "D" refer to the DOS partition on the first or second hard disk.

The commands operate on the following kinds of disks:

DOS partitions on a hard disk
5 1/4 inch DOS
8 or 9 sectors per track
40 tracks per side
1 or 2 sides
DOS versions 1.0, 2.0 or 3.0

Conversions

All DOS text files use a carriage-return/linefeed combination, CR-LF , to indicate a newline. XENIX uses a single newline LF character. When the *doscat* and *doscp* commands transfer DOS text files to XENIX , they automatically strip the CR. When text files are transferred to DOS , the commands insert a CR before each LF character.

Under some circumstances the automatic newline conversions do not occur. The **-m** option may be used to insure the newline conversion. The **-r** option can be used to override the automatic conversion and force the command to perform a true byte copy regardless of file type.

Examples

```
doscat /dev/fd0:/docs/memo.txt
doscat /tmp/f1 /tmp/f2 /dev/fd0:/src/file.asm

dosdir /dev/fd0:/src
dosdir A:/src A:/dev

doscp /tmp/myfile.txt /dev/fd0:/docs/memo.txt
doscp /tmp/f1 /tmp/f2 /dev/fd0:/mydir

dosformat A:
dosformat /dev/fd0

dosls /dev/fd0:/src
dosls B:

dosmkdir /dev/fd0:/usr/docs

dosrm /dev/fd0:/docs/memo.txt
dosrm A:/docs/memo1.txt

dosrmdir /dev/fd0:/usr/docs
```

Files

/etc/default/msdos	Default information
/dev/fd*	Floppy disk devices
/dev/hd*	Hard disk devices

See Also

assign(C), dtype(C)

Notes

It is not possible to refer to DOS directories with wild card specifications. The programs mentioned above cooperate among themselves so no two programs will access the same DOS disk. Only one process will access a given DOS disk at any time, while other processes wait. If a process has to wait too long, it displays the error message, "can't seize a device," and exits with an exit code of 1.

The following hard disk devices:

/dev/hd0d
/dev/rhd0d
/dev/hd1d
/dev/rhd1d

are similar to **/dev/hd0a** in that the disk driver determines which partition is the DOS partition and uses that as *hd?d*. This means that software using the DOS partition does not need to know which partition is DOS (the disk driver determines that).

The XENIX Development System supports the creation of DOS executable files, using *cc* (CP). Refer to the XENIX *C User's Guide* and *C Library Guide* for more information on using XENIX to create programs suitable for DOS systems.

Name

dparam – Displays/changes hard disk characteristics.

Syntax

dparam [**–w**]
dparam /dev/rhd[**01**]**0** [characteristics]

Description

The *dparam* command displays or changes the hard disk characteristics currently in effect. These changes go into effect immediately and are also written to the master boot block for subsequent boots. If a non-standard hard disk is used, this utility must be called before accessing the drive.

-w Causes a copy of **/etc/masterboot** to be copied to disk to ensure that non-standard hard disks are supported for the specified drive. This call must precede a call to write non-standard disk parameters for the desired parameters to be saved correctly in the masterboot block.

When called without options or disk characteristics, *dparam* prints the current disk characteristics (on the standard output) for the specified hard disk. These values are printed in the same order as the argument list.

When writing characteristics for the specified hard disk, *dparam* changes the current disk controller status and updates the masterboot block. The argument ordering is critical and must be entered as specified below. All characteristics must be entered when writing disk characteristics, otherwise an error is returned. Hard disk characteristics (in respective order) are:

number of cylinders	total number of cylinders on the hard disk
number of heads	number of heads
write cylinder	hardware specific, consult your hardware manual
write precompensation cylinder	hardware specific, consult your hardware manual

ecc	number of bits of error correction on I/O transfers, consult your hardware manual
control	very hardware specific, consult your hardware manual
landing zone cylinder	where to park heads after shutting down the system
number of sectors per track	number of sectors per track on the hard disk

Examples

dparam -w

dparam /dev/rhd10

dparam /dev/rhd00 700 4 256 180 5 0 640 17

Notes

This utility changes the kernel's view of the hard disk parameters. It may be subject to restrictions imposed by the hardware configuration.

Name

dtype – Determines disk type.

Syntax

dtype **[–s]** device ...

Description

dtype determines type of disk, prints pertinent information on the standard output unless the silent (–s) option is selected, and exits with a corresponding code (see below). When more than one argument is given, the exit code corresponds to the last argument.

Disk Type	Exit Code	Message (optional)
Misc.	60	error (specified)
	61	empty or unrecognized data
Storage	70	dump format, volume n
	71	tar format[, extent e of n]
	72	cpio format
	73	cpio character (–c) format
MS-DOS	80	DOS 1.x, 8 sec/track, single sided
	81	DOS 1.x, 8 sec/track, dual sided
	90	DOS 2.x, 8 sec/track, single sided
	91	DOS 2.x, 8 sec/track, dual sided
	92	DOS 2.x, 9 sec/track, single sided
	93	DOS 2.x, 9 sec/track, dual sided
	94	DOS 2.x fixed disk
XENIX	120	XENIX 2.x filesystem [needs fsck]
	130	XENIX 3.x or later filesystem [needs fsck]

Notes

word-swapped refers to byte ordering of long words in relation to the host system.

XENIX file systems and dump and cpio binary formats may not be recognized if created on a foreign system. This is due to such system differences as byte and word swapping and structure alignment.

This utility only works reliably for floppy diskettes.

Name

du – Summarizes disk usage.

Syntax

du [**−afrsu**] [names]

Description

du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, the current directory is used.

The optional argument **−s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **−a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

The **−f** option causes **du** to display the usage of files in the current file system only. Directories containing mounted file systems will be ignored. The **−u** option causes *du* to ignore files that have more than one link.

du is normally silent about directories that cannot be read, files that cannot be opened, etc. The **−r** option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

Notes

If the **−a** option is not used, nondirectories given as arguments are not listed.

If there are too many distinct linked files, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count.

This utility reports sizes in 512 byte blocks. Systems which define a block as 1024 characters, "round-off" the size of files containing 511 or fewer bytes to 1 block. *du* interprets 1 block from a 1024 byte block system as 2 of its own 512 byte blocks. Thus a 500 byte file is interpreted as 2 blocks rather than 1. Refer to the **machine**(HW) manual page for the block size used by your system.

Name

dump – Performs incremental file system backup.

Syntax

dump [key [arguments] filesystem]

Description

dump copies to the specified device all files changed after a certain date in the *filesystem*. The *key* specifies the date and other options about the backup, where a *key* consists of characters from the set **0123456789kfusd**. The meanings of these characters are described below:

f Places the backup on the next *argument* file instead of the default device.

u If the backup completes successfully, writes the date of the beginning of the backup to the file **/etc/ddate**. This file records a separate date for each file system and each backup level.

0–9 This number is the "backup level". Backs up all files modified since the last date stored in the file **/etc/ddate** for the same file system at lesser levels. If no date is determined by the level, the beginning of time is assumed; thus the option **0** causes the entire file system to be backed up.

s For backups to magnetic tape, the size of the tape specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, *dump* will wait for reels to be changed. The default size is 2,300 feet.

d For backups to magnetic tape, the density of the tape, expressed in BPI, is taken from the next *argument*. This is used in calculating the amount of tape used per write. The default is 1600.

k This option is used when backing up to a block-structured device, such as a floppy disk. The size (in K-bytes) of the volume being written is taken from the next *argument*. If the **k** argument is specified, any s and **d** arguments are ignored. The default is to use s and **d**.

If no arguments are given, the *key* is assumed to be **9u** and a default file system is backed up to the default device.

The first backup should be a full level-0 backup:

dump 0u

Next, periodic level 9 backups should be made on an exponential progression of tapes or floppies:

dump 9u

This progression is shown as follows:

1 2 1 3 1 2 1 4 ...

where backup 1 is used every other time, backup 2 every fourth, backup 3 every eighth, etc.) When the level-9 incremental backup becomes unmanageable because a tape is full or too many floppies are required, a level-1 backup should be made:

dump 1u

After this, the exponential series should progress as if uninterrupted. These level-9 backups are based on the level-1 backup, which is based on the level-0 full backup. This progression of levels of backups can be carried as far as desired.

The default file system and the backup device depend on the settings of the variables DISK and TAPE, respectively, in the file **/etc/default/dump.**

Files

/etc/ddate	Records backup dates of file system/level
etc/default/dump	Default dump information

See Also

XENIX *Operations Guide*
cpio(C), default(M), dumpdir(C), restore(C), dump(F)

Diagnostics

If the backup requires more than one volume (where a volume is likely to be a floppy disk or tape), you will be asked to change volumes. Press RETURN after changing volumes.

Notes

Sizes are based on 1600 BPI for blocked tape; the raw magnetic tape device has to be used to approach these densities. Write errors to the backup device are usually fatal. Read errors on the file system are ignored.

It is not possible to successfully *restore* an entire active root file system.

Warning

When backing up to floppy disks, be sure to have enough *formatted* floppies ready before starting a backup.

Name

dumpdir – Prints the names of files on a backup archive.

Syntax

dumpdir [**f** filename]

Description

dumpdir is used to list the names and inode numbers of all files and directories on an archive written with the *backup* command. This is most useful when attempting to determine the location of a particular file in a set of backup archives.

The **f** option causes *filename* to be used as the name of the backup device instead of the default. The backup device depends on the setting of the variable TAPE in the file **/etc/default/dumpdir.** The device specified as TAPE can be any type of backup device supported by the system (for example, a floppy drive or cartridge tape drive).

Files

rst* Temporary files

See Also

backup(C), restore(C), default(M)

Diagnostics

If the backup extends over more than one volume (where a volume is likely a floppy disk or tape), you will be asked to change volumes. Press RETURN after changing volumes.

Name

echo – Echoes arguments.

Syntax

echo [arg] ...
/bin/echo [arg] ...

Description

echo writes its arguments separated by blanks and terminated by a newline on the standard output. *echo* also understands C-like escape conventions. The following escape sequences need to be quoted so that the shell interprets them correctly:

\b Backspace

\c Prints line without newline

\f Form feed

\n Newline

\r Carriage return

\t Tab

\v Vertical tab

**** Backslash

*****n* The 8-bit character whose ASCII code is the 1, 2 or 3-digit octal number *n* must start with a zero

echo is useful for producing diagnostics in command files and for sending known data into a pipe.

See Also

sh(C)

Notes

The *csh*(C) has a built-in *echo* utility which has a different syntax than this *echo*. Be aware that users running under *csh* will get the built-in *echo* unless they specify **/bin/echo** .

Name

ed – Invokes the text editor.

Syntax

ed [–] [–p string **] [** file **]**

Description

ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. *ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

The options are:

– Suppresses the printing of character counts by the *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and the **!** prompt after a !*shell command*.

–p Allows the user to specify a prompt string.

ed supports formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty -tabs** or **stty tab3** mode (see *stty*(C), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: While inputting text, tab characters are expanded to every eighth column as the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by entering a period (.) alone at the beginning of a line.

ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression specifies a set of character strings. A member of this set of strings is said to be *matched* by the regular expression. The regular expressions allowed by *ed* are constructed as follows:

The following one-character regular expressions match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character regular expression that matches itself.

1.2 A backslash (\) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

  a. ., *, [, and \ (dot, star, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4 below).

  b. ^ (caret), which is special at the *beginning* of an *entire* regular expression (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

  c. $ (dollar sign), which is special at the *end* of an entire regular expression (see 3.2 below).

  d. The character used to bound (i.e., delimit) an entire regular expression, which is special for that regular expression (for example, see how slash (/) is used in the *g* command below).

1.3 A period (.) is a one-character regular expression that matches any character except newline.

1.4 A nonempty string of characters enclosed in square brackets ([]) is a one-character regular expression that matches *any one* character in that string. If, however, the first character of the string is a caret (^), the one-character regular expression matches any character *except* newline and the remaining characters in the string. The star (*) has this special meaning *only* if it occurs first in the string. The dash (**–**) may be used to indicate a range of consecutive ASCII characters; for example, [0–9] is equivalent to [0123456789]. The dash (**–**) loses this special meaning if it occurs first (after an initial caret (^), if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial caret (^), if any); e.g., **[]a–f]** matches either a right square bracket (]) or one of the letters "a" through "f" inclusive. Dot, star, left bracket, and the backslash lose their special meaning within such a string of characters.

The following rules may be used to construct regular expressions from one-character regular expressions:

2.1 A one-character regular expression matches whatever the one-character regular expression matches.

2.2 A one-character regular expression followed by a star (*) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character regular expression followed by **\{***m***\}**, **\{***m***,\}**, or **\{***m,n***\}** is a regular expression that matches a *range* of occurrences of the one-character regular expression. The values of *m* and *n* must be nonnegative integers less than 255; **\{***m***\}** matches *exactly m* occurrences; **\{***m***,\}** matches *at least m* occurrences; **\{***m,n***\}** matches any number of occurrences between *m* and *n,* inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

2.4 The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

2.5 A regular expression enclosed between the character sequences **\(** and **\)** is a regular expression that matches whatever the unadorned regular expression matches. See 2.6 below for a discussion of why this is useful.

2.6 The expression *****n* matches the same string of characters as was matched by an expression enclosed between **\(** and **\)** *earlier* in the same regular expression. Here *n* is a digit; the subexpression specified is that beginning with the *n*-th occurrence of **\(** counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

Finally, an *entire regular expression* may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A caret (^) at the beginning of an entire regular expression constrains that regular expression to match an *initial* segment of a line.

3.2 A dollar sign ($) at the end of an entire regular expression constrains that regular expression to match a *final* segment of a line. The construction ^*entire regular expression*$ constrains the entire regular expression to match the entire line.

The null regular expression (e.g., //) is equivalent to the last regular expression encountered.

To understand addressing in *ed* , it is necessary to know that there is a *current line* at all times. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.

2. The character $ addresses the last line of the buffer.

3. A decimal number *n* addresses the *n*-th line of the buffer.

4. '*x* addresses the line marked with the mark name character *x*, which must be a lowercase letter. Lines are marked with the *k* command described below.

5. A regular expression enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched.

6. A regular expression enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the regular expression. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *Files* below.

7. An address followed by a plus sign (+) or a minus sign (−) followed by a decimal number specifies that address plus or minus the indicated number of lines. The plus sign may be omitted.

8. If an address begins with **+** or **−**, the addition or subtraction is taken with respect to the current line; e.g, **−5** is understood to mean **.−5**.

9. If an address ends with **+** or **−**, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address **−** refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to **−**.) Moreover, trailing **+** and **−** characters have a cumulative effect, so **−−** refers to the current line less 2.

10. For convenience, a comma (**,**) stands for the address pair **1,$**, while a semicolon (**;**) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last address(es) are used.

Typically, addresses are separated from each other by a comma (**,**). They may also be separated by a semicolon (**;**). In the latter case, the current line (**.**) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5 and 6 above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **p** or by **l**, in which case the current line is either printed or listed, respectively, as discussed below under the *p* and *l* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; dot is left at the last inserted line, or, if there were no inserted lines, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer.

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; dot is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e ***file***

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; dot is set to the last line of the buffer. If no filename is given, the currently remembered filename, if any, is used (see the *f* command). The number of characters read is typed; ***file*** is remembered for possible use as a default filename in subsequent *e*, *r*, and *w* commands. If ***file*** begins with an exclamation (!), the rest of the line is taken to be a shell command. The output of this command is read for the *e* and *r* commands. For the *w* command, the file is used as the standard input for the specified command. Such a shell command is *not* remembered as the current filename.

E ***file***

The *E*dit command is like *e*, except the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f ***file***

If ***file*** is given, the *f*ilename command changes the currently remembered filename to ***file***; otherwise, it prints the currently remembered filename.

(1,$)g*/regular-expression/command list*

In the *g*lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multiline list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted; the . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *Notes* and the last paragraph before *Files* below.

(1,$)G/*regular-expression*/

In the interactive *G*lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, that line is printed, dot (.) is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. 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 entering an INTERRUPT.

h

The *h*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous diagnostic if there was one. The *H* command alternately turns this mode on and off; it is initially on.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; dot is left at the last inserted line, or if there were no inserted lines, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command.

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate newline characters. If only one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lowercase letter. The address '*x* then addresses this line; dot is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few nonprinting characters (e.g., tab, backspace) are represented by mnemonic overstrikes, all other nonprinting characters are printed in octal, and long lines are folded. An *l* command may be appended to any command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines; dot is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; dot is left at the last line printed. The *n* command may be appended to any command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; dot is left at the last line printed. The *p* command may be appended to any command other than *e*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially on.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done.

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

($)r ***file***

The *r*ead command reads in the given file after the addressed line. If no filename is given, the currently remembered filename, if any, is used (see *e* and *f* commands). The currently remembered filename is *not* changed unless *file* is the very first filename mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; dot is set to the last line read in. If *file* begins with !, the rest of the line is taken to be a shell (*sh*(C)) command whose output is to be read. Such a shell command is *not* remembered as the current filename.

(.,.)s/*regular-expression*/*replacement*/ or

(.,.)s/*regular-expression*/*replacement*/g or

(.,.)s/*regular-expression*/*replacement*/n n=1-512

The *s*ubstitute command searches each addressed line for an occurrence of the specified regular expression. In each line in which a match is found, all (nonoverlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or newline may be used instead of **/** to delimit the regular expression and the *replacement*; dot is left at the last line on which a substitution occurred.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the regular expression on the current line. The special meaning of the ampersand in this context may be suppressed by preceding it with a backslash. The characters ***n***, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified regular expression enclosed between **\\(** and **\\)**. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of **\\(** starting from the left. When the character **%** is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The **%** loses its special meaning when it is in a replacement string of more than one character or is preceded by a ****.

A line may be split by substituting a newline character into it. The newline in the *replacement* must be escaped by preceding it with a ****. Such a substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); dot is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a, c, d, g, i, j, m, r, s, t, v, G,* or *V* command.

(1,$)v/*regular-expression***/***command list*

This command is the same as the global command *g* except that the *command list* is executed with dot initially set to every line that does *not* match the regular expression.

(1,$)V/*regular-expression***/**

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the regular expression.

(1,$)w *file*

The *write* command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writeable by everyone), unless the *umask* setting (see *sh*(C)) dictates otherwise. The currently remembered filename is *not* changed unless *file* is the very first filename mentioned since *ed* was invoked. If no filename is given, the currently remembered filename, if any, is used (see *e* and *f* commands); dot is unchanged. If the command is successful, the number of characters written is displayed. If *file* begins with an exclamation (!), the rest of the line is taken to be a shell command to which the addressed lines are supplied as the standard input. Such a shell command is *not* remembered as the current filename.

($)=

The line number of the addressed line is typed; dot is unchanged by this command.

!*shell command*

The remainder of the line after the ! is sent to the XENIX shell (*sh*(C)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered filename; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; dot is unchanged.

(.+1)

An address alone on a line causes the addressed line to be printed. A RETURN alone on a line is equivalent to **.+1p**. This is useful for stepping forward through the editing buffer a line at a time.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a question mark (?) and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per filename, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last newline. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on), cannot be edited by *ed*.

If the closing delimiter of a regular expression or of a replacement string (e.g., /) would be the last character before a newline, that delimiter may be omitted, in which case the addressed line is printed. Thus, the following pairs of commands are equivalent:

```
s/s1/s2s/s1/s2/p
g/s1g/s1/p
?s1?s1?
```

Files

/tmp/e#	Temporary; # is the process number
ed.hup	Work is saved here if the terminal is hung up

See Also

grep(C), sed(C), sh(C), stty(C), regexp(S)

Diagnostics

?	Command errors
? *file*	An inaccessible file

Use the *h*elp and *H*elp commands for detailed explanations.

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands: it prints **?** and allows you to continue editing. A second *e* or *q* command at this point will take effect. The dash (**−**) command-line option inhibits this feature.

Notes

An exclamation (!) command cannot be subject to a *g* or a *v* command.

The ! command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(C)).

The sequence **\n** in a regular expression does not match any character.

The *l* command mishandles DEL.

Because 0 is an illegal address for the *w* command, it is not possible to create an empty file with *ed*.

Characters are mashed to 7 bits on input.

If the editor input is coming from a command file (i.e., *ed file* < *ed-cmd-file*), the editor will exit at the first failure of a command in the command file.

Name

enable – Turns on terminals and line printers.

Syntax

enable tty ...
enable printers

Description

For terminals this program manipulates the **/etc/ttys** file and signals *init* to allow logins on a particular terminal.

For line printers, *enable* activates the named printers and enables them to print requests taken by *lp*(C). Use *lpstat*(C) to find the status of the printers.

Examples

A simple command to enable **tty01** follows:

```
enable tty01
```

Files

/dev/tty*

/etc/ttys

/usr/spool/lp/*

See Also

disable(C), getty(M), init(M), login(M), lp(C), lpstat(C), ttys(M), ungetty(M)

Name

env – Sets environment for command execution.

Syntax

env [–] [name=value] ... [command args]

Description

env obtains the current *environment,* modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The – flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

See Also

sh(C), exec(S), profile(F), environ(M)

Notes

The 2.3 *printenv* command has been replaced in XENIX 3.0 and System V by the *env* command. The *printenv* shipped is a link to the command *env.*

Name

ex – Invokes a text editor.

Syntax

ex [**–**] [**–v**] [**–**t tag] [**–r**] [**+***lineno*] name ...

Description

ex is the root of the editors *ex* and *vi*. *ex* is a superset of *ed*, whose most notable extension is a display editing facility. Display based editing is the focus of *vi*.

If you have not used *ed*, or if you are a casual user, you will find that *edit* is most convenient for you. It avoids some of the complexities of *ex* which is used mostly by systems programmers and persons very familiar with *ed*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(C), a command which focuses on the display editing portion of *ex*.

For ed Users

If you have used *ed* you will find that *ex* has a number of new features. Intelligent terminals and high-speed terminals are very pleasant to use with *vi*. Generally, the *ex* editor uses far more of the capabilities of terminals than *ed* does. It uses the terminal capability database *termcap* (M) and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The *ex* editor makes use of features such as insert and delete character and line in its **visual** command mode, which can be abbreviated **vi** , which is the central mode of editing when using *vi*(C). There is also an interline editing **open** command, (**o**) that works on all terminals.

ex contains a number of features for easily viewing the text of a file. The **z** command gives easy access to windows of text. Hitting Ctrl-D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting the RETURN key. Of course, the screen-oriented **visual** mode gives constant access to editing context.

ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change. *ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents the overwriting of existing files unless you have edited them, so that you do not accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

ex has several features for editing more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to edit each in turn. You can also give the **next** command a list of filenames, or a pattern used by the shell to specify a new set of files to be edited. In general, filenames in the editor may be formed with full shell metasyntax. The metacharacter "%" is also available in forming filenames and is replaced by the name of the current file. For editing large groups of related files, you can use *ex's* **tag** command to quickly locate functions and other important points in any of the files. This is useful when you want to find the definition of a particular function in a large program. The command *ctags*(CP) builds a *tags* file or a group of C programs.

For moving text between files and within a file, the editor has a group of buffers named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

The command **&** repeats the last **substitute** command. There is also a confirmed substitute command. You give a range of substitutions to be done and the editor interactively prompts you whether each substitution is desired.

You can use the **substitute** command in *ex* to systematically convert the case of letters between uppercase and lowercase. It is possible to ignore case in searches and substitutions. *ex* also allows regular expressions that match words to be constructed. This is convenient, for example, when searching for the word "edit" if your document also contains the word "editor."

ex has a set of *options* that you can set. One option which is very useful is the *autoindent* option that allows the editor to automatically supply leading white space to align text. You can then press Ctrl-D to backtab, space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join** (**j**) command which supplies whitespace between joined lines automatically, the commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort.*

Files

/usr/lib/ex3.7strings	Error messages
/usr/lib/ex3.7recover	Recover command
/usr/lib/ex3.7preserve	Preserve command
/etc/termcap	Describes capabilities of terminals
$HOME/.exrc	Editor startup file
/tmp/Ex*nnnnn*	Editor temporary
/tmp/Rx*nnnnn*	Named buffer temporary
/usr/preserve	Preservation directory

See Also

awk(C), ctags(CP), ed(C), grep(C), sed(C), termcap(M), vi(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line "–" option is used.

There is no easy way to do a single scan ignoring case.

Because of the implementation of the arguments to *next,* only 512 bytes of argument list are allowed there.

The format of **/etc/termcap** and the large number of capabilities of terminals used by the editor cause terminal type setup to be rather slow.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

Name

expr – Evaluates arguments as an expression.

Syntax

expr arguments

Description

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that zero is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Expressions should be quoted by the shell, since many of the characters that have special meaning in the shell also have special meaning in *expr.* The list is in order of increasing precedence, with equal precedence operators grouped within braces ({ and }).

expr **|** *expr*
: Returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr **&** *expr*
: Returns the first *expr* if neither *expr* is null nor **0**, otherwise returns **0**.

expr { **=**, **>**, **>=**, **<**, **<=**, **!=** } *expr*
: Returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { **+**, **–** } *expr*
: Addition or subtraction of integer-valued arguments.

expr { *****, **/**, **%** } *expr*
: Multiplication, division, or remainder of the integer-valued arguments.

expr **:** *expr*
: The matching operator **:** compares the first argument with the second argument which must be a regular expression; regular expression syntax is the same as that of *ed*(C), except that all patterns are "anchored" (i.e., begin with a caret (^))

and therefore the caret is not a special character in that context. (Note that in the shell, the caret has the same meaning as the pipe symbol (|).) Normally the matching operator returns the number of characters matched (zero on failure). Alternatively, the **\(...\)** pattern symbols can be used to return a portion of the first argument.

Examples

1. a=`expr $a + 1`

   Adds 1 to the shell variable **a**.

2. # For $a equal to either "/usr/abc/file" or just "/file"
   ´expr $a : .*/\(.*\) | $a´

   Returns the last segment of a pathname (i.e., file). Watch out for the slash alone as an argument: *expr* will take it as the division operator (see *Notes* on the next page).

3. expr $VAR : ´.*´

   Returns the number of characters in **$VAR**.

See Also

ed(C), sh(C)

Diagnostics

As a side effect of expression evaluation, *expr* returns the following exit values:

0	If the expression is neither null nor zero
1	If the expression is null or zero
2	For invalid expressions

Other diagnostics include:

syntax error For operator/operand errors

nonnumeric argument
If arithmetic is attempted on such a string

Notes

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an equals sign (=), the command:

```
expr $a = =
```

looks like:

```
expr = = =
```

Thus the arguments are passed to *expr* (and will all be taken as the = operator). The following permits comparing equals signs:

```
expr X$a = X=
```

Name

factor – Factor a number.

Syntax

factor [number]

Description

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{46} (about 7.2×10^{13}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

The time it takes to factor a number, *n*, is proportional to $\sqrt{n}$. It usually takes longer to factor a prime or the square of a prime, than to factor other numbers.

Diagnostics

factor returns an error message if the supplied input value is greater than 2^{46} or is not an integer number.

Name

false – Returns with a nonzero exit value.

Syntax

false

Description

false does nothing except return with a nonzero exit value. *true*(C), *false's* counterpart, does nothing except return with a zero exit value. "False" is typically used in shell procedures such as:

```
until false
do
        command
done
```

See Also

sh(C), true(C)

Diagnostics

false is any non-zero value.

Name

fdisk – Maintain disk partitions.

Syntax

fdisk [[**-p**] [**-ad** *partition*] [**-c** *partition start size*] [**-f** *devicename*]]

Description

fdisk displays information about disk partitions. *fdisk* also creates and deletes disk partitions and changes the active partition. *fdisk* functionality is a superset of the MS-DOS command of the same name. *fdisk* is usually used interactively from a menu.

The hard disk has at most four partitions. Only one partition is active at any given time. It is possible to assign a different operating system to each partition. Once a partition is made active, the operating system resident in that partition boots automatically once the current operating system is halted.

To use XENIX, at least one partition must be assigned to XENIX.

The *fdisk* utility does not allocate the first track or the last cylinder on the hard disk when the "Use Entire Disk for XENIX" option is used. The "Use Entire Disk for XENIX" option always leaves the first track unassigned, but allows you to allocate the last cylinder on the hard disk. The first track on the hard disk is reserved for masterboot and the last cylinder is generally used when running hard disk diagnostics. You should not allocate the last cylinder if you plan to run diagnostics on your hard disk.

For example, if a disk has 2442 tracks, *fdisk* reports these as tracks 0-2441. If your hard disk has 4 heads, *fdisk* will assign (using the "Use Entire Disk for XENIX" option) tracks 1-2437. (Track 0 is reserved for masterboot.) The last cylinder (tracks 2438-2441) is not assigned with the "Use Entire Disk for XENIX" option.

Partitions are defined by a "partition table" at the end of the master boot block. The partition table provides the location and size of the partitions on the disk. The partition table also defines the active partition. Each partition can be assigned to XENIX, DOS, or some other operating system. Once a DOS partition is set up, DOS files and directories resident in the DOS partition may be accessed while running XENIX by means of the *dos*(C) commands. DOS may be booted without the DOS partition being active via the "boot:dos" command. See *boot*(HW).

Arguments

−p, -a, -d, -c
These flags are used to invoke *fdisk* non-interactively:

−p	prints out the disk partition table.
−a *number*	activates the specified partition number.
−d *number*	deletes the specified partition number.
−c *number start size*	creates partition with specified start and size.

−f *name*
Open device *name* and read the partition table associated with that device's partition. The default is **/dev/hd00.**

Options

The *fdisk* command displays a prompt and a menu of five options. Updates to the disk are not made until you enter "q" from the main menu.

1. Display Partition Table.
   This option displays a table of information about each partition on the hard disk. The PARTITION column gives the partition number. The STATUS column tells whether the partition is active (A) or inactive (I). TYPE tells whether the partition is XENIX, DOS, or "other". The option also displays the starting track, ending track and total number of tracks in each partition.

2. Use Entire Disk for XENIX.
   fdisk creates one partition that includes all the tracks on the disk, except the first track and the last cylinder. This partition is assigned to XENIX and is designated the active partition.

3. Create XENIX Partition
   This option allows the creation of a partition by altering the partition table. *fdisk* reports the number of tracks available for each partition and the number of tracks in use. *fdisk* prompts for the partition to create, the starting track and size in tracks. The change is written to the operating system and the hard disk when you enter "q" from the main menu.

4. Activate Partition
   This option activates the specified partition. Only one partition may be active at a time. The change is not effective until you exit. The operating system residing in the newly activated partition boots once the current operating system is halted.

5. Delete Partition
   This option requests which partition you wish to delete. *fdisk* reports the new available amount of disk space in tracks. The change is not effective until you exit.

Exit the *fdisk* program by typing a 'q' at the main *fdisk* menu. Your changes are now written to the operating system and the hard disk.

Notes

The minimum recommended size for a XENIX partition is 5 megabytes.

Since *fdisk* is intended for use with DOS, it may not work with all operating system combinations.

See also

dos(C), hd(HW).

Name

file – Determines file type.

Syntax

file [**−m**] file ...

file [**−m**] **−f** namesfile

Description

file performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language.

If the **−f** option is given, *file* takes the list of filenames from *namesfile*. If the **−m** option is given, *file* sets the access time for the examined file to the current time. Otherwise, the access time remains unchanged.

Several object file formats are recognized. For **a.out** and **x.out** format object files, the relationship of *cc* flags to file classification is **−i** for "separate", **−n** for "pure", and **−s** for not "not stripped."

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

It can make mistakes: in particular it often mistakes command files for C programs.

Name

find – Finds files.

Syntax

find pathname-list expression

Description

find recursively descends the directory hierarchy for each pathname in the *pathname-list* (i.e., one or more pathnames) seeking files that match a Boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where **+***n* means more than *n*, **−***n* means less than *n* and *n* means exactly *n*.

−name *file* — True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for the left bracket (**[**), the question mark (**?**) and the star (*****).

−perm *onum* — True if the file permission flags exactly match the octal number *onum* (see *chmod*(C)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(S)) become significant and the flags are compared.

−type *x* — True if the type of the file is *x*, where *c* is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, first-in-first-out, or plain file respectively.

−links *n* — True if the file has *n* links.

−user *uname* — True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

−group *gname* — True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.

−size *n* — True if the file is *n* blocks long (512 bytes per block).

−atime *n* — True if the file has been accessed in *n* days.

−mtime *n* True if the file has been modified in *n* days.

−ctime *n* True if the file has been changed in *n* days.

−exec *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument { } is replaced by the current path name.

−ok *cmd* Like **−exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

−cpio*device* Always true; write the current file on *device* in *cpio*(F) format (5120-byte records).

−print Always true; causes the current path name to be printed.

−newer *file* True if the current file has been modified more recently than the argument *file*.

(*expression*) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

negation The negation of a primary is specified with the exclamation (!) unary *not* operator.

AND The AND operation is implied by the juxtaposition of two primaries.

OR The OR operation is specified with the **−o** operator given between two primaries.

Example

The following removes all files named **a.out** or ***.out** that have not been accessed for a week:

```
find / \( −name a.out −o −name '*.out' \) −atime +7 −exec rm {}
\;
```

Files

/etc/passwd
/etc/group

See Also

cpio(C)(F), sh(C), stat(S), test(C)

Name

finger – Finds information about users.

Syntax

finger [-bfilpqsw] [login1 [login2 ...]]

Description

By default *finger* lists the login name, full name, terminal name and write status (as a "*" before the terminal name if write permission is denied), idle time, login time, office location, and phone number (if they are known) for each current XENIX user. (Idle time is minutes if it is a single integer, hours and minutes if a colon (:) is present, or days and hours if a "d" is present.)

A longer format also exists and is used by *finger* whenever a list of names is given. (Account names as well as first and last names of users are accepted.) This is a multiline format; it includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* which is also in the home directory.

finger options are:

−b Briefer long output format of users.

−f Suppresses the printing of the header line (short format).

−i Quick list of users with idle times.

−l Forces long output format.

−p Suppresses printing of the *.plan* files.

−q Quick list of users.

−s Forces short output format.

−w Forces narrow format list of specified users.

Files

/etc/utmp	Who file
/etc/passwd	User names, offices, phones, login directories, and shells

$HOME/.plan	Plans
$HOME/.project	Projects

See Also

who(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Only the first line of the *.project* file is printed.

Entries in the **/etc/passwd file** have the following format:

login name:user password(coded):user ID:group ID:comments:home directory:login shell

The comment field corresponds to configurable columns in the *finger* output. For example, in the following **/etc/passwd** entry:

blf:Tg6bLFzOwgfbA:47:5:Brian Foster, Mission, x70, 767-1234
:/u/blf:/bin/shV

the comment field, "Brian Foster, Mission, x70, 767-1234" , contains data for the "In Real Life" , "Office" , and "Home Phone" , columns of the *finger* listings.

Idle time is computed as the elapsed time since any activity on the given terminal. This includes previous invocations of *finger* which may have modified the terminal's corresponding device file **/dev/tty**??.

Name

fixhdr – Changes executable binary file headers.

Syntax

fixhdr **option** *files*

Description

fixhdr changes the header of output files created by link editors or assemblers. The kinds of modifications include changing the format of the header, the fixed stack size, the standalone load address, and symbol names.

Using *fixhdr* allows the use of binary executable files, created under other versions or machines, by simply changing the header information so that it is usable by the target cpu.

These are the options to *fixhdr* :

-xa Change the *x.out* format of the header to the *a.out* format.

-xb Change the *x.out* format of the header to the *b.out* format.

-x4 Change the *x.out* format of the header to the 4.2BSD *a.out* format.

-x5 [-n] Change the *x.out* format of the header to 5.2 (UNIX™ System V release 2) *a.out* format. The **-n** flag causes leading underscores on symbol names to be passed with no modifications.

-ax -c [11,86]
Change the *a.out* format of the header to the *x.out* format. The **-c** flag specifies the target cpu. 11 specifies a PDP-11 cpu. 86 specifies one of the 8086 family of cpus (8086, 8088, 80186, or 80286).

-bx Change the *b.out* format of the header to the *x.out* format.

-5x [-n] Change the 5.2 (UNIX System V release 2) *a.out* format of the header to the *x.out* format. The **-n** flag causes leading underscores on symbol names to be passed with no modifications.

-86x Add the *x.out* header format to the *86rel* object module format. See *86rel*(F).

-F *num* Add (or change) the fixed stack size specified in the *x.out* format of the header. *num* must be a hexadecimal number.

-A *num* Add (or change) the standalone load address specified in the *x.out* format of the header. *num* must be a hexadecimal number.

-M[smlh] Change the model of the *x.out* or *86rel* format. Model refers to the compiler model specified when creating the binary. **s** refers to small model, **m** refers to medium model, **l** refers to large model, and **h** refers to huge model.

-v [2,3,5,7]
Change the version of XENIX specified in the header. XENIX version 2 was based on UNIX Version 7.

-s s1=s2 [-s s3=s4]
Change symbol names, where symbol name *s1* is changed to *s2*.

-r Ensure that the resolution table is of non-zero size.

-C *cpu* Set the *cpu* type. *cpu* can be 186, 286, 286, 8086, others.

Files

/usr/bin/fixhdr

See Also

a.out(F), 86rel(F)

Notes

Give *fixhdr* one option at a time. If you need to make more than one kind of modification to a file, use *fixhdr* on the original file. Then use it again on the *fixhdr* output, specifying the next option. Copy the original file if you need an unmodified version as *fixhdr* makes the modifications directly to the file.

Name

format – format floppy disks

Syntax

format [**-f**] [**-e**] [**-q**] [device] [**-i** interleave]

Description

format formats diskettes for use with XENIX. It may be used either interactively or from the command line. The default drive is **/dev/rfd0.**

Options

The following command line options are available:

−f Suppresses the interactive feature. The *format* program does not wait for user-confirmation before starting to format the diskette. Regardless of whether or not you run *format* interactively, track and head information is displayed.

−e
Erases the servo information on a mini-cartridge.

device
This specifies the device to be formatted. The default device is **/dev/rfd0 .**

−i interleave
Specifies the interleave factor.

−q
Quiet option. Suppresses the track and head output information normally displayed. Although this option does not suppress the interactive prompt, it would typically be used with **−f** to produce no output at all.

Usage

To run *format* interactively, enter:

```
format
```

followed by any of the legal options except **-f**, and press RETURN. When you run *format* interactively, you see the prompt:

insert diskette in drive and press return when ready

When you press RETURN at this prompt, *format* begins to format the diskette.

If you specify the **-f** option, you do not see this prompt. Instead, the program begins formatting immediately upon invocation.

Unless you specify the **-q** option, *format* displays which track and head it is currently on:

track # head #

The number signs above are replaced by the actual track and head information.

Files

/dev/rfd[0 - n]

See Also

fd(**M**)

Notes

The *format* utility does not format floppies for use under DOS. Also, XENIX requires error free floppies.

It is not advisable to format a low density (48tpi) diskette on a high density (96tpi) floppy drive. Diskettes written on a high density drive should be read on high density drives. A low density diskette written on a high density drive may not be unreadable on a low density drive.

Name

fsck – Checks and repairs file systems.

Syntax

/bin/fsck [options] [file-system] ...

Description

fsck audits and interactively repairs inconsistent conditions for XENIX System V file systems. If the file system is consistent, the the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent, the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions result in some loss of data. The amount and severity of the loss may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond "yes" or "no". If the operator does not have write permission *fsck* defaults to the action of the **-n** option.

The following flags are interpreted by *fsck*:

-y Assumes a yes response to all questions asked by *fsck*.

-n Assumes a no response to all questions asked by *fsck*; do not open the file system for writing.

-s*b:c* Ignores the actual free list and (unconditionally) reconstructs a new one by rewriting the super-block of the file system. The file system *must* be unmounted while this is done.

The **-s***b:c* option allows for creating an optimal free-list organization. The following forms are supported:

-s
-sBlocks-per-cylinder:Blocks-to-skip (file system interleave)
(for anything else)

If *b:c* is not given, then the values used when the file system was created are used. If these values were not specified, then a reasonable default value is used.

-S Conditionally reconstructs the free list. This option is like -s*b:c* above except that the free list is rebuilt only if there are no discrepancies discovered in the file system. Using **-S** forces a "no" response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

-t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **-t** flag, *fsck* prompts the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes. If the system has a large hard disk there may not be enough space on another filesystem for the scratch file. In such cases, if the system has a floppy drive, use a blank, formatted floppy in the floppy drive with (for example) **/dev/fd0** specified as the scratch file.

-q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced **fifoS** files will selectively be removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list salvaged.

-D Directories are checked for bad blocks. Useful after system crashes.

-f Fast check. Check block and sizes (Phase 1) and check the free list (Phase 5). The free list will be reconstructed (Phase 6) if it is necessary.

-rr Recovers the root file system. The required *filesystem* argument must refer to the root file system, and preferably to the block device (normally **/dev/root**). This switch implies **-y** and overrides **-n** . If any modifications to the file system are required, the system will be automatically shutdown to ensure the integrity of the file system.

-c Causes any supported file system to be converted to the type of the current file system. The user is prompted to verify the request for each file system that requires conversion unless the **-y** option is specified. It is recommended that every file system be checked with this option *while unmounted* if it is to be used with the current version of XENIX. To update the active root file system, it should be checked with:

```
fsck -c -rr /dev/root
```

If no *file-systems* are specified, *fsck* reads a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are as follows:

- Blocks claimed by more than one inode or the free list
- Blocks claimed by an inode or the free list outside the range of the file system
- Incorrect link counts
- Size checks:
  Incorrect number of blocks
  Directory size not 16-byte aligned
- Bad inode format
- Blocks not accounted for anywhere
- Directory checks:
  File pointing to unallocated inode
  Inode number out of range
- Super block checks:
  More than 65536 inodes
  More blocks for inodes than there are in the file system
- Bad free block list format
- Total free block or free inode count incorrect

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

dfsck allows two file system checks on two different drives simultaneously. *Options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A - is the separator between file system groups.

Files

/etc/checklist	Contains default list of file systems to check
/etc/default/boot	Automatic boot control

See Also

checklist(F), filesystem(F), autoboot(M), init(M)

Diagnostics

The diagnostics produced by *fsck* are intended to be self-explanatory.

Notes

fsck will not run on a mounted non-raw file system unless the file system is the root file system or unless the **-n** option is specified and no writing out of the file system will take place. If any such attempt is made, a warning is displayed and no further processing of the file system is done for the specified device.

Although checking a raw device is almost always faster, there is no way to tell if the file system is mounted. And cleaning a mounted file system will almost certainly result in an inconsistent superblock.

Warning

File systems created under XENIX-86 version 3.0 are not supported under XENIX System V because the word ordering in type *long* variables has changed. *fsck* is capable of auditing and repairing XENIX version 3.0 file systems if the word ordering is correct.

For the root file system, "fsck -rr /dev/root" should be run. For all other file systems, "fsck /dev/??" on the *unmounted* block device should be used.

Name

getopt – Parses command options.

Syntax

set -- `getopt optstring $*`

Description

getopt is used to check and break up options in command lines for parsing by shell procedures. *Optstring* is a string of recognized option letters (see *getopt*(S)). If a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by whitespace. The special option **--** is used to delimit the end of the options. *getopt* will place **--** in the arguments at the end of the options, or recognize it if used explicitly. The shell arguments ($1 $2 . . .) are reset so that each option is preceded by a dash (**-**) and in its own shell argument; each option argument is also in its own shell argument.

Example

The following code fragment shows how one can process the arguments for a command that can take the options **a** and **b**, and the option **o**, which requires an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)    FLAG=$i; shift;;
        -o)         OARG=$2;    shift; shift;;
        --)         shift;  break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

See Also

sh(C), getopt(S)

Diagnostics

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

Notes

The "Syntax" given for this utility assumes the user has a *sh*(C) shell.

Name

grep, egrep, fgrep – Searches a file for a pattern.

Syntax

grep [**-bchlnsvy**] [expression] [files]

egrep [**-bchlnv**] [expression] [files]

fgrep [**-bclnvxy**] [strings] [files]

Description

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *grep* patterns are limited regular *expressions* in the style of *ed*(C); it uses a compact nondeterministic algorithm. *egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

–v All lines but those matching are displayed.

–x Displays only exact matches of an entire line. (*fgrep* only.)

–c Only a count of matching lines is displayed.

–l Only the names of files with matching lines are displayed, separated by newlines.

–h Prevents the name of the file containing the matching line from being appended to that line. Used when searching multiple files.

–n Each line is preceded by its relative line number in the file.

–b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

–s Suppresses error messages produced for nonexistent or unreadable files. (*grep* only.)

–y Turns on matching of letters of either case in the input so that case is insignificant. Does not work for *egrep*.

−e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a dash (**−**).

−f *file*
The regular *expression* for *grep* or *egrep*, or *strings* list (for *fgrep*) is taken from the *file*.

In all cases, the filename is output if there is more than one input file. Care should be taken when using the characters $, *, [, ^, |, (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotation marks.

Fgrep searches for lines that contain one of the *strings* separated by newlines.

Egrep accepts regular expressions as in *ed*(C), except for \(and \), with the addition of the following:

- A regular expression followed by a plus sign (+) matches one or more occurrences of the regular expression.
- A regular expression followed by a question mark (?) matches 0 or 1 occurrences of the regular expression.
- Two regular expressions separated by a vertical bar (|) or by a newline match strings that are matched by either regular expression.
- A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is [ ], then * ? +, then concatenation, then the backslash (\) and the newline.

See Also

ed(C), sed(C), sh(C)

Diagnostics

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

Notes

Ideally there should be only one *grep*, but there isn't a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to 256 characters; longer lines are truncated.

Egrep does not recognize ranges, such as **[a−z]**, in character classes.

When using *grep* with the **−y** option, the search is not made totally case insensitive in character ranges specified within brackets.

Multiple strings can be specified in *fgrep* without using a separate strings file by using the quoting conventions of the shell to imbed newlines in the *single* string argument. For example, you might enter the following on the command line:

```
fgrep 'string1
string2
string3' text.file
```

Similarly, multiple strings can be specified in *egrep* by doing:

```
egrep 'string1|string2|string3' text.file
```

Thus *egrep* can do almost anything that *grep* and *frep* can do.

Name

grpcheck – Checks group file.

Syntax

grpcheck [file]

Description

grpcheck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

Files

/etc/group

/etc/passwd

See Also

pwcheck(C), group(M), passwd(M)

Diagnostics

Group entries in **/etc/group** with no login names are flagged.

Name

haltsys, reboot – Closes out the file systems and shuts down the system.

Syntax

/etc/haltsys
/etc/reboot

Description

The *haltsys* utility performs a *uadmin*() system call (see *uadmin*(S)) to flush out pending disk I/O, mark the file systems clean, and halt the processor. *haltsys* takes effect immediately, so user processes should be killed beforehand. *shutdown*(C) is recommended for normal system shutdown, since it warns users, terminates processes, then calls *haltsys*. Use *haltsys* directly only if you cannot run shutdown; for example, because of some system problem.

The *reboot* command performs the same function as *haltsys*, except the system is rebooted automatically afterwards.

Notes

haltsys locks hard disk heads.

See Also

shutdn(S), uadmin(S), shutdown(C)

Name

hd – Displays files in hexadecimal format.

Syntax

hd [**-**format [**-**s offset] [**-n** count] [file] ...

Description

The *hd* command displays the contents of files in hexadecimal, octal, decimal, and character formats. Control over the specification of ranges of characters is also available. The default behavior is with the following flags set: "–abx –A". This says that addresses (file offsets) and bytes are printed in hexadecimal and that characters are also printed. If no *file* argument is given, the standard input is read.

Options include:

-s *offset* Specify the beginning offset in the file where printing is to begin. If no 'file' argument is given, or if a seek fails because the input is a pipe, 'offset' bytes are read from the input and discarded. Otherwise, a seek error will terminate processing of the current file.

The *offset* may be given in decimal, hexadecimal (preceded by '0x'), or octal (preceded by a '0'). It is optionally followed by one of the following multipliers: **w**, **l**, **b**, or **k**; for words (2 bytes), long words (4 bytes), half kilobytes (512 bytes), or kilobytes (1024 bytes). Note that this is the one case where "b" does *not* stand for bytes. Since specifying a hexadecimal offset in blocks would result in an ambiguous trailing 'b', any offset and multiplier may be separated by an asterisk (*).

-n *count* Specify the number of bytes to process. The *count* is in the same format as *offset,* above.

Format Flags

Format flags may specify addresses, characters, bytes, words (2 bytes) or longs (4 bytes) to be printed in hex, decimal, or octal. Two special formats may also be indicated: text or ascii. Format and base specifiers may be freely combined and repeated as desired in order to specify different bases (hexadecimal, decimal or octal) for different output formats (addresses, characters, etc.). All format flags appearing in a single argument are applied as appropriate to all other flags in that argument.

acbwlA
Output format specifiers for addresses, characters, bytes, words, longs and ascii respectively. Only one base specifier will be used for addresses; the address will appear on the first line of output that begins each new offset in the input.

The character format prints printable characters unchanged, special C escapes as defined in the language, and the remaining values in the specified base.

The ascii format prints all printable characters unchanged, and all others as a period (.). This format appears to the right of the first of other specified output formats. A base specifier has no meaning with the ascii format. If no other output format (other than addresses) is given, **bx** is assumed. If no base specifier is given, *all* of **xdo** are used.

hxdo
Output base specifiers for hexadecimal, decimal and octal. If no format specifier is given, *all* of **acbwl** are used.

t Print a text file, each line preceded by the address in the file. Normally, lines should be terminated by a **\n character**, but long lines will be broken up. Control characters in the range 0x00 to 0x1f are printed as '^@' to '^_'. Bytes with the high bit set are preceded by a tilde (~) and printed as if the high bit were not set. The special charcters (^, ~, \) are preceded by a backslash (\) to escape their special meaning. As special cases, two values are represented numerically as '\177' and '\377'. This flag will override all output format specifiers except addresses.

Name

head – Prints the first few lines of a stream.

Syntax

head [–count] [file ...]

Description

This filter prints the first *count* lines of each of the specified files. If no files are specified, *head* reads from the standard input. If no *count* is specified, then 10 lines are printed.

See Also

tail(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Name

id – Prints user and group IDs and names.

Syntax

id

Description

Id writes a message on the standard output, giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

See Also

logname(C), getuid(S)

Name

imacct – Generate an IMAGEN accounting report.

Syntax

imacct acctfile

Description

imacct reads the IMAGEN accounting file *acctfile* and generates a report on the number of pages and files printed. It tallies this information per each user on each host computer, and provides totals and percentages.

The accounting file is generated by the serial "sequence packet protocol" IMAGEN printer handler *ips*(M).

Files

/usr/adm/imagen	Default *acctfile* written by *imagen.spp*.

See Also

imagen(M),
ips(M)

Notes

No sorting option is available.

Author

IMAGEN Corporation.

Name

imprint – Prints text files on an IMAGEN printer.

Syntax

imprint [options] [file...]

Description

imprint queues the specified *files* for printing on an IMAGEN printer using either *pr*(C) or *cat*(C), and passes the correct options to *ipr* (C). If no *files* are given, the standard input is read.

The *options* are:

−I*flag*
Pass *flag* to *ipr*.

−p*flag*
Pass *flag* to either *pr* or *cat*.

−P*printer*
Print the output on *printer*. The default *printer* is specified as **PRINTER** in the file **/etc/default/imagen**, which is read by *ipr*.

−c*n*
Print *n* copies. This turns on pagecollation.

−h*banner*
The string *banner* is passed to both *pr* (**−h**) and *ipr* (**−f**) as the header for this job.

−l*n*
Set the page length to *n* lines. This may also set the printer's interline spacing.

−n
Use *cat* rather than *pr* to print the file.

−w*n*
Set the the line width to *n* characters. A line width of more than 80 characters is printed in landscape (132 column) mode.

−2
Print two logical pages per physical page ("2-up").

−C
Suppress pagecollation (see **−c** above).

−F
Suppress pagereversal (which is on by default).

−J
Suppress generation of the job header page.

−L
Print in landscape mode, 132 columns wide.

−O
Print page borders.

−R
Print page rules (one every two lines).

−d
Prints some information for debugging purposes.

−o*n*
The output is offset *n* character positions from the left margin.

−T*n*
The output begins *n* 1/48's of an inch from the top of the each page.

See Also

cat(C), ipr(C), pr(C)

Notes

Certain parameters can be overridden by document control language in the file itself. Also, a **−c** flag after a **−C** flag turns pagecollation on once more.

If the job contains errors detected by the printer, the job header page is always generated.

The **−T** option is meaningful only if the IMAGEN printer **language** is "daisy." This can be set by **−I−Ldaisy**. If the printer **language** is **daisy**, then the **−o** option uses units of 1/120 of an inch, rather a character width.

Older versions of *imprint* passed any unrecognized *option* on to either *pr* or *cat*. This is no longer supported, and either **-p** or the "end of *options*" delimiter **--** must be used to pass an unmolested *option* to either *pr* or *cat*.

Author

IMAGEN Corporation.

Name

ipcrm – Removes a message queue, semaphore set or shared memory ID.

Syntax

ipcrm [options]

Description

ipcrm removes one or more specified messages, a semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

-q *msqid* removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

-m *shmid* removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-s *semid* removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

-Q *msgkey* removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

-M *shmkey* removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-S *semkey* removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(S), *shmctl*(S), and *semctl*(S). The identifiers and keys may be found by using *ipcs*(C).

See Also

ipcs(C), msgctl(S), msgget(S), msgop(S), semctl(S), semget(S), semop(S), shmctl(S), shmget(S), shmop(S)

Note

ipcrm cannot be used to remove semaphores created using *creatsem*(S) or to remove shared memory created using *sdget*(S).

Name

ipcs – Reports the status of inter-process communication facilities.

Syntax

ipcs [options]

Description

ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

−q Print information about active message queues.
−m Print information about active shared memory segments.
−s Print information about active semaphores.

If any of the options **−q**, **−m**, or **−s** are specified, information about only those indicated are displayed. If none of the three options are specified, information about all three are displayed.

−b
Print biggest allowable size information (maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores). See below, for the meaning of columns in a listing.

−c
Print creator's login name and group name. See below.

−o
Display information on outstanding usage (number of messages on queue, total number of bytes in messages on queue, and the number of processes attached to shared memory segments).

−p
Display process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues. It displays the process ID of the creating process and the process ID of the last process to attach or detach on shared memory segments.) See below.

−t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, and last *semop*(S) on semaphores.) See below.

−a
Use all print *options*. (This is a shorthand notation for **−b**, **−c**, **−o**, **−p**, and **−t**.)

−C *corefile*
Use the file *corefile* in place of **/dev/kmem.**

−N *namelist*
The argument will be taken as the name of an alternate *namelist* (**/xenix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:

- **q** message queue;
- **m** shared memory segment;
- **s** semaphore.

ID (all) The identifier for the facility entry. Note that ID is "X" for facilities created using *creatsem*(S) or *sdget*(S).

KEY (all) The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** from when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:
The first two characters are:

- **R** if a process is waiting on a *msgrcv*;
- **S** if a process is waiting on a *msgsnd*;
- **D** if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
- **C** if the associated shared memory segment is to be cleared when the first attach is executed;
- **−** if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

- **r** if read permission is granted;
- **w** if write permission is granted;
- **a** if alter permission is granted;
- **-** if the indicated permission is *not* granted.

OWNER	(all)	The login name of the owner of the facility entry.
GROUP	(all)	The group name of the group of the owner of the facility entry.
CREATOR	(a,c)	The login name of the creator of the facility entry.
CGROUP	(a,c)	The group name of the group of the creator of the facility entry.
CBYTES	(a,o)	The number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o)	The number of messages currently outstanding on the associated message queue.
QBYTES	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	(a,p)	The process ID of the last process to send a message to the associated queue.
LRPID	(a,p)	The process ID of the last process to receive a message from the associated queue.
STIME	(a,t)	The time the last message was sent to the associated queue.
RTIME	(a,t)	The time the last message was received from the associated queue.
CTIME	(a,t)	The time when the associated entry was created or changed.
NATTCH	(a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

Files

/xenix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

See Also

creatsem(S), msgop(S), sdget(S), semop(S), shmop(S)

Notes

Things can change while *ipcs* is running; the picture it gives is only a close approximation.

Name

ipr, oldipr – Put files onto the IMAGEN printer queue.

Syntax

ipr [options] [file...]
oldipr [options] [file...]

Description

ipr causes the named files to be queued for printing on an IMAGEN printer, using *lp*(C), with the appropriate document control language strings prepended. Some of the information in the document header includes the number of **copies**, the names of the printed **files**, and the IMAGEN printer **language** used. If no files are named, the standard input is read.

oldipr is identical to *ipr*, but implies the **-o** option and is used with old-style imPRESS files.

The *options* are:

-L*language*
Causes the specified *language* declaration to be included in the document control **language** string for the queued files. This should correspond to the language in which the document was prepared.

-D*string*
Causes *string* to be included in the document control language for the queued files.

-P*printer*
This file is to be printed on *printer*. The default *printer* is specified as **PRINTER** in the file **/etc/default/imagen**.

-f*name*
Imbed the identifier *name* as the value of the variable **files** in the document control language. If not specified, the names of the input files are used. This is printed on the banner page.

-m
Causes *mail*(C) to be sent when the job is complete.

-r
The files will be unlinked after being queued for printing.

−c*n*
Prints *n* copies.

−d
Additional information is printed for debugging purposes.

−o
Specifies that the file being queued is an old style (prior to version 1) imPRESS-**language** format file.

ipr reads **/etc/default/imagen** to obtain various default settings. The values obtained and the default values are:

PRINTER=imagen
The name of the IMAGEN printer. This can be overridden with the **−P** option.

JAMPROOF=no
Whether or not paper-jam resistance measures should be used. If such steps are taken, printing is usually slowed down.

The values for the default settings can be changed to reflect the local system configuration. If **/etc/default/imagen** does not exist or cannot be read, the above default values are used.

Files

/usr/bin/lp
The XENIX printer spooling system.

See Also

imagen(M), imprint(C), ips(M), itroff(CT), lp(C)

Notes

The number of copies to be printed and other parameters can be overridden by document-control information contained within the document itself.

Author

IMAGEN Corporation.

Name

iprint – Converts text files to DVI format.

Syntax

iprint [options] [file...]

Description

iprint converts the input text *files* to DVI format. The DVI output must first be converted to imPRESS format before it can be printed on an IMAGEN printer. Unless the **-i** option is given, *dviimp*(CT) is used to automatically preform this conversion and print the results. If no *file* names are given, the standard input is read.

The *options* are:

-i*output*
The imPRESS is saved in file *output* instead of being printed.

-b*banner*
The string *banner* is passed on to *dviimp* as the argument to its **-b** option. The default *banner* is the name of the first input *file*.

-c*n*
Print *n* copies.

-B
Print the first non-blank line on each page in a bold type face, and ignore leading blank lines. This is for use with programs like *pr*(C) which generate page headers.

-f*font*
Use the following argument as the name of a font file for the text. A variable-pitch font will generally produce ugly results.

-F*font*
Use the following argument as the name of a font file for the bold header line. See the **-B** option).

-o*n*
Print with a page offset (left margin) of *n* spaces.

-l*n*
Take the page length to be *n* lines.

−D*raster*
The directory containing the raster images is *raster*. The default raster image directory is specified by **RASTER** in the file **/etc/default/imagen**, and has an assumed resolution of 240 pixels per inch.

−d
Produces extensive output for debugging.

−v
Produces more verbose debugging output.

iprint reads **/etc/default/imagen** to obtain various default settings. The values obtained and the default values are:

RASTER=/usr/lib/imagen/raster
The font rasterization files are to be found in directory **240** within this directory (i.e. *raster***/240**). This can be overridden by the **−D** option.

TMPDIR=/tmp
Directory in which temporary files are kept.

The values of the default settings can be changed to reflect the local system configuration. If **/etc/default/imagen** does not exist or cannot be read, the above default values are used.

Files

tmpdir **/dvi***??????*
Temporary file used to hold the DVI output that *dviimp* processes. The value of *tmpdir* is set by **TMPDIR** in **/etc/default/imagen**.

raster **/240/***
Raster images of host resident fonts. The default values for *raster* is specified by **RASTER** in **/etc/default/imagen**, and can be overridden by the **−D** option.

/usr/bin/dviimp
The DVI to imPRESS format conversion program.

See Also

dviimp(CT), imprint(C), ipr(C)

Notes

The resolution of the IMAGEN printer is assumed to be 240 pixels per inch.

"Font *f* version *n*" means this font file is not a version 0 RAS-format file. Other diagnostics should be self-explanatory.

Author

IMAGEN Corporation.

Name

ips – Imagen serial sequence packet protocol handler
ipbs– Imagen parallel byte stream protocol handler

Syntax

/usr/lib/ips [-D string] [-a file] [-l file] [-i printer] [-oprs] [-u uid] [file]

Description

ips and *ipbs* are the lowest level of Imagen-provided support software. Each handles a different form of supported communications. They present a quite similar view to higher level software, allowing that to be relatively independent of communications method.

ips sends files to Imagen printers using the sequence packet protocol (see the appropriate IMAGEN system manual for further information). This protocol provides for error detection, retransmission, status reporting, and detection of unrecoverable errors.

ipbs supports the parallel byte stream communications method, which provides for flow control but no error detection and correction.

The following information is common to all these programs.

If no file name is given, stdin is read.

The following options are recognized:

-D
The next argument is taken as a string to be prepended to the file being sent. If an unrecoverable failure occurs during transmission and the file can be resent, this string will be resent as well.

-a The next argument is taken as the name of a file in which to store printer status information.

-i The following argument is the device name of the printer.

-l The next argument is taken as the name of a file in which to store logging information.

-o Normally *ips* expects to send to stdout and does not expect to have to set port characteristics. If an explicit device name is given with the **-i** switch, both of these assumptions are reversed. This switch serves to toggle the assumption on the need to set up port characteristics. This switch does not apply to *ipbs* which does not set up its ports.

-r If *ips* is given an explicit file name to send, it will assume the file to be rewindable. Stdin is assumed to be not rewindable. In a manner similar to the **-o** switch above, this switch toggles this assumption.

-s Regardless of what other indications may have been given, use stdout as the printer.

-u*uid*
Uid is a string representing the user identification number of the person to be credited with this printing job.

See Also

imprint(C), lp(C), lpinit(C)

Notes

An interface for Imagen printers is found in the directory **/usr/spool/lp/model**. The file **imagen.s** provides an interface to an Imagen printer in serial mode. The file **imagen.p** provides an interface to an Imagen printer in parallel mode.

The *lpinit* program can be used to initialize an Imagen printer in either serial or parallel mode.

When using an IMAGEN printer in parallel printer mode (using **ipbs** , you must specify the quote character as ASCII 2 and the EOF character as ASCII 4. Control characters must be "taken as is". Refer to the IMAGEN system manuals provided with the printer, for information on specifying these characters in the printer configuration.

Name

join – Joins two relations.

Syntax

join [options] file1 file2

Description

join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is a dash (**–**), the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

–a*n* In addition to the normal output, produces a line for each unpairable line in file *n*, where *n* is 1 or 2.

–e *s* Replaces empty output fields by string *s*.

–j*n m* Joins on the *m*th field of file *n*. If *n* is missing, uses the *m*th field in each file.

–o *list* Each output line comprises the fields specifed in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number.

–t*c* Uses character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

See Also

awk(C), comm(C), sort(C)

Notes

With default field separation, the collating sequence is that of *sort* **-b**. With **−t**, the sequence is that of a plain sort.

Name

kill – Terminates a process.

Syntax

kill [−signo] processid ...

Description

kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(C).

For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by − is given as the first argument, that signal is sent instead of the terminate signal (see *signal*(S)). In particular "kill −9 . . ." is a sure kill.

See Also

ps(C), sh(C), kill(S), signal(S)

Name

l – Lists information about contents of directory.

Syntax

l [**−ACFRabcdfgilnopqrstu**] name ...

Description

For each directory argument, *l* lists the contents of the directory; for each file argument, *l* repeats its name and other requested information. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents. Information is listed in the format of the "ls −l" command, which is identical to the *l* command. This format and all provided switches are described in *ls*(C) and *lc*(C), to which should you should refer for a complete discussion of the capabilities of *l*.

Files

/etc/passwd	Contains user IDs
/etc/group	Contains group IDs

Notes

Newline and tab are considered printing characters in filenames.

The output device is assumed to be 80 columns wide.

Name

lc – Lists directory contents in columns.

Syntax

lc [−1ACFRabcdfgilmnopqrstux] *name* ...

Description

lc lists the contents of files and directories, in columns. If *name* is a directory name, *lc* lists the contents of the directory; if *name* is a filename, *lc* repeats the filename and any other information requested. Output is given in columns and sorted alphabetically. If no argument is given, the current directory is listed. If several arguments are given, they are sorted alphabetically, but file arguments appear before directories.

Files that are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns. A stream output format is available in which files are listed across the page, separated by commas. The **−m** option enables this format.

The options are:

−1
Forces an output format with one entry per line.

−A
If not the root directory, this option displays all files that begin with "." (except "." and ".." themselves). Otherwise, files are displayed normally.

−C
Forces columnar output, even if redirected to a file.

−F
Causes directories to be marked with a trailing "/" and executable files to be marked with a trailing "*"

−R
Recursively lists subdirectories.

−a
Lists all entries; "." and ".." are not suppressed.

−b
Forces printing of nongraphic characters in the *ddd* notation, in octal.

−c
Sorts by time of file creation.

−d
If the argument is a directory, lists only its name, not its contents (mostly used with **−l** to get status on directory).

−f Forces each argument to be interpreted as a directory and lists the name found in each slot. This option turns off **−l**, **−t**, **−s**, and **−r**, and turns on **−a**; the order is the order in which entries appear in the directory.

−g
The same as **−l**, except that the owner is not printed.

−i Prints inode number in first column of the report for each file listed.

−l Lists in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file. If the file is a special file, the size field instead contains the major and minor device numbers.

−m
Forces stream output format.

−n
Same as the **−l** switch, but the owner's user ID appears instead of the owner's name. If used in conjunction with the **−g** switch, the owner's group ID appears instead of the group name.

−o
The same as **−l**, except that the group is not printed.

−p
Pad output with spaces.

−q
Forces printing of nongraphic characters in filenames as the character "?".

−r
Reverses the order of sort to get reverse alphabetic or oldest first as appropriate.

−s
Gives size in 512-byte blocks, including indirect blocks for each entry.

−t Sorts by time modified (latest first) instead of by name, as is normal.

−u
Uses time of last access instead of last modification for sorting (**−t**) or printing (**−l**).

−x
Forces columnar printing to be sorted across rather than down the page.

The following are alternate invocations of the **lc** command:

lf Produces the same output as **lc -F**.

lr Produces the same output as **lc -R**.

lx Produces the same output as **lc -x**.

The mode printed under the **−l** option contains 11 characters. The first character is:

− If the entry is a plain file

d If the entry is a directory

b If the entry is a block-type special file

c If the entry is a character-type special file

p If the entry is a named pipe

s If the entry is a semaphore

m If the entry is shared data (memory)

The next 9 characters are interpreted as 3 sets of 3 bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set, the 3 characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r If the file is readable

w If the file is writable

x If the file is executable

− If the indicated permission is not granted

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise the user-execute permission character is given as s if the file has set-user-ID mode.

The last character of the mode (normally "x" or "–") is **t** if the 1000 bit of the mode is on. See *chmod*(C) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is displayed.

Files

/etc/passwd	To get user IDs for "lc –o"
/etc/group	To get group IDs for "lc –g"

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Newline and tab are considered printing characters in filenames. The output device is assumed to be 80 columns wide. Column width choices are poor for terminals that can tab.

This utility reports sizes in 512 byte blocks. On systems which use 1024 byte blocks, this means a file of 500 bytes uses 2 blocks. *lc -s* will report 2 blocks used, rather than 1 block, since the file uses one system block of 1024 bytes. Refer to the **machine**(M) manual page for the block size used by your system.

Name

line – Reads one line.

Syntax

line

Description

line copies one line (up to a newline) from the standard input and writes it on the standard output. It returns an exit code of 1 on end-of-file and always prints at least a newline. It is often used within shell files to read from the user's terminal.

See Also

gets(CP), sh(C)

Name

ln – Makes a link to a file.

Syntax

ln file1 file2
ln file1 ... directory

Description

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc). may have several links to it. There is no way to distinguish a link to a file from its original directory entry. Any changes to the file are effective independent of the name by which the file is known.

In the first case, *ln* creates a link to the existing file, *file1*. The *file2* argument is a new name referring to the same file contents as *file1*.

In the second case, *directory* is the location of a directory into which one or more links are created with corresponding file names.

You cannot link to a directory or link across file systems.

See Also

cp(C), mv(C), rm(C)

Name

logname – Gets login name.

Syntax

logname

Description

logname returns the value of *getlogin*(S) or *getuid*(S) which is set when a user logs into the system.

See Also

env(C), getlogin(S), getuid(S), login(M), logname(S)

Name

lp, lpr, cancel – Send/cancel requests to lineprinter.

Syntax

lp [options...][name...]
or
lpr [options...][name...]
cancel [request ID s] [printers]

Description

lp causes the named files and associated information (collectively called a "request") to be printed by a lineprinter. *lp* and *lpr* are equivalent commands and may be used interchangeably. If no file names are mentioned, the standard input is assumed. The file name - stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

lp associates a unique request ID with each request and prints it on the standard output. This request ID can be used later to cancel (see *cancel*) or find the status of the request (see *lpstat*(C)).

The following options to *lp* may appear in any order and may be intermixed with file names:

-c Makes copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the **-c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety; any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

-d*dest* Chooses *dest* as the printer or class of printers to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (for example, printer unavailability or file space limitation), requests for specific destinations may not be accepted (see *accept*(C) and *lpstat*(C)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(C)).

-m Sends mail (see *mail*(C)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

-n*number* Prints *number* of copies of the output. The default is one.

-o*option* Specifies printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **-o** keyletter more than once. For more information about what is valid for *options*, see *lpadmin*(C).

-r Removes file after sending it.

-s Suppresses messages from *lp*(C) such as "request id is ...".

-t*title* Prints *title* on the banner page of the output.

-w Writes a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail is sent instead.

The file **/etc/default/lpd** contains the setting of the variable BANNERS, whose value is the number of pages printed as a banner identifying each printout. This is normally set to either 1 or 2.

Cancel cancels line printer requests that were made by the *lp*(C) command. The command line arguments may be either request IDs (as returned by *lp*(C)) or *printer* names (for a complete list, use *lpstat*(C)). Specifying a request ID cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request. User's identification and accounting data spool area contains BANNERS setting.

Files

/etc/passwd
/usr/spool/lp/*
/etc/default/lpd

See Also

enable(C), lpstat(C), mail(C), accept(C), lpadmin(C), lpsched(C)

Name

lpadmin – Configures the lineprinter spooling system.

Syntax

/usr/lib/lpadmin -p printer [options...]
/usr/lib/lpadmin -x dest
/usr/lib/lpadmin - d[dest]

Description

lpadmin configures the lineprinter spooling system to describe printers, classes, and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs, and to change the system default destination. System managers may also use *lpinit*(C) to add new printing destinations to the system. *lpadmin* may not be used when the lineprinter scheduler, *lpsched*(C), is running, except where noted below.

Exactly one of the **-p**, **-d**, or **-x** options must be present for every legal invocation of *lpadmin*.

-d[*dest*] Makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(C) is running. No other *options* are allowed with **-d**.

-x*dest* Removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **-x**.

-p*printer* Names a *printer* to which all of the *options* below refer. If *printer* does not exist then it will be created.

The following *options* are only useful with **-p** and may appear in any order. For ease of discussion, the printer will be referred to as *p* below.

-c*class* Inserts printer *p* into the specified *class*. *Class* will be created if it does not already exist.

-e*printer* Copies an existing *printer*'s interface program to be the new interface program for *p*.

-h Indicates that the device associated with *p* is hardwired. This *option* is assumed when creating a new printer unless the **-l** *option* is supplied.

-i*interface* Establishes a new interface program for *p*. *Interface* is the pathname of the new program.

-l Indicates that the device associated with *p* is a login terminal. The lineprinter scheduler, *lpsched*(C), disables all login terminals automatically each time it is started. Before re-enabling *p*, its current *device* should be established using *lpadmin*.

-m*model* The model printer interface program, *dumb*, is supplied with XENIX lineprinter software. It is a shell procedure which interfaces *lpsched*(C) and print devices. It can be found in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **-m** or *lpinit*(C). This program is an interface for a line printer without special functions and protocol. Form feeds are assumed. System managers may modify copies of *dumb* and then use *lpadmin* **-i** to associate the copies with printers.

-r*class* Removes printer *p* from the specified *class*. If *p* is the last member of the *class*, then the *class* will be removed.

-v*device* Associates a new *device* with printer *p*. *Device* is the pathname of a file that is writable by the system manager, *lp*. Note that there is nothing to stop a system manager from associating the same *device* with more than one *printer*. If only the **-p** and **-v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions

When creating a new printer, the **-v** option and one of the **-e**, **-i**, or **-m** options must be supplied. Only one of the **-e**, **-i**, or **-m** options may be supplied. The **-h** and **-l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters A - Z , a - z , 0 - 9 and _ (underscore).

Models

Model printer interface programs are shell procedures which interface between *lpsched*(C) and devices. Models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **-m**.

Models should have 644 permission if owned by *lp* & bin, or 664 permission if owned by bin & bin. System managers may modify copies of models and then use *lpadmin* **-i** to associate them with printers. If printers have special options, these can be included in the interface program. Users can then choose an option with the *lp* **-o** command.

One model interface program is supplied with XENIX lineprinter software: *dumb*. This is an interface program for a lineprinter without special functions and protocol. Form feeds are assumed. This is a good model for system managers to copy and modify.

Serial printers that need delays or other special *stty*(C) options (such as maping CR to newline) should have this string included in the model interface program:

```
stty [ options ... ] 0<&1
```

Files

/usr/spool/lp/*

See Also

accept(C), enable(C), lp(C), lpinit(C), lpsched(C), lpstat(C)

Name

lpinit – Adds, reconfigures and maintains lineprinters.

Syntax

/etc/lpinit

Description

lpinit is a shell script for configuring and adding new lineprinters to a system, and for maintaining and reconfiguring existing printers. It should only be executed by the system manager.

lpinit displays a menu with the following options:
1) Add a new printer
2) Remove a printer
3) Reconfigure an existing printer
4) Assign a system default printer
5) Print lp status information

When reconfiguring an existing printer the following options are given:
1) Insert a printer into a class
2) Remove a printer from a class
3) Install a new interface program for a printer
4) Associate a new device with a printer

Information which the system manager may be asked to supply includes:
- The printer device (e.g. **/dev/lp0**).
- The printer name (default is *printer*).
- The pathname of the interface program (several example programs are supported).
- The name of a class into which to insert or remove a printer.
- Whether the printer being added or reconfigured is a parallel or serial printer.

The printer name can be any combination of up to 14 alphanumeric characters or underscores. A printer interface program can be a shell script, C program, or any executable program; or the model interface program, **/usr/spool/lp/model/dumb**, can be copied and modified. (See the "Models" section of the *lpadmin*(C) manual page.)

When adding a new printer, *lpinit* changes the acceptance status of the new lineprinter to "accept," and enables it to print files. **/etc/lpinit** then asks if the new printer will be the default printing destination. All nonspecific print requests are routed to the default destination (see *lp*(C)).

If the line printer scheduler is running when *lpinit* is invoked, the user is reminded that any jobs which are printing may be interrupted and the user is asked if he wants to continue. The scheduler is restarted when *lpinit* exits only if it was running when *lpinit* was invoked or if a new printer was added.

The steps to configure a new printer can be taken separately, (see *lpadmin*(C), *accept*(C), *enable*(C), and *lpsched*(C) for more information).

Files

/etc/lpinit

See Also

accept(C), enable(C), lp(C), lpadmin(C), lpsched(C)

Name

lpr – Sends files to the lineprinter queue for printing.

Syntax

lpr [option ...] [name ...]

Description

lpr causes the named files to be queued for printing on a lineprinter. If no names appear, the standard input is assumed; thus *lpr* may be used as a filter.

The following options may be given (each as a separate argument and in any order) before any filename arguments:

-c Makes a copy of the file and prints the copy and not the original. Normally files are linked whenever possible.

-r Removes the file after sending it.

-m When printing is complete, reports that fact by *mail*(C).

-n Prints number of copies of output.

The file **/etc/default/lpd** contains the setting of the variable BANNERS, whose value is the number of pages printed as a banner identifying each printout. This is normally set to either 1 or 2.

Files

/etc/passwd	User's identification and accounting data
/usr/lib/lpd	Lineprinter daemon
/usr/spool/lpd/*	Spool area
/etc/default/lpd	Contains BANNERS default setting
/etc/lpopen	On some systems - sets modes on a serial line

See Also

banner(C)

Notes

Once a file has been queued for printing, it should not be changed or deleted until printing is complete. If you want to alter the contents of the file or to remove the file immediately, use the **−c** option to force *lpr* to make its own copy of the file.

Name

lpsched, lpshut, lpmove – Starts/stops the lineprinter request scheduler and moves requests.

Syntax

/usr/lib/lpsched
/usr/lib/lpslmt
/usr/lib/lpmove requests destinations
/usr/lib/lpmove dest1 dest2

Description

lpsched schedules requests taken by *lp*(C) for printing on lineprinters.

lpshut shuts down the lineprinter scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again. All lineprinter commands perform their functions even when *lpsched* is not running.

lpmove moves requests that were queued by *lp*(C) between lineprinter destinations. This command may be used only when *lpsched* is not running. The first form of the command moves the named *requests* to the lineprinter *destinations*, *dest*. *Requests* are request IDs as returned by *lp*(C). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp*(C) will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status for the new destination when moving requests (see *accept*(C)).

Files

/usr/spool/lp/*

See Also

accept(C), enable(C), lp(C), lpadmin(C), lpinit(C), lpstat(C)

Name

lpstat – prints lineprinter status information

Syntax

lpstat [options ...]

Description

lpstat prints information about the current status of the lineprinter system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(C) by the user. Any arguments that are not *options* are assumed to be request IDs (as returned by *lp*). *lpstat* prints the status of these requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the following options may be followed by *list* which can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

-u"user1, user2, user3"

The omission of a *list* following such options causes all information relevant to the option to be printed, for example:

lpstat -o

prints the status of all output requests.

-a[*list*] Prints acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

-c[*list*] Prints class names and their members. *List* is a list of class names.

-d Prints the system default destination for *lp*.

-o[*list*] Prints the status of output requests. *List* is a list of intermixed printer names, class names, and request IDs.

-p[*list*] Prints the status of printers. *List* is a list of printer names.

-r Prints the status of the lineprinter scheduler, *lpsched*.

-s Prints a status summary, including the status of the line-printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

-t Prints all status information.

-u[*list*] Prints status of output requests for users. *List* is a list of login names.

-v[*list*] Prints the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

Files

/usr/spool/lp/*

See Also

enable(C), lp(C)

Name

ls – Gives information about contents of directories.

Syntax

ls [**-ACFRabcdfgilmnopqrstux**] [names]

Description

For each directory named, *ls* lists the contents of that directory; for each file named, *ls* repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **-C** and **-x** options enable multi-column formats, and the **-m** option enables stream output format in which files are listed across the page, separated by commas. In order to determine output format for the **-C**, **-x**, and **-m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *termcap* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

There are many options:

-A List all entries; entries whose name begin with a period (.) are listed. Does not list current directory (.) and directory above (..).

-a Lists all entries; entries whose name begin with a period (.) are listed.

-R Recursively lists subdirectories encountered.

-d If an argument is a directory, lists only its name (not its contents); often used with **-l** to get the status of a directory.

-C Multi-column output with entries sorted down the columns.

-x Multi-column output with entries sorted across rather than down the page.

-m Stream output format.

-l Lists in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will contain the major and minor device numbers, rather than a size.

-n The same as **-l**, except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.

-o The same as **-l**, except that the group is not printed.

-g The same as **-l**, except that the owner is not printed.

-r Reverses the order of sort to get reverse alphabetic or oldest first, as appropriate.

-t Sorts by time modified (latest first) instead of by name.

-u Uses time of last access instead of last modification for sorting use with the **-t** option.

-c Uses time of last modification of the inode (file created, mode changed, etc.) for sorting use with **-t** option.

-p Puts a slash (/) after each filename if that file is a directory.

-F Puts a slash (/) after each filename if that file is a directory and puts an asterisk (*) after each filename if that file is executable.

-b Forces printing of non-graphic characters to be in the octal \ddd notation.

-q Forces printing of non-graphic characters in file names as the character (?).

-i For each file, prints the inode number in the first column of the report.

-s Gives size in blocks, including indirect blocks, for each entry.

-f Forces each argument to be interpreted as a directory and lists the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.

The mode printed under the **-l** option consists of 11 characters. The first character is:

- \- If the entry is an ordinary file.
- **d** If the entry is a directory.
- **b** If the entry is a block special file.
- **c** If the entry is a character special file.
- **p** If the entry is a named pipe.
- **s** If the entry is a semaphore.
- **m** If the entry is a shared data (memory) file.

The next 9 characters are interpreted as 3 sets of 3 bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the 3 characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

- **r** If the file is readable.
- **w** If the file is writable.
- **x** If the file is executable.
- \- If the indicated permission is *not* granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **-**) is **t** if the 1000 (octal) bit of the mode is on; see *chmod*(C) for the meaning of this mode. The indications of set-ID and 1000 bit of the mode are capitalized if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks including indirect blocks is printed.

Files

/etc/passwd	Gets user IDs for ls -l and ls -o
/etc/group	Gets group IDs for ls -l and ls -g

/etc/termcap/* Gets terminal information

See Also

chmod(C), find(C), l(C), lc(C), termcap(C)

Notes

Newline and tab are considered printing characters in filenames.

Unprintable characters in filenames may confuse the columnar output options.

This utility reports sizes in 512 byte blocks. Systems which define a block as 1024 characters, "round-off" the size of files containing 511 or fewer bytes to 1 block. *ls -s* interprets 1 block from a 1024 byte block system as 2 of its own 512 byte blocks. Thus a 500 byte file is interpreted as 2 blocks rather than 1. Refer to the **machine**(M) manual page for the block size used by your system.

Name

mail – Sends, reads or disposes of mail.

Syntax

mail [[-u user] [-f mailbox]] [-e] [-R] [-i] [users ...]

mail [-s subject] [-i] [user ...]

Description

mail is a mail processing system that supports composing of messages, and sending and receiving of mail between multiple users. When sending mail, a *user* is the name of a user or of an alias assigned to a machine or to a group of users.

Options include:

-u *user*
Tells *mail* to read the system mailbox belonging to the specified *user*.

-f *mailbox*
Tells *mail* to read the specified *mailbox* instead of the default user's system mailbox.

-e Allows escapes from compose mode when input comes from a file.

-R
Makes the mail session "read-only" by preventing alteration of the mailbox being read. Useful when accessing system-wide mailboxes.

-i Tells *mail* to ignore interrupts sent from the terminal. This is useful when reading or sending mail over telephone lines where "noise" may produce unwanted interrupts.

-s *subject*
Specifies *subject* as the text of the *Subject:* field for the message being sent.

Sending mail

To send a message to one or more other people, invoke *mail* with arguments which are the names of people to send to. You are then expected to type in your message, followed by a Ctrl-D at the beginning of a line.

Reading Mail

To read mail, invoke *mail* with no arguments. This will check your mail out of the system-wide directory so that you can read and dispose of the messages sent to you. A message header is printed out for each message in your mailbox The current message is initially the last numbered message and can be printed using the **print** command (which can be abbreviated **p**). You can move among the messages much as you move between lines in *ed,* with the commands **+** and **-** moving backwards and forwards, and simple numbers typing the addressed message.

If new mail arrives during the mail session, you can read in the new messages with the **restart** command.

Disposing of Mail

After examining a message, you can **delete** (**d**) the message or **reply** (**r**) to it. Deletion causes the *mail* program to forget about the message. This is not irreversible, the message can be **undeleted** (**u**) by giving its number, or the *mail* session can be aborted by giving the **exit** (**x**) command. Deleted messages will, however, disappear.

Specifying Messages

Commands such as **print** and **delete** often can be given a list of message numbers as arguments to apply to a number of messages at once. Thus "delete 1 2" deletes messages 1 and 2, while "delete 1-5" deletes messages 1 through 5. The special name "*" addresses all messages, and "$" addresses the last message; thus the command **top** which prints the first few lines of a message could be used in "top *" to print the first few lines of all messages.

Replying to or Originating Mail

You can use the **reply** command to set up a response to a message, sending it back to the person who sent it. Then you can enter in the text of the reply, and press Ctrl-D to send it. While you are composing a message, *mail* treats lines beginning with a tilde (~) as special. For instance, typing "~m" alone on a line, places a copy of the current message into the response, right shifting it by one tabstop. Other escapes set up subject fields, add and delete recipients to the message, and allow you to escape to an editor to revise the message or to a shell to run some commands. (These options are given in the summary below.)

Ending a Mail Session

You can end a *mail* session with the **quit (q)** command. Messages that have been examined go to your *mbox* file unless they have been deleted, in which case they are discarded. Unexamined messages go back to the post office. The **-f** option causes *mail* to read in the contents of your *mbox* (or the specified file) for processing; when you **quit**, *mail* writes undeleted messages back to this file. The **-i** option causes *mail* to ignore interrupts.

Using Aliases and Distribution Lists

It is also possible to create a personal distribution list. For instance, you can send mail to "cohorts" and have it go to a group of people. Such lists can be defined by placing a line like

alias cohorts bill bob barry bobo betty beth bobbi

in the file .mailrc in your home directory. The current list of such aliases can be displayed by the **alias (a)** command in *mail*. System-wide distribution lists can be created by editing /usr/lib/mail/aliases, see *aliases*(M); these are kept in a slightly different syntax. In mail you send, personal aliases will be expanded in mail sent to others so that they will be able to **reply** to the recipients. System-wide *aliases* are not expanded when the mail is sent, but any reply returned to the machine will have the system-wide alias expanded.

mail has a number of options which can be **set** in the *.mailrc* file to alter its behavior; thus "set askcc" enables the "askcc" feature. (These options are summarized below.)

Summary

Each mail command is entered on a line by itself, and may take arguments following the command word. The command need not be entered in its entirety; the first command which matches the typed prefix is used. For the commands that take message lists as arguments; if no message list is given, then the next message forward that satisfies the command's requirements is used. If there are no messages forward of the current message, the search proceeds backwards, and if there are no messages at all, *mail* types "No applicable messages" and aborts the command.

- Goes to the previous message and prints it out. If given a numeric argument *n*, goes to the *n*th previous message and prints it.

+	Goes to the next message and prints it out. If given a numeric argument *n*, goes to the *n*th next message and prints it.
RETURN	Goes to the next message and prints it out.
?	Prints a brief summary of commands.
!	Executes the shell command which follows.
=	Prints out the current message number.
^	Prints out the first message.
$	Prints out the last message.
alias	(a) With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, adds the users named in the second and later arguments to the alias named in the first argument.
Alias *users*	Prints system-wide list of aliases for users. At least one user must be specified.
cd	(c) Changes the user's working directory to that specified, if given. If no directory is given, then changes to the user's login directory.
delete	(d) Takes a list of messages as an argument and marks them all as deleted. Deleted messages are not retained in the system mailbox after a quit, nor are they available to any command other than the *undelete* command.
dp	Deletes the current message and prints the next message. If there is no next message, *mail* says "no more messages."
echo *path*	Expands shell metacharacters.
edit	(e) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.
exit	(x) Effects an immediate return to the shell without modifying the user's system mailbox, his *mbox* file, or his edit file in **-f**.
file	(fi) Prints the name of the file mail is reading. If it is a mailbox, the name of the owner is returned.

forward (f) Forwards the current message to the named users. Current message is indented within forwarded message.

Forward (F) Forwards the current message to the named users. Current message is *not* indented within forwarded message.

headers (h) Lists the current range of headers, which is an 18 message group. If a "+" argument is given, then the next 18 message group is printed, and if a "-" argument is given, the previous 18 message group is printed. Both "+" and "-" may take a number to view a particular window. If a message-list is given, it prints the specified headers.

hold (ho) Takes a message list and marks each message therein to be saved in the user's system mailbox instead of in *mbox*. Use only when the switch *autombox* is set. Does not override the **delete** command.

list Prints list of **mail** commands.

lpr (l) Prints out each message in a message-list on the lineprinter.

mail (m) Takes as arguments login names and distribution group names and sends mail to those people.

mbox (mb) Marks messages in a message list so that they are saved in the user mailbox after leaving mail.

move *mesg-list mesg-num*
Places the messages specified in *mesg-list* after the message specified in *mesg-num*. If *mesg-num* is 0, *mesg-list* moves to the top of the mailbox.

next (n like + or RETURN) Goes to the next message in sequence and prints it. With an argument list, types the next matching message.

print (p) Prints out each message in a message-list on the terminal display.

quit (q) Terminates the session, retaining all undeleted, unsaved messages in the system mailbox and removing all other messages. Files marked with a star (*) are saved; files marked with an "M" are saved in the user mailbox. If new mail has arrived during the session, the message "You have new mail" is given. If given while editing a mailbox file with the **-f** flag, then the mailbox file is rewritten. The user returns to the shell, unless the rewrite of the mailbox file fails, in which case the user can escape with the **exit** command.

reply (r) Takes a message list and sends mail to each message author. The default message must not be deleted.

Reply (R) Takes a message list and sends mail to each message author *and each member of the message* just like the **mail** command. The default message must not be deleted.

restart Reads in messages that arrived during the current mail session.

save (s) Takes a message list and a filename and appends each message in turn to the end of the file. The filename, in quotation marks, followed by the line count and character count is echoed on the user's terminal.

set (se) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form "option=value" or "option".

shell (sh) Invokes an interactive version of the shell.

size (si) Takes a message list and prints out the size in characters of each message.

source (so) Reads mail commands from the file given as its only argument.

string *string mesg-list*
Searches for *string* in *mesg-list*. If no *mesg-list* is specified, all undeleted messages are searched. Case is ignored in search.

top (t) Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable **toplines** and defaults to six.

undelete (u) Takes a message list and marks each one as *not* being deleted.

unset (uns) Takes a list of option names and discards their remembered values; the inverse of **set**.

visual (v) Takes a message list and invokes vi on each message.

whois Looks up a list of target mail recipients and prints the real names or descriptions of each recipient. If the first character of the first argument is alphabetic, the arguments are looked up without change. Otherwise, the arguments are assumed to be a message list, in the format specified in the *Mail User's Guide*. For each message in the list, the "From" person is extracted from the header and added to the list of users to be searched.

If a target mail recipient contains a machine and user name, nothing is printed. If it is a private alias, "private alias" is printed. If it is a global alias, the name or description of the recipient is printed (contents of the $n field in the alias file). If all of the above fail, the user is looked up in /etc/passwd; if the user is a local user, "local user" is printed. Finally, if none of the above tests and searches succeed, "unknown" is printed.

write *filename*
(**w**) Saves the body of the message in the named file.

Here is a summary of the compose escapes, which are used when composing messages to perform special functions. Compose escapes are only recognized at the beginning of lines.

~~*string* Inserts the string of text in the message prefaced by a single tilde (~). If you have changed the escape character, then you should double that character instead.

~? Prints out help for compose escapes.

~. Same as Ctrl-D on a new line.

~!*cmd* Executes the indicated shell command, then returns to the message.

~ |*cmd* Pipes the message through the command as a filter. If the command gives no output or terminates abnormally, retains the original text of the message.

~_ *mail-command*
Executes a mail command, then returns to compose mode.

~: *mail-command*
Executes a mail command, then returns to compose mode.

~alias Prints list of private aliases

~alias *aliasname*
Prints names included in private *aliasname*.

~Alias Performs aliasing by first examining private aliases and then system-wide aliases using all three global alias files (aliases.hash, faliases, and maliases). Only the final result is printed (non-local mail recipients will have the complete delivery path printed). The user list is taken from header fields.

~Alias *users* Performs aliasing by first examining private aliases and then system-wide aliases using all three global alias files (aliases.hash, faliases, and maliases). Only the final result is printed (non-local mail recipients will have the complete delivery path printed). At least one user must be specified.

~b *name* ... Adds the given names to the list of blind carbon copy recipients.

~c *name* ... Adds the given names to the list of carbon copy recipients.

~cc *name* ... Same as ~c above.

~d Reads the file *dead.letter* from your home directory into the message.

~e Invokes the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.

~h Edits the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field with the current terminal erase and kill characters.

~m *mesg-list* Reads the named messages into the message buffer, shifted right one tab. If no messages are specified, reads the current message.

~M *mesg-list* Reads the named messages into the message buffer, with no indentation. If no messages are specified, reads the current message.

~p Prints out the messages collected so far, prefaced by the message header fields.

~**Print**	Prints the real names or descriptions (in parentheses) after each recipient in a header field.
~**q**	Aborts the message being sent, copying the message to *dead.letter* in your home directory if **save** is set.
~**r** *filename*	Reads the named file into the message buffer.
~**Return** *name*	Adds the given names to the Return-receipt-to field.
~**s** *string*	Causes the named string to become the current subject field.
~**t** *name* ...	Adds the given names to the direct recipient list.
~**v**	Invokes a visual editor (defined by the VISUAL option) on the message buffer. After you quit the editor, you may resume appending text to the end of your message.
~**w** *filename*	Writes the body of the message to the named file.

Options are controlled with the **set** and **unset** commands. An option may be either a switch, in which case it is either on or off, or a string, in which case the actual value is of interest. The switch options include the following:

askcc	Causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.
asksubject	Causes *mail* to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field is sent.
autombox	Causes all examined messages to be saved in the user mailbox unless deleted or saved.
autoprint	Causes the **delete** command to behave like **dp** - thus, after deleting a message, the next one will be entered automatically.
chron	Causes messages to be displayed in chronological order.
dot	Permits use of dot (.) as the end of file character when composing messages.

execmail Causes the underbar prompt to return before *mail* is finished being sent. This frees the user to continue while *mail* performs mailing functions in background.

ignore Causes interrupt signals from your terminal to be ignored and echoed as at-signs (@).

mchron Causes messages to be listed in numerical order (most recently received first), but displayed in chronological order.

metoo Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.

nosave Prevents aborted messages from being appended to the file *dead.letter* in your home directory on receipt of two interrupts (or a ~**q**).

quiet Suppresses the printing of the version header when first invoked.

verify Causes each target mail recipient to be verified in the manner decribed in the **whois** command. This option permits errors made while composing messages to be corrected or ignored.

The following options have string values:

EDITOR Pathname of the text editor to use in the **edit** command and ~e escape. If not defined, then a default editor (*/bin/ed*) is used.

SHELL Pathname of the shell to use in the **!** command and the ~! escape. A default shell (*/bin/sh*) is used if this option is not defined.

VISUAL Pathname of the text editor (*/bin/vi*) to use in the **visual** command and ~v escape.

escape If defined, the first character of this option gives the character to use in the place of the tilde (~) to denote escapes.

page=*n* Specifies the number of lines (*n*) to be printed in a "page" of text when displaying messages.

record If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not saved.

toplines	If defined, gives the number of lines of a message to be printed out with the **top** command; normally, the first six lines are printed.

Files

/usr/spool/mail/*	System mailboxes
/usr/*name*/dead.letter	File where undeliverable mail is deposited
/usr/*name*/mbox	Your old mail
/usr/*name*/.mailrc	File giving initial mail commands
/usr/lib/mail/aliases	System-wide aliases
/usr/lib/mail/aliases.hash	System-wide alias database
/usr/lib/mail/faliases	Forwarding aliases for the local machine
/usr/lib/mail/maliases	Machine aliases
/usr/lib/mail/mailhelp.cmd	Help file
/usr/lib/mail/mailhelp.esc	Help file
/usr/lib/mail/mailhelp.set	Help file
/usr/lib/mail/mailrc	System initialization file (defaults)
/usr/bin/mail	The mail command

See Also

aliases(M), aliashash(M), netutil(C) Chapter 3, "Mail", in the *XENIX User's Guide.*

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Name

mesg – Permits or denies messages sent to a terminal.

Syntax

mesg [n] [y]

Description

mesg with argument **n** forbids messages via *write*(C) by revoking nonuser write permission on the user's terminal. *mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

Files

/dev/tty*

See Also

write(C)

Diagnostics

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

Name

mkdev - Calls scripts to add peripheral devices.

Syntax

/etc/mkdev lp
/etc/mkdev hd
/etc/mkdev serial
/etc/mkdev fs [*device file*]
/etc/mkdev fd
/etc/mkdev tape
/etc/mkdev om
/etc/mkdev shl

Description

mkdev calls the scripts to create the requested type of device file(s). *mkdev* may call *lpinit*(C), *hdinit*, *serinit*, *fdinit*, *fsinit*, *tapeinit*, *ominit*, or *shlinit*. If no arguments are listed, *mkdev* prints a usage message.

/etc/mkdev lp creates device files for use with line printers. (See *lpinit*(C).)

/etc/mkdev hd creates device files for use with a peripheral hard disk. The device files for an internal hard disk already exist. *hdinit* invokes the following utilities: *dparam*(C), *badtrk*(M), *fdisk*(C), and *divvy*(C).

/etc/mkdev serial creates device files for use with serial cards. The device files for the first and second ports already exist. Additional device files must be created for the ports added when expansion cards are added to the system. The **/etc/ttys** and **/etc/ttytype** files are updated.

/etc/mkdev fs performs the system maintenance tasks required to add a new filesystem to the system once the device is created (*mknod*(C)) and the filesystem is made (*mkfs*(C)). It creates the */file* and */file/***lost+found** directories, reserves slots in the **lost+found** directory, and modifies **/etc/checklist, /etc/default/filesys** and **/etc/default** to check (*fsck*(C)) and mount (*mount*(C),*mnt*(C),*rc*(C)) the filesystem as appropriate. It is usually used in conjunction with *mkdev hd* when adding a second hard disk to the system or with *mkdev fd* when creating a mountable filesystem on a floppy, but can be used on any additional filesystem (for example, on a large internal hard disk).

/etc/mkdev fd creates bootable and root file system floppy disks. The three basic options are: boot and root on a single disk (96 tpi only), boot and root pair (48 tpi) or filesystem only. Use with *mkdev fs* when creating a filesystem-only floppy.

/etc/mkdev tape configures the tape driver in preparation for linking a new kernel that includes tape support. It adds a standard quarter-inch cartridge tape driver and/or a mini-cartridge tape driver.

The current driver configurations can be displayed, and changed if necessary. A zero in any of the fields means the driver automatically detects the type of tape device installed and uses the built-in values for that device. If the autoconfiguration values are not correct for your drive, refer to your hardware manual for the correct values, configure the driver and relink the new kernel. *mkdev tape* can also be used to remove a tape driver from the existing kernel.

/etc/mkdev shl initializes necessary devices and configures kernel parameters associated with the number of shell layers sessions available on the system.

Once the driver is configured, you are prompted for re-linking the kernel. The appropriate devices in **/dev** are created.

The various *init* scripts prompt for the information necessary to create the devices.

Files

/usr/lib/mkdev/fsinit
/usr/lib/mkdev/hdinit
/usr/lib/mkdev/lpinit
/usr/lib/mkdev/serinit
/usr/lib/mkdev/tapeinit
/usr/lib/mkdev/fdinit
/usr/lib/mkdev/shlinit

See Also

badtrk(M), divvy(C), dparam(C), fd(HW), fdisk(C), filesys(F), format(C), hd(HW), lp(HW), lpinit(C), mkfs(C), mknod(C), mount(C), serial(HW), tape(HW), and the "Using Peripheral Devices" chapter in the XENIX *Operations Guide.*

Notes

ominit is not supplied with the standard XENIX distribution.

Name

mkdir – Makes a directory.

Syntax

mkdir dirname ...

Description

mkdir creates directories. The standard entries "dot" (.), for the directory itself, and "dot dot" (..), for its parent, are made automatically.

mkdir requires write permission in the parent directory. The permissions assigned to the new directory are modified by the current file creation mask set by *umask* (C).

See Also

rmdir(C), umask(C)

Diagnostics

mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns nonzero.

Name

mkfs – Constructs a file system.

Syntax

/etc/mkfs [**-y**] [**-n**] special blocks[: inodes] [gap inblocks]
/etc/mkfs [**-y**] [**-n**] special proto [gap inblocks]
[-s blocks [: inodes]]

Description

mkfs constructs a file system by writing on the special file *special* , according to the directions found in the remainder of the command line.

If it appears that the special file contains a file system, operator confirmation is requested before overwriting the data. The **-y** "yes" option overrides this, and writes over any existing data without question. The **-n** option causes *mkfs* to terminate without question if the target contains an existing file system. The check used is to read block one from the target device (block one is the super-block) and see whether the bytes are the same. If they are not, this is taken to be meaningful data and confirmation is requested.

If the second argument is given as a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. The boot program is left uninitialized. If the number of inodes is specified, then this number should be the same as the estimated number of files in the file system. If the optional number of inodes is not given, the number of inodes is calculated as a function of the system file size.

If the second argument is a file name that can be opened, *mkfs* assumes it to be a prototype file, *proto*, and takes its directions from that file. The prototype file contains tokens separated by spaces or newlines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The bootstrap program specified should already be stripped of the header (see *strip*(CP)). If the header has not been stripped from the bootstrap program, then *mkfs* issues a warning. The second token is a number specifying the size of the created file system. Typically, it will have been the number of blocks on the device, perhaps diminished by space for swapping. The next token is the i-list size in blocks. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **-bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or **-** to specify set-user-ID mode or not. The third is **g** or **-** for the set-group-ID mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions; see *chmod*(C).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whose contents and size are copied. If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr  d--777 3 1
     sh   ---755 3 1 /bin/sh
     ken  d--755 6 1
          $
     b0   b--644 3 1 0 0
     c0   c--644 3 1 0 0
     $
$
```

In the second version of the command the **-s** option is a command-line override of the size and number of inodes in the *proto* file.

In both commands, the disk interleaving factors, *gap* and *inblocks* , can be specified. The interleaving factors are a disk hardware function and are described in detail in the XENIX *Operations Guide*.

See Also

chmod(C), filesystem(F), dir(F), strip(CP)

Notes

There is no way to specify links when using a prototype file. If the number of inodes is specified on the command line, then the maximum number of inodes in the file system is 65500.

This utility uses BSIZE blocks. Refer to the *machine (HW)* manual page for the size of filesystem blocks.

Name

mknod – Builds special files.

Syntax

/etc/mknod name [**c**] [**b**] major minor

/etc/mknod name **p**

/etc/mknod name **s**

/etc/mknod name **m**

Description

mknod makes a directory entry and corresponding inode for a special file. The first argument is the *name* of the entry. In the first case, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g., unit, drive, or line number), which may be either decimal or octal.

The assignment of major device numbers is specific to each system. Major device numbers can be found in the system source file **c.c**.

mknod can also be used to create named pipes with the **p** option; semaphores with the **s** option; and shared data (memory) with the **m** option.

Only the super-user can use the first form of the syntax.

System Compatibility

The **s** and **m** options can only be used to create XENIX version 3.0 semaphores and shared data, not XENIX System V semaphores and shared data.

See Also

mknod(S)

Name

mkuser – Adds a login ID to the system.

Syntax

/etc/mkuser

Description

mkuser is used to add more user login IDs to the system. It is the preferred method for adding new users to the system, since it handles all directory creation and password file update. To add a new user to the system, *mkuser* requires five pieces of information: the login name, the initial password, the group identification, the user's login shell and an optional comment string for the password file. It also allows the new user to be assigned to a group if required, although in most cases a default group is suitable. The program prompts for these five items and validates the given data. The login name is checked against certain criteria (i.e., it must be at least three characters and begin with a lowercase letter). The password must follow standard XENIX conventions, see *passwd*(C). The password file comment field can be up to 35 characters of information.

mkuser takes some of its parameters from a default file, **/etc/default/mkuser** . Currently, one may set the root path of home directories. An example default file is:

```
HOME=/usr
HOMEMODE=0755
PROFMODE=0640
```

Where HOME entry is the user's home directory, the HOMEMODE entry is the permissions for the user's home directory, and the PROFMODE entry is the permissions for the *.login, .profile and .cshrc files.*

This file can be edited (by the super-user) to change this default. There are five other files in the directory **/usr/lib** which may also be altered to suit local options. They are *mkuser.help* which is the introductory explanation given by *mkuser* on startup, *mkuser.mail* which is the initial mail message sent to new users, *mkuser.prof*, the standard *.profile* file given to new **sh** and **rsh** shell users, *mkuser.login*, the standard *.login* file given to new **csh** users, and *mkuser.cshrc*, the standard *.cshrc* file given to new **csh** users.

mkuser prompts for the shell type to assign to the new user. The shell types available are standard (Bourne) *sh* (option 1), visual shell *vsh* (option 2), c-shell *csh* (option 3), restricted *rsh* (option 4), and *uucp login* (option 5).

mkuser allocates user IDs starting at 200, or the largest number used in the password file. The default group ID for new users is 50. The minimum group ID allowed for user accounts is 50. The program prompts the operator for an optional group specification. This can either be a numeric group ID, or a group name. If the group exists, the user is added to it. If it does not exist, a new entry in **/etc/group** is created. A new group cannot have a numeric ID less than 51. If a new group is to be created, and the operator only specifies the group name, a free group ID is assigned. Alternatively, the operator can specify the group ID too.

mkuser can only be executed by the super-user.

The minimum length of a legal password, and the minimum and maximum number of weeks used in password aging are specified in **/etc/default/passwd** by the variables PASSLENGTH, MINWEEKS and MAXWEEKS. For example, these variables might be set as follows:

```
PASSLENGTH=6
MINWEEKS=2
MAXWEEKS=6
```

Files

/etc/passwd

/usr/spool/mail/*username*

/etc/default/mkuser

/etc/default/passwd

/usr/lib/mkuser/mkuser.cshrc

/usr/lib/mkuser/mkuser.help

/usr/lib/mkuser/mkuser.login

/usr/lib/mkuser/mkuser.prof

/usr/lib/mkuser/mkuser.mail

See Also

chmod(C), group(M), passwd(C), pwadmin(C), rmuser(C)

Name

more - Views a file one screen full at a time.

Syntax

more [**-cdflrsuw**] [**-*n***] [**+**linenumber] [**+/**pattern] [name ...]

Description

This filter allows examination of a continuous text one screen full at a time. It normally pauses after each full screen, displaying:

--More--

at the bottom of the screen. If the user then presses a carriage return, one more line is displayed. If the user presses the SPACE bar, another full screen is displayed. Other possibilities are described below.

The command line options are:

-n An integer which is the size (in lines) of the window which *more* will use instead of the default.

-c *more* draws each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

-d *more* prompts with the message "Hit space to continue, Rubout to abort" at the end of each full screen. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be inexperienced.

-f This option causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters that would ordinarily occupy screen positions, but do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are and fold lines erroneously.

-l Does not treat Ctrl-L (form feed) specially. If this option is not given, *more* pauses after any line that contains a Ctrl-L, as if the end of a full screen has been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

-s Squeezes multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

-u Normally, *more* handles underlining, such as that produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The **-u** option suppresses this processing.

-r Normally, *more* ignores control characters that it does not interpret in some way. The *-r* option causes these to be displayed as ^C where "C" stands for any such character.

-w Normally, *more* exits when it comes to the end of its input. With *-w* however, *more* prompts and waits for any key to be struck before exiting.

+linenumber
Starts up at *linenumber*.

+/pattern
Starts up two lines before the line containing the regular expression *pattern*.

more looks in the file **/etc/termcap** to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

more looks in the environment variable *MORE* to preset any flags desired. For example, if you prefer to view files using the **-c** mode of operation, the shell command "MORE=-c" in the **.profile** file causes all invocations of *more* to use this mode.

If *more* is reading from a file, rather than a pipe, a percentage is displayed along with the "--More--" prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be entered when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1):

i <space>
Displays *i* more lines, (or another full screen if no argument is given).

Ctrl-D
Displays 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d Same as Ctrl-D.

*i*z Same as entering a space except that *i*, if present, becomes the new window size.

*i*s Skips *i* lines and displays a full screen of lines.

*i*f Skips *i* full screens and displays a full screen of lines.

q or Q
Exits from *more.*

= Displays the current line number.

v Starts up the screen editor *vi* at the current line. Note that *vi* may not be available with your system.

h or ?
Help command; Gives a description of all the *more* commands.

i/expr
Searches for the *i*th occurrence of the regular expression *expr.* If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a full screen is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n Searches for the *i*th occurrence of the last regular expression entered.

(Single quotation mark) Goes to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!*command*
Invokes a shell with *command*. The characters % and ! in "command" are replaced with the current filename and the previous shell command respectively. If there is no current filename, % is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n
Skips to the *i*th next file given in the command line (skips to last file if *i* doesn't make sense).

i:p
Skips to the *i*th previous file given in the command line. If this command is given in the middle of printing out a file, *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell rings and nothing else happens.

:f Displays the current filename and line number.

:q or :Q
Exits from *more* (same as q or Q).

. Repeats the previous command.

The commands take effect immediately. It is not necessary to enter a carriage return. Up to the time when the command character itself is given, the user may enter the line kill character to cancel the numerical argument being formed. In addition, the user may enter the erase character to redisplay the "--More--(*xx*%)" message.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you enter will not show on your terminal, except for the slash (/) and exclamation (!) commands.

If the standard output is not a teletype, *more* acts just like *cat*, except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n |more -s
```

Files

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

See Also

csh(CP), sh(C), environ(M)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

The *vi* and *help* options may not be available.

Before displaying a file, *more* attempts to detect whether it is a non-printable binary file such as a directory or executable binary image. If *more* concludes that a file is unprintable, it refuses to print it. However, *more* cannot detect all possible kinds of non-printable files.

Name

mount – Mounts a file structure.

Syntax

/etc/mount [special-device directory [**-r**]]

/etc/umount special-device

/etc/mnt [**-rc**]

Description

mount announces to the system that a removable file structure is present on *special-device*. The file structure is mounted on *directory*. The *directory* must already exist; it becomes the name of the root of the newly mounted file structure. *directory* should be empty. If *directory* contains files, they will appear to have been removed while the *directory* is mounted and reappear when the *directory* is unmounted.

The *mount* and *umount* commands maintain a table of mounted devices. If each special device is invoked without any arguments, *mount* displays the name of the device, and the directory name of the mounted file structure, whether the file structure is read-only, and the date it was mounted.

The optional last argument indicates that the file is to be mounted read-only. Physically write-protected file structures must be mounted in this way or errors occur when access times are updated, whether or not any explicit write is attempted.

umount removes the removable file structure previously mounted on device *special-device*.

mnt is invoked from */etc/rc* with the **-rc** flag to mount filesystems when the system comes up multi-user. *mnt* uses the file */etc/default/filesys* for information on the filesystems.

Files

/etc/mnttab	Mount table
/etc/default/filesys	Filesystem data

See Also

umount(C), mount(S), mnttab(F), default(M)

Diagnostics

mount issues a warning if the file structure to be mounted is currently mounted under another name.

Busy file structures cannot be dismounted with *umount.* A file structure is busy if it contains an open file or some user's working directory.

Notes

Some degree of validation is done on the file structure, however it is generally unwise to mount corrupt file structures.

Be warned that when in single-user mode, the commands that look in **/etc/mnttab** for default arguments (for example *df, ncheck, quot, mount,* and *umount*) give either incorrect results (due to a corrupt **/etc/mnttab** from a non-shutdown stoppage) or no results (due to an empty mnttab from a *shutdown* stoppage).

When multi-user, this is not a problem; **/etc/rc** initializes **/etc/mnttab** to contain only **/dev/root** and subsequent mounts update it appropriately.

The *mount*(C) and *umount*(C) commands use a lock file to guarantee exclusive access to **/etc/mnttab.** The commands which just read it (those mentioned above) do not, so it is possible that they may hit a window, which is corrupt. This is not a problem in practice since *mount* and *umount* are not frequent operations.

When mounting a file system on a floppy disk you need not use the same *directory* each time. However, if you do, the full pathnames for the files are consistent with each use.

Floppy disks must be unprotected (no write-protect tab) to be mounted as a filesystem. Always **unmount** filesystems on floppy disks before removing them from the floppy drive. Failure to do so requires running **fsck** the next time the disk is **mounted .**

Name

mv – Moves or renames files and directories.

Syntax

mv [**-f**] file1 file2

mv [**-f**] file ... directory

Description

mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(S)) and reads the standard input to obtain a line. If the line begins with **y**, the move takes place; if not, *mv* exits.

In the second form, one or more *files* are moved to the *directory* with their original filenames.

No questions are asked when the **-f** option is given.

mv refuses to move a file onto itself.

See Also

cp(C), chmod(S), copy(C)

Notes

If *file1* and *file2* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Name

mvdir – Moves a directory.

Syntax

/etc/mvdir dirname name

Description

mvdir moves directories within a file system. *Dirname* must be a directory; *name* must not exist. Neither name may be a sub-set of the other (**/x/y** can not be moved to **/x/y/z**, nor vice versa).

See Also

mkdir(C)

Name

ncheck – Generates names from inode numbers.

Syntax

ncheck [**–i** numbers] [**–a**] [**–s**] [file-system]

Description

ncheck with no argument generates a pathname and inode number list of all files on the set of file systems specified in **/etc/mnttab.** The two characters "/." are appended to the names of directory files. The **–i** option reduces the report to only those files whose inode numbers follow. The **–a** option allows printing of the names . and .., which are ordinarily suppressed. The **–s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy. A single *filesystem* may be specified rather than the default list of mounted file systems.

Files

/etc/mnttab

See Also

fsck(C), sort(C)

Diagnostics

When the file system structure is improper, **??** denotes the "parent" of a parentless file and a pathname beginning with **...** denotes a loop.

Notes

See *Notes* under *mount*(C).

Name

netutil – Administers the XENIX network.

Syntax

netutil [-x] [-e] [–option]

Description

The *netutil* command allows the user to create and maintain a network of XENIX machines. A Micnet network is a link through serial lines of two or more XENIX systems. It is used to send mail between systems with the *mail*(C) command, transfer files between systems with the *rcp*(C) command, and execute commands from a remote system with the *remote*(C) command.

The **netutil** command is used to create and distribute the data files needed to implement the network. It is also used to start and stop the network. The *option* argument may be any one of **install**, **save**, **restore**, **start**, **stop**, or the numbers 1 through 5 respectively. The **-x** option logs transmissions and the **-e** options logs errors.

The **install** option interactively creates the data files needed to run the network. The **save** option saves these files on floppy or hard disks, allowing them to be distributed to the other systems in the network. If you save the micnet files to the hard disk, you can then use *uucp*(C) to transfer the files to the other machines. This option specifies the name of the backup device and prompts for whether this is the desired device to use. The user can specify an alternate device, including a file on the hard disk. The name of the default backup device is located in the file **/etc/default/micnet**. This can be changed depending on system configuration. The **restore** option copies the data files from floppy disk back to a system. The **start** option starts the network. The **stop** option stops the network. An *option* may also be any decimal digit in the range 1 to 5. If invoked without an *option*, the command displays a menu from which to choose one. Once an option is selected, it prompts for additional information if needed.

A network must be installed before it can be started. Installation consists of creating appropriate configuration files with the **install** option. This option requires the name of each machine in the network, the serial lines to be used to connect the machines, the speed of transmission for each line, and the names of the users on each machine. Once created, the files must be distributed to each computer in the network with the **save** and **restore** options. The network is started by using the **start** option on each machine in the network. Once started, mail and remote commands can be passed along the network. A record of the transmissons between computers in a network can be kept in the network log files. Installation of the network is described in the XENIX *Operations Guide*.

Files

/bin/netutil
/etc/default/micnet

See Also

aliases(M), aliashash(M), mail(C), micnet(M), remote(C), rcp(C), systemid(M), top(M)
XENIX *Operations Guide*

Name

newform – Changes the format of a text file.

Syntax

newform [**-s**] [**-i**tabspec] [**-o**tabspec] [**-bn**] [**-en**] [**-pn**] [**-an**] [**-f**] [**-c**char] [**-ln**] [file ...]

Description

newform reads lines from the named *files,* or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **-s**, command line options may appear in any order, may be repeated, and may be intermingled with *files*. Command line options are processed in the order typed. This means that option sequences like "**-e15** **-l60**" will yield results different from "**-l60** **-e15**". Options are applied to all *files* on the command line.

-i*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described below. In addition, *tabspec* may be **--**, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** expects no tabs; if any are found, they are treated as **-1**.

-o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **-i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** means that no spaces will be converted to tabs on output.

-l*n* Sets the effective line length to *n* characters. If *n* is not typed, -l defaults to 72. The default line length without the **-l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **-i** to expand tabs to spaces).

-b*n* Truncates *n* characters from the beginning of the line when the line length is greater than the effective line length (see **-l***n*). The default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **-b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

newform -l1 -b7 file-name

The option **-l1** must be used to set the effective line length shorter than any existing line in the file so that the **-b** option is activated.

-e*n* Truncates *n* characters from the end of the line.

-c*k* Changes the prefix/append character to *k*. Default character for *k* is a space (see options -p and -c).

-p*n* Prefixes *n* characters (see **-c***k*) to the beginning of a line when the line length is less than the effective line length. The default is to prefix the number of characters necessary to obtain the effective line length.

-a*n* Appends *n* characters to the end of a line. The default is to append the number of characters necessary to get the effective line length.

-f Writes the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **-o** option. If no **-o** option is specified, the line which is printed will contain the default specification of **-8**.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

Tabs

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. The lowest column number is 1. For *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g. the DASI 300, DASI 300S, and DASI 450.

The "canned" tabs are given as -*code* where *code* (and its meaning) is from the following list:

-a 1,10,16,36,72
Assembler, IBM S/370, first format

-a2 1,10,16,40,72
Assembler, IBM S/370, second format

-c 1,8,12,16,20,55
COBOL, normal format

-c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:
<:t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columnms 1-6 omitted), with more tabs than COBOL **-c2**. This is the recommended format for COBOL. The appropriate format specification is:
<:t-c3 m6 s66 d:>

-f 1,7,11,15,19,23
FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,53,57,61
PL/I

-s 1,10,55
SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

-*n* A repetitive specification requests tabs at columns 1+*n*, 1+2**n*, etc. Note that such a setting leaves a left margin of *n* columns on TermiNet terminals *only*. Of particular importance is the value -8: this represents the XENIX system "standard" tab setting, and is the most likely tab setting to found at a terminal. It is required for use with *nroff*(CT) **-h** option for high-speed output. Another special case is the value **-0**, implying no tabs at all.

n1,n2,... The arbitrary format permits the user to type any chosen set of number, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

- *-file*

If the name of a file is given, *newform* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings.

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-T*type*

newform usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in *term*(CT). If no **-T** flag is supplied, *newform* searches for the **$TERM** value in the *environment* (see *environ*(M)). If no *type* can be found, *newform* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n*+1 the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Example

In the following example, *newform* converts a file named *text* with leading digits, one or more tabs, and text on each line to a file beginning with the text and the leading digits placed at the end of each line in column 73 (-s option). All tabs after the first one are expanded to spaces (**-i** option). To reach the line length of 72 characters (**-l** option), spaces are appended to each line up to column 72 (**-a** option) or lines are truncated at column 72 (**-e** option). To reformat the sample file **text** in this manner, enter:

```
newform -s -i -l -a -e text
```

Exit Codes

0 - normal execution
1 - for any error

See Also

csplit(C)

Diagnostics

All diagnostics are fatal.

usage: ...	*newform* was called with a bad option.
not -s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

Notes

newform normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was "-o--" , and was preceded by either "-o--" or a "-i--" , the tab specification format line will be incorrect.

Name

newgrp – Logs user into a new group.

Syntax

newgrp [group]

Description

newgrp changes the group identification of its caller. The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

newgrp without an argument changes the group identification to the group in the password file; in effect it changes the group identification back to the caller's original group.

A password is demanded if the group has a password and the user himself does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

When most users log in, they are members of the group named **group**.

Files

/etc/group

/etc/passwd

See Also

login(M), group(M)

Notes

There is no convenient way to enter a password into **/etc/group**.

Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Shell variables are not preserved when invoking this command unless they are explicitly exported. (See *sh*(C).)

Name

ncheck – Generates names from inode numbers.

Syntax

ncheck [**–i** numbers] [**–a**] [**–s**] [file-system]

Description

ncheck with no argument generates a pathname and inode number list of all files on the set of file systems specified in **/etc/mnttab**. The two characters "**/.**" are appended to the names of directory files. The **–i** option reduces the report to only those files whose inode numbers follow. The **–a** option allows printing of the names . and .., which are ordinarily suppressed. The **–s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy. A single *filesystem* may be specified rather than the default list of mounted file systems.

Files

/etc/mnttab

See Also

fsck(C), sort(C)

Diagnostics

When the file system structure is improper, **??** denotes the "parent" of a parentless file and a pathname beginning with **...** denotes a loop.

Notes

See *Notes* under *mount*(C).

Name

news – Print news items.

Syntax

news [**−a**] [**−n**] [**−s**] [items]

Description

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

The **−a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The **−n** option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The **−s** option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time.

All other arguments are assumed to be specific news items that are to be printed.

If the INTERRUPT key is struck during the printing of a news item, printing stops and the next item is started. Another INTERRUPT within one second of the first causes the program to terminate.

Files

/usr/news/*
$HOME/.news_time

See Also

profile(M), environ(M).

Name

nice – Runs a command at a different priority.

Syntax

nice [**−**increment] command [arguments]

Description

nice executes *command* with a lower CPU scheduling priority. Priorities range from 0 to 39, where 0 is the highest priority and 39 is the lowest. By default, commands have a "nice value" of 20. If an **−increment** argument is given where *increment* is in the range 1-19, *increment* is added to the default priority of 20 to produce a numerically higher priority, meaning a *lower* scheduling priority. If no *increment* is given, an increment of 10 to produce a priority of 30 is assumed.

The super-user may run commands with priority *higher* than normal by using a double negative increment. For example, an argument of **--10** would decrement the default to produce a nice value of 10, which is a higher scheduling priority than the default of 20.

See Also

nohup(C), nice(S)

Diagnostics

nice returns the exit status of the subject command.

Notes

An *increment* larger than 19 is equivalent to 19.

Name

nl − Adds line numbers to a file.

Syntax

nl [**−h**type] [**−b**type] [**−f**type] [**−v**start#] [**−i**incr] [**−p**] [**−l**num] [**−s**sep] [**−w**width] [**−n**format] file

Description

nl reads lines from the named *file*, or the standard input if no *file* is named, and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. 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).

The start of logical page sections is signaled by input lines containing nothing but the following character(s):

Page Section	*Line Contents*
Header	\:\:\:
Body	\:\:
Footer	\:

Unless signaled otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional filename. Only one file may be named. The options are:

−b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p***string*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).

−h*type* Same as **−b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

−f*type* Same as **−b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

−p Does not restart numbering at logical page delimiters.

−v*start#* *Start#* is the initial value used to number logical page lines. Default is **1**.

−i*incr* *Incr* is the increment value used to number logical page lines. Default is **1**.

−s*sep* *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

−w*width* *Width* is the number of characters to be used for the line number. Default *width* is **6**.

−n*format* *Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes supressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

−l*num* *Num* is the number of blank lines to be considered as one. For example, **−l2** results in only the second adjacent blank being numbered (if the appropriate **−ha**, **−ba**, and/or **−fa** option is set). Default is **1**.

See Also

pr(C)

Name

nohup – Runs a command immune to hangups and quits.

Syntax

nohup command [arguments]

Description

nohup executes *command* with hangups and quits ignored. If output is not redirected by the user, it will be sent to **nohup.out**. If **nohup.out** does not have write permission in the current directory, output is redirected to **$HOME/nohup.out.**

See Also

nice(C), signal(S)

Name

od – Displays files in octal format.

Syntax

od [−bcdox] [file] [[**+**]offset[**.**][**b**]]

Description

od displays *file* in one or more formats as selected by the first argument. If the first argument is missing, **−o** is default. The meanings of the format options are:

−b Interprets bytes in octal.

−c Interprets bytes in ASCII. Certain nongraphic characters appear as C escapes: null=**\0**, backspace=**\b**, form feed=**\f**, newline=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.

−d Interprets words in decimal.

−o Interprets words in octal.

−x Interprets words in hex.

The *file* argument specifies which file is to be displayed. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where displaying is to start. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks. If the file argument is omitted, the offset argument must be preceded by **+**.

The display continues until end-of-file.

See Also

hd(C), adb(CP)

Name

pack, pcat, unpack – Compresses and expands files.

Syntax

pack [–] name ...

pcat name ...

unpack name ...

Description

pack attempts to store the specified files in a compressed form. Wherever possible, each input file *name* is replaced by a packed file *name***.z** with the same access modes, access and modified dates, and the owner of *name.* If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very scattered, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

- The file appears to be already packed
- The filename has more than 12 characters

- The file has links
- The file is a directory
- The file cannot be opened
- No disk storage blocks will be saved by packing
- A file called *name*.**z** already exists
- The .**z** file cannot be created
- An I/O error occurred during processing

The last segment of the filename must contain no more than 12 characters to allow space for the appended .**z** extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(C) does for ordinary files. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.**z** use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name*.**z** without destroying *name*.**z**, enter the command:

```
pcat name >nnn
```

Pcat returns the number of files it was unable to unpack. Failure may occur if:

- The filename (exclusive of the .**z**) has more than 12 characters
- The file cannot be opened
- The file does not appear to be the output of *pack*

unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.**z** (or just *name*, if *name* ends in .**z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .**z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as in a file where the "unpacked" name already exists, or if the unpacked file cannot be created.

Name

passwd – Changes login password.

Syntax

passwd name

Description

This command changes (or installs) a password associated with the login *name*.

The program prompts for the old password (if any) and then for the new one (twice). The user must supply these. Passwords can be of any reasonable length, but only the first eight characters of the password are significant. The minimum number of characters allowed in a new password is determined by the PASSLENGTH variable. Although the minimum can be 3, a minimum of 5 characters is strongly recommended since passwords shorter than this are much easier to guess or discover by trial and error.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password. Only the super-user can create a null password.

The password file is not changed if the new password is the same as the old password, or if the password has not "aged" sufficiently; see *passwd*(M)).

The minimum length of a legal password, and the minimum and maximum number of weeks used in password aging are specified in **/etc/default/passwd** by the variables PASSLENGTH, MINWEEKS and MAXWEEKS. If not explicitly set, the default values for these variables are:

PASSLENGTH=5
MINWEEKS=2
MAXWEEKS=4

MINWEEKS and MAXWEEKS values must be in the range 0 to 63. If PASSLENGTH is not in the range 3 to 8, it is set to 5.

Files

/etc/default/passwd
/etc/passwd

See Also

default(M), login(M), passwd(M), pwadmin(C)

Name

pg - File perusal filter for soft-copy terminals.

Syntax

pg [- number] [**-p** string] [**-cefns**] [+ linenumber] [+/ pattern /] [files ...]

Description

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a soft-copy terminal. (The dash (-) command line option and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If you press the RETURN key, another page is displayed; other possibilities are listed below. This command is different from previous paginators because it allows you to back up and review something that has already passed.

To determine terminal attributes, *pg* scans the *termcap*(M) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

- *number* Specifies the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23.)

-p *string* Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is a colon (:).

-c Homes the cursor and clears the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *termcap*(M) data base.

-e Causes *pg not* to pause at the end of each file.

-f Inhibits *pg* from splitting lines. In the absence of the **-f** option, *pg* splits lines longer than the screen width, but some sequences of characters in the displayed text (for example, escape sequences for underlining) give undesirable results.

-n Normally, commands must be terminated by pressing the RETURN key (ASCII newline character). This option causes an automatic end of command as soon as a command letter is entered.

-s Causes *pg* to display all messages and prompts in standout mode (usually inverse video).

+*linenumber*
Starts up at *linenumber*.

+/*pattern***/** Starts up at the first line containing the regular expression pattern.

The responses that may be entered when *pg* pauses can be divided into three categories: those that cause further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address* (an optionally signed number indicating the point from which further text should be displayed). *pg* interprets this *address* in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address if no address is provided.

The perusal commands and their defaults are as follows:

(+1)RETURN key or *<blank>*
Causes one page to be displayed. The *address* is specified in pages.

(+1) **l**
With a signed *address*, causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an unsigned *address* this command displays a full screen of text beginning at the specified line.

(+1) **d** or **Ctrl-D**
Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*:

. or **Ctrl-L**
Causes the current page of text to be redisplayed.

$ Displays the last window full in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(C) are available. They must always be terminated by a newline character, even if the *-n* option is specified.

i/pattern/
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
i?pattern?
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The caret (^) notation is useful for terminals which will not properly handle the question mark (?).

After searching, *pg* displays the line found at the top of the screen. You can modify this by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. Use the suffix **t** to restore the original situation.

The following commands modify the environment of perusal:

*i***n** Begins perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begins perusing the *i*th previous file in the command line. The *i* is an unsigned number, default is 1.

*i***w** Displays another window of text. If *i* is present, set the window size to *i*.

s *filename*
: Saves the input in the named file. Only the current file being perused is saved. The white space between the s and *filename* is optional. This command must always be terminated by a newline character, even if the *-n* option is specified.

h Help displays abbreviated summary of available commands.

q or **Q** Quit *pg*.

!*command*
: *command* is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a newline character, even if the *-n* option is specified.

At any time when output is being sent to the terminal, the user can press the quit key (normally Ctrl-\) or the INTERRUPT (BREAK) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, because any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(C), except that a header is printed before each file (if there is more than one).

Example

To use *pg* to read system news, enter:

news | pg -p "(Page %d):"

Files

/etc/termcap	Terminal information data base
/tmp/pg*	Temporary file when input is from a pipe

See Also

ed(C), grep(C), termcap(M)

Notes

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

While waiting for terminal input, *pg* responds to "BREAK, DEL," and the caret (^) by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place you in prompt mode. Use these signals with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

The *z* and *f* commands used with *more* are available, and the terminal slash (/), caret (^), or question mark (?) may be omitted from the searching commands.

Name

pr – Prints files on the standard output.

Syntax

pr [options] [files]

Description

pr prints the named files on the standard output. If *file* is **–**, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **–s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

Options may appear singly or combined in any order. Their meanings are:

+*k* Begins printing with page *k* (default is 1).

–*k* Produces *k*-column output (default is 1). The options **–e** and **–i** are assumed for multicolumn output.

–a Prints multicolumn output across the page.

–m Merges and prints all files simultaneously, one per column (overrides the **–***k*, and **–a** options).

–d Double-spaces the output.

–e*ck* Expands *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every 8th position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any nondigit character) is given, it is treated as the input tab character (default for *c* is the tab character).

–i*ck* In *output*, replaces whitespace wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every 8th position are assumed. If *c* (any nondigit character) is given, it is treated as the output tab character (default for *c* is the tab character).

−n*ck* Provides *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **−m** output. If *c* (any nondigit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

−w*k* Sets the width of a line to *k* character positions (default is 72 for equal-width multicolumn output, no limit otherwise).

−o*k* Offsets each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

−l*k* Sets the length of a page to *k* lines (default is 66).

−h Uses the next argument as the header to be printed instead of the filename.

−p Pauses before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

−f Uses form feed character for new pages (default is to use a sequence of linefeeds). Pauses before beginning the first page if the standard output is associated with a terminal.

−r Prints no diagnostic reports on failure to open files.

−t Prints neither the 5-line identifying header nor the 5-line trailer normally supplied for each page. Quits printing after the last line of each file without spacing to the end of the page.

−s*c* Separates columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

Examples

The following prints *file1* and *file2* as a double-spaced, three-column listing headed by "file list":

```
pr −3dh "file list" file1 file2
```

The following writes **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, . . . :

```
pr -e9 -t <file1 >file2
```

See Also

cat(C)

Name

ps – Reports process status.

Syntax

ps [options]

Description

ps prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. Otherwise, the information that is displayed is controlled by the following *options*:

–e Prints information about all processes.

–d Prints information about all processes, except process group leaders.

–a Prints information about all processes, except process group leaders and processes not associated with a terminal.

–f Generates a *full* listing. (Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed.) See below for meaning of columns in a full listing.

–l Generates a *long* listing. See below.

–c *corefile* Uses the file *corefile* in place of **/dev/mem**.

–s *swapdev* Uses the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*.

–n *namelist* The argument is taken as the name of an alternate *namelist* (**/xenix** is the default).

–t *tlist* Restricts listing to data about the processes associated with the terminals given in *tlist*, where *tlist* can be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

−p *plist* Restricts listing to data about processes whose process ID numbers are given in *plist*, where *plist* is in the same format as *tlist*.

−u *ulist* Restricts listing to data about processes whose user ID numbers or login names are given in *ulist*, where *ulist* is in the same format as *tlist*. In the listing, the numerical user ID is printed unless the **−f** option is used, in which case the login name is printed.

−g *glist* Restricts listing to data about processes whose process groups are given in *glist*, where *glist* is a list of process group leaders and is in the same format as *tlist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options only determine what information is provided for a process; they do *not* determine which processes will be listed.

F (l) A status word consisting of flags associated with the process. Each flag is associated with a bit in the status word. These flags are added to form a single octal number. Process flag bits and their meanings are:

01	in core;
02	system process;
04	locked in core (e.g., for physical I/O);
10	being swapped;
20	being traced by another process.

S (l) The state of the process:

0	non-existent;
S	sleeping;
W	waiting;
R	running;
I	intermediate;
Z	terminated;
T	stopped.

UID (f,l) The user ID number of the process owner; the login name is printed under the **−f** option.

PID (all) The process ID of the process; it is possible to kill a process if you know this datum.

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

STIME (f) Starting time of the process.

PRI (l) The priority of the process; higher numbers mean lower priority.

NI (l) Nice value; used in priority computation.

ADDR (l) The memory address of the process, if resident; otherwise, the disk address.

SZ	(l)	The size in blocks of the core image of the process, but not including the size of text shared with other processes. Since this size includes the current size of the stack, it will vary as the stack size varies.
WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
TTY	(all)	The controlling terminal for the process.
TIME	(all)	The cumulative execution time for the process.
CMD	(all)	The command name; the full command name and its arguments are printed under the **−f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **−f** option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **−f** option, is printed in square brackets.

Files

/xenix	system namelist
/dev/mem	memory
/dev	searched to find swap device and terminal ("tty") names.

See Also

kill(C), nice(C)

Notes

Things can change while *ps* is running; the picture it gives is only a close approximation to reality.

Some data printed for defunct processes are irrelevant.

Name

pstat – Reports system information.

Syntax

pstat [**−aixpf**] [**−u** ubase] [**−c** corefile]
[**−n** namelist] [file]

Description

pstat interprets the contents of certain system tables. *pstat* searches for these tables in **/dev/mem** and **/dev/kmem.** With the *file* given, the tables are sought in the specified *file* rather than **/dev/mem.** Similarly, the **−c** option allows one to specify a *corefile* rather than **/dev/kmem** for the search. The required namelist is taken from **/xenix.** Options are:

−a Under **−p**, describe all process slots rather than just active ones.

−i Print the inode table with these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

- L Locked
- U Update time *filesystem*(F) must be corrected
- A Access time must be corrected
- M File system is mounted here
- W Wanted by another process (L flag is on)
- T Contains a text file
- C Changed time must be corrected

CNT Number of open file table entries for this inode.

DEV Major and minor device number of file system in which this inode resides.

INO I-number within the device.

MODE Mode bits, see *chmod*(S).

NLK Number of links to this inode.

UID User ID of owner.

SIZ/DEV Number of bytes in an ordinary file, or major and minor device of special file.

−x Prints the text table with these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

- T *ptrace*(S) in effect
- W Text not yet written on swap device
- L Loading in progress

K Locked
w Wanted (L flag is on)

DADDR
Disk address in swap, measured in multiples of BSIZE bytes. (286 only)

CADDR
Core address, measured in units of memory management resolution.

SIZE
Size of text segment, measured in units of memory management resolution. (286 only)

IPTR
Core location of corresponding inode.

CNT
Number of processes using this text segment.

CCNT
Number of processes in core using this text segment.

-p Prints process table for active processes with these headings:

LOC
The core location of tbis table entry.

S
Run state encoded thus:
0 No process
1 Waiting for some event
3 Runnable
4 Being created
5 Being terminated
6 Stopped under trace

F
Miscellaneous state variables, ORed together:
01 Loaded
02 The scheduler process
04 Locked
010
Swapped out
020
Traced
040
Used in tracing
0100
Locked in by *lock*(S).

PRI
Scheduling priority, see *nice*(S).

SIGNAL
Signals received (signals 1-16 coded in bits 0-15).

UID
Real user ID.

TIM
Time resident in seconds; times over 127 coded as 127.

CPU
Weighted integral of CPU time, for scheduler.

NI
Nice level, see *nice*(S).

PGRP
Process number of root of process group (the opener of the controlling terminal).

PID
The process ID number.

PPID
The process ID of parent process.

ADDR	If in core, the physical address of the "u-area" of the process measured in units of memory management resolution. If swapped out, the position in the swap area is measured in multiples of BSIZE bytes.
SIZE	Size of process image, measured in units of memory management resolution.
WCHAN	Wait channel number of a waiting process.
LINK	Link pointer in list of runnable processes.
TEXTP	If text is pure, pointer to location of text table entry.
CLKT	Countdown for *alarm*(S) measured in seconds.

−u *ubase*

Print information about a user process. *Ubase* is the hexadecimal location of the process in main memory. The address may be obtained by using the long listing (**−l** option) of the *ps*(C) command.

−c *corefile*

Use the file *corefile* in place of **/dev/kmem.**

−n *namelist*

Use the file *namelist* as an alternate namelist in place of **/xenix.**

−f Print the open file table with these headings:

LOC	The core location of this table entry.
FLG	Miscellaneous state variables: R Open for reading W Open for writing P Pipe
CNT	Number of processes that know this open file.
INO	The location of the inode table entry for this file.
OFFS	The file offset, see *lseek*(S).

Files

/xenix Namelist

/dev/mem Default source of tables

See Also

ps(C), stat(S), filesystem(F)

Name

pwadmin – Performs password aging administration.

Syntax

pwadmin [**–min** weeks **–max** weeks] options

Description

pwadmin is used to examine and modify the password aging information in the password file. The options one can specify are as follows:

–d *user*
: Displays the password aging information.

–f *user*
: Forces the user to change his password at the next login.

–c *user*
: Prevents the user from changing his password.

–a *user*
: Enables password aging for the given user. This option sets the minimum number of weeks that the user must wait before changing his password and the maximum number of weeks that a user can keep his current password for the values defined by the MINWEEKS and MAXWEEKS variables in the **/etc/default/passwd** file. If the file is not found or the defined values are not in the range 0 to 63, the default values 2 and 4 are used.

–n *user*
: Disables the password aging feature.

–min *weeks*
: Enables password aging and sets the minimum number of weeks before the user can change his password to *weeks*. (This prevents him from changing his password back to the old one).

–max *weeks*
: Enables password aging and sets the number of weeks so the user can keep his current password set for *weeks*.

Files

/etc/passwd

/etc/default/passwd

See Also

passwd(C), passwd(M)

Notes

The user must not attempt to force a new password by setting both the **−min** and **−max** values to zero. To force a password, use the **−f** option.

The user must not attempt to prevent further password changes by setting the **−min** value greater than the **−max** value. To prevent changes, use the **−c** option.

Name

pwcheck – Checks password file.

Syntax

pwcheck [file]

Description

pwcheck scans the password file and checks for any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The default password file is **/etc/passwd**.

Files

/etc/passwd

See Also

grpcheck(C), group(M), passwd(M)

Name

pwd – Prints working directory name.

Syntax

pwd

Description

pwd prints the pathname of the working (current) directory.

See Also

cd(C)

Diagnostics

"Cannot open .." and "Read error in .." indicate possible file system trouble. In such cases, see the XENIX *Operations Guide* for information on fixing the file system.

Name

quot - Summarizes file system ownership.

Syntax

quot [option] ... [filesystem]

Description

quot prints the number of blocks in the named *filesystem* currently owned by each user. If no *filesystem* is named, the file systems given in **/etc/mnttab** are examined.

The following options are available:

-n Processes pipeline input for display. Specifically, the following pipeline:

ncheck filesystem | sort +0n | quot -n filesystem

produces a list of all files and their owners.

-c Prints three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file. Data for files of size greater than 499 blocks are included in the figures for files of exactly size 499.

-f Prints a count of the number of files as well as space owned by each user.

Files

/etc/passwd	Gets user names
/etc/mnttab	Contains list of mounted file systems

See Also

cmchk(C), du(C), ls(C), machine(M)

Notes

Holes in files are counted as if they actually occupied space.

Blocks are reported in 512 byte blocks. On filesystems that use 1024 byte blocks, a file of 26 bytes is reported as using 2 blocks, since it uses one 1024 byte block, or two 512 byte blocks. See *machine*(M) or use *cmchk*(C) to determine the filesystem block size.

See also *Notes* under *mount*(C).

Name

random – Generates a random number.

Syntax

random [−s] [scale]

Description

random generates a random number on the standard output. and returns the number as its exit value. By default, this number is either 0 or 1 (i.e., *scale* is 1 by default). If *scale* is given a value between 1 and 255, then the range of the random value is from 0 to *scale*. If *scale* is greater than 255, an error message is printed.

When the **−s** , "silent" option is given, the random number is returned as an exit value but is not printed on the standard output. If an error occurs, *random* returns an exit value of zero.

See Also

rand(S)

Notes

This command does not perform any floating point computations.

random uses the time of day as a seed.

Name

rcp – Copies files across XENIX systems.

Syntax

rcp [options] [srcmachine:]srcfile [destmachine:]destfile

Description

rcp copies files between systems in a Micnet network. The command copies the *srcmachine:srcfile* to *destmachine:destfile*, where *srcmachine*: and *destmachine*: are optional names of systems in the network, and *srcfile* and *destfile* are pathnames of files. If a machine name is not given, the name of the current system is assumed. If **–** is given in place of *srcfile, rcp* uses the standard input as the source. Directories named on the destination machine must have write permission, and directories and files named on a remote source machine must have read permission.

The available options are:

–m
: Mails and reports completion of the command, whether there is an error or not.

–u [*machine*:]*user*
: Any mail goes to the named *user* on *machine*. The default *machine* is the machine on which the *rcp* command is completed or on which an error was detected. If an alias for *user* exists in the system alias files on that *machine*, the mail will be redirected to the appropriate mailbox(es). Since system alias files are usually identical throughout the network, any specified *machine* will most likely be overridden by the aliasing mechanism. To prevent aliasing, *user* must be escaped with at least two \ characters (at least four if given as a shell command).

rcp is useful for transferring small numbers of files across the network. The network consists of daemons that periodically awaken and send files from one system to another. The network must be installed using *netutil* (C) before *rcp* can be used.

Also, to enable transfer of files from a remote system, either:

This line should be in */etc/default/micnet* on the systems in the network:

rcp=/usr/bin/rcp

Or, these lines should be in that file:

executeall
execpath=PATH=*path*

where *path* must contain */usr/bin*.

Example

rcp −m machine1:/etc/mnttab /tmp/vtape

See Also

mail(C), micnet(M), netutil(C), remote(C)

Diagnostics

If an error occurs, mail is sent to the user.

Notes

Full pathnames must be specified for remote files.

rcp handles binary data files transparently, no extra options or protocols are needed to handle them. Wildcards are not expanded on the remote machine.

Name

red – Invokes a restricted version of *ed*(C).

Syntax

red [file]

Description

red is a restricted version of *ed*(C). It will only allow editing of files in the current directory. It prohibits executing *sh*(C) commands via the ! command. *red* displays an error message on any attempt to bypass these restrictions.

In general, *red* does not allow commands like

    !date

or

    !sh

Furthermore, *red* will not allow pathnames in its command line. For example, the command:

    red /etc/passwd

when the current directory is not **/etc** causes an error.

See Also

ed(C), rsh(C)

Name

remote – Executes commands on a remote XENIX system.

Syntax

remote [–] [–f file] [–m] [–u user] machine
command [arguments]

Description

remote is a limited networking facility that permits execution of XENIX commands across serial lines. Commands on any connected system may be executed from the host system using *remote.* A command line consisting of *command* and any blank-separated *arguments* is executed on the remote *machine*. A machine's name is located in the file **/etc/systemid.** Note that wild cards are *not* expanded on the remote machine, so they should not be specified in *arguments*. The optional **–m** switch causes mail to be sent to the user telling whether the command is successful.

The available options follow:

– A dash signifies that standard input is used as the standard input for *command* on the remote *machine*. Standard input comes from the local host and not from the remote machine.

–f *file* Use the specified *file* as the standard input for *command* on the remote *machine*. The *file* exists on the local host and not on the remote machine.

–m Mails the user to report completion of the command. By default, mail reports only errors.

–u *user* Any mail goes to the named *user* on *machine.* The default *machine* is the machine on which an error was detected, or on which the *remote* command was completed. The mail will be redirected to the appropriate mailbox(es), if an alias for *user* exists in the system alias files on that *machine* . Since system alias files are usually identical throughout the network, any specified *machine* will most likely be overridden by the aliasing mechanism. To prevent aliasing, *user* must be escaped with at least two \ characters (at least four if given as a shell command).

Before *remote* can be successfully used, a network of systems must first be set up and the proper daemons initialized using *netutil* (C). Also, entries for the command to be executed using *remote* must be added to the */etc/default/micnet* files on each remote machine.

Example

The following command executes an *ls* command on the directory **/tmp** of the machine *machine1*:

```
remote machine1 ls /tmp
```

See Also

rcp(C), mail(C), netutil(C), micnet(M)

Notes

The *mail* command uses the equivalent of *remote* to send mail between machines.

Name

restore, restor – Invokes incremental file system restorer.

Syntax

restore key [arguments]

restor key [arguments]

Description

restore is used to read archive media backed up with the *backup*(C) command.

The *key* specifies what is to be done. *Key* is one of the characters **cC**, **rR**, **tT**, or **xX** optionally combined with **k** and/or f or **F**. *restor* is an alternate spelling for the same command.

c,C
: Verify (check) a dump tape. Used after a dump is made to make sure the tape has no I/O errors or bad checksums. **C** is the same as **c** except that it provides a higher level of checking.

f Uses the first *argument* as the name of the archive (backup device /dev/*) instead of the default.

F **F** is the number of the first file on the tape to read. All files up to that point are skipped.

k Follow this option with the size of the backup volume. This allows for reading multivolume dumps from media such as floppies.

r,R
: The archive is read and loaded into the file system specified in *argument*. This should not be done lightly (see below). If the key is **R**, *restore* asks which archive of a multivolume set to start on. This allows *restore* to be interrupted and then restarted (an *fsck* must be done before the restart).

t Prints the date the archive was written and the date the file system was backed up.

T Prints a full listing of a dump tape. Similar to t.

x Each file on the archive named by an *argument* is extracted. The filename has all "mount" prefixes removed; for example, if **/usr** is a mounted file system, **/usr/bin/lpr** is named **/bin/lpr** on the archive.

The extracted file is placed in a file with a numeric name supplied by *restore* (actually the inode number). In order to keep the amount of archive read to a minimum, the following procedure is recommended:

1. Mount volume 1 of the set of backup archives.

2. Type the *restore* command with the appropriate key and arguments.

3. *restore* will check *dumpdir*, then announce whether or not it found the files, give the numeric name that it will assign to the file, and in the case of a tape, rewind to the start of the archive.

4. It then asks you to "mount the desired tape volume". Type the number of the volume you choose. On a multivolume backup, the recommended procedure is to mount the last through the first volumes, in that order. *restore* checks to see if any of the requested files are on the mounted archive (or a later archive, thus the reverse order). If the requested files are not there, *restore* doesn't read through the tape. If you are working with a single-volume backup or if the number of files being restored is large, respond to the query with 1 and *restore* will read the archives in sequential order.

X Same as **x** except that files are replaced in original location.

The **r** option should only be used to restore a complete backup archive onto a clear file system, or to restore an incremental backup archive onto a file system so created. Thus:

```
/etc/mkfs /dev/hd1 10000
restore r /dev/hd1
```

is a typical sequence to restore a complete backup. Another *restore* can be done to get an incremental backup in on top of this.

A *backup* followed by a *mkfs* and a *restore* is used to change the size of a file system.

Files

rst*	Temporary files
/etc/default/restor	Name of default archive device

The default archive unit varies with installation.

Notes

It is not possible to successfully *restore* an entire active root file system.

Diagnostics

There are various diagnostics involved with reading the archive and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the dump extends over more than one disk or tape, *restor* may ask you to change disks or tapes. Reply with a newline when the next unit has been mounted.

See Also

backup(C), dumpdir(C), fsck(C), mkfs(C), sddate(C)

Name

rm, rmdir – Removes files or directories.

Syntax

rm [**-fri**] file ...

rmdir dir ...

Description

rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with **y**, the file is deleted, otherwise the file remains. No questions are asked when the **-f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **-i** (interactive) option is in effect, *rm* asks whether to delete each file, and if the **-r** option is in effect, whether to examine each directory.

rmdir removes empty directories.

See Also

rmdir(C)

Diagnostics

Generally self-explanatory. It is forbidden to remove the file **..** to avoid the consequences of inadvertently doing something like:

```
rm -r .*
```

It is also forbidden to remove the root directory of a given file system.

No more than 17 levels of subdirectories can be removed using the **−r** option.

Name

rmdir – Removes directories.

Syntax

rmdir dir ...

Description

rmdir removes the entries for one or more subdirectories from a directory. A directory must be empty before it can be removed. *rmdir* enforces a standard and *safe* procedure for removing a directory; the contents of the directory must be removed before the directory itself can be deleted with *rmdir* . Note that the "rm -r *dir*" command is a more dangerous alternative to *rmdir.*

rmdir removes entries for the named directories, which must be empty.

See Also

rm(C)

Notes

rmdir will refuse to remove the root directory of a mounted file system.

Name

rmuser – Removes a user account from the system.

Syntax

/etc/rmuser

Description

rmuser removes users from the system. It begins by prompting for a user name; after receiving a valid user name as a response, it then deletes the named user's entry in the password file, and removes the user's mailbox file, the **.profile** file, and the entire home directory. It will also remove the users group entry in **/etc/group** if the user was the only remaining member of that group, and the group ID was greater than 50.

Before removing a user ID from the system, make sure its mailbox is empty and that all files belonging to that user ID have been saved or deleted as required.

The *rmuser* program will refuse to remove a user ID or any of its files if one or more of the following checks fails:

- The user name given is one of the "system" user names such as root, sys, sysinfo, cron, or uucp. All user IDs less than 200 are considered reserved for system use, and cannot be removed using *rmuser*. Likewise, all group IDs less than 50 are not removable using *rmuser*.

- The user's mailbox exists and is not empty.

- The user's home directory contains files other than **.profile** .

rmuser can only be executed by the super-user.

Files

/etc/passwd

/usr/spool/mail/*username*

$HOME

See Also

mkuser(C), backup(C)

Name

rsh – Invokes a restricted shell (command interpreter).

Syntax

rsh [flags] [name [arg1 ...]]

Description

rsh is a restricted version of the standard command interpreter *sh*(C). It is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that changing directory with *cd*, setting the value of $PATH, using command names containing slashes, and redirecting output using > and >> are all disallowed.

When invoked with the name **–rsh**, *rsh* reads the user's **.profile** (from **$HOME/.profile**). It acts as the standard *sh* while doing this, except that an interrupt causes an immediate exit, instead of causing a return to command level. The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end user shell procedures that have access to the full power of the standard shell, while restricting him to a limited menu of commands; this scheme assumes that the end user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions, then leaving the user in an appropriate directory (probably *not* the login directory).

rsh is actually just a link to *sh* and any *flags* arguments are the same as for *sh*(C).

The system administrator often sets up a directory of commands that can be safely invoked by *rsh*.

See Also

sh(C), profile(M)

Name

runbig - Runs a command that may require more memory than normal.

Syntax

runbig command [arguments]

Description

runbig executes commands that may require more memory than is normally available to a user process. While *runbig* is executing the specified *command*, it ignores the restriction on the default of memory available to the user process. The *command* will run normally until it grows to be larger than the amount of memory available to a user process. It is then locked in core memory and not swapped until it either exits or shrinks to a size less than or equal to the size of a default user process.

The removal of the process size restriction during execution of *runbig* will be preserved during an *exec*(S) system call, but not for a *fork*(S) system call.

See Also

exec(S), fork(S)

Notes

Running programs greater than the default process size, and therefore, possibly greater than the size of the disk swap area, may severely impact system performance.

runbig has no effect on systems whose memory size is much less than the size of the disk swap area.

Name

sddate – Prints and sets backup dates.

Syntax

sddate [name lev date]

Description

If no argument is given, the contents of the backup date file **/etc/ddate** are printed. The backup date file is maintained by *backup*(C) and contains the date of the most recent backup for each backup level for each filesystem.

If arguments are given, an entry is replaced or made in **/etc/ddate.** *name* is the last component of the device pathname, *lev* is the backup level number (from 0 to 9), and *date* is a time in the form taken by *date*(C):

mmddhhmm[yy]

Where the first *mm* is a two-digit month in the range 01-12, *dd* is a two-digit day of the month, *hh* is a two-digit military hour from 00-23, and the final *mm* is a two-digit minute from 00-59. An optional two-digit year, *yy,* is presumed to be an offset from the year 1900, i.e., 19*yy*.

Some sites may wish to back up file systems by copying them verbatim to backup media. *sddate* could be used to make a "level 0" entry in **/etc/ddate**, which would then allow incremental backups.

For example:

```
sddate rhd0 5 10081520
```

makes an **/etc/ddate** entry showing a level 5 backup of **/dev/rhd0** on October 8, at 3:20 PM.

Files

/etc/ddate

See Also

backup(C), dump(C), date(C)

Diagnostics

bad conversion If the date set is syntactically incorrect.

Name

sdiff – Compares files side-by-side.

Syntax

sdiff [options ...] file1 file2

Description

sdiff uses the output of *diff*(C) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x    |    y
a         a
b    <
c    <
d         d
     >    c
```

The following options exist:

–w *n* Uses the next argument, *n*, as the width of the output line. The default line length is 130 characters.

–l Only prints the left side of any lines that are identical.

–s Does not print identical lines.

–o *output* Uses the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(C), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a **%** and waits for one of the following user-typed commands:

l Appends the left column to the output file

r Appends the right column to the output file

s Turns on silent mode; does not print identical lines

v Turns off silent mode

e l
Calls the editor with the left column

e r
Calls the editor with the right column

e b
Calls the editor with the concatenation of left and right

e Calls the editor with a zero length file

q Exits from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

See Also

diff(C), ed(C)

Name

sed – Invokes the stream editor.

Syntax

sed [**−n**] [**−e** script] [**−f** sfile] [files]

Description

sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **−f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **−e** option and no **−f** options, the flag **−e** may be omitted. The **−n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a D command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **−n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a **/***regular expression***/** in the style of *ed*(C) modified as follows:

- In a context address, the construction *?regular expression?*, where *?* is any character, is identical to **/***regular expression***/**. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

- The escape sequence **\n** matches a newline *embedded* in the pattern space.

- A period **.** matches any character except the *terminal* newline of the pattern space.

- A command line with no addresses selects every pattern space.

- A command line with one address selects each pattern space that matches the address.

- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter, the process is repeated, looking again for the first address.

Editing commands can be applied only to nonselected pattern spaces by use of the negation function ! (below).

In the following list of functions, the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with backslashes to hide the newlines. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1)**a**
text Appends *text*, placing it on the output before reading the next input line.

(2)**b** *label* Branches to the **:** command bearing the *label*. If *label* is empty, branches to the end of the script.

(2)**c**
text Changes text by deleting the pattern space and then appending *text*. With 0 or 1 address or at the end of a 2-address range, places *text* on the output and starts the next cycle.

(2)**d** Deletes the pattern space and starts the next cycle.

(2)**D** Deletes the initial segment of the pattern space through the first newline and starts the next cycle.

(2)**g** Replaces the contents of the pattern space with the contents of the hold space.

(2)**G** Appends the contents of the hold space to the pattern space.

(2)**h** Replaces the contents of the hold space with the contents of the pattern space.

(2) **H** Appends the contents of the pattern space to the hold space.

(1)**i**
text Insert. Places *text* on the standard output.

(2) **l** Lists the pattern space on the standard output with nonprinting characters spelled in two-digit ASCII and long lines folded.

(2) n Copies the pattern space to the standard output. Replaces the pattern space with the next line of input.

(2) **N** Appends the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2) **p** Prints (copies) the pattern space on the standard output.

(2) **P** Prints (copies) the initial segment of the pattern space through the first newline to the standard output.

(1) **q** Quits *sed* by branching to the end of the script. No new cycle is started.

(2) r *rfile* Reads the contents of *rfile* and places them on the output before reading the next input line.

(2) s/*regular expression*/*replacement*/*flags*
Substitutes the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of **/**. For a more detailed description, see *ed*(C). *Flags* is zero or more of:

g Globally substitutes for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Prints the pattern space if a replacement was made.

w *wfile*
Writes the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Branches to the colon (**:**) command bearing *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t** command. If *label* is empty, **t** branches to the end of the script.

(2) w *wfile* Writes the pattern space to *wfile*.

(2) **x** Exchanges the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/
Replaces all occurrences of characters in *string1* with the corresponding characters in *string2.* The lengths of *string1* and *string2* must be equal.

(2)**!** *function*
Applies the *function* (or group, if *function* is {) only to lines *not* selected by the address(es).

(0) **:** *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1) **=** Places the current line number on the standard output as a line.

(2) **{** Executes the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

See Also

awk(C), ed(C), grep(C)

Notes

This command is explained in detail in *XENIX Text Processing Guide.*

Name

setcolor – Set screen color.

Syntax

setcolor -[nbrgopc] argument [argument]

Description

setcolor allows the user to set the screen color on a color screen. Both foreground and background colors can be set independently in a range of 16 colors. **setcolor** can also set the reverse video and graphics character colors. **setcolor** with no arguments produces a usage message that displays all available colors, then resets the screen to its previous state.

For example, the following strings are possible colors.

blue	magenta	brown	black
lt_blue	lt_magenta	yellow	gray
cyan	white	green	red
lt_cyan	hi_white	lt_green	lt_red

The following flags are available. In the arguments below, "color" is taken from the above list.

–n
Set the screen to "normal" white characters on black background.

color [color]
Set the foreground to the first color. Sets background to second color if a second color choice is specified.

–b color
Set the background to the specified color.

–r color color
Set the foreground reverse video characters to the first color. Set reverse video characters' background to second color.

–g color color
Set the foreground graphics characters to the first color. Set graphics characters' background to second color.

–o
Set the color of the screen b order (overscan region).

−p pitch duration
Set the pitch and duration of the bell. Pitch is the period in microseconds, and duration is measured in fifths of a second.

−c first last
Set the first and last scan lines of the cursor. (For more information see *screen*(HW).)

Notes

The ability of *setcolor* to set any of these described functions is ultimately dependent on the ability of devices to support them. *setcolor* emits an escape sequence that may or may not have an effect on monochrome devices.

Occasionally changing the screen color can help prolong the life of your monitor.

See Also

screen(HW)

Name

setkey – Assigns the function keys.

Syntax

setkey *keynum string*

Description

The **setkey** command assigns the given ANSI *string* to be the output of the computer function key given by *keynum*. For example, the command:

setkey 1 date

assigns the string "date" as the output of function key 1. The *string* can contain control characters, such as a newline character, and should be quoted to protect it from processing by the shell. For example, the command:

setkey 2 "pwd ; lc\n"

assigns the command sequence "pwd ; lc" to function key 2. Notice how the newline character is embedded in the quoted string. This causes the commands to be carried out when function key 2 is pressed. Otherwise, the Enter key would have to be pressed after pressing the function key, as in the previous example.

Files

/bin/setkey

See Also

keyboard(HW)

Notes

setkey works only on the *console* keyboard.

The string mapping table is where the function keys are defined. It is an array of 256 bytes (typedef *strmap_t*) where null terminated strings can be put to redefine the function keys. The first null terminated string is assigned to the first string key and the second string key and so on.

There is no limit on the length of any particular string as long as the whole table does not exceed 256 bytes, including nulls. Strings can be made null by the introduction of extra null characters.

Name

setmnt – Establishes /etc/mnttab table.

Syntax

/etc/setmnt

Description

setmnt creates the **/etc/mnttab** table (see *mnttab*(F)), which is needed for both the *mount*(C) and *umount*(C) commands. *setmnt* reads the standard input and creates a *mnttab* entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the file system's *special file* (e.g., "hd0") and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the *mnttab*(F) entry.

Files

/etc/mnttab

See Also

mnttab(F)

Notes

If *filesys* or *node* are longer than 128 characters, errors can occur.

setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

setmnt is normally invoked by **/etc/rc** when the system boots up.

Name

settime – Changes the access and modification dates of files.

Syntax

settime mmddhhmm [yy] [**–f** fname] name ...

Description

Sets the access and modification dates for one or more files. The dates are set to the specified date, or to the access and modification dates of the file specified via **–f**. Exactly one of these methods must be used to specify the new date(s). The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last two digits of the year and is optional. For example:

settime 1008004583 ralph pete

sets the access and modification dates of files *ralph* and *pete* to Oct 8, 12:45 AM, 1983. Another example:

settime –f ralph john

This sets the access and modification dates of the file *john* to those of the file *ralph*.

Notes

Use of *touch* in place of *settime* is encouraged.

Name

sh – Invokes the shell command interpreter.

Syntax

sh [**–ceiknrstuvx**] [args]

Description

The shell is the standard command programming language that executes commands read from a terminal or a file. See *Invocation* below for the meaning of arguments to the shell.

Commands

A *simple–command* is a sequence of nonblank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(S)). The *value* of a simple–command is its exit status if it terminates normally, or (octal) 1000+*status* if it terminates abnormally (i.e., if the failure produces a core file). See *signal*(S) for a list of status values.

A *pipeline* is a sequence of one or more *commands* separated by a vertical bar (**|**). (The caret (^), also has the same effect.) The standard output of each command but the last is connected by a *pipe*(S) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;** or **&**. Of these four 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 **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (nonzero) exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following commands. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command:

for *name* [**in** *word* ...]
do
list
done

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in**
[*pattern* [|*pattern*] ...) *list*
;;]
esac

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see *Filename Generation* below).

if *list* **then**
list
[**elif** *list* **then**
list]
[**else** *list*]
fi

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list*
do
list
done

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(list)
Executes *list* in a subshell.

{*list*;}
list is simply executed.

name () {*list;*}
Define a function which is referenced by *name*. The body of functions is the *list* of commands between { and }. Execution of functions is described later (see *Execution*.)

The following words are recognized only as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (\`\`) may be used as part or all of a word; trailing newlines are removed.

Parameter Substitution

The character $ is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

name =*value* [*name* =*value*] . . .

Pattern-matching is not performed on *value*.

${*parameter*}
A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *, @, #, ?, –, $, and !. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is * or @, then all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:−*word*}
If *parameter* is set and is not a null argument, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
If *parameter* is not set or is null, then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
If *parameter* is set and is not a null argument, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}
If *parameter* is set and is not a null argument, substitute *word*; otherwise substitute nothing. In the above, *word* is not evaluated unless it is to be used as the substituted string, so that in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal

− Flags supplied to the shell on invocation or by the **set** command

? The decimal value returned by the last synchronously executed command

$ The process number of this shell

! The process number of the last background command invoked

The following parameters are used by the shell:

CDPATH
Defines search path for the *cd* command. See the section *Special Commands*, "cd".

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

PATH
The search path for commands (see *Execution* below)

MAIL
If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by **%** and a message that will be printed when the modification time changes. The default message is *you have mail.*

PS1
Primary prompt string, by default "**$** "

PS2
Secondary prompt string, by default "**>** "

IFS
Internal field separators, normally **space**, **tab**, and **newline**

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(C) and *accton*(C) can be used to anaylze the data collected.

SHELL
When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an 'r' in the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, and **IFS**, while **HOME** and **MAIL** are not set at all by the shell (although **HOME** *is* set by *login*(M)).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Filename Generation

Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character . at the start of a filename or immediately following a /, as well as the character / itself, must be matched explicitly. These characters and their matching patterns are:

* Matches any string, including the null string.

? Matches any single character.

[...]
Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening bracket ([) is an exclamation mark (!), then any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > newline space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \\. The pair **\\newline** is ignored. All characters enclosed between a pair of single quotation marks (´´), except a single quotation mark, are quoted. Inside double quotation marks (""), parameter and command substitution occurs and \\ quotes the characters \\, `, ", and $. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2"** ...

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Spelling Checker

When using *cd*(C) the shell checks spelling. For example, if you change to a different directory using *cd* and misspell the directory name, the shell repsonds with an alternative spelling of an existing directory. Enter "y" and press RETURN 9 or just press RETURN) to change to the offered directory. If the offered spelling is incorrect, enter "n", then retype the command line. In this example the sh(C) response is boldfaced:

$ cd /usr/spol/uucp
cd /usr/spool/uucp?y
ok

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command*. They are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<word Use file *word* as standard input (file descriptor 0).

>word Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists, output is appended to it (by first seeking the end-of-file); otherwise, the file is created.

<<[−]word The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\newline** is ignored, and **** must be used to quote the characters ****, **$**, **\`**, and the first character of *word*. If **−** is appended to << , all leading tabs are stripped from *word* and from the document.

<&digit The standard input is duplicated from file descriptor *digit* (see *dup*(S)). Similarly for the standard output using >.

<&− The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by **&**, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see *environ*(M)) is a list of name–value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affect the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is composed of any unmodified name–value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple–command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

TERM=450 cmd args

and

(export TERM; TERM=450; cmd args)

are equivalent (as far as the above execution of *cmd* is concerned).

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name.

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11. See the **trap** command below.

Execution

Each time a command is executed, the above substitutions are carried out. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1, $2, ...** are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(S).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a **/**, then 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 an **a.out** file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Shell procedures are often used by users running the **csh**. However, if the first character of the procedure is a # (comment character), **csh** assumes the procedure is a **csh** script, and invokes */bin/csh* to execute it. Always start **sh** procedures with some other character if **csh** users are to run the procedure at any time. This invokes the standard shell */bin/sh* .

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re–determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash −r** command is executed (see below).

Special Commands

Input/output redirection is permitted for these commands:

: No effect; the command does nothing. A zero exit code is returned.

. *file*
Reads and executes commands from *file* and returns. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exits from the enclosing **for** or **while** loop, if any. If *n* is specified, it breaks *n* levels.

continue [*n*]
Resumes the next iteration of the enclosing **for** or **while** loop. If *n* is specified, it resumes at the *n*-th enclosing loop.

cd [*arg*]
Changes the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, the search path is not used. Otherwise, each directory in the path is searched for *arg*.

If the shell is reading its commands from a terminal, and the specified directory does not exist (or some component cannot be searched), spelling correction is applied to each component of *directory*, in a search for the "correct" name. The shell then asks whether or not to try and change directory to the corrected directory name; an answer of *n* means "no", and anything else is taken as "yes".

echo [arg]
Writes arguments separated by blanks and terminated by a newline on the standard output. Arguments may be enclosed in quotes. Quotes are required so that the shell correctly interprets these special escape sequences:

\b Backspace

\c Prints line without newline.

\f Form feed

\n Newline

\r Carriage return

\t Tab

\v Vertical tab

**** Backslash

*****n* The 8-bit character whose ASCII code is the 1, 2 or 3-digit octal number *n* must start with a zero

eval [*arg* . . .]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An end-of-file will also cause the shell to exit.

export [*name* ...]
The given *name*s are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, a list of all names that are exported in this shell is printed.

hash [**-r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* ...

pwd
Print the current working directory. See *pwd*(C) for usage and description.

read [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**-eknuvx** [*arg* ...]]

−e
If the shell is noninteractive, exits immediately if a command exits with a nonzero exit status.

−f Disables file name generation.

−h
Locates and remembers fuction commands as functions are defined (function commands are normally located when the function is executed).

−k
Places all keyword arguments in the environment for a command, not just those that precede the command name.

−n
Reads commands but does not execute them.

−u
Treats unset variables as an error when substituting.

−v
Prints shell input lines as they are read.

−x
Prints commands and their arguments as they are executed. Although this flag is passed to subshells, it does not enable tracing in those subshells.

− −
Does not change any of the flags; useful in setting $1 to −.

Using **+** rather than **−** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**. The remaining arguments are positional parameters and are assigned, in order, to **$1, $2, . . .** If no arguments are given, the values of all names are printed.

shift
The positional parameters from **$2** . . . are renamed **$1** . . .

test
Evaluates conditional expressions. See *test*(C) for usage and description.

times
Prints the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] . . .
arg is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. The highest signal number allowed is 16. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by

the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [[**−f**] *n*]
imposes a size limit of *n* blocks on files.

−f imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). Any user may decrease the file size limit, but only the super−user (root) can increase the limit. With no argument, the current limit is printed.

If no option is given and a number is specified, **−f** is assumed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

umask [*ooo*]
The user file−creation mask is set to the octal number *ooo* where *o* is an octal digit (see *umask*(C)). If *ooo* is omitted, the current value of the mask is printed.

wait [*n*]
Waits for the specified process to terminate, and reports the termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is always 0.

Invocation

If the shell is invoked through *exec*(S) and the first character of argument 0 is **−**, commands are initially read from **/etc/profile** and then from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; note that unless the **−c** or **−s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

−c *string* If the **−c** flag is present, commands are read from *string*.

−s If the **−s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.

−t If the **−t** flag is present, a single command is read and executed, and the shell exits. This flag is intended for use by C programs only and is not useful interactively.

−i If the **−i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case, TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

−r If the **−r** flag is present, the shell is a restricted shell (see *rsh*(C)).

The remaining flags and arguments are described under the **set** command above.

Exit Status

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used noninteractively, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed. See the **exit** command above.

Files

/etc/profile	system default *profile* if none is present
$HOME/.profile	read by login shell at login
/tmp/sh*	temporary file for <<
/dev/null	source of empty file

See Also

cd(C), env(C), login(M), newgrp(C), rsh(C), test(C), umask(C), dup(S), exec(S), fork(S), pipe(S), signal(S), umask(S), wait(S), a.out(F), profile(M), environ(M)

Notes

The command **readonly** (without arguments) produces the same output as the command **export**.

If << is used to provide standard input to an asynchronous process invoked by **&**, the shell gets mixed up about naming the input document; a garbage file **/tmp/sh*** is created and the shell complains about not being able to find that file by another name.

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Name

shV – Invokes the shell command interpreter.

Syntax

shV [**−ceiknrstuvx**] [args]

Description

The shell is the standard command programming language that executes commands read from a terminal or a file. See *Invocation* below for the meaning of arguments to the shell.

Commands

A *simple–command* is a sequence of nonblank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(S)). The *value* of a simple–command is its exit status if it terminates normally, or (octal) 1000+*status* if it terminates abnormally (i.e., if the failure produces a core file). See *signal*(S) for a list of status values.

A *pipeline* is a sequence of one or more *commands* separated by a vertical bar (|). (The caret (^), also has the same effect.) The standard output of each command but the last is connected by a *pipe*(S) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;** or **&**. Of these four 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 **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (nonzero) exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple–command or one of the following commands. Unless otherwise stated, the value returned by a command is that of the last simple–command executed in the command:

for *name* [**in** *word* . . .]
do
list
done

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in**
[*pattern* [|*pattern*] . . .) *list*
;;]
esac

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see *Filename Generation* below).

if *list* **then**
list
[**elif** *list* **then**
list]
[**else** *list*]
fi

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list*
do
list
done

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(list)
Executes *list* in a subshell.

{*list*;}
  list is simply executed.

name () {*list;*}
  Define a function which is referenced by *name*. The body of functions is the *list* of commands between { and }. Execution of functions is described later (see *Execution*.)

The following words are recognized only as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing newlines are removed.

Parameter Substitution

The character $ is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

  name=*value* [*name*=*value*] . . .

Pattern–matching is not performed on *value*.

${*parameter*}
  A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *, @, #, ?, –, $, and !. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is * or @, then all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:−*word*}
If *parameter* is set and is not a null argument, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
If *parameter* is not set or is null, then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
If *parameter* is set and is not a null argument, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}
If *parameter* is set and is not a null argument, substitute *word*; otherwise substitute nothing. In the above, *word* is not evaluated unless it is to be used as the substituted string, so that in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal

− Flags supplied to the shell on invocation or by the **set** command

? The decimal value returned by the last synchronously executed command

$ The process number of this shell

! The process number of the last background command invoked

The following parameters are used by the shell:

CDPATH
Defines search path for the *cd* command. See the section *Special Commands*, "cd".

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

PATH
The search path for commands (see *Execution* below)

MAIL
If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail.*

PS1
Primary prompt string, by default "$ "

PS2
Secondary prompt string, by default "> "

IFS
Internal field separators, normally **space**, **tab**, and **newline**

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(C) and *accton*(C) can be used to anaylze the data collected.

SHELL
When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an 'r' in the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH, PS1, PS2,** and **IFS,** while **HOME** and **MAIL** are not set at all by the shell (although **HOME** *is* set by *login* (M)).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Filename Generation

Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character . at the start of a filename or immediately following a /, as well as the character / itself, must be matched explicitly. These characters and their matching patterns are:

* Matches any string, including the null string.

? Matches any single character.

[...]
Matches any one of the enclosed characters. A pair of characters separated by **-** matches any character lexically between the pair, inclusive. If the first character following the opening bracket ([) is an exclamation mark (!), then any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > newline space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a ****. The pair **\newline** is ignored. All characters enclosed between a pair of single quotation marks (´´), except a single quotation mark, are quoted. Inside double quotation marks (""), parameter and command substitution occurs and \ quotes the characters \, `, ", and $. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2"** ...

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Spelling Checker

When using *cd*(C) the shell checks spelling. For example, if you change to a different directory using *cd* and misspell the directory name, the shell repsonds with an alternative spelling of an existing directory. Enter "y" and press RETURN 9 or just press RETURN) to change to the offered directory. If the offered spelling is incorrect, enter "n", then retype the command line. In this example the **sh**(C) response is boldfaced:

```
$ cd /usr/spol/uucp
cd /usr/spool/uucp?y
ok
```

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command*. They are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word* — Use file *word* as standard input (file descriptor 0).

>*word* — Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.

>>*word* — Use file *word* as standard output. If the file exists, output is appended to it (by first seeking the end-of-file); otherwise, the file is created.

<<[–]*word* — The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\newline** is ignored, and **** must be used to quote the characters ****, $, `, and the first character of *word*. If – is appended to << , all leading tabs are stripped from *word* and from the document.

<&*digit* — The standard input is duplicated from file descriptor *digit* (see *dup*(S)). Similarly for the standard output using >.

<&– — The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by **&**, the default standard input for the command is the empty file **/dev/null.** Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see *environ*(M)) is a list of name–value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affect the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is composed of any unmodified name–value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple–command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

TERM=450 cmd args

and

(export TERM; TERM=450; cmd args)

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name.

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11. See the **trap** command below.

Execution

Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec*(S).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a **/**, then 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 an **a.out** file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Shell procedures are often used by users running the **csh**. However, if the first character of the procedure is a # (comment character), **csh** assumes the procedure is a **csh** script, and invokes */bin/csh* to execute it. Always start **sh** procedures with some other character if **csh** users are to run the procedure at any time. This invokes the standard shell */bin/sh* .

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is permitted for these commands:

: No effect; the command does nothing. A zero exit code is returned.

. *file*
Reads and executes commands from *file* and returns. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exits from the enclosing **for** or **while** loop, if any. If *n* is specified, it breaks *n* levels.

continue [*n*]
Resumes the next iteration of the enclosing **for** or **while** loop. If *n* is specified, it resumes at the *n*–th enclosing loop.

cd [*arg*]
Changes the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, the search path is not used. Otherwise, each directory in the path is searched for *arg*.

If the shell is reading its commands from a terminal, and the specified directory does not exist (or some component cannot be searched), spelling correction is applied to each component of *directory*, in a search for the "correct" name. The shell then asks whether or not to try and change directory to the corrected directory name; an answer of *n* means "no", and anything else is taken as "yes".

echo [arg]
Writes arguments separated by blanks and terminated by a newline on the standard output. Arguments may be enclosed in quotes. Quotes are required so that the shell correctly interprets these special escape sequences:

\b Backspace

\c Prints line without newline.

\f Form feed

\n Newline

\r Carriage return

\t Tab

\v Vertical tab

**** Backslash

*****n* The 8-bit character whose ASCII code is the 1, 2 or 3-digit octal number *n* must start with a zero

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An end-of-file will also cause the shell to exit.

export [*name* ...]
The given *names* are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, a list of all names that are exported in this shell is printed.

hash [**-r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* ...

pwd
Print the current working directory. See *pwd*(C) for usage and description.

read [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**-eknuvx** [*arg* ...]]

-e
If the shell is noninteractive, exits immediately if a command exits with a nonzero exit status.

-f Disables file name generation.

−h
: Locates and remembers fuction commands as functions are defined (function commands are normally located when the function is executed).

−k
: Places all keyword arguments in the environment for a command, not just those that precede the command name.

−n
: Reads commands but does not execute them.

−u
: Treats unset variables as an error when substituting.

−v
: Prints shell input lines as they are read.

−x
: Prints commands and their arguments as they are executed. Although this flag is passed to subshells, it does not enable tracing in those subshells.

− −
: Does not change any of the flags; useful in setting $1 to −.

Using **+** rather than **−** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given, the values of all names are printed.

shift
: The positional parameters from **$2** ... are renamed **$1** ...

test
: Evaluates conditional expressions. See *test*(C) for usage and description.

times
: Prints the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
: *arg* is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. The highest signal number allowed is 16. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [[**−f**] *n*]
imposes a size limit of *n* blocks on files.

−f imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). Any user may decrease the file size limit, but only the super−user (root) can increase the limit. With no argument, the current limit is printed.

If no option is given and a number is specified, **−f** is assumed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

umask [*ooo*]
The user file−creation mask is set to the octal number *ooo* where *o* is an octal digit (see *umask*(C)). If *ooo* is omitted, the current value of the mask is printed.

wait [*n*]
Waits for the specified process to terminate, and reports the termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is always 0.

Invocation

If the shell is invoked through *exec*(S) and the first character of argument 0 is **−**, commands are initially read from **/etc/profile** and then from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh.** The flags below are interpreted by the shell on invocation only; note that unless the **−c** or **−s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

−c *string* If the **−c** flag is present, commands are read from *string*.

−s If the **−s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.

−t If the **−t** flag is present, a single command is read and executed, and the shell exits. This flag is intended for use by C programs only and is not useful interactively.

−i If the **−i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive.* In this case, TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

−r If the **−r** flag is present, the shell is a restricted shell (see *rsh*(C)).

The remaining flags and arguments are described under the **set** command above.

Exit Status

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used noninteractively, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed. See the **exit** command above.

Files

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

See Also

cd(C), env(C), login(M), newgrp(C), rsh(C), test(C), umask(C), dup(S), exec(S), fork(S), pipe(S), signal(S), umask(S), wait(S), a.out(F), profile(M), environ(M)

Notes

The command **readonly** (without arguments) produces the same output as the command **export.**

If << is used to provide standard input to an asynchronous process invoked by **&**, the shell gets mixed up about naming the input document; a garbage file **/tmp/sh*** is created and the shell complains about not being able to find that file by another name.

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Name

shl – Shell layer manager

Syntax

shl

Description

shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers,* using the commands described below.

The *current layer* is the layer that can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the *stty*(C) option **loblk** may be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *shl* has its own prompt, >>>, to help distinguish it from a layer.

A *layer* is a shell that has been bound to a virtual tty device (**/dev/sxt???**). The virtual device can be manipulated like a real tty device using *stty*(C) and *ioctl*(S). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or new-line. Only the first eight characters are significant. The *names* (1) through (7) cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

create *name*
Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where # is the last digit of the virtual device bound to the layer. The shell prompt variable **PS1** is set to the name of the layer followed by a space, or, if superuser, the name

followed by a sharp (#) and a space. A maximum of seven layers can be created.

block *name* [*name* ...]
For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **loblk** within the layer.

delete *name* *name* ...
For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(2)).

help (or **?**)
Print the syntax of the *shl* commands.

layers **−l** *name* ...
For each *name*, list the layer name and its process group. The **−l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume *name*
Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle
Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **loblk** within the layer.

quit
Exit *shl*. All layers are sent the SIGHUP signal.

name
Make the layer referenced by *name* the current layer.

Files

/dev/sxt???	Virtual tty devices
$SHELL	Variable containing path name of the shell to use (default is /bin/sh).

See Also

ioctl(S), sh(C), signal(S), stty(C), sxt(M)

Note

It is inadvisable to kill *shl.*

Name

sleep – Suspends execution for an interval.

Syntax

sleep time

Description

sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

See Also

alarm(S), sleep(S)

Notes

It is recommended that *time* be less than 65536 seconds.

Name

shutdown – Terminates all processing.

Syntax

/etc/slmtdown [time] [su]

Description

shutdown is part of the XENIX operation procedures. Its primary function is to terminate all currently running processes in an orderly and cautious manner. The *time* argument is the number of minutes before a shutdown will occur. The optional *su* argument lets the user go single-user, without completely shutting down the system. However, the system is shut down for multi-user use. *shutdown* goes through the following steps. First, all users logged on the system are notified to log off the system by a broadcasted message. All file system super-blocks are updated before the system is stopped (see *sync*(C)). This must be done before rebooting the system, to insure file system integrity.

See Also

sync(C), umount(C), wall(C)

Diagnostics

The most common error diagnostic that will occur is *device busy*. This diagnostic appears when a particular file system could not be unmounted. See *umount*(C).

Notes

Once *shutdown* has been invoked, it must be allowed to run to completion and must *not* be interrupted by pressing BREAK or DEL.

shutdown locks the hard disk heads.

Name

sort – Sorts and merges files.

Syntax

sort [**-cmu**] [**-o**output] [**-y**kmem] [**-z**recsz] [**-dfiMnr**] [**-btx**] [**+**pos1] [**-**pos2] [files]

Description

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if **-** is used as a file name or if no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in the machine's collating sequence.

The following options alter the default behavior:

-c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-m Merge only, the input files are already sorted.

-u Unique: suppress all but one in each set of lines having equal keys.

-o*output*
The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **-o** and *output*.

-y*kmem*
The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **-y0** is guaranteed to start with minimum memory. By convention, **-y** (with no argument) starts with maximum memory.

- ***zrecsz***

The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **-c** or **-m** options, a popular system default size will be used. Lines longer than the buffer size 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.

The following options override the default ordering rules.

-d "Dictionary" order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

-f Fold lower case letters into upper case.

-i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

-M Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **-M** option implies the **-b** option (see below).

-n An initial numeric string, consisting of optional blanks, an optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

-r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation **+***pos1* **-***pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing -*pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field (a minimal sequence of characters followed by a field separator or a newline). By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

-t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

-b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first **+***pos1* argument, it will be applied to all **+***pos1* arguments. Otherwise, the **b** flag may be attached independently to each **+***pos1* or **-***pos2* argument (see below).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags b, **d**, **f**, **i**, **n**, or **r**. A starting position specified by **+***m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect, *n* is counted from the first non-blank in the *m*+1st field; **+***m*.0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by **-***m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect, *n* is counted from the last leading blank in the *m*+1st field; **-***m*.1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

Examples

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

sort -r +1.0b -1.1b infile1 infile2

Print the password file (*passwd*(M)) sorted by the numeric user ID (the third colon-separated field):

sort -t: +2n -3 /etc/passwd

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

sort -um +2 -3 infile

Files

/usr/tmp/stm???

See Also

comm(C), join(C), uniq(C)

Diagnostics

Comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorders discovered under the **-c** option. When the last line of an input file is missing a **newline** character, *sort* appends one, prints a warning message, and continues.

Name

split – Splits a file into pieces.

Syntax

split [–*n*] [file [name]]

Description

split reads *file* and writes it in as many *n*-line pieces as necessary (default 1000), onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if a dash (**–**) is given instead, the standard input file is used.

See Also

bfs(C), csplit(C)

Name

stty – Sets the options for a terminal.

Syntax

stty [**−a**] [**−g**] [options]

Description

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options. With the **−a** option, *stty* reports all of the option settings; with the **−g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first four groups may be found in *termio*(M). *options* in the last group are implemented using *options* in the previous groups. Refer to *stty*(HW) for hardware specific information that describes control modes for the video monitor and other display devices.

Common Control Modes

parenb (−parenb)
Enables (disables) parity generation and detection.

parodd (−parodd)
Selects odd (even) parity.

cs5 cs6 cs7 cs8
Selects character size (see *tty*(**M**)).

0 Hangs up phone line immediately.

50 75 110 134 150 200 300 600
1200 1800 2400 4800 9600 exta
Sets terminal baud rate to the number given, if possible.

hupcl (−hnpcl)
Hangs up (does not hang up) phone connection on last close.

hup (−hup)
Same as **hupcl (−hupcl)**.

cstopb (−cstopb)
Uses two(one) stop bits per character.

cread (−cread)
Enables (disables) the receiver.

clocal (−clocal)
Assumes a line without (with) modem control.

Input Modes

ignbrk (−ignbrk)
Ignores (does not ignore) break on input.

brkint (−brkint)
Signals (does not signal) INTERRUPT on break.

ignpar (−ignpar)
Ignores (does not ignore) parity errors.

parmrk (−parmrk)
Marks (does not mark) parity errors (see *tty*(M)).

inpck (−inpck)
Enables (disables) input parity checking.

istrip (−istrip)
Strips (does not strip) input characters to 7 bits.

inlcr (−inlcr)
Maps (does not map) NL to CR on input.

igncr (−igncr)
Ignores (does not ignore) CR on input.

icrnl (−icrnl)
Maps (does not map) CR to NL on input.

iuclc (−iuclc)
Maps (does not map) uppercase alphabetics to lowercase on input.

ixon (−ixon)
Enables (disables) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (−ixany)
Allows any character (only DC1) to restart output.

ixoff (−ixoff)
Requests that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (−opost)
Post-processes output (does not post-process output; ignores all other output modes).

olcuc (−olcuc)
Maps (does not map) lowercase alphabetics to uppercase on output.

onlcr (−onlcr)
Maps (does not map) NL to CR-NL on output.

ocrnl (−ocrnl)
Maps (does not map) CR to NL on output.

onocr (−onocr)
Does not (does) output CRs at column zero.

onlret (−onlret)
On the terminal NL performs (does not perform) the CR function.

ofill (−ofill)
Uses fill characters (use timing) for delays.

ofdel (−ofdel)
Fill characters are DELETEs (NULs).

cr0 cr1 cr2 cr3
Selects style of delay for RETURNs (see *tty*(M)).

nl0 nl1
Selects style of delay for LINEFEEDs (see *tty*(M)).

tab0 tab1 tab2 tab3
Selects style of delay for horizontal TABs (see *tty*(M)).

bs0 bs1
Selects style of delay for BACKSPACEs (see *tty*(M)).

ff0 ff1
Selects style of delay for FORMFEEDs (see *tty*(M)).

vt0 vt1
Selects style of delay for Vertical TABs (see *tty*(M)).

Local Modes

isig (−isig)
Enables (disables) the checking of characters against the special control characters INTERRUPT and QUIT.

icanon (−icanon)
Enables (disables) canonical input (ERASE and KILL processing).

xcase (−xcase)
Canonical (unprocessed) upper/lowercase presentation.

echo (−echo)
Echoes back (does not echo back) every character typed.

echoe (−echoe)
Echoes (does not echo) ERASE character as a SPACEBAR string. Note: this mode will erase the ERASE character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, TABs, and BACKSPACEs.

echok (−echok)
Echoes (does not echo) NL after KILL character.

lfkc (−lfkc)
The same as **echok (−echok)**; obsolete.

echonl (−echonl)
Echoes (does not echo) NL.

noflsh (−noflsh)
Disables (enables) flush after INTERRUPT or QUIT.

stwrap (-stwrap)
Disables (enables) truncation of lines longer than 79 characters on a synchronous line.

stflush (-stflush)
Enables (disables) flush on a synchronous line after every *write*(S).

stappl (-stappl)
Uses application (line) mode on a synchronous line.

Control Assignments

control-character-C
Sets *control-character* to *C*, where *control-character* is ERASE, KILL, INTERRUPT, QUIT, EOF, EOL. If *C* is preceded by a caret (^) (escaped from the shell), then the value used is the corresponding CTRL character (e.g., "^D" is a CTRL-D); "^?" is interpreted as DELETE and "^–" is interpreted as undefined.

min *i*, **time** *i* (0<*i*<127)
When **–icanon** is set, and one character has been received, read requests are not satisfied until at least **min** characters have been received or the timeout value **time** has expired and one character has been received. See *tty*(C).

line *i*
Sets the line discipline to *i* (0 < *i* < 127). There are currently no line disciplines implemented.

Combination Modes

evenp or **parity**
Enables **parenb** and **cs7.**

oddp
Enables **parenb, cs7,** and **parodd.**

–parity, –evenp, or **–oddp**
Disables **parenb,** and sets **cs8.**

raw (–raw or **cooked)**
Enables (disables) raw input and output (no ERASE, KILL, INTERRUPT, QUIT, EOT, or output post-processing).

nl (–nl)
Unsets (sets) **icrnl, onlcr.** In addition **–nl** unsets **inlcr, igncr, ocrnl,** and **onlret.**

lcase (–lcase)
Sets (unsets) **xcase, iuclc,** and **olcuc.**

LCASE (–LCASE)
Same as **lcase (–lcase).**

tabs (–tabs or **tab3)**
Preserves (expands to spaces) tabs when printing.

ek Resets ERASE and KILL characters back to normal CTRL-H and CTRL-U .

sane
Resets all modes to some reasonable values. Useful when a terminal's settings have been hopelessly scrambled.

term
Sets all modes suitable for the terminal type, TERM, where TERM is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

See Also

console(M), ioctl(S), stty(HW), tty(M), termio(M)

Notes

Many combinations of options make no sense, but no checking is performed.

Name

su – Makes the user a super-user or another user.

Syntax

su [**–**] [name [arg . . .]]

Description

su allows you to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless you are already a super-user). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file (**/bin/sh** if none is specified (see *sh*(C)). To restore normal user ID privileges, press EOF (Ctrl-D) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(C), an *arg* of the form **–c** *string* executes *string* via the shell and an arg of **–r** gives the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(C). If the first argument to *su* is a **–**, the environment is changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is **–**, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of $PATH, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **–sh** or **–su** to determine if it was invoked by *login*(M) or *su*(C), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(M) and *su*(C).

If you want to log all attempts by users to become root, create the file /etc/default/su. In this file, plase a string similar to: SULOG=/usr/adm/sulog This causes all attempts by any user to switch user id's to be recorded in the file **/usr/adm/sulog.** This can be any arbitrary filename. The *su* logfile records the original user, the UID of the su attempt, and the time of the attempt. If the attempt is successful, a plus sign (+) is placed on the line describing the attempt. A minus sign (-) indicates an unsuccessful attempt.

Examples

To become user **bin** while retaining your previously exported environment, enter:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, enter:

```
su - bin
```

To execute *command* with the temporary environment and permissions of user **bin**, enter:

su - bin -c "*command args*"

Files

/etc/passwd	The system password file
/etc/default/su	Optional file giving location of sulog.
/etc/profile	The system profile
$HOME/.profile	The user profile

See Also

env(C), environ(M), login(M), passwd(M), profile(M), sh(C)

Name

sum – Calculates checksum and counts blocks in a file.

Syntax

sum [**-r**] file

Description

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of BSIZE blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over a transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum.

See Also

cmchk(C), machine(M), wc(C)

Diagnostics

"Read error" is indistinguishable from end-of-file on most devices; check the block count.

Notes

Refer to *machine*(M) or use the *cmchk (C)* utility to determine BSIZE for your system.

Name

swapctl – Specifies additional devices for paging and swapping.

Description

This command is available only in XENIX-386. If you have XENIX-386, see your *Release Notes* for the complete version of this reference page.

Name

sync – Updates the super-block.

Syntax

sync

Description

sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to ensure file system integrity. Note that *shutdown* (C) automatically calls *sync* before shutting down the system.

See Also

sync(S)

Name

sysadmin – Performs file system backups and restores files.

Syntax

/etc/sysadmin

Description

sysadmin is a utility for performing file system backups and for restoring files from backup volumes. It can do a daily incremental backup (level 9), or a periodic full backup (level 0). It can provide a listing of the files backed up and also has facilities for restoring individual files and complete filesystems from backups.

Any supported archive medium may be used to create backups. Any filesystem may be backed up. Menus of these devices are created for each option from the files **/tmp/backup.list**, **/etc/default/archive**, and **/etc/default/filesys**.

You must be the super-user to use this program.

Files

/tmp/backup.list
/etc/default/archive
/etc/default/filesys

See Also

backup(C), dumpdir(C), mkfs(C), restore(C), archive(F), **filesys(F)**

Notes

/tmp/backup.list, **/etc/default/archive** and **/etc/default/filesys** may be edited to add devices, or to delete entries for devices that are no longer used.

Name

sysadmsh − Menu driven system administration utility

Syntax

sysadmsh

Description

sysadmsh is an easy-to-use menu interface designed to provide novice users with the tools needed for day-to-day system administration of the XENIX system.

WARNING: *sysadmsh* does not replace the XENIX documentation. It provides an overview of available system administration features and a reminder of tasks which need to be performed regularly. An understanding of the XENIX *Installation Guide*, the XENIX *Operations Guide,* and the XENIX *User's Guide* is necessary to use *sysadmsh.*

Usage

To use this utility enter:

sysadm

at the login prompt. This sets your login shell to be the *sysadmsh* menu hierarchy. You may access many useful commands and submenus, all presented in simple, descriptive terms.

Alternately, *sysadmsh* menus may also be invoked by entering:

root

at the login prompt, and then entering:

sysadmsh

at the shell prompt. Note, however, that you are logged in as a different user (not *sysadm*) and some things may behave differently.

Once you are in *sysadmsh,* on-line instructions for its use may be obtained by selecting the Help Menu.

Some *sysadmsh* options must be run from the system console device (screen F1). Some options must be run while in single user (system maintenance) mode. Check the documentation manual page referenced in the on-line help file for more information.

Files

See Also

XENIX Operations Guide
XENIX User's Guide
XENIX Installation Guide

acctcom(C), accton(C), alias(M), asktime(C), at(C), badtrk(C), checklist(F), chgrp(C), chmod(S), chown(C), configure(C) copy(C), cron(C), csh(C), custom(C), df(C), diff(C), dircmp(C), disable(C), diskcmp(C), diskcp(C), dmesg(C), dos(C), dtype(C), du(C), enable(C), fdisk(C), find(C), finger(C), fixperm(M), format(C), fsck(C), fstab(F), grpcheck(C), init(M), kill(C), login(M), lp(C), lpadmin(C), lpinit(C), lpstat(C), mail(C), mkdev(C), mkuser(C), more(C), mount(C), netutil(C), ps(C), pwadmin(C), pwcheck(C), quot(C), rmuser(C), shutdown(C), sysadmin(C), systemid(M), tar(C), ttys(M), umount(C), uuinstall(C), vi(C), wall(C), who(C), write(C)

Notes

A knowledge of *vi*(C) is assumed for file edit selections, although the SCO Lyrix® editor is used when available.

Acknowledgements

This utility takes its design from the SCO Lyrix Word Processing System.

Name

tail – Delivers the last part of a file.

Syntax

tail [±[number][**lbc**] [**−f**]] [file]

Description

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance **+***number* from the beginning, or **−***number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **−f** ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f file
```

will print the last ten lines of *file*, followed by any lines that are appended to *file* between the time *tail* is initiated and killed.

See Also

dd(C)

Notes

Tails relative to the end of the file are kept in a buffer, and thus are limited in length. Unpredictable results can occur if character special files are "tailed".

Name

tape – Magnetic tape maintenance program.

Syntax

tape command [devicefile]

Description

tape sends commands and receives status to and from the tape subsystem. *tape*(HW) lists the drives supported. The available commands are listed below.

amount
Report amount of data in current or last transfer.

erase
Erase tape cartridge. Also retensions.

reset
Reset tape controller and tape drive. Clears error conditions and returns tape subsystem to power-up state.

reten
Retension tape cartridge. Should be used periodically to remedy slack tape problems. Tape slack can cause an unusually large number of tape errors.

rewind
Rewind to beginning of tape.

rfm
Wind tape forward to the next file mark.

status
Return tape status.

wfm
Write a file mark at the current tape position.

The **amount, reset** and **status** commands can be used while the tape is busy with other operations. The **erase, reten, rewind, rfm** and **wfm** commands wait until the current command has been completed before proceeding.

When you are using the non-rewinding tape device or the *tape* commands **rfm** and **wfm**, the tape drive light remains on after the command has been completed, indicating that more operations may be performed on the tape. The *tape* **rewind** command may be used to clear this condition.

For more information on devicefiles, (listed below), see the *tape* (HW) manual page.

Files

/dev/rct0
/dev/nrct0
/dev/rct2
/dev/nrct2
/dev/rctmini

See Also

backup(C), cpio(C), dd(C), dump(C), restore(C), tape(HW), tar(C)

Name

tar – Archives files.

Syntax

tar [key] [files]

Description

tar saves and restores files to and from an archive medium, which is typically a storage device such as floppy disk or tape, or a regular file. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Valid function letters are **c**, **t**, **x**, and **e**. Other arguments to the command are *files* (or directory names) specifying which files are to be backed up or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory. The **r** and **u** option cannot be used with tape devices.

The function portion of the key is specified by one of the following letters:

r The named *files* are written to the end of the archive. The **c** function implies this function.

x The named *files* are extracted from the archive. If a named file matches a directory whose contents had been written onto the archive, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire contents of the archive are extracted. Note that if several files with the same name are on the archive, the last one overwrites all earlier ones.

t The names of the specified files are listed each time that they occur on the archive. If no *files* argument is given, all the names on the archive are listed.

u The named *files* are added to the archive if they are not already there, or if they have been modified since last written on that archive.

c Creates a new archive; writing begins at the beginning of the archive, instead of after the last file. This command implies the **r** function.

The following characters may be used in addition to the letter that selects the desired function:

0,...,7 This modifier selects the drive on which the archive is mounted. The default is found in the file **/etc/default/tar**.

v Normally, *tar* does its work silently. The **v** (verbose) option causes it to display the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the archive entries than just the name.

w Causes *tar* to display the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with y is given, the action is performed. Any other input means "no".

f Causes *tar* to use the next argument as the name of the archive instead of the default device listed in **/etc/default/tar**. If the name of the file is a dash (**−**), *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *tar* can also be used to move hierarchies with the command:

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

b Causes *tar* to use the next argument as the blocking factor for archive records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (key letters **x** and **t**).

F Causes *tar* to use the next argument as the name of a file from which succeeding arguments are taken.

l Tells *tar* to display an error message if it cannot resolve all of the links to the files being backed up. If **l** is not specified, no error messages are displayed.

m Tells *tar* to not restore the modification times. The modification time of the file is the time of extraction.

k Causes *tar* to use the next argument as the size of an archive volume in kilobytes. The minimum value allowed is 250. This option is useful when the archive is not intended for a magnetic tape device, but for some fixed size device, such as floppy disk (See **f** above). Very large files are split into "extents" across volumes. When restoring from a multivolume archive, *tar* only prompts for a new volume if a split file has been partially restored.

To override the value of **k** in the **default** file, specify **k** as 0 on the command line.

e Prevents files from being split across volumes (tapes or floppy disks). If there is not enough room on the present volume for a given file, *tar* prompts for a new volume. This is only valid when the **k** option is also specified on the command line.

n Indicates the archive device is not a magnetic tape. The **k** option implies this. Listing and extracting the contents of an archive are sped because *tar* can seek over files it wishes to skip. Sizes are printed in kilobytes instead of tape blocks.

p Indicates that files are extracted using their original permissions. It is possible that a non-super-user may be unable to extract files because of the permissions associated with the files or directories being extracted.

A Suppresses absolute filenames. Any leading "/"characters are removed from filenames. During extraction arguments given should match the relative (rather than the absolute) pathnames. With the **c, r, u** options the **A** options can be used to inhibit putting leading slashes in the archive headers.

tar reads **/etc/default/tar** to obtain default values for the device, blocking factor, volume size, and the device type (tape or non-tape). If no numeric key is specified on the command, *tar* looks for a line in the **default** file beginning with the string *archive0=*. Following this pattern are 4 blank separated strings indicating the values for the device, blocking factor, volume size and device type, in that order. A volume size of '0' indicates infinite volume length, (the previous default value of volume) and is suitable for magnetic tape media. An example **/etc/default/tar** line follows:

archive0=/dev/fd0 1 400 *n*

The *n* in the last field, means that this device is not a tape. Use *y* for tape devices. Any default value may be overridden on the command line. The numeric keys (0-7) select the line from the default value beginning with *archive#=*, where # is the numeric key. When the **f** key letter is specified on the command line, the entry "*archivef=*" is used. In this case, the default file entry must still contain 4 strings, but the first entry (specifying the device) is not significant. The default file **/etc/default/tar** need not exist if a device is specified on the command line.

Examples

If the name of a floppy disk device is **/dev/fd1**, then a tar format file can be created on this device by entering:

```
assign /dev/fd
tar cvfk /dev/fd1 360 files
```

where *files* are the names of files you want archived and 360 is the capacity of the floppy disk in kilobytes. Note that arguments to key letters are given in the same order as the key letters themselves, thus the **fk** key letters have corresponding arguments **/dev/fd1** and **360**. Note that if a *file* is a directory, the contents of the directory are recursively archived. To display a listing of the archive, enter:

```
tar tvf /dev/fd1
```

At some later time you will likely want to extract the files from the archive floppy. You can do this by entering:

```
tar xvf /dev/fd1
```

The above command extracts all files from the archive, using the exact same pathnames as used when the archive was created. Because of this behavior, it is normally best to save archive files with relative pathnames rather than absolute ones, since directory permissions may not let you read the files into the absolute directories specified. (See the **A** flag under *Options*.)

In the above examples, the **v** verbose option is used simply to confirm the reading or writing of archive files on the screen. Also, a normal file could be substituted for the floppy device **/dev/fd1** shown in the examples.

Files

/etc/default/tar	Default devices, blocking and volume sizes, device type
/tmp/tar*	

Diagnostics

Displays an error message about bad key characters and archive read/write errors.

Displays an error message if not enough memory is available to hold the link tables.

Notes

There is no way to ask for the *n*th occurrence of a file.

The **u** option can be slow.

The limit on filename length is 100 characters.

When archiving a directory that contains subdirectories, *tar* will only access those subdirectories that are within 17 levels of nesting. Subdirectories at higher levels will be ignored after *tar* displays an error message.

Systems with a 1K-byte file system cannot specify raw disk devices unless the **b** option is used to specify an even number of blocks. This means that one cannot update a raw-mode disk partition.

Do not enter:

tar xfF – –

This would imply taking two things from the standard input at the same time.

Use error-free floppy disks for best results with *tar*.

Name

tee – Creates a tee in a pipe.

Syntax

tee [**–i**] [**–a**] [**–u**] [file] ...

Description

tee transcribes the standard input to the standard output and makes copies in the *files*. The **–i** option ignores interrupts; the **–a** option causes the output to be appended to the *files* rather than overwriting them. The **–u** option causes the output to be unbuffered.

Examples

The following example illustrates the creation of temporary files at each stage in a pipeline:

```
grep ABC | tee ABC.grep | sort | tee ABC.sort | more
```

This example shows how to tee output to the terminal screen:

```
grep ABC | tee /dev/ttyxx | sort | uniq >final.file
```

Name

telinit, mkinittab – Alternative method of turning terminals on and off.

Syntax

telinit state
mkinittab [ttysfile]...

Description

telinit directs the actions of *init*(M). It is an alternative to using *enable*(C) and *disable*(C) to allow and disallow logins on terminals.

telinit generates a new **/etc/ttys** file from the **/etc/inittab** file. Only those lines from *inittab*(F) which apply in *state* are converted to their *ttys*(M) equivalent. *init* is then signaled to allow or disallow logins on terminals according to **/etc/ttys**.

The recognized *state* arguments are:

0–6
Generate **/etc/ttys** using the lines in **/etc/inittab** which apply to the specified *state*.

q, Q
Do not generate a new **/etc/ttys** file, but signal *init* to examine the existing **/etc/ttys** file.

s, S
Signal *init* to enter System Maintenance (single-user) mode.

Only the superuser can run *telinit*. Users currently logged onto terminals that are disabled are abruptly killed. Logins are not allowed on terminals not listed in **/etc/ttys**.

mkinittab writes on the standard output an *inittab*-format file generated from the specified *ttysfiles*. Each *ttysfile* must be in *ttys* format. If no *ttysfile* is specified, the standard input is read.

Files

/etc/ttys

/etc/inittab

See Also

disable(C), enable(C), getty(M), init(M), inittab(F), login(M), ttys(M)

Notes

inittab is provided for users more familiar with the *telinit* approach to terminal administration, as opposed to the standard XENIX *enable* and *disable* approach. It is intended that a full integration of these two approaches will be provided in a future version of XENIX.

Name

test – Tests conditions.

Syntax

test expr

[expr]

Description

test evaluates the expression *expr* , and if its value is true, returns a zero (true) exit status; otherwise, *test* returns a nonzero exit status if there are no arguments. The following primitives are used to construct *expr*:

–r *file*	True if *file* exists and is readable.
–w *file*	True if *file* exists and is writable.
–x *file*	True if *file* exists and is executable.
–f *file*	True if *file* exists and is a regular file.
–d *file*	True if *file* exists and is a directory.
–c *file*	True if *file* exists and is a character special file.
–b *file*	True if *file* exists and is a block special file.
–u *file*	True if *file* exists and its set-user-ID bit is set.
–g *file*	True if *file* exists and its set-group-ID bit is set.
–k *file*	True if *file* exists and its sticky bit is set.
–s *file*	True if *file* exists and has a size greater than zero.
–t [*fildes*]	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–z *s1*	True if the length of string *s1* is zero.
–n *s1*	True if the length of the string *s1* is nonzero.
s1 **=** *s2*	True if strings *s1* and *s2* are identical.

s1 **!=** *s2*	True if strings *s1* and *s2* are *not* identical.
s1	True if *s1* is *not* the null string.
n1 **-eq** *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, and **-le** may be used in place of **-eq**.

These primaries may be combined with the following operators:

!	Unary negation operator
-a	Binary *and* operator
-o	Binary *or* operator (**-a** has higher precedence than **-o**)
(expr)	Parentheses for grouping

Notice that all the operators and flags are separate arguments to *test*. Notice also, that parentheses are meaningful to the shell and, therefore, must be escaped.

See Also

find(C), sh(C)

Warning

In the second form of the command (i.e., the one that uses [ ], rather than the word *test*), each square bracket must be separated from any other characters by spaces.

Name

tic – Terminfo compiler.

Syntax

tic [**−v** [*n*] [**−p** permlist]] file ...

Description

tic translates terminfo files from the source format into the compiled format. The results are placed in the directory **/usr/lib/terminfo.**

The **−v** (verbose) option causes *tic* to output trace information showing its progress. If the optional digit *n* is appended, the level of verbosity can be increased.

The **−p** option directs *tic* to create a permissions file **permlist** for use with *fixperm*(M).

tic compiles all terminfo descriptions in the given files. When a **use=** field is discovered, *tic* first searches the current file and then the master file **./terminfo.src.**

If the environment variable TERMINFO is set, the results are placed there instead of **/usr/lib/terminfo.**

Some limitations: the total size of a description cannot exceed 4096 bytes; the name field cannot exceed 128 bytes.

Files

/usr/lib/terminfo/*/* -Compiled terminal capability database.

See Also

terminfo(M), terminfo(S), terminfo(F), tid(C)

Notes

Use of the **−p** option is not recommended. The functionality may change in future versions of XENIX.

Name

tid – Terminfo decompiler.

Syntax

tid [*term*]

Description

tid decompiles the description of terminal *term* originally compiled by *tic*(C). If *term* is not specified, the setting of the **TERM** environment variable is used.

Files

/usr/lib/terminfo/*/* – Compiled terminal descriptions.

See Also

tic(C), terminfo(F), terminfo(M).

Notes

The output of *tid* is not acceptable input to *tic*.

Name

touch – Updates access and modification times of a file.

Syntax

touch [**–amc**] [mmddhhmm[yy]] files

Description

touch causes the access and modification times of each argument to be updated. If no time is specified (see *date*(C)) the current time is used. The first *mm* refers to the month, *dd* refers to the day, *hh* refers to the hour, the second *mm* refers to the minute, and *yy* refers to the year. The **–a** and **–m** options cause touch to update only the access or modification times respectively (default is **–am**). The **–c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

See Also

date(C), utime(S)

Name

tput – Queries the terminfo database.

Syntax

tput [**-T***type*] attribute

Description

The command *tput* uses the terminfo database to make the values of terminal-dependent *attributes* available to the shell. *tput* outputs a string if the terminal *attribute* is of type string, or an integer if the *attribute* is of type integer. If the *attribute* is of type Boolean, *tput* simply sets the exit code (0 for true if the terminal has the capability, 1 for false if it does not) and produces no output.

The **-T** flag indicates the type of the terminal. Normally this option is unnecessary, as the default is taken from the environment variable **TERM**.

attribute is the terminal capability name from the terminfo database.

Examples

tput clear	Echo clear-screen sequence for the current terminal.
tput cols	Print the number of columns for the current terminal.
tput -T450 cols	Print the number of columns for the 450 terminal.

bold='tput smso'
offbold='tput rmso' Set the shell variables "bold" to begin standout mode sequence and "offbold" to end standout mode sequence for the current terminal. This might be followed by a prompt, such as:

```
echo "${bold}Name: ${offbold}\c"
```

tput hc Set exit code to indicate if the current terminal is a hardcopy terminal.

Files

/usr/lib/terminfo/*/* –Compiled terminal capability database.

See Also

terminfo(M), terminfo(S), tic(C), stty(C)

Notes

If the *attribute* is of type boolean, a value of 0 is returned for TRUE and a value of 1 for FALSE.

If the *attribute* is of type string or integer, a value of 0 is returned upon successful completion. Any other value returned indicates an error. For example, the specification of a bad *attribute* (any capability name that is not found in the terminfo database) produces an error.

Name

tr – Translates characters.

Syntax

tr [**–cds**] [string1 [string2]]

Description

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **–cds** may be used:

–c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal

–d Deletes all input characters in *string1*

–s Squeezes all strings of repeated output characters that are in *string2* to single characters

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a–z] Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a**n***]** Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character **** may be used as in the shell to remove special meaning from any character in a string. In addition, **** followed by 1, 2, or 3 octal digits, stands for the character whose ASCII code is given by those digits.

The following example creates a list of all the words in *file1*, one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline:

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

See Also

ed(C), sh(C), ascii(M)

Notes

Won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

Name

true – Returns with a zero exit value.

Syntax

true

Description

true does nothing except return with a zero exit value. *false*(C), *true*'s counterpart, does nothing except return with a nonzero exit value. *true* is typically used in shell procedures such as:

```
while true
do
        command
done
```

See Also

sh(C), false (C)

Diagnostics

true has exit status zero.

Name

tset - Sets terminal modes.

Syntax

tset [**–**] [**–hrsuIQS**] [**–e**[*c*]] [**–E**[*c*]] [**–k**[*c*]]
[–m [*ident*][*test baudrate*]:*type*] [*type*]

Description

tset causes terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and the like. It is driven by the **/etc/ttytype** and **/etc/termcap** files.

The type of terminal is specified by the *type* argument. The type may be any type given in **/etc/termcap.** If *type* is not specified, the terminal type is the value of the environment variable TERM, unless the **–h** flag is set or any **–m** argument is given. In this case, the type is read from **/etc/ttytype** (the port name to terminal type database). The port name is determined by a *ttyname*(S) call on the diagnostic output. If the port is not found in **/etc/ttytype** the terminal type is set to *unknown*.

Ports for which the terminal type is indeterminate are identified in **/etc/ttytype** as *dialup, plugboard,* etc. The user can specify how these identifiers should map to an actual terminal type. The mapping flag, **–m**, is followed by the appropriate identifier (a four-character or longer substring is adequate), an optional test for baud rate, and the terminal type to be used if the mapping conditions are satisfied. If more than one mapping is specified, the first correct mapping prevails. A missing identifier matches all identifiers. Baud rates are specified as with stty(C), and are compared with the speed of the diagnostic output. The test may be any combination of: >, =, <, @, and !. (Note: @ is a synonym for = and ! inverts the sense of the test. Remember that escape characters are meaningful to the shell.)

If the *type* as determined above begins with a question mark, the user is asked if he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. (The question mark must be escaped to prevent filename expansion by the shell.)

tset is most useful when included in the **.login** (for *csh*(C)) or **.profile** (for *sh*(C)) file executed automatically at login, with –m mapping used to specify the terminal type you most frequently dial in on.

Options

−e
This option sets the erase character to the named character, *c*, with *c* defaulting to Ctrl-H.

−E
This flag is identical to **−e** except that it only operates on terminals that can backspace.

−k
This option sets the kill character to the named character, *c*, with *c* defaulting to Ctrl-U. In all of these flags, "^X" where X is any character is equivalent to Ctrl-X .

− This option prints the terminal type on the standard output; this can be used to get the terminal type by entering:

```
set termtype = `tset -`
```

If no other options are given, *tset* operates in "fast mode" and *only* outputs the terminal type, bypassing all other processing.

−h
Forces *tset* to search **/etc/ttytype** for information and to overlook the environment variable, **TERM.**

−s
This option outputs "setenv" commands (if your default shell is *csh*(C) or "export" and assignment commands (if your default shell is *sh*(C));

For the −s option with the Bourne shell, enter:

```
tset -s ... > /tmp/tset$$
/tmp/tset$$
rm /tmp/tset$$
```

−S
This option only outputs the strings to be placed in the environment variables.

If you are using csh, enter:

```
set noglob
set term=(‘tset -S ....‘)
setenv TERM $term[1]
setenv TERMCAP "$term[2]"
unset term
unset noglob
```

−r
This option displays the terminal type on the diagnostic output.

−Q
This option suppresses displaying the "Erase set to" and "Kill set to" messages.

−I This option suppresses outputting the terminal initialization strings.

−m
This option is the mapping flag. It is used to specify the terminal type you most frequently use. It is followed by the appropriate identifier for your terminal, listed in **/etc/ttytype** . When you log on the system, it sets the terminal type to *ident* unless you specify otherwise.

Examples

tset gt42

Sets the terminal type to gt42.

tset −mdialup\>300:adm3a −mdialup:dw2 −Qr −e#

If the entry in **/etc/ttytype** corresponding to the login port is "dialup", and the port speed is greater than 300 baud, set the terminal type to adm3a. If the **/etc/ttytype** entry is "dialup" and the port speed is less than or equal to 300 baud, set the terminal type to dw2. Set the erase character to "#", and display the terminal type (but not the erase character) on standard error.

tset −m dial:ti733 −m plug:\?hp2621 −m unknown:\? −e −k^U

If the **/etc/ttytype** entry begins with "dial", the terminal type becomes ti733. If the entry begins with "plug", *tset* prompts with:

TERM = (hp2621)

Enter the correct terminal type if it is different than that shown. If the entry is "unknown", *tset* prompts with:

TERM = (unknown)

In any case erase is set to the terminal's backspace character, and the terminal type is displayed on standard error and the kill character is set to Ctrl-U.

Files

/etc/ttytype Port name to terminal type map database

/etc/termcap Terminal capability database

See Also

tty(M), termcap(M), stty(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Name

tty – Gets the terminal's name.

Syntax

tty [**–s**]

Description

The *tty* command prints the pathname of the user's terminal on the standard output. The **–s** option inhibits printing, allowing you to test just the exit code.

Exit Codes

0 if the standard input is a terminal, 1 otherwise.

Diagnostics

not a tty	If the standard input is not a terminal and **–s** is not specified

Name

umask – Sets file-creation mode mask.

Syntax

umask [ooo]

Description

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively. Only the low-order 9 bits of *cmask* and the file mode creation mask are used. The value of each specified digit is "subtracted" from the corresponding "digit" specified by the system for the creation of any file (see *umask*(S) or *creat*(S)). This is actually a binary masking operation, and thus the name "umask". In general, binary ones remove a given permission, and zeros have no effect at all. For example, **umask 022** removes *group* and *others* write permission (files normally created with mode 777 become mode 755 ; files created with mode 666 become mode 644).

If *ooo* is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell. By default, login shells have a umask of 022.

See Also

chmod(C), sh(C), chmod(S), creat(S), umask(S)

Name

umount – Dismounts a file structure.

Syntax

/etc/umount special-device

Description

umount announces to the system that the removable file structure previously mounted on device *special-device* is to be removed. Any pending I/O for the file system is completed, and the file structure is flagged clean. For a detailed explanation of the mounting process, see *mount*(C).

Files

/etc/mnttab Mount table

See Also

mount(C), mount(S), mnttab(F)

Diagnostics

device busy An executing process is using a file on the named file system

Name

uname – Prints the name of the current XENIX system.

Syntax

uname [**-snrmvdupa**]

Description

uname prints the current system name of the XENIX system on the standard output file. It is primarily used to determine which system you are using. The options cause selected information returned by *uname*(S) to be printed:

-s Prints the system name (default).

-n Prints the nodename (the nodename may be a name that the system is known by to a communications network).

-r Prints the operating system release.

-m Manufacturer prints original supplier (number) of XENIX system.

-v Prints the operating system version.

-d Distributor prints OEM (number) for the system.

-u Prints user serial number.

-p Prints processor of the machine.

-a Prints all the above information.

See Also

uname(S)

Name

uniq – Reports repeated lines in a file.

Syntax

uniq [**–udc** [**+**n] [**–**n]] [input [output]]

Description

uniq reads the *input* file and compares adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(C). If the **–u** flag is used, just the lines that are not repeated in the original file are output. The **–d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **–u** and **–d** mode outputs.

The **–c** option supersedes **–u** and **–d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of nonspace, nontab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

See Also

comm(C), sort(C)

Name

units – Converts units.

Syntax

units

Description

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division is shown by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, as well as the following:

pi Ratio of circumference to diameter

c Speed of light

e Charge on an electron

g Acceleration of gravity

force Same as **g**

mole
Avogadro's number

water
Pressure head per unit height of water

au Astronomical unit

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g. **lightyear**). British units that differ from their US counterparts are prefixed with "br". For a complete list of units, enter:

cat /usr/lib/unittab

Files

/usr/lib/unittab

Name

uucico - Executes work files for uucp data transmission.

Syntax

uucico [options] ...

Description

The *uucico* program performs the following major functions:

- Scan the spool directory for work.
- Place a call to a remote site.
- Negotiate a line protocol to be used.
- Execute all requests from both sites.
- Log work requests and work completions.

uucico may be started by a system daemon such as *cron*(C), by the user (usually for testing), or by a remote site.

Options

The *uucico* program must generally be started directly by the user or by another program, such as a shell script invoked by *cron* . There are several options used for execution:

−r1
: Start the program in MASTER mode. This is used when *uucico* is started by a program or *cron* shell.

  In MASTER mode, a connection is made to a remote site. MASTER mode operates in one of two ways. If no site name is specified (the **−s** option not specified), the program scans the spool directory for sites to call. If a site name is specified, that site is called, and work is only done for that site.

  If started by a remote site, the program is considered to be in SLAVE mode.

−s*sitename*
: Do work only for site *sitename* . If **−s** is specified, a call to the specified site is made even if there is no work for site *sitename* in the spool directory, but call only when times in the **L.sys** file permit it. This is useful for polling sites that do not have the hardware to initiate a connection.

−S*sitename*
Do work only for site *sitename* . If **−S** is specified, a call to the specified site is made even if there is no work for the site in the site in the spool directory. Unlike **−s**, this option ignores the call times for the *sitename* given in the **L.sys** file.

The following options are used primarily for debugging:

−d*dir*
Use directory *dir* for the spool directory.

−x*num*
Use *num* (1-9) as the level of debugging output. 9 specifies the highest level of debugging.

Notes

The *uucico* program should be specified as the shell field in the **/etc/passwd** file for the uucp logins.

If you want your machine to immediately forward files from a remote machine to other machines, the command */usr/lib/uucp/uucico* must be present in the **L.cmds** file. Otherwise, files will be forwarded the next time your machine connects with the remote machine.

On a dial-in site, *uucico* is always started whenever a calling site logs in.

However, on a dial-out site, *uucico* is only started when explicitly invoked.

Files

/usr/spool/uucp/AUDIT
/usr/spool/uucp/LOGFILE
/usr/spool/uucp/SEQF
/usr/spool/uucp/SYSLOG
/usr/spool/uucp/X.*
/usr/lib/uucp/Cmds
/usr/lib/uucp/L-devices
/usr/lib/uucp/L.cmds
/usr/lib/uucp/L.sys
/usr/lib/uucp/L_stat
/usr/lib/uucp/L_sub
/usr/lib/uucp/R_stat
/usr/lib/uucp/R_sub
/usr/lib/uucp/USERFILE

/usr/lib/uucp/uucico

See Also

uux(C), mail(C), uuinstall(C), uuto(C)

Name

uuclean – Clean-up the uucp spool directory.

Syntax

uuclean [options] ...

Description

uuclean scans the spool directory for files with the specified prefix and deletes all those which are older than the specified number of hours.

The following options are available:

–d*directory*
Clean *directory* instead of the spool directory.

–p*pre* Scan for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following will cause all files older than the specified time to be deleted.

–n*time* Files whose age is more than *time* hours are deleted if the prefix test is satisfied. (Default time is 72 hours.)

–m Send mail to the owner of the file when it is deleted.

–x*num* Provides debugging information based on the value of *num*. For increasing values of *num*, more information is given.

This program will typically be started by *cron*(C).

Files

/usr/lib/uucp	directory with commands used by *uuclean* internally
/usr/spool/uucp	spool directory

See Also

uucp(C), uux(C).

Name

uucp, uulog, uuname - Copies files from XENIX to XENIX.

Syntax

uucp [option] ... source-file ... destination-file

uulog [option] ...

uuname [**-l**]

Description

uucp copies files named by the source-file arguments to the destination-file argument. A filename may be a pathname on your machine, or may have the form:

system-name!pathname

where "system-name" is taken from a list of system names which *uucp* knows about. Shell metacharacters ?*[] appearing in *pathname* will be expanded on the appropriate system.

Pathnames may be a a full pathname, or a pathname preceded by ~*user* where *user* is a user ID on the specified system and is replaced by that user's login directory. Anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system, the copy will fail. If the destination file is a directory, the last part of the source filename is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(S)). *uucp* makes no distinction between binary and text files. However, the set uid and set gid flags will not accompany the binary file and must be set once the binary has arrived at its destination. The appropriateness of the file permissions and ownership should be checked.

The following options are interpreted by *uucp*:

-d Makes all necessary directories for the file copy.

-c Uses the source file when copying out rather than copying the file to the spool directory.

-m Sends mail to you when the copy is complete.

-r	Queues the job but does not start the file transfer process. By default, a file transfer process is started each time *uucp* is invoked.
-x *num*	Provides debugging based on the value of *num*. For increasing values of *num*, more information is given.
-n *user*	Notifies user on remote system. If sending files, the file's owner and group id's will be set to those of the user.

uulog maintains a summary log of *uucp* and *uux*(C) transactions in the file **/usr/spool/uucp/LOGFILE** by gathering information from partial log files named **/usr/spool/uucp/LOG.*.?**. *uulog* removes the partial log files.

The options cause *uulog* to display log information:

-s*sys*
Displays information about work involving system *sys*.

-u*user*
Displays information about work done for the specified *user*.

uuname displays the *uucp* names of known systems. The **-l** option returns the local system name. The format of **ADMIN** is:

sysname tab description tab

Files

/usr/spool/uucp	Spool directory
/usr/spool/uucppublic	Public directory for receiving and sending
/usr/lib/uucp/*	Other data and program files

See Also

uux(C), mail(C), uuinstall(C), uuto(C)

Notes

For security reasons, all files received by *uucp* should be owned by uucp.

The **-m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ?*[] will not activate the **-m** option.

This version of *uucp* is based on a version 7 implementation.

Warning

For security reasons, file access should be, and usually is, severely restricted. You probably will not be able to copy or manipulate arbitrary filcs, or cxccutc many commands, on a remote machine.

Similarly, you may not be able to send files to arbitrary, remote pathnames. As distributed, the remotely accessible files are in **/usr/spool/uucppublic.**

Name

uuinstall – Administers UUCP control files.

Syntax

/etc/uuinstall [-r]

Description

The *uuinstall* program is used to manage the content of the control files used by the *uucp* communications system. It allows the user to change the contents of these files without using a text editor. The user need not know the detailed format of each of the control files, although he must be familiar with the function of the various fields within the files. These details are explained in the XENIX *User's Guide* .

The *uuinstall* program can only be executed by the super-user. When invoked with the optional **-r** flag, *uuinstall* will not allow any of the files to be modified whether or not the user has made changes to the files.

If *uuinstall* finds any of the required **uucp** control files missing from the system, it will create them with the correct access permissions and ownership.

Files

/etc/systemid
/usr/lib/uucp/USERFILE
/usr/lib/uucp/L.sys
/usr/lib/uucp/L-devices
/usr/lib/uucp/L-dialcodes

See Also

mkuser(C),
XENIX *User's Guide*

Name

uustat – uucp status inquiry and job control.

Syntax

uustat [-c] [-j] [-k] [-m] [-osuvy] . . .

Description

uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The following options are recognized:

–c*hour*	Remove the status entries which are older than *hour* hours.
–j*all*	Report the status of all the *uucp* requests.
–k*jobn*	Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless he is the superuser.
–m*mch*	Report the status of accessibility of machine *mch*. If *mch* is specified as all, the status of all machines known to the local *uucp* are provided.
–o*hour*	Report the status of all *uucp* requests which are older than *hour* hours.
–s*sys*	Report the status of all *uucp* requests which communicate with remote system *sys*.
–u*user*	Report the status of all *uucp* requests issued by *user*.
–v	Report the *uucp* status verbosely. If this option is not specified, a status code is displayed with each *uucp* request.
–y*hour*	Report the status of all *uucp* requests which are younger than *hour* hours.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options **–j**, **–m**, **–k**, or **–c** may be specified at a time.

For example, the command:

```
uustat -uhdc -smhtsa -y72 -v
```

prints the verbose status of all *uucp* requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The job request status format is:

```
job-number user remote-system command-time status-time
```

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
00001	the copy failed, but the reason cannot be determined
00002	permission to access local file is denied
00004	request forwarded
00020	job in progress
00100	cannot copy to local directory - file left in pubdir/user/file
00200	local system cannot create temporary file
00400	cannot execute *uucp*
01000	copy succeeded
02000	copy finished, job deleted
04000	job is queued

The machine accessibility status format is:

system name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

Files

/usr/spool/uucp spool directory
/usr/lib/uucp/L_stat
system status file
/usr/lib/uucp/R_stat
request status file

See Also

uucp(C).

Name

uusub – Monitor uucp network.

Syntax

uusub [options]

Description

uusub defines a *uucp* subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

–a*sys*	Add *sys* to the subnetwork.
–d*sys*	Delete *sys* from the subnetwork.
–l	Report the statistics on connections.
–r	Report the statistics on traffic amount.
–f	Flush the connection statistics.
–u*hr*	Gather the traffic statistics over the past *hr* hours.
–c*sys*	Exercise the connection to the system *sys*. If *sys* is specified as **all**, then exercise the connection to all the systems in the subnetwork.

The connections report format is:

sys #call #ok time #dev #login #nack #other

where *sys* is the remote system name, *#call* is the number of times the local system tries to call *sys* since the last flush was done, *#ok* is the number of successful connections, *time* is the the latest successful connect time, *#dev* is the number of unsuccessful connections because of no available device (e.g. ACU), *#login* is the number of unsuccessful connections because of login failure, *#nack* is the number of unsuccessful connections because of no response (e.g., line busy, system down), and *#other* is the number of unsuccessful connections because of other reasons.

The traffic statistics format is:

sfile sbyte rfile rbyte

where *sfile* is the number of files sent and *sbyte* is the number of bytes sent over the period of time indicated in the latest *uusub* command with the **–u***hr* option. Similarly, *rfile* and *rbyte* are the numbers of files and bytes received.

The command:

uusub −c all −u 24

is typically started by *cron*(C) once a day.

Files

/usr/spool/uucp/SYSLOG	system log file
/usr/lib/uucp/L_sub	connection statistics
/usr/lib/uucp/R_sub	traffic statistics

See Also

uucp(C), uustat(C).

Name

uuto, uupick – Public XENIX-to-XENIX file copy.

Syntax

uuto [options] source-files destination
uupick [**–s** system]

Description

uuto sends *source-files* to *destination*. *uuto* uses the *uucp*(CP) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the following format:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*(CP)). *Logname* is the login name of someone on the specified system.

Two options are available:

–p Copy the source file into the spool directory before transmission.
–m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(C) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] **?**

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line> Go on to next entry.

d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir* (current directory is default).
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick invoked with the **−s***system* option will only search /usr/spool/uucppublic for files sent from *system*.

Files

/usr/spool/uucppublic   public directory

See Also

mail(C), uuclean(C), uucp(C), uuname(C), uustat(C), uux(C).

Name

uux – Executes command on remote XENIX.

Syntax

uux [**–**] command-string

Description

uux gathers 0 or more files from various systems, executes commands on a specified system, and sends the standard output to a file on a specified system.

The command-string is made up of one or more arguments that look like a shell command line, except that the command and filenames may be prefixed by system-name!. A null system-name is interpreted as the local system.

Filenames may be (1) a full pathname; (2) a pathname preceded by ˜*xxx*; where *xxx* is a user ID on the specified system and is replaced by that user's login directory; or (3) anything else prefixed by the current directory.

The "–" option causes the standard input to the *uux* command to be the standard input to the command-string.

For example, the command:

    uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !fi.diff"

will get the **f1** files from the usg and pwba machines, execute a *diff* command and put the results in **f1.diff** in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

Files

/usr/uucp/spool   Spool directory

/usr/uucp/*   Other data and programs

See Also

uucp(C)

Warning

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an incoming request from *uux*. Typically, a restricted site will permit little other than the receipt of mail via *uux*.

Notes

Only the first command of a shell pipeline may have a system-name!. All other commands are executed on the system of the first command.

The shell metacharacter * will probably not perform as expected.

The shell tokens << and >> are not implemented.

There is no notification of denial of execution on the remote machine.

Name

vi, view, vedit – Invokes a screen-oriented display editor.

Syntax

vi [–option ...] [command ...] [filename ...]

view [–option ...] [command ...] [filename ...]

vedit [–option ...] [command ...] [filename ...]

Description

vi offers a powerful set of text editing operations based on a set of mnemonic commands. Most commands are single keystrokes that perform simple editing functions. *vi* displays a full screen "window" into the file you are editing. The contents of this window can be changed quickly and easily within *vi*. While editing, visual feedback is provided (the name vi itself is short for "visual").

The *view* command is the same as *vi* except that the read-only option (**–R**) is set automatically. The file cannot be changed with *view*.

The *vedit* command is the same as *vi* except for differences in the option settings. *vedit* uses **novice** mode, turns off the **magic** option, sets the option **report=1** calls the **showmode** and sets **redraw**.

The **showmode** option informs the *vedit* user, in a message in the lower right hand corner of the screen, which mode is being used. For instance after the **ESC–i** command is used, the message reads "INSERT MODE".

Note that you can not set the **novice** option from within *vi* or *ex*. If you want to use the **novice** option you must use the *vedit* utility. (It is possible to set the **nonovice** option from within *vedit*.)

vi and the line editor *ex* are one and the same editor: the names *vi* and *ex* identify a particular user interface rather than any underlying functional difference. The differences in user interface, however, are quite striking. *ex* is a powerful line-oriented editor, similar to the editor *ed*. However, in both *ex* and *ed*, visual updating of the terminal screen is limited, and commands are entered on a command line. *vi*, on the other hand, is a screen-oriented editor designed so that what you see on the screen corresponds exactly and immediately to the contents of the file you are editing. In the following discussion, *vi* commands and options are printed in boldface type.

Options available on the *vi* command line include:

-t Equivalent to an initial *tag* command; edits the file containing the tag and positions the editor at its definition.

-r Used in recovering after an editor or system crash, retrieving the last saved version of the named file. If no file is specified, this option prints a list of saved files.

-l Specific to editing LISP, this option sets the **showmatch** and **lisp** options.

-w*n* Sets the default window size to *n*. Useful on dialups to start in small windows.

-R Sets a read-only option so that files can be viewed but not edited.

The Editing Buffer

vi performs no editing operations on the file that you name during invocation. Instead, it works on a copy of the file in an "editing buffer."

When you invoke *vi* with a single filename argument, the named file is copied to a temporary editing buffer. The editor remembers the name of the file specified at invocation, so that it can later copy the editing buffer back to the named file. The contents of the named file are not affected until the changes are copied back to the original file.

Modes of Operation

Within *vi* there are three distinct modes of operation:

Command Mode — Within command mode, signals from the keyboard are interpreted as editing commands.

Insert Mode — Insert mode can be entered by typing any of the *vi* insert, append, open, substitute, change, or replace commands. Once in insert mode, letters typed at the keyboard are inserted into the editing buffer.

ex Escape Mode — The *vi* and *ex* editors are one and the same editor differing mainly in their user interface. In *vi* , commands are usually single keystrokes. In *ex*, commands are lines of text terminated by a RETURN. *vi* has a special "escape" command that gives access to many of these line-oriented *ex* commands. To use the *ex* escape mode, type a colon (:). The colon is echoed on the status line as a prompt for the *ex* command. An executing command can be aborted by pressing INTERRUPT. Most file manipulation commands are executed in *ex* escape mode (for example, the commands to read in a file and to write out the editing buffer to a file).

Special Keys

There are several special keys in *vi*. The following keys are used to edit, delimit, or abort commands and command lines.

ESC Used to return to *vi* command mode or to cancel partially formed commands.

RETURN Terminates *ex* commands when in *ex* escape mode. Also used to start a newline when in insert mode.

INTERRUPT
Often the same as the DEL or RUBOUT key on many terminals. Generates an interrupt, telling the editor to stop what it is doing. Used to abort any command that is executing.

/ Used to specify a string to be searched for. The slash appears on the status line as a prompt for a search string. The question mark (?) works exactly like the slash key, except that it is used to search backward in a file instead of forward.

: The colon is a prompt for an *ex* command. You can then type in any *ex* command, followed by an ESC or RETURN, and the given *ex* command is executed.

The following characters are special in insert mode:

BKSP Backs up the cursor one character on the current line. The last character typed before the BKSP is removed from the input buffer, but remains displayed on the screen.

Ctrl-U Moves the cursor back to the first character of the insertion and restarts insertion.

Ctrl-V Removes the special significance of the next typed character. Use Ctrl-V to insert control characters. Linefeed and Ctrl-J cannot be inserted in the text except as newline characters. Ctrl-Q and Ctrl-S are trapped by the operating system before they are interpreted by *vi*, so they too cannot be inserted as text.

Ctrl-W Moves the cursor back to the first character of the last inserted word.

Ctrl-T During an insertion, with the **autoindent** option set and at the beginning of the current line, entering this character will insert *shiftwidth* whitespace.

Ctrl-@ If entered as the first character of an insertion, it is replaced with the last text inserted, and the insertion terminates. Only 128 characters are saved from the last insertion. If more than 128 characters were inserted, then this command inserts no characters. A Ctrl-@ cannot be part of a file, even if quoted.

Starting and Exiting vi

To enter *vi*, enter:

vi	*Edits empty editing buffer*
vi file	*Edits named file*
vi +123 file	*Goes to line 123*
vi +45 file	*Goes to line 45*
vi +/word file	*Finds first occurrence of "word"*
vi +/tty file	*Finds first occurrence of "tty"*

There are several ways to exit the editor:

ZZ The editing buffer is written to the file *only* if any changes were made.

:x The editing buffer is written to the file *only* if any changes were made.

:q! Cancels an editing session. The exclamation mark (!) tells *vi* to quit unconditionally. In this case, the editing buffer is not written out.

vi Commands

vi is a visual editor with a window on the file. What you see on the screen is *vi*'s notion of what the file contains. Commands do not cause any change to the screen until the complete command is entered. Most commands may take a preceding count that specifies repetition of the command. This count parameter is not given in the following command descriptions, but is implied unless overridden by some other prefix argument. When *vi* gets an improperly formatted command, it rings a bell.

Cursor Movement

The cursor movement keys allow you to move your cursor around in a file. Note in particular the direction keys (if available on your terminal), the H, J, K, and L cursor keys, and SPACEBAR, BKSP, Ctrl-N, and Ctrl-P. These three sets of keys perform identical functions.

Forward Space – l, SPACEBAR, or right direction key

Syntax: **l**
SPACEBAR
right direction key

Function:
Moves the cursor forward one character. If a count is given, move forward count characters. You cannot move past the end of the line.

Backspace – h, BKSP, or left direction key

Syntax: **h**
BKSP
left direction key

Function: Moves cursor backward one character. If a count is given, moves backward *count* characters. Note that you cannot move past the beginning of the current line.

Next Line – +, RETURN, j, Ctrl-N, and LF

Syntax: **+**
RETURN

Function: Moves the cursor down to the beginning of the next line.

Syntax: **j**
Ctrl-N
LF
down direction key

Function: Moves the cursor down one line, remaining in the same column. Note the difference between these commands and the preceding set of next line commands which move to the *beginning* of the next line.

Previous Line — k, Ctrl-P, and up direction key

Syntax: **k**
Ctrl-P
up direction key

Function: Moves the cursor up one line, remaining in the same column. If a count is given, the cursor is moved *count* lines.

Syntax: **–**

Function: Moves the cursor up to the beginning of the previous line. If a count is given, the cursor is moved up a *count* lines.

Beginning of Line — 0 and ^

Syntax: **^**
0

Function: Moves the cursor to the beginning of the current line. Note that **0** always moves the cursor to the first character of the current line. The caret (^) works somewhat differently: it moves to the first character on a line that is not a tab or a space. This is useful when editing files that have a great deal of indentation, such as program texts.

End of Line — $

Syntax: **$**

Function: Moves the cursor to the end of the current line. Note that the cursor resides on top of the last character on the line. If a count is given, the cursor is moved forward *count*–1 lines to the end of the line.

Goto Line — G

Syntax: [*linenumber*]**G**

Function: Moves the cursor to the beginning of the line specified by *linenumber*. If no *linenumber* is given, the cursor moves to the beginning of the *last* line in the file. To find the line number of the current line, use Ctrl-G.

Column – |

Syntax: [*column*] |

Function: Moves the cursor to the column in the current line given by *column*. If no *column* is given, the cursor is moved to the first column in the current line.

Word Forward – w and W

Syntax: **w**
W

Function: Moves the cursor forward to the beginning of the next word. The lowercase **w** command searches for a word defined as a string of alphanumeric characters separated by punctuation or whitespace (i.e., tab, newline, or space characters). The uppercase **W** command searches for a word defined as a string of nonwhitespace characters.

Back Word – b and B

Syntax: **b**
B

Function: Moves the cursor backward to the beginning of a word. The lowercase **b** command searches backward for a word defined as a string of alphanumeric characters separated by punctuation or whitespace (i.e., tab, newline, or space characters). The uppercase **B** command searches for a word defined as a string of nonwhitespace characters. If the cursor is already within a word, it moves backward to the beginning of that word.

End – e and E

Syntax: **e**
E

Function: Moves the cursor to the end of a word. The lowercase **e** command moves the cursor to the last character of a word, where a word is defined as a string of alphanumeric characters separated by punctuation or whitespace (i.e., tab, newline, or space characters). The uppercase **E** moves the cursor to the last character of a word where a word is defined as a string of nonwhitespace characters. If the cursor is already within a word, it moves to the end of that word.

Sentence – (and)

Syntax: (
)

Function: Moves the cursor to the beginning (left parenthesis) or end of a sentence (right parenthesis). A sentence is defined as a sequence of characters ending with a period (.), question mark (?), or exclamation mark (!), followed by either two spaces or a newline. A sentence begins on the first nonwhitespace character following a preceding sentence. Sentences are also delimited by paragraph and section delimiters. See below.

Paragraph – { and }

Syntax: }
{

Function: Moves the cursor to the beginning ({) or end (}) of a paragraph. A paragraph is defined with the *paragraphs* option. By default, paragraphs are delimited by the nroff macros ".IP", ".LP", ".P", ".QP", and ".bp". Paragraphs also begin after empty lines.

Section – [[and]]

Syntax:]]
[[

Function: Moves the cursor to the beginning ([[) or end (]]) of a section. A section is defined with the *sections* option. By default, sections are delimited by the nroff macros ".NH" and ".SH". Sections also start at formfeeds (Ctrl-L) and at lines beginning with a brace ({).

Match Delimiter – %

Syntax: %

Function: Moves the cursor to a matching delimiter, where a delimiter is a parenthesis, a bracket, or a brace. This is useful when matching pairs of nested parentheses, brackets, and braces.

Home – H

Syntax: [*offset*]H

Function: Moves the cursor to the upper left corner of the screen. Use this command to quickly move to the top of the screen. If an *offset* is given, the cursor is homed *offset*-1 number of lines from the top of the screen. Note that the command "dH" deletes all lines from the current line to the top line shown on the screen.

Middle Screen – M

Syntax: **M**

Function: Moves the cursor to the beginning of the screen's middle line. Use this command to quickly move to the middle of the screen from either the top or the bottom. Note that the command "dM" deletes from the current line to the line specified by the M command.

Lower Screen – L

Syntax: [*offset*]L

Function: Moves the cursor to the lowest line on the screen. Use this command to quickly move to the bottom of the screen. If an *offset* is given, the cursor is homed *offset*-1 number of lines from the bottom of the screen. Note that the command "dL" deletes all lines from the current line to the bottom line shown on the screen.

Previous Context – `` and ''

Syntax:
''
'*character*
``
`*character*

Function: Moves the cursor to previous context or to context marked with the m command. If the single quotation mark or back quotation mark is doubled, the cursor is moved to previous context. If a single character is given after either quotation mark, the cursor is moved to the location of the specified mark as defined by the **m** command. Previous context is the location in the file of the last "nonrelative" cursor movement. The single quotation mark (') syntax is used to move to the beginning of the line representing the previous context. The back quotation mark (`) syntax is used to move to the previous context *within* a line.

The Screen Commands

The screen commands are *not* cursor movement commands and cannot be used in delete commands as the delimiters of text objects. However, the screen commands do move the cursor and are useful in paging or scrolling through a file. These commands are described below:

Page – Ctrl-U and Ctrl-D

Syntax: [*size*]**Ctrl-U**
[*size*]**Ctrl-D**

Function: Scrolls the screen up a half window (Ctrl-U) or down a half window (Ctrl-D). If *size* is given, the scroll is *size* number of lines. This value is remembered for all later scrolling commands.

Scroll – Ctrl-F and Ctrl-B

Syntax: **Ctrl-F**
Ctrl-B

Function: Pages screen forward and backward. Two lines of continuity are kept between pages if possible. A preceding count gives the number of pages to move forward or backward.

Status – Ctrl-G

Syntax: **BELL**
Ctrl-G

Function: Displays *vi* status on status line. This gives you the name of the file you are editing, whether it has been modified, the current line number, the number of lines in the file, and the percentage of the file (in lines) that precedes the cursor.

Zero Screen – z

Syntax: [*linenumber*]z[*size*]**RETURN**
[*linenumber*]z[*size*]**.**
[*linenumber*]z[*size*]**–**

Function: Redraws the display with the current line placed at or "zeroed" at the top, middle, or bottom of the screen, respectively. If you give a *size*, the number of lines displayed is equal to *size*. If a preceding *linenumber* is given, the given line is placed at the top of the screen. If the last argument is a RETURN, the current line is placed at the top of the screen. If the last argument is a period (.), the current line is placed in the middle of the screen. If the last argument is a minus sign (–), the current line is placed at the bottom of the screen.

Redraw – Ctrl-R or Ctrl-L

Syntax: **Ctrl-R**
Ctrl-L

Function: Redraws the screen. Use this command to erase any system messages that may scramble your screen. Note that system messages do not affect the file you are editing.

Text Insertion

The text insertion commands always place you in insert mode. Exit from insert mode is always done by pressing ESC. The following insertion commands are "pure" insertion commands; no text is deleted when you use them. This differs from the text modification commands, change, replace, and substitute, which delete and then insert text in one operation.

Insert – i and I

Syntax: **i**[*text*]**ESC**
I[*text*]**ESC**

Function: Insert *text* in editing buffer. The lowercase **i** command places you in insert mode. *Text* is inserted *before* the character beneath the cursor. To insert a newline, press a RETURN. Exit insert mode by typing the ESC key. The uppercase I command places you in insert mode, but begins text insertion at the beginning of the current line, rather than before the cursor.

Append – a and A

Syntax: **a**[*text*]**ESC**
A[*text*]**ESC**

Function: Appends *text* to the editing buffer. The lowercase **a** command works exactly like the lowercase **i** command, except that text insertion begins after the cursor and not before. This is the one way to add text to the end of a line. The uppercase **A** command begins appending text at the end of the current line rather than after the cursor.

Open New Line – o and O

Syntax: **o**[*text*]**ESC**
O[*text*]**ESC**

Function: Opens a new line and inserts text. The lowercase **o** command opens a new line below the current line; uppercase **O** opens a new line *above* the current line. After the new line has been opened, both these commands work like the **I** command.

Text Deletion

Many of the text deletion commands use the D key as an operator. This operator deletes text objects delimited by the cursor and a cursor movement command. Deleted text is always saved away in a buffer. The delete commands are described below:

Delete Character – x and X

Syntax: **x**
X

Function: Deletes a character. The lowercase **x** command deletes the character beneath the cursor. With a preceding count, *count* characters are deleted to the right beginning with the character beneath the cursor. This is a quick and easy way to delete a few characters. The uppercase **X** command deletes the character just before the cursor. With a preceding count, *count* characters are deleted backward, beginning with the character just before the cursor.

Delete – d and D

Syntax: **d***cursor-movement*
dd
D

Function: Deletes a text object. The lowercase **d** command takes a *cursor-movement* as an argument. If the *cursor-movement* is an intraline command, deletion takes place from the cursor to the end of the text object delimited by the *cursor-movement*. Deletion forward deletes the character beneath the cursor; deletion backward does not. If the *cursor-movement* is a multi-line command, deletion takes place from and including the current line to the text object delimited by the *cursor-movement*.

The **dd** command deletes whole lines. The uppercase **D** command deletes from and including the cursor to the end of the current line.

Deleted text is automatically pushed on a stack of buffers numbered 1 through 9. The most recently deleted text is also placed in a special delete buffer that is logically buffer 0. This special buffer is the default buffer for all (put) commands using the double quotation mark (") to specify the number of the buffer for delete, put, and yank commands. The buffers 1 through 9 can be accessed with the **p** and **P** (put) commands by appending the double quotation mark (") to the number of the buffer. For example:

    "4p

puts the contents of delete buffer number 4 in your editing buffer just below the current line. Note that the last deleted text is "put" by default and does not need a preceding buffer number.

Text Modification

The text modification commands all involve the replacement of text with other text. This means that some text will necessarily be deleted. All text modification commands can be "undone" with the **u** command:

Undo – u and U

Syntax: **u**
U

Function: Undoes the last insert or delete command. The lowercase **u** command undoes the last insert or delete command. This means that after an insert, **u** deletes text; and after a delete, **u** inserts text. For the purposes of undo, all text modification commands are considered insertions.

The uppercase **U** command restores the current line to its state before it was edited, no matter how many times the current line has been edited since you moved to it.

Repeat – .

Syntax: .

Function: Repeats the last insert or delete command. A special case exists for repeating the **p** and **P** "put" commands. When these commands are preceded by the name of a delete buffer, successive **u** commands display the contents of the delete buffers.

Change — c and C

Syntax: **c***cursor-movement text***ESC**
C*text***ESC**
cc*text***ESC**

Function: Changes a text object and replaces it with *text* . Text is inserted as with the **i** command. A dollar sign ($) marks the extent of the change. The **c** command changes arbitrary text objects delimited by the cursor and a *cursor-movement* . The **C** and **cc** commands affect whole lines and are identical in function.

Replace — r and R

Syntax: **r***char*
R*text***ESC**

Function: Overstrikes character or line with *char* or *text* , respectively. Use **r** to overstrike a single character and **R** to overstrike a whole line. A count multiplies the replacement text count times.

Substitute — s and S

Syntax: **s***text***ESC**
S*text***ESC**

Function: Substitutes current character or current line with *text*. Use **s** to replace a single character with new text. Use **S** to replace the current line with new text. If a preceding count is given, *text* substitutes for count number of characters or lines depending on whether the command is **s** or **S**, respectively.

Filter — !

Syntax: **!***cursor-movement cmd* **RETURN**

Function: Filters the text object delimited by the cursor and *cursor-movement* through the XENIX command, *cmd*. For example, the following command sorts all lines between the cursor and the bottom of the screen, substituting the designated lines with the sorted lines:

!Lsort

Arguments and shell metacharacters may be included as part of *cmd*; however, standard input and output are always associated with the text object being filtered.

Join Lines – J

Syntax: **J**

Function: Joins the current line with the following line. If a *count* is given, *count* lines are joined.

Shift – < and >

Syntax: >[*cursor-movement*]
<[*cursor-movement*]
>>
<<

Function: Shifts text left (>) or right (<). Text is shifted by the value of the option *shiftwidth*, which is normally set to eight spaces. Both the > and < commands shift all lines in the text object delimited by the current line and *cursor-movement*. The >> and << commands affect whole lines. All versions of the command can take a preceding count that acts to multiply the number of objects affected.

Text Movement

The text movement commands move text in and out of the named buffers *a-z* and out of the delete buffers *1-9*. These commands either "yank" text out of the editing buffer and into a named buffer or "put" text into the editing buffer from a named buffer or a delete buffer. By default, text is put and yanked from the "unnamed buffer", which is also where the most recently deleted text is placed. Thus it is quite reasonable to delete text, move your cursor to the location where you want the deleted text placed, and then put the text back into the editing buffer at this new location with the **p** or **P** command.

The named buffers are most useful for keeping track of several chunks of text that you want to keep on hand for later access, movement, or rearrangement. These buffers are named with the letters *a* through *z*. To refer to one of these buffers (or one of the numbered delete buffers) in a command, use a quotation mark. For example, to yank a line into the buffer named *a*, enter:

```
"ayy
```

To put this text back into the file, enter:

```
"ap
```

If you delete text in the buffer named *A* rather than *a*, text is appended to the buffer.

Note that the contents of the named buffers are not destroyed when you switch files. Therefore, you can delete or yank text into a buffer, switch files, and then do a put. Buffer contents are *destroyed* when you exit the editor, so be careful.

Put – p and P

Syntax: ["*alphanumeric*]**p**
["*alphanumeric*]**P**

Function: Puts text from a buffer into the editing buffer. If no buffer name is specified, text is put from the unnamed buffer. The lowercase **p** command puts text either below the current line or after the cursor, depending on whether the buffer contains a partial line or not. The uppercase **P** command puts text either above the current line or before the cursor, again depending on whether the buffer contains a partial line or not.

Yank – y and Y

Syntax: ["*letter*]**y***cursor-movement*
["*letter*]**yy**
["*letter*]**Y**

Function: Copies text in the editing buffer to a named buffer. If no buffer name is specified, text is yanked into the unnamed buffer. If an uppercase *letter* is used, text is appended to the buffer and does not overwrite and destroy the previous contents. When a *cursor-movement* is given as an argument, the delimited text object is yanked. The **Y** and **yy** commands yank a single line, or, if a preceding count is given, multiple lines can be yanked.

Searching

The search commands search either forward or backward in the editing buffer for text that matches a given regular expression.

Search – / and ?

Syntax: **/**[*pattern*]**/**[*offset*]**RETURN**
/[*pattern*]**RETURN**
?[*pattern*]**?**[*offset*]**RETURN**
?[*pattern*]**RETURN**

Function: Searches forward (**/**) or backward (?) for *pattern*. A string is actually a regular expression. The trailing delimiter is not required. If no *pattern* is given, then last *pattern* searched for is used. After the second delimiter, an *offset* may be given, specifying the beginning of a line relative to the line on which *pattern* was found. For example:

/word**/**-

finds the beginning of the line immediately preceding the line containing "word" and the following command:

/word**/**+2

finds the beginning of the line two lines after the line containing "word". See also the *ignorecase* and *magic* options.

Next String – n and N

Syntax: **n**
N

Function: Repeats the last search command. The **n** command repeats the search in the same direction as the last search command. The **N** command repeats the search in the opposite direction of the last search command.

Find Character – f and F

Syntax: **f***char*
F*char*
;
,

Function: Finds character *char* on the current line. The lowercase **f** searches forward on the line; the uppercase **F** searches backward. The semicolon (;) repeats the last character search. The comma (,) reverses the direction of the search.

To Character — t and T

Syntax: t*char*
T*char*
;
,

Function: Moves the cursor up to but not on *char*. The semicolon (;) repeats the last character search. The comma (,) reverses the direction of the search.

Mark — m

Syntax: **m***letter*

Function: Marks a place in the file with a lowercase *letter*. You can move to a mark using the "to mark" commands described below. It is often useful to create a mark, move the cursor, and then delete from the cursor to the mark "a" with the following command:

d´a

To Mark — ´ and `

Syntax: *´letter*
`letter

Function: Move to *letter*. These commands let you move to the location of a mark. Marks are denoted by single lowercase alphabetic characters. Before you can move to a mark, it must first be created with the **m** command. The back quotation mark (`) moves you to the exact location of the mark within a line; the forward quotation mark (´) moves you to the beginning of the line containing the mark. Note that these commands are also legal cursor movement commands.

Exit and Escape Commands

There are several commands that are used to escape from *vi* command mode and to exit the editor. These are described in the following section.

ex Escape – :

Syntax: :

Function: Enters *ex* escape mode to execute an *ex* command. The colon appears on the status line as a prompt for an *ex* command. You then can enter an *ex* command line terminated by either a RETURN or an ESC and the *ex* command will execute. You are then prompted to type RETURN to return to *vi* command mode. During the input of the *ex* command line or during execution of the *ex* command, you may press INTERRUPT to stop what you are doing and return to *vi* command mode.

Exit Editor – ZZ

Syntax: **ZZ**

Function: Exit *vi* and write out the file if any changes have been made. This returns you to the shell from which you started *vi*.

Quit to ex – Q

Syntax: **Q**

Function: Enters the *ex* editor. When you do this, you will still be editing the same file. You can return to *vi* by entering the *vi* command from *ex*.

ex Commands

Entering the colon (:) escape command when in command mode produces a colon prompt on the status line. This prompt is for a command available in the line-oriented editor, *ex*. In general, *ex* commands let you write out or read in files, escape to the shell, or switch editing files.

Many of these commands perform actions that affect the "current" file by default. The current file is normally the file that you named when you started *vi*, although the current file can be changed with the "file" command, **f**, or with the "next" command, **n**. In most respects, these commands are identical to similar commands for the editor, *ed*. All such *ex* commands are aborted by either RETURN or ESC. We shall use RETURN in our examples. Command entry is terminated by typing INTERRUPT.

Command Structure

Most *ex* command names are English words, and initial prefixes of the words are acceptable abbreviations. In descriptions, only the abbreviation is discussed, since this is the most frequently used form of the command. The ambiguity of abbreviations is resolved in favor of the more commonly used commands. As an example, the command **substitute** can be abbreviated **s** , while the shortest available abbreviation for the **set** command is **se.**

Most commands accept prefix addresses specifying the lines in the file that they are to affect. A number of commands also may take a trailing *count* specifying the number of lines to be involved in the command. Counts are rounded down if necessary. Thus, the command "10p" displays the tenth line in the buffer while "move 5" moves the current line after line 5.

Some commands take other information or parameters, stated after the command name. Examples might be option names in a **set** command, such as "set number", a filename in an **edit** command, a regular expression in a **substitute** command, or a target address for a **copy** command. For example:

    1,5 copy 25

A number of commands have variants. The variant form of the command is invoked by placing an exclamation mark (!) immediately after the command name. Some of the default variants may be controlled by options; in this case, the exclamation mark turns off the meaning of the default.

In addition, many commands take flags, including the characters "p" and "l". A "p" or "l" must be preceded by a blank or tab. In this case, the command abbreviated by these characters is executed after the command completes. Since *ex* normally displays the new current line after each change, **p** is rarely necessary. Any number of plus (+) or minus (-) characters may also be given with these flags. If they appear, the specified offset is applied to the current line value before the printing command is executed.

Most commands that change the contents of the editor buffer give feedback if the scope of the change exceeds a threshold given by the **report option.** This feedback helps to detect undesirably large changes so that they may be quickly and easily reversed with the **undo** command. After commands with global effect, you will be informed if the net change in the number of lines in the buffer during this command exceeds this threshold.

Command Addressing

The following specifies the line addressing syntax for *ex* commands:

. The current line. Most commands leave the current line as the last line which they affect. The default address for most commands is the current line, thus "." is rarely used alone as an address.

n The *n*th line in the editor's buffer, lines being numbered sequentially from 1.

$ The last line in the buffer.

% An abbreviation for "1,$", the entire buffer.

+*n* or −*n* An offset, *n* relative to the current buffer line. The forms ".+3" "+3" and "+++" are all equivalent. If the current line is line 100 they all address line 103.

/*pattern***/** or ?*pattern*?
Scan forward and backward respectively for a text matching the regular expression given by *pattern*. Scans normally wrap around the end of the buffer. If all that is desired is to print the next line containing *pattern*, the trailing slash (/) or question mark (?) may be omitted. If *pattern* is omitted or explicitly empty, the string matching the last specified regular expression is located. The forms "RETURN" and "?RETURN" scan using the last named regular expression. After a substitute, "RETURN" and "??RETURN" would scan using that substitute's regular expression.

″ or *´x* Before each nonrelative motion of the current line dot (.), the previous current line is marked with a label, subsequently referred to with two single quotation marks (″). This makes it easy to refer or return to this previous context. Marks are established with the *vi* **m** command, using a single lowercase letter as the name of the mark. Marked lines are later referred to with the following notation:

´x.

where *x* is the name of a mark.

Addresses to commands consist of a series of addresses, separated by a colon (,) or a semicolon (;). Such address lists are evaluated left to right. When addresses are separated by a semicolon (;) the current line (.) is set to the value of the previous addressing expression before the next address is interpreted. If more addresses are given than the command requires, all but the last one or two are ignored. If the command takes two addresses, the first addressed line must precede the second in the buffer. Null address specifications are permitted in a list of addresses, the default in this case is the current line "."; thus ",100" is equivalent to ".,100". It is an error to give a prefix address to a command which expects none.

Command Format

The following is the format for all *ex* commands:

[*address*] [*command*] [!] [*parameters*] [*count*] [*flags*]

All parts are optional depending on the particular command and its options. The following section describes specific commands.

Argument List Commands

The argument list commands allow you to work on a set of files, by remembering the list of filenames that are specified when you invoke *vi*. The **args** command lets you examine this list of filenames. The **file** command gives you information about the current file. The **n** (next) command lets you either edit the next file in the argument list or change the list. And the **rewind** command lets you restart editing the files in the list. All of these commands are described below:

args — The members of the argument list are displayed, with the current argument delimited by brackets. For example, a list might look like this:

file1 file2 [file3] file4 file5

The current file is *file3*.

f — Displays the current filename, whether it has been modified since the last **write** command, whether it is read-only, the current linenumber, the number of lines in the buffer, and the percentage of the buffer that you have edited. In the rare case that the current file is "[Not edited]", this is noted also; in this case you have to use **w!** to write to the file, since the editor is not sure that a **w** command will not destroy a file unrelated to the current contents of the buffer.

f *file* The current filename is changed to *file* which is considered "[Not edited]".

n The next file in the command line argument list is edited.

n! This variant suppresses warnings about the modifications to the buffer not having been written out, discarding irretrievably any changes that may have been made.

n [+*command*] *filelist*
The specified *filelist* is expanded and the resulting list replaces the current argument list; the first file in the new list is then edited. If *command* is given (it must contain no spaces), then it is executed after editing the first such file.

rew The argument list is rewound, and the first file in the list is edited.

rew! Rewinds the argument list discarding any changes made to the current buffer.

Edit Commands

To edit a file other than the one you are currently editing, you will often use one of the variations of the **e** command.

In the following discussions, note that the name of the current file is always remembered by vi and is specified by a percent sign (%). The name of the *previous* file in the editing buffer is specified by a number sign (#).

The edit commands are described below:

e *file* Used to begin an editing session on a new file. The editor first checks to see if the buffer has been modified since the last w command was issued. If it has been, a warning is issued and the command is aborted. The command otherwise deletes the entire contents of the editor buffer, makes the named file the current file, and displays the new filename. After ensuring that this file is sensible, (i.e., that it is not a binary file, directory, or a device), the editor reads the file into its buffer. If the read of the file completes without error, the number of lines and characters read is displayed on the status line. If there were any non-ASCII characters in the file, they are stripped of their non-ASCII high bits, and any null characters in the file are discarded. If none of these errors occurred, the file is considered edited. If the last line of the input file is missing the trailing newline character, it is supplied and a complaint issued. The current line is initially the first line of the file.

e! *file* — This variant form suppresses the complaint about modifications having been made and not written from the editor buffer, thus discarding all changes that have been made before editing the new file.

e *+n file* — Causes the editor to begin editing at line *n* rather than at the first line. The argument *n* may also be an editor command containing no spaces; for example, "+/pattern".

Ctrl-^ — This is a shorthand equivalent for ":e #RETURN", which returns to the previous position in the last edited file. If you do not want to write the file, you should use ":e! #RETURN" instead.

Write Commands

The write commands let you write out all or part of your editing buffer to either the current file or to some other file. These commands are described below:

w *file* — Writes changes made back to *file*, displaying the number of lines and characters written. Normally, *file* is omitted and the buffer is written to the name of the current file. If *file* is specified, text is written to that file. The editor writes to a file only if it is the current file and is edited, or if the file does not exist. Otherwise, you must give the variant form **w!** to force the write. If the file does not exist it is created. The current filename is changed only if there is no current filename; the current line is never changed.

If an error occurs while writing the current and edited file, the editor displays:

No write since last change

even if the buffer had not previously been modified.

w>> *file* — Appends the buffer contents at the end of an existing file. Previous file contents are not destroyed.

w! *name* — Overrides the checking of the normal **write** command, and writes to any file that the system permits.

w !*command*

Writes the specified lines into *command*. Note the difference between

w! *file*

which overrides checks and

w !*cmd*

which writes to a command. The output of this command is displayed on the screen and not inserted in the editing buffer.

Read Commands

The read commands let you read text into your editing buffer at any location you specify. The text you read in must be at least one line long, and can be either a file or the output from a command.

r *file* Places a copy of the text of the given file in the editing buffer after the specified line. If no file is given, the current filename is used. The current filename is not changed unless there is none, in which case the file becomes the current name. If the file buffer is empty and there is no current name, this is treated as an **e** command.

Address 0 is legal for this command and causes the file to be read at the beginning of the buffer. Statistics are given as for the **e** command when the **r** successfully terminates. After an **r** the current line is the last line read.

r !*command* Reads the output of *command* into the buffer after the specified line. A blank or tab before the exclamation mark (!) is mandatory.

Quit Commands

There are several ways to exit *vi*. Some abort the editing session, some write out the editing buffer before exiting, and some warn you if you decide to exit without writing out the buffer. All of these ways of exiting are described below:

q Exits *vi*. No automatic write of the editor buffer to a file is performed. However, *vi* displays a warning message if the file has changed since the last **w** command was issued, and does not quit. *vi* also displays a diagnostic if there are more files in the argument list left to edit.

Normally, you will wish to save your changes, and you should enter a **w** command. If you wish to discard them, enter the **q!** command variant.

q! Quits from the editor, discarding changes to the buffer without complaint.

wq *name* Like a **w** and then a **q** command.

wq! *name* Overrides checking normally made before execution of the **w** command to any file. For example, if you own a file but do not have write permission turned on, the **wq!** allows you to update the file anyway.

x *name* If any changes have been made and not written, writes the buffer out and then quits. Otherwise, it just quits.

Global and Substitute Commands

The global and substitute commands allow you to perform complex changes to a file in a single command. Learning how to use these commands is a must for an experienced *vi* user.

g/*pattern***/***cmds*

The **g** command has two distinct phases. In the first phase, each line matching *pattern* in the editing buffer is marked. Next, the given command list is executed with the current line, dot (.), initially set to each marked line.

The command list consists of the remaining commands on the current input line and may continue to multiple lines by ending all but the last such line with a backslash (\). This multiple-line option will not work from within *vi*, you must switch to *ex* to do it. If *cmds* (or the trailing slash (/) delimiter) is omitted, each line matching *pattern* is displayed.

The **g** command itself may not appear in *cmds*. The options **autoprint** and **autoindent** are inhibited during a global command and the value of the **report** option is temporarily infinite, in deference to a **report** for the entire global. Finally, the context mark (´) or (`) is set to the value of the current line (.) before the global command begins and is not changed during a global command.

The following global commands, most of them substitutions, cover the most frequent uses of the global command.

g/*s1***/p** — This command simply prints all lines that contain the string "s1" .

g/*s1***/s//***s2***/** — This command substitutes the *first* occurrence of "s1" on all lines that contain it with the string "s2".

g/*s1***/s//***s2***/g** — This command substitutes all occurrences of "s1" with the string "s2". This includes multiple occurrences of "s1" on a line.

g/*s1***/s//***s2***/gp** — This command works the same as the preceding example, except that in addition, all changed lines are displayed on the screen.

g/*s1***/s//***s2***/gc** — This command prompts you to confirm that you want to make each substitution of the string "s1" with the string "s2". If you enter a Y , the given substitution is made, otherwise it is not.

g/*s0***/s/***s1***/***s2***/g** — This command marks all those lines that contain the string "s0", and then for those lines only, substitutes all occurrences of the string "s1" with "s2".

g!/*pattern***/***cmds* — This variant form of **g** runs *cmds* at each line not matching *pattern*.

s/ *pattern***/***repl***/***options*

On each specified line, the first instance of text matching the regular expression *pattern* is replaced by the replacement text *repl*. If the **global** indicator option character **g** appears, all instances on a line are substituted. If the **confirm** indication character **c** appears, before each substitution the line to be substituted is printed on the screen with the string to be substituted marked with caret (^) characters. By entering Y , you cause the substitution to be performed; any other input causes no change to take place. After an **s** command, the current line is the last line substituted.

v/*pattern***/***cmds* — A synonym for the **global** command variant **g!**, running the specified *cmds* on each line that does not match *pattern*.

Text Movement Commands

The text movement commands are largely superseded by commands available in *vi* command mode. However, the following two commands are still quite useful:

co *addr flags* — A copy of the specified lines is placed after *addr*, which may be "0". The current line "." addresses the last line of the copy.

[*range*]**m***addr* — The **m** command moves the lines specified by *range* after the line given by *addr*. For example, **m+** swaps the current line and the following line, since the default range is just the current line. The first of the moved lines becomes the current line (dot).

Shell Escape Commands

You will often want to escape from the editor to execute normal XENIX commands. You may also want to change your working directory so that your editing can be done with respect to a different working directory. These operations are described below:

cd *directory* — The specified *directory* becomes the current directory. If no directory is specified, the current value of the *home* option is used as the target directory. After a **cd** , the current file is not considered to have been edited so that write restrictions on preexisting files still apply.

sh — A new shell is created. You may invoke as many commands as you like in this shell. To return to *vi*, enter a Ctrl-D to terminate the shell.

!*command* — The remainder of the line after the exclamation (!) is sent to a shell to be executed. Within the text of *command* , the characters "%" and "#" are expanded as the filenames of the current file and the last edited file and the character "!" is replaced with the text of the previous command. Thus, in particular, "!!" repeats the last such shell escape. If any such expansion is performed, the expanded line is echoed. The current line is unchanged by this command.

If there has been "[No write]" of the buffer contents since the last change to the editing buffer, a diagnostic is displayed before the command is executed as a warning. A single exclamation (!) is displayed when the command completes.

Other Commands

The following command descriptions explain how to use miscellaneous *ex* commands that do not fit into the above categories:

abbr Maps the first argument to the following string. For example, the following command

:abbr rainbow yellow green blue red

maps "rainbow" to "yellow green blue red". Abbreviations can be turned off with the **unabbreviate** command, as in:

:una rainbow

map, map!
Maps any character or escape sequence to an existing command sequence. Characters mapped with **map!** work in both command and insert mode, while characters mapped with **map** work only in command mode. Characters mapped with **map!** cannot be unmapped using **unmap**.

nu Displays each specified line preceded by its buffer line number. The current line is left at the last line displayed. To get automatic line numbering of lines in the buffer, set the *number* option.

preserve The current editor buffer is saved as though the system had just crashed. This command is for use only in emergencies when a **w** command has resulted in an error and you do not know how to save your work.

= Displays the line number of the addressed line. The current line is unchanged.

recover *file*
Recovers *file* from the system save area. The system saves a copy of the editing buffer only if you have made changes to the file, the system crashes, or you execute a **preserve** command. When you use **preserve** , you are notified by mail when a file is saved.

set *argument*
With no arguments, **set** displays those options whose values have been changed from their defaults; with the argument **all**, it displays all of the option values.

Giving an option name followed by a question mark (?) causes the current value of that option to be displayed. The question mark is unnecessary unless the option is a Boolean value. Switch options are given values either with:

set *option*

to turn them on or:

set *nooption*

to turn them off. String and numeric options are assigned with:

set *option*=value

More than one parameter may be given to *set* ; all are interpreted from left to right.

tag *label*
The focus of editing switches to the location of *label.* If necessary, *vi* will switch to a different file in the current directory to find *label*. If you have modified the current file before giving a **tag** command, you must first write it out. If you give another **tag** command with no argument, the previous *label* is used.

Similarly, if you press Ctrl-], *vi* searches for the word immediately after the cursor as a tag. This is equivalent to entering ":tag", the word following the cursor, and then pressing the RETURN key.

The **tags** file is normally created by a program such as **ctags**, and consists of a number of lines with three fields separated by blanks or tabs. The first field gives the name of the tag, the second the name of the file where the tag resides, and the third gives an addressing form which can be used by the editor to find the tag. This field is usually a contextual scan using */pattern/* to be immune to minor changes in the file. Such scans are always performed as if the **nomagic** option was set. The tag names in the tags file must be sorted alphabetically. There are a number of options that can be set to affect the *vi* environment. These can be set with the *ex* **set** command either while editing or immediately after *vi* is invoked in the *vi* start-up file, **.exrc**.

unmap
Unmaps any character or escape sequence that has been mapped using the map command.

The first thing that must be done before you can use *vi*, is to set the terminal type so that *vi* uuderstands how to talk to the particular terminal you are using.

Each time *vi* is invoked, it reads commands from the file named **.exrc** in your home directory. This file normally sets the user's preferred options so that they need not be set manually each time you invoke *vi*. Each of the options is described in detail below.

Options

There are only two kinds of options: switch options and string options. A switch option is either on or off. A switch is turned off by prefixing the word *no* to the name of the switch within a **set** command. String options are strings of characters that are assigned values with the syntax *option=string*. Multiple options may be specified on a line. *vi* options are listed below:

autoindent, ai default: **noai**

Can be used to ease the preparation of structured program text. For each line created by an append, change, insert, open, or substitute operation, *vi* looks at the preceding line to determine and insert an appropriate amount of indentation. To back the cursor up to the preceding tab stop, press Ctrl-D. The tab stops going backward are defined as multiples of the **shiftwidth** option. You cannot backspace over the indent, except by pressing Ctrl-D.

Specially processed in this mode is a line with no characters added to it, which turns into a completely blank line (the whitespace provided for the **autoindent** is discarded). Also, specially processed in this mode are lines beginning with a caret (^) and immediately followed by a Ctrl-D. This causes the input to be repositioned at the beginning of the line, but retains the previous indent for the next line. Similarly, a "0" followed by a Ctrl-D, repositions the cursor at the beginning without retaining the previous indent. **Autoindent** doesn't happen in global commands.

autoprint ap default: **ap**

Causes the current line to be displayed after each *ex* **copy, move,** or **substitute** command. This has the same effect as supplying a trailing "p" to each such command. **Autoprint** is suppressed in globals, and only applies to the last command on a line.

autowrite, aw default: **noaw**

Causes the contents of the buffer to be automatically written to the current file if you have modified it when you give a **next, rewind, tag,** or **!** command, or a Ctrl-^ (switch files) or Ctrl-] (tag go to) command.

beautify, bf default: **nobeautify**

Causes all control characters except tab, newline and formfeed to be discarded from the input. A complaint is registered the first time a backspace character is discarded. **Beautify** does not apply to command input.

directory, dir default: **dir=/tmp**
Specifies the directory in which *vi* places the editing buffer file. If the directory does not have write permission, the editor will exit abruptly when it fails to write to the buffer file.

edcompatible default: **noedcompatible**
Causes the presence or absence of **g** and **c** suffixes on substitute commands to be remembered, and to be toggled on and off by repeating the suffixes. The suffix **r** causes the substitution to be like the tilde (~) command, instead of like the ampersand command (&).

errorbells, eb default: **noeb**
Error messages are preceded by a bell. If possible, the editor always places the error message in inverse video instead of ringing the bell.

hardtabs, ht default: **ht=8**
Gives the boundaries on which terminal hardware tabs are set or on which tabs the system expands.

ignorecase, ic default: **noic**
Maps all uppercase characters in the text to lowercase in regular expression matching. In addition, all uppercase characters in regular expressions are mapped to lowercase except in character class specifications enclosed in brackets.

lisp default: **nolisp**
Autoindent indents appropriately for LISP code, and the ( ) { } [[and]] commands are modified to have meaning for LISP.

list default: **nolist**
All printed lines are displayed, showing tabs and end-of-lines.

magic default: **magic**
If **nomagic** is set, the number of regular expression metacharacters is greatly reduced, with only up-arrow (^) and dollar sign ($) having special effects. In addition, the metacharacters "~" and "&" in replacement patterns are treated as normal characters. All the normal metacharacters may be made **magic** when **nomagic** is set by preceding them with a backslash (\).

mesg default: **nomesg**
Causes write permission to be turned off to the terminal while you are in visual mode, if **nomesg** is set. This prevents people writing to your screen with the XENIX **write** command and scrambling your screen as you edit.

number, n default: **nonumber**
Causes all output lines to be printed with their line numbers.

open default: **open**
If set to **noopen**, the commands **open** and **visual** are not permitted from *ex*. This is set to prevent confusion resulting from accidental entry to open or visual mode.

optimize, opt default: **optimize**
Output of text to the screen is expedited by setting the terminal so that it does not perform automatic carriage returns when displaying more than one line of output, thus greatly speeding output on terminals without addressable cursors when text with leading whitespace is printed.

paragraphs, para default: **para=**IPLPPPQPP TPbp
Specifies paragraph delimiters for the { and } operations. The pairs of characters in the option's value are the names of the nroff macros that start paragraphs.

prompt default: **prompt**
ex input is prompted for with a colon (:). If **noprompt** is set, when *ex* command mode is entered with the **Q** command, no colon prompt is displayed on the status line.

redraw default: **noredraw**
The editor simulates (using great amounts of output), an intelligent terminal on a dumb terminal. Useful only at very high speed.

remap default: **remap**
If on, mapped characters are repeatedly tried until they are unchanged. For example, if *o* is mapped to *O* and *O* is mapped to *I*, *o* will map to *I* if remap is set, and to *O* if **noremap** is set.

report default: **report=5**
Specifies a threshold for feedback from commands. Any command that modifies more than the specified number of lines will provide feedback as to the scope of its changes. For global commands and the undo command, the net change in the number of lines in the buffer is presented at the end of the command. Thus notification is suppressed during a **g** command on the individual commands performed.

scroll default: **scroll=½ window**
Determines the number of logical lines scrolled when Ctrl-D is received from a terminal input in command mode, and the number of lines displayed by a command mode **z** command (double the value of *scroll*).

sections default: **sections**=SHNHH HU
Specifies the section macros for the [[and]] operations. The pairs of characters in the option's value are the names of the nroff macros that start paragraphs.

shell, sh default: **sh**=/bin/sh
Gives the pathname of the shell forked for the shell escape command (!), and by the **shell** command. The default is taken from SHELL in the environment, if present.

shiftwidth, sw default:sw=8
Gives the width of a software tab stop, used in reverse tabbing with Ctrl-D when using **autoindent** to append text, and by the shift commands.

showmatch, sm default: **nosm**
When a) or } is typed, moves the cursor to the matching (or { for one second if this matching character is on the screen.

tabstop, ts default: **ts**=8
The editor expands tabs in the input file to be on *n* boundaries for the purposes of display.

taglength, tl default: **tl**=0
The first *n* characters in a tag name are significant, but all others are ignored. A value of zero (the default) means that all characters are significant.

tags default: **tags**=tags /usr/lib/tags
A path of files to be used as tag files for the **tag** command. A requested tag is searched for in the specified files, sequentially. By default, files named *tag* are searched for in the current directory and in **/usr/lib**.

term default=value of shell TERM variable
The terminal type of the output device.

terse default: **noterse**
Shorter error diagnostics are produced for the experienced user.

warn default: **warn**
Warn if there has been "[No write since last change]" before a shell escape command (!).

window default: **window** = speed dependent
This specifies the number of lines in a text window. The default is 8 at slow speeds (600 baud or less), 16 at medium speed (1200 baud), and the full screen (minus one line) at higher speeds.

w300, w1200, w9600
These are not true options but set **window** (above) only if the speed is slow (300), medium (1200), or high (9600), respectively.

wrapscan, ws default: **ws**
Searches, using the regular expressions in addressing, will wrap around past the end of the file.

wrapmargin, wm default: **wm=0**
Defines the margin for automatic insertion of newlines during text input. A value of zero specifies no wrap margin.

writeany, wa default: **nowa**
Inhibits the checks normally made before **write** commands, allowing a write to any file that the system protection mechanism will allow.

Regular Expressions

A regular expression specifies a set of strings of characters. A member of this set of strings is said to be "matched" by the regular expression. *vi* remembers two previous regular expressions: the previous regular expression used in a substitute command and the previous regular expression used elsewhere, referred to as the previous *scanning* regular expression. The previous regular expression can always be referred to by a null regular expression: e.g., "//" or "??".

The regular expressions allowed by *vi* are constructed in one of two ways depending on the setting of the **magic** option. The *ex* and *vi* default setting of **magic** gives quick access to a powerful set of regular expression metacharacters. The disadvantage of **magic** is that the user must remember that these metacharacters are **magic** and precede them with the backslash (\) to use them as "ordinary" characters. With **nomagic** set, regular expressions are much simpler, there being only two metacharacters. The power of the other metacharacters is still available by preceding the now ordinary character with a "\". Note that "\" is always a metacharacter. In this discussion, the magic option is assumed. With **nomagic** , the only special characters are the caret (^) at the beginning of a regular expression, the dollar sign ($) at the end of a regular expression, and the backslash (\). The tilde (~) and the ampersand (&) also lose their special meanings related to the replacement pattern of a substitute.

The following basic constructs are used to construct **magic** mode regular expressions.

char An ordinary character matches itself. Ordinary characters are any characters except a caret (^) at the beginning of a line, a dollar sign ($) at the end of line, a star (*) as any character other than the first, and any of the following characters:

. \ [~

These characters must be preceded by a backslash (\) if they are to be treated as ordinary characters.

^ At the beginning of a pattern, forces the match to succeed only at the beginning of a line.

$ At the end of a regular expression, forces the match to succeed only at the end of the line.

. Matches any single character except the newline character.

\< Forces the match to occur only at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> Similar to "\<", but matching the end of a "word", i.e., either the end of the line or before a character which is not a letter, a digit, or the underline character.

[*string*]
Matches any single character in the class defined by *string*. Most characters in *string* define themselves. A pair of characters separated by a dash (–) in *string* defines the set of characters between the specified lower and upper bounds, thus "[a–z]" as a regular expression matches any single lowercase letter. If the first character of *string* is a caret (^) then the construct matches those characters which it otherwise would not. Thus "[^a–z]" matches anything but a lowercase letter or a newline. To place any of the characters caret, left bracket, or dash in *string* they must be escaped with a preceding backslash (\).

The concatenation of two regular expressions first matches the leftmost regular expression and then the longest string that can be recognized as a regular expression. The first part of this new regular expression matches the first regular expression and the second part matches the second. Any of the single character matching regular expressions mentioned above may be followed by a star () to form a regular expression that matches zero or more adjacent occurrences of the characters matched by the prefixing regular expression. The tilde (~) may be used in a regular expression to match the text that defined the replacement part of the last s command. A regular expression may be enclosed between the sequences "\(" and "\)" to remember the text matched by the enclosed regular expression. This text can later be interpolated into the replacement text using the following notation:

digit

where *digit* enumerates the set of remembered regular expressions.

The basic metacharacters for the replacement pattern are the ampersand (&) and the tilde (~); these are given as "\&" and "\~" when **nomagic** is set. Each instance of the ampersand is replaced by the characters matched by the regular expression. In the replacement pattern, the tilde stands for the text of the previous replacement pattern.

Other metasequences possible in the replacement pattern are always introduced by a backslash (\). The sequence "*n*" is replaced by the text matched by the *n*th regular subexpression enclosed between "\(" and "\)". When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of "\(" starting from the left. The sequences "\u" and "\l" cause the immediately following character in the replacement to be converted to uppercase or lowercase, respectively, if this character is a letter. The sequences "\U" and "\L" turn such conversion on, either until "\E" or "\e" is encountered, or until the end of the replacement pattern.

Limitations

When using *vi,* you should note the following limits:

250K lines in a file

510 characters per line

256 characters per global command list

128 characters per filename

128 characters in the previous inserted and deleted text

100 characters in a shell escape command

63 characters in a string valued option

30 characters in a tag name

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

The **/usr/lib/ex3.7preserve** program can be used to restore *vi* buffer files that were lost as a result of a system crash. The program searches the **/tmp** directory for *vi* buffer files and places them in the directory **/usr/preserve**. The owner can retrieve these files using the **−r** option.

The **/usr/lib/ex3.7preserve** program must be placed in the system startup file, **/etc/rc**, before the command that cleans out the **/tmp** directory. See the XENIX *Operations Guide* for more information on **/etc/rc.**

Name

vsh - menu driven visual shell

Syntax

vsh

Description

vsh is a highly interactive, visually oriented shell which eases many XENIX activities. The *vsh* features both standard and customizable XENIX command menus and on-line help. The *vsh* displays information and menus in windows on the screen. To enter *vsh,* simply enter:

vsh

from a shell prompt. *vsh* can also be made a user's default shell by changing their shell entry in **/etc/passwd** (the last colon-separated field). Help is available from all menus by typing the question mark character.

The very last line of the screen is a status line. The status line displays the current pathname, the date, time and operating system name. If you have new mail, the status line will indicate so. Above the status line is the message line, which displays messages, error or otherwise, from *vsh.*

A command menu is displayed at the bottom of the screen. The standard menu contains a range of commonly used XENIX commands. Above the command menu is the output window. This window contains a scrolling display of the output from commands. This window is not visible at start-up, but is displayed while running certain commands such as '='.

In the top of the screen is a window with a listing of the current working directory. To alter the size of this window, use the *Window* command from the main command menu. Items in the listing window may be selected using standard key commands (q.v.). Two special key commands are used with the listing window. The equals sign '=' ('SHOW') key, displays the contents of the currently selected file or directory. The minus sign '-' ('GOAWAY') key, returns you to the listing window.

Commands may be invoked in one of two ways. A command can be selected by pressing the first letter of its name. Alternatively, press the space bar. Each time the space bar is pressed, the next menu item is highlighted. This highlighting indicates that the command has been selected. Backspace moves to the previous selection.

Once a command is selected, press the return key. A menu is displayed which gives the valid arguments for the particular command. The default choice is shown in parentheses, e.g.:

recursive: Yes (No)

To send the output to another program, you may enter a vertical bar in the "output:" field of the commands' menu.

When the menu is filled in, press RETURN to start the command.

Main Menu Commands

The following menu options are available from the standard main menu. Certain sub-commands are available under the Options selection. These are described in the next section.

Copy
: Copy a file to a new file. Copy the contents of a directory to a new directory.

Delete
: Delete a file or directory.

Edit
: Invoke an editor for a file. Default is the visual editor vi(C).

Help
: Get help on diverse topics. A menu is displayed at the bottom of the screen of available help topics.

Mail
: Send or read XENIX mail.

Name
: Rename a directory or file.

Options
: Perform various commands. See OPTIONS section.

Print
: Print file or files on systems' lineprinter.

Quit
: Quit the visual shell.

Run
: Run a specified XENIX command or applications program.

View
: View a specified file or directory listing. This file or directory listing will be displayed in the upper window. Use the *vsh* scrolling commands to move around (see KEY COMMANDS Section).

Window
: Reset upper window 'redraw' characteristics and height.

Options Subcommand

The Options selection on the main menu has several important commands grouped under the selections Directory, Filesystem, Output, and Permissions. These are as follows:

Directory

Make
: Make a directory under current working directory.

Usage
: Display disk usage by number of blocks in current working directory.

Filesystem

Create
: Create a filesystem.

FilesCheck
: Check file system consistency.

Mount
: Mount a file system on a specified mount-point.

SpaceFree
: Report number of disk blocks available on all or some mounted file systems.

Unmount
: Unmount specified file system if it is not currently busy.

Output

VShell
Echo vsh commands in output window (default).

XENIX
Echo actual XENIX commands in output window. For instance, if running "Options Filesystem FilesCheck", the command *fsck* will be displayed in the output window if "Options Output Xenix" is set.

Permissions

Change permissions on a file or directory.

Key Commands

The following keyboard commands allow editing of menus and fields, and give access to various vsh features.

<Ctrl-E>
Move the cursor up one line.

<Ctrl-X>
Move the cursor down one line.

<Ctrl-S>
Move the cursor left one character.

<Ctrl-D>
Move the cursor right one character.

<Ctrl-R><Ctrl-E>
Scroll page up.

<Ctrl-R><Ctrl-X>
Scroll page down.

<Ctrl-R><Ctrl-S>
Scroll page left.

<Ctrl-R><Ctrl-D>
Scroll page right.

<Ctrl-Q>
Home. Go to start of menu.

<Ctrl-Z>
End. Go to the end of menu.

<Ctrl-C>
Cancel. Stop present operation and return to the main command menu.

<RETURN>
Start the present command.

<TAB>, <Ctrl-I>, or <Ctrl-A>
Move to and select entire contents of next field in command line.

<SPACE>
Select next item in menu.

<BACKSPACE> or <Ctrl-H>
Select previous menu item. In editing command lists, deletes character. Replacement text may then be typed.

<Ctrl-Y> or <DEL>
Delete selected character.

<Ctrl-L>
Move to next character to right of current cursor position.

<Ctrl-K>
Move to next character to left of current cursor position.

<Ctrl-P>
Move to next word to right of current cursor position.

<Ctrl-O>
Move to next word to left of current cursor position.

? Help. Request information about the selected command or command in progress at the time of the request.

= Show. Display sub-directory listings and text files in directory listings. Display submenus for commands in main menu.

\- Goaway. Return listing window to current or parent directory after a show command.

@ Display the Modify menu.

! Redraw the screen.

| Display filter menu.

Files

menu.def	standard menu definition file.
.mnu	extension for customized command menus.
/usr/lib/vsh/VSHELL.HPP	help file
/usr/lib/vsh/VSHELL.HPT	yet another help file

Name

vmstat – Reports virtual memory statistics.

Description

This command is available only in XENIX-386. If you have XENIX-386, see your *Release Notes* for the complete version of this reference page.

Name

wait – Awaits completion of background processes.

Syntax

wait

Description

Waits until all background processes started with an ampersand (**&**) have finished, and reports on abnormal terminations.

Because the *wait*(S) system call must be executed in the parent process, the shell itself executes *wait*, without creating a new process.

See Also

sh(C)

Notes

Not all the processes of a pipeline with three or more stages are children of the shell, and thus cannot be waited for.

Name

wall – Writes to all users.

Syntax

/etc/wall

Description

wall reads a message from the standard input until an end-of-file. It then sends this message to all users currently logged in preceded by "Broadcast Message from ...". *wall* is used to warn all users, for example, prior to shutting down the system.

The sender should be super-user to override any protections the users may have invoked.

Files

/dev/tty*

See Also

mesg(C), write(C)

Diagnostics

Cannot send to ...	The open on a user's tty file has failed.

Name

wc – Counts lines, words and characters.

Syntax

wc [**–lwc**] [names]

Description

wc counts lines, words and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or newlines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **–lwc**.

When *names* are specified on the command line, they are printed along with the counts.

Name

what – Identifies files.

Syntax

what files ...

Description

what searches the given files for all occurrences of the pattern @(#) and prints out what follows until the first tilde (~), greater-than sign (>), new-line, backslash (\) or null character. The SCCS command *get*(CP) substitutes this string as part of the @(#) string.

For example, if the shell procedure in file **print** contains

```
# @(#)this is the print program
# @(#)syntax: print [files]
pr $* | lpr
```

then the command

```
what print
```

displays the name of the file **print** and the identifying strings in that file:

```
print:
        this is the print program
        syntax: print [files]
```

what is intended to be used with the *get*(CP) command, which automatically inserts identifying information, but it can also be used where the information is inserted manually.

See Also

admin(CP), get(CP)

Name

who – Lists who is on the system.

Syntax

who [**−uTHldtasq**] [file]

who am i

who am I

Description

who can list the user's name, terminal line, login time, and the elapsed time since activity occurred on the line; it also lists the process ID of the command interpreter (shell) for each current XENIX system user. It examines the **/etc/utmp** file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

who with the **am i** or **am I** option identifies the invoking user.

Except for the default **−s** option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

−u This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process ID of the user's shell. The *comment* is the comment field. It can contain information about where the terminal is located, the telephone number of the dataset, the type of terminal if hard-wired, etc.

−T This option is the same as the **−u** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A plus character (**+**) appears if the terminal is writable by anyone; a minus character (**−**) appears if it is not. **Root** can write to all lines having a plus character (**+**) or a minus character (**−**) in the *state* field. If a bad line is encountered, a question mark (**?**) is displayed.

−l This option lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

−H This option displays column headings above the regular output.

−q This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

−d This option displays all processes that have expired and have not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(S)), of the dead process. This can be useful in determining why a process terminated.

−t This option indicates the last change to the system clock (via the *date*(C) command) by **root**. See *su*(C).

−a This option processes the **/etc/utmp** file or the named *file* with all options turned on.

−s This option is the default and lists only the *name, line,* and *time* fields.

Files

/etc/utmp
/etc/wtmp

See Also

date(C), login(C), mesg(C), su(C), utmp(F), wait(S)

Name

whodo – Determines who is doing what.

Syntax

/etc/whodo

Description

whodo produces merged, reformatted, and dated output from the *who*(C) and *ps*(C) commands.

See Also

ps(C), who(C)

Name

write – Writes to another user.

Syntax

write user [tty]

Description

write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *your-logname your-tty* ...

The recipient of the message should write back at this point. Communication continues until an end-of-file is read from the terminal or an interrupt is sent. At that point, *write* displays:

(end of message)

on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *tty* argument may be used to indicate the appropriate terminal.

Permission to write may be denied or granted by use of the *mesg*(C) command. At the outset, writing is allowed. Certain commands, in particular *nroff*(CT) and *pr*(C), disallow messages in order to prevent messy output.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first write to another user, wait for him or her to write back before starting to send. Each party should end each message with a distinctive signal (**(o)** for "over" is conventional), indicating that the other may reply; **(oo)** for "over and out" is suggested when conversation is to be terminated.

Files

/etc/utmp	To find user
/bin/sh	To execute !

See Also

mail(C), mesg(C), who(C)

Name

xargs – Constructs and executes commands.

Syntax

xargs [flags] [command [initial-arguments]]

Description

xargs combines the fixed *initial-arguments* with arguments read from the standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for using the shell $PATH variable. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or newlines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted: Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings, a backslash (****) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (exception: see **–i** flag). Flags **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **–l** vs. **–n**), the last flag has precedence. *Flag* values are:

–l*number* *Command* is executed for each *number* lines of nonempty arguments from the standard input. This is instead of the default single line of input for each *command*. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first newline *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next nonempty line. If *number* is omitted, 1 is assumed. Option **–x** is forced.

−i*replstr* Insert mode: *command* is executed for each line from the standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **−x** is also forced. **{}** is assumed for *replstr* if not specified.

−n*number* Executes *command* , using as many standard input arguments as possible, up to the *number* of arguments maximum. Fewer arguments are used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **−x** is also coded, each *number* of arguments must fit in the *size* limitation, or *xargs* terminates execution.

−t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

−p Prompt mode: The user is prompted whether to execute *command* at each invocation. Trace mode (**−t**) is turned on to display the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything), will execute the command; anything else, including a carriage return, skips that particular invocation of *command*.

−x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **−x** is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.

−s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **−s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

−e*eofstr* *Eofstr* is taken as the logical end-of-file string. Underscore (_) is assumed for the logical **EOF** string if **−e** is not coded. **−e** with no *eofstr* coded turns off the logical **EOF** string capability (underscore is taken literally). *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

xargs terminates if it either receives a return code of **−1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(C)) with an appropriate value to avoid accidentally returning with **−1**.

Examples

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is prompted to enter which files in the current directory are to be printed and prints them one at a time:

```
ls | xargs -p -l lpr
```

Or many at a time:

```
ls | xargs -p -l | xargs lpr
```

The following will execute *diff*(C) with successive pairs of arguments originally entered as shell arguments:

```
echo $* | xargs -n2 diff
```

Name

yes – Prints string repeatedly.

Syntax

yes [string]

Description

yes repeatedly outputs "y", or if a single string argument is given, *arg* is output repeatedly. The command will continue indefinitely unless aborted. Useful in pipes to commands that prompt for input and require a "y" response for a yes. In this case, *yes* terminates when the command it pipes to terminates, so that no infinite loop occurs.

Contents

Miscellaneous (M)

passwd	The password file.
profile	Sets up an environment at login time.
setclock	Sets system real time clock.
sxt	Pseudo-device driver.
systemid	The Micnet system identification file.
systty	System maintenance device.
termcap	Terminal capability data base.
terminals	List of supported terminals.
terminfo	Terminal capability data base.
termio	General terminal interface.
top, top.next	The Micnet topology files.
tty	General terminal interface.
ttys	Login terminals file.
tz	Time zone variable.
ungetty	Suspends/restarts a *getty* process.
utmp, wtmp	Formats of utmp and wtmp entries.

Name

intro – Introduction to miscellaneous features and files.

Description

This section contains miscellaneous information useful in maintaining the system. Included are descriptions of files, devices, tables and programs that are important in maintaining the entire system.

Name

aliases, aliases.hash, maliases, maliases.hash, faliases – Micnet aliasing files.

Description

These files contain the alias definitions for a Micnet network. Aliases are short names or abbreviations that may be used in the *mail* command to refer to specific machines or users in a network. Aliasing allows a complex combination of site, machine, and user names to be represented by a single name.

The **aliases, maliases,** and **faliases** files each define a different type of alias. The **aliases** file defines the standard aliases which are names for specific systems and users and, in some case, for commands. The **maliases** file defines machine aliases, names, and paths for specific systems. The **faliases** file defines forwarding aliases which are temporary names for forwarding mail intended for one system or user to another.

The **aliases.hash** file is the hashed version of the **aliases** file created by the *aliashash* command. The file is used by the *mail* command to resolve all standard aliases and is identical to the **aliases** file except for a hash table at the beginning of the file. The hash table allows for more efficient access to the entries in the file. The **aliases** file need only be present to generate the **aliases.hash** file. The **aliases** file is not required to run the network.

The **maliases.hash** file is the hashed version of the **maliases** file. It is an optional file created by executing the following command:

/usr/lib/mail/aliashash /usr/lib/mail/maliases

If the **maliases.hash** file is created, **maliases** is no longer necessary to run the network. If the number of machines in the network is large, and particularly if several types of networks are in use, it is recommended that the **maliases** file be hashed. In such a network, the configuration is no longer homogeneous, aliases are likely to be fairly complex and machine aliases are likely to differ between machines. The use of machine aliases allows the standard alias file to be identical on all machines in the network. In such an environment, *netutil* can only generate network files that can be used as a starting point. The rest of the network maintenance should be done manually with a text editor.

Each file contains zero or more lines. If hashing is to be performed, at least one alias is required. Each line lists the alias and its meaning. The alias meaning can have site, machine, and user login names and other aliases (its exact composition depends on the type of alias). A colon (:) separating the alias and meaning is required.

In the **aliases** file, a line can have the forms:

alias:[[site!]machine:]user[,[[site!]machine:]user]...

alias:[[site!]machine:]command-pipeline

alias:error-message

Site and *machine* are the site and machine names of the system to which the user belongs or on which the specified command is to be executed. The site and machine names must end with an exclamation mark (!) or colon (:) respectively, and must be defined in a **systemid** file. A machine alias may be used in place of a site and machine name if it is followed by a question mark.

User is a user login name or another alias. User names in a list must be separated by commas. A newline may immediately follow a comma. Spaces and tabs are allowed, but only immediately before or after a comma or newline.

Command-pipeline is any valid command (with necessary arguments) preceded by a pipe symbol (|) and enclosed in double quotation marks. Spaces may separate the command and arguments, but there must be no space between the first double quotation mark and the pipe symbol.

Error-message is any sequence of letters, numbers, and punctuation marks (except a double quotation mark), preceded by a number sign (#) and enclosed in double quotation marks.

In the **faliases** file, each line can have the same form as lines in the **aliases** file except that no more than one user name can be given for any one alias. To prevent alias expansion on a remote machine, the meaning should be escaped with "\\", as in:

foo: mach?\\foo

Failure to do the escape may result in an infinite forwarding loop. If this happens and the loop does not invoke a **uucp** connection, looping will be detected, and the mail will be returned to the sender.

The **alias.hash** file has already been searched at this point. If there is no explicit machine given as part of the meaning, the recipient will be assumed to be local. After forward aliasing is complete, machine aliasing is performed as necessary.

In the **maliases** file, a line has the form:

alias:[[site!]machine:]...

Site and *machine* are the site and machine names for a specific network and system. Multiple site and machine names direct messages along the specified path of systems. If no site or machine name is given, the alias is ignored.

Before the *mail* program sends a message, it searches the **aliases.hash**, **faliases**, and **maliases** files to see if any of the names given with the command are aliases. Each file is searched in turn (**aliases.hash**, **faliases**, then **maliases**) and if a match is found, the alias is replaced with its meaning. If no match is found, the name is assumed to be the valid login name of a user on that machine. The search in the **aliases.hash** file continues until all aliases have been replaced, so it is possible for several replacements to occur for a single name. Alias loops are now detected. If a loop exists, any recipients involved in the alias loop are dropped from the mail recipient list, and an error message is displayed. The **faliases** file is searched once, from beginning to end, even if it is empty. The **maliases** file is searched only if the alias contains a machine alias.

When an alias is a user or a list of users, the *mail* command sends the message to each user in the list. When it is a command-pipeline, the *mail* command starts execution of the command on the specified machine and sends the message as input. When the alias is an error-message, the *mail* command ignores the message and instead, displays the alias and its meaning at the standard error.

In all files, any line beginning with a number sign (#) is considered a comment and is ignored.

As a special feature, any alias that contains a site name as the first component of its meaning is automatically prepended with the machine alias **uucp?**. This alias may be explicitly defined in the **maliases** file to help direct mail between networks to the system performing the *uucp* link.

Directives

Though alias directives are never included in an alias expansion, they can be used to restrict the expansion to a class of users, forward the unexpanded alias to another machine, or produce error messages. An **aliases** file may include directives of the form:

testalias: $xalaska, mikem, georger, terih

sams: "$e ambiguous, use samst or samsm"

Fields on the right-hand side of an alias (after the colon) that begin with a dollar sign ($) character, are alias directives. Fields containing any blanks or tabs must be enclosed in quotes. The directive must precede all normal right-hand fields as shown in the example above. The character following the dollar sign ($) specifies the directive type:

$n <real name or description>

$x <machine>

$e <error message>

$p <permissions>

$r <restrictions>

None of the above directives are currently supported in **/usr/lib/mail/faliases**. Only the $e is supported in **/usr/lib/mail/maliases** and **maliases.hash.** Unrecognized directives do not create error messages and are treated as if they do not exist. The above directives are described in detail as follows:

$n For a user alias, this field should contain the full real name of the user associated with the alias. For a group alias, a description of the group should be given.

$x Causes the alias to be forwarded, unexpanded, to the machine specified in this field. White space is only allowed immediately following the $x. Since machine aliasing will be performed, the appropriate machine alias must exist in the **maliases** file.

$e This field contains an error message to be printed. The left side of the alias will be removed from the list of users to be aliased. An alternate form of $e is #.

$p This field contains the character star (*) or a string of upper and lowercase alphabetic characters. Each character indicates that the user on the left-hand side of the alias belongs to a special "class" of users. The star (*) character implies membership in all such classes.

$r This field contains a string of upper and lower case alphabetic characters, each character indicating a "class" of users to be granted expansion permision. The absence of a $r field means that any user can expand the alias. If the $r field exists, expansion is only allowed if:

1) the user requesting expansion has a $p field and it contains one or more of the charaters found in the $r field.

2) the user has a $p field and it contains a "*".

3) the real user ID is 0 (super user).

If expansion is not allowed, no error messages result; the alias in question is treated as if it were not present.

To send mail delivery problems to root, the following alias could be used:

network: "$n the network mail recipient," root

To forward a group alias called *testalias* to a machine called *alaska* and expand it there, the following alias may be used:

testalias: $xalaska, mikem, georger, terih

Files

/usr/lib/mail/aliases

/usr/lib/mail/aliases.hash

/usr/lib/mail/maliases

/usr/lib/mail/faliases

/usr/lib/mail/maliases.hash

See Also

aliashash(M), netutil(C), systemid(M), top(M)

Name

aliashash – Micnet alias hash table generator.

Syntax

aliashash [**-v**] [**-o** output-file] [input-file]

Description

The *aliashash* command reads the *input-file* and generates an *output-file* containing a hash table of alias definitions for a Micnet network. The *input-file* must name a file containing alias definitions in the form described for the **aliases** file (see *aliases*(M)). If the **−o** option is not used to specify an *output-file*, the command creates a file with the same name as the *input-file* but with **.hash** appended to it. If no *input-file* is given, the command reads the file named **/usr/lib/mail/aliases** and creates the file named **/usr/lib/mail/aliases.hash**.

If invoked with the **−v** option, the command lists information about the hash table.

The *output-file* will contain both the alias definitions given in the *input-file* and the new hash table. The hash table appears at the beginning of the file and is separated from the alias definitions by a blank line. The hash table has three or more lines. The first line is:

    #<hash>

The second line has 4 entries: the bytes per table entry, the maximum number of items per hash value, the number of entries in the table, and the offset (in bytes) from the beginning of the file to the beginning of the alias definitions.

The next lines (up to the end of the hash table) contain the hash table entries. Each line has 8 entries (separated by spaces) and each entry has 2 fields. The first field (1 byte) is a checksum (represented as a printable character); the second field is a pointer (in bytes) to the alias definition. The pointer is represented as a hexadecimal number with leading blanks if necessary and is always relative to the start of the definitions.

The *aliashash* command is normally invoked by the **install** option of the *netutil* command. If the alias definitions of a network must be changed, the definitions in the **aliases** file should be changed and a new **aliases.hash** file created using the *aliashash* command. The new **aliases.hash** file must then be copied to all other computers in the network.

Files

/usr/lib/mail/aliashash
/usr/lib/mail/aliases
/usr/lib/mail/aliases.hash
/usr/lib/mail/maliases.hash

See Also

aliases(M), netutil(C)

Warning

Do not use the *aliashash* command to create the **aliases.hash** file while the network is running. If necessary, create a temporary output file, **aliases.hash-** , using the **-o** option, then enter:

```
mv aliases.hash- aliases.hash
```

This will prevent disruption of the network.

Name

ascii – Map of the ASCII character set.

Description

ascii is a map of the 7-bit ASCII character set. It lists both octal and hexadecimal equivalents of each character. It contains:

Octal							
000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 '
050 (	051)	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

Hexadecimal							
00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

The extended 8-bit ASCII character set is shown here, again with the octal and hexadecimal value of each character. The *mapchan*(C) utility allows access to these characters. Display of these characters is dependent on the capabilities of the hardware device. (A ▒ indicates an unassigned character.)

Octal							
200 ▒	201 ▒	202 ▒	203 ▒	204 ind	205 nel	206 ssa	207 esa
210 hts	211 htj	212 vts	213 pld	214 plu	215 ri	216 ss2	217 ss3
220 dcs	221 pu1	222 pu2	223 sts	224 cch	225 mw	226 spa	227 epa
230 ▒	231 ▒	232 ▒	233 csi	234 st	235 osc	236 pm	237 apc
240 nbsp	241 ¡	242 ¢	243 £	244 ¤	245 ¥	246 ¦	247 §
250 ¨	251 ©	252 ª	253 «	254 ¬	255 shy	256 ®	257 ¯
260 °	261 ±	262 ²	263 ³	264 ´	265 µ	266 ¶	267 •
270 ,	271 ¹	272 º	273 »	274 ¼	275 ½	276 ¾	277 ¿
300 À	301 Á	302 Â	303 Ã	304 Ä	305 Å	306 Æ	307 Ç
310 È	311 É	312 Ê	313 Ë	314 Ì	315 Í	316 Î	317 Ï
320 Đ	321 Ñ	322 Ò	323 Ó	324 Ô	325 Õ	326 Ö	327 ▒
330 Ø	331 Ù	332 Ú	333 Û	334 Ü	335 Ý	336 Þ	337 ß
340 à	341 á	342 â	343 ã	344 ä	345 å	346 æ	347 ç
350 è	351 é	352 ê	353 ë	354 ì	355 í	356 î	357 ï
360 đ	361 ñ	362 ò	363 ó	364 ô	365 õ	366 ö	367 ▒
370 ø	371 ù	372 ú	373 û	374 ü	375 ý	376 þ	377 ÿ

Hexadecimal							
80 ▒	81 ▒	82 ▒	83 ▒	84 ind	85 nel	86 ssa	87 esa
88 hts	89 htj	8a vts	8b pld	8c plu	8d ri	8e ss2	8f ss3
90 dcs	91 pu1	92 pu2	93 sts	94 cch	95 mw	96 spa	97 epa
98 ▒	99 ▒	9a ▒	9b csi	9c st	9d osc	9e pm	9f apc
a0 nbsp	a1 ¡	a2 ¢	a3 £	a4 ¤	a5 ¥	a6 ¦	a7 §
a8 ¨	a9 ©	aa ª	ab «	ac ¬	ad shy	ae ®	af ¯
b0 °	b1 ±	b2 ²	b3 ³	b4 ´	b5 µ	b6 ¶	b7 •
b8 ,	b9 ¹	ba º	bb »	bc ¼	bd ½	be ¾	bf ¿
c0 À	c1 Á	c2 Â	c3 Ã	c4 Ä	c5 Å	c6 Æ	c7 Ç
c8 È	c9 É	ca Ê	cb Ë	cc Ì	cd Í	ce Î	cf Ï
d0 Đ	d1 Ñ	d2 Ò	d3 Ó	d4 Ô	d5 Õ	d6 Ö	d7 ▒
d8 Ø	d9 Ù	da Ú	db Û	dc Ü	dd Ý	de Þ	df ß
e0 à	e1 á	e2 â	e3 ã	e4 ä	e5 å	e6 æ	e7 ç
e8 è	e9 é	ea ê	eb ë	ec ì	ed í	ee î	ef ï
f0 đ	f1 ñ	f2 ò	f3 ó	f4 ô	f5 õ	f6 ö	f7 ▒
f8 ø	f9 ù	fa ú	fb û	fc ü	fd ý	fe þ	ff ÿ

Files

/usr/pub/ascii

Name

autoboot – Automatically boots the system.

Description

The system can be set up to go through the *boot* stages automatically (as defined in **/etc/default/boot** when the computer is turned on (booted), provided no key is pressed at the *boot*(HW) prompt.

If boot times out and LOADXENIX=YES, then XENIX is passed the word "auto" in its boot string and *init*(M), *fsck*(C), and *asktime*(C) are passed an **-a** flag.

The *autoboot* procedure checks the file **/etc/default/boot** for instructions on autobooting:

LOADXENIX=YES or NO	Whether or not *boot*(HW) times out and loads XENIX. *boot* looks for this variable in the **/etc/default/boot** file on its default device.
FSCKFIX=YES or NO	Whether or not *fsck*(C) fixes any root system problems by itself. If the variable is set at YES, then *fsck*(C) is run on the root filesystem with the **-rr** flag.
MULTIUSER=YES or NO	Whether or not *init*(M) invokes *sulogin* or proceeds to multiuser mode.
PANICBOOT=YES or NO	Whether or not the system reboots after a panic(). This variable is read from **/etc/default/boot** by *init*.
RONLYROOT=YES or NO	Whether or not the root filesystem is mounted *readonly*. This must be used only during installation, and not for a normal boot. It will effectively prevent writing to the filesystem.
DEFBOOTSTR=*bootstring*	Set default bootstring to *bootstring*. This is the string used by boot when the user presses <RETURN> only to the "boot:" prompt, or when boot times out.

SYSTTY=*x*	If *x* is zero (0), the system console device is set to the serial adapter at COM. If *x* is one (1), the system console is set to the main display adapter.

If either the **/etc/default/boot** file or the variable needed cannot be found, the variable is assumed to be NO. However, if the filesystem cannot be found, PANICBOOT is YES.

The **/etc/default/boot** file is shipped with the following default figuration:

LOADXENIX=YES
FSCKFIX=YES
MULTIUSER=YES
PANICBOOT=NO

A scratch file is needed by *fsck* to check large filesystems. The user is informed during the installation of XENIX if the system needs a scratch file to *fsck* the root filesystem. If necessary, the installation procedure creates the filesystem **/dev/scratch** to write the *fsck* temporary file. *fsck* uses the file named on the **/etc/default/boot** line:

SCRATCH=

as a scratch file. If the installation procedure creates the scratch filesystem, the entry in the **/etc/default/boot** is automatically made.

SCRATCH need only be specified if the root filesystem is large enough to need a temporary file. If a file is specfied, it is always passed to *fsck* when checking the root filesystem, even if the system is *booted* manually. The only exception is the first time XENIX is booted from the hard disk, when the user must specify the scratch file. The file specified as **SCRATCH** must not be on the filesystem being checked by *fsck*. **SCRATCH** also can not be on an unmounted filesystem.

If the XENIX mail system, *mail*(C), is installed on the system, the output of the boot sequence is mailed to *root*. Otherwise, the system administrator should check the file **/etc/bootlog** for the boot sequence output. The output of *fsck*(C) is temporarily saved in the file **/dev/recover** before it is moved to **/etc/bootlog** and finally may be sent to the system administrator via *mail*.

Other *boot* options which take affect during *autoboot* are documented on the *boot*(HW) manual page.

Files

/etc/bootlog	*boot* output log for autobooting systems
/etc/default/boot	boot information file
/etc/rc	instructions for entering multi-user mode, includes mounting and checking additional file systems
/dev/recover	allows saving of *fsck* output
/dev/scratch	temporary *fsck* file for large filesystems

See Also

boot(HW), fsck(C), init(M)

Notes

The utilities invoked during the *boot* procedure are passed the **-a** flag and time out only when the system *autoboots*. For example, *asktime*(C) times out after 30 seconds when the system *autoboots*, but waits for a response from the user any other time it is invoked.

The previous *boot* modes of AUTO=CLEAN, DIRTY, NEVER have been retained for backwards compatibility, but are ignored if any of the newer modes are present.

Name

badtrk – Scans fixed disk for flaws and creates bad track table.

Syntax

badtrk [**-e**] [-s *qtdn*] [**-f** */dev/rhd**]

Description

Used chiefly during system installation, *badtrk* scans the media surface for flaws, creates a new bad track table, prints the current table, and adds and deletes entries to the table.

WARNING: The **-e** flag should not be invoked by the user. It is called by *hdinit* during installation to change the space allocated for bad tracks. Use of the **-e** flag at any other time may restructure the hard disk, rendering much of the information stored on it unusable.

To use *badtrk*, you must be in single user mode. (See *shutdown*(C)). To address the active XENIX partition on your *primary* fixed disk, enter:

```
badtrk -f /dev/rhd0a
```

To address the active XENIX partition on your *secondary* fixed disk, enter:

```
badtrk -f /dev/rhd1a
```

Usage

When *badtrk* is executed, the program first displays the main menu:

```
1. Print Current Bad Track Table
2. Scan Disk (You may choose Read-Only or Destructive later)
3. Add Entries to Current Bad Track Table by Cylinder/Head Number
4. Add Entries to Current Bad Track Table by Sector Number
5. Delete Entries Individually From Current Bad Track Table
6. Delete All Entries From Bad Track Table

   Enter your choice or 'q' to quit:
```

You are prompted for option numbers, and, depending upon the option, more information may be queried for later.

A bad track table (option '1') might look like this:

Defective Tracks

	Cylinder	Head	Sector Number(s)
1.	190	3	12971–12987

Press <RETURN> to continue.

Option '2' scans the disk for flaws. If *badtrk* thinks changes may have been made to your bad track table since entering *badtrk* or updating your table, you will be asked if you want to update the device with the new table before scanning. You should answer 'y' to save your changes, 'n' if you don't want to save changes made up to this point. Next you are prompted for more information. After you respond to these prompts, *badtrk* begins its scan. You can interrupt a scan by typing "q" at any time. You are then prompted to continue the scan or return to the main menu.

As the program finds flawed tracks, it displays the location of each bad track. An example error message might be:

```
error on dev Fixed Disk (0/47), block=12954 cmd=0003 status=0018
sector = 12971, cylinder/head = 190/3
```

(You may see this kind of message if there is a read or write error during the scanning procedure.)

When the scan is complete, the main menu reappears. The program automatically enters any detected flaws in the bad track table.

If there are no entries in your bad track table and a scan does not reveal any flaws, but your disk is furnished with a flaw map, you should enter these flaws into the bad track table. Select either option '3' or '4' to add the entries. (See next paragraph.)

To add flaw locations to an existing bad track table, select either option '3' or option '4', depending upon the format of the flaw map furnished with your disk. Enter the defective tracks, one per line.

When you are satisfied that *badtrk* contains a table of the desired flaws, quit the *badtrk* program by entering 'q' at the main menu.

If *badtrk* was invoked with the **-e** option (which should only occur when called by *hdinit,* during the XENIX installation procedure), you are prompted for the the number of bad tracks to allocate space for. There will be a recommended number of replacement tracks to allocate based on the number of known bad tracks plus an allowance for tracks that will go bad in the future. You should choose to allocate at least as many as the recommended number of replacement tracks. Make your choice carefully, because if you want to change this amount later, you will have to reinstall XENIX.

At this point, you are asked if you want to "update". This is *badtrk*'s way of asking if any changes which were made should be saved. You should answer 'y' to save your changes, 'n' to leave the bad track table as it was when last updated.

Arguments

-f *name*
Opens the partition *name* and reads the bad track table associated with that partition. The default is **/dev/rhd0a.**

-s *options*
Invokes *badtrk* non-interactively. Valid options for this flag are:

[q]uick
[t]horough
[d]estructive
[n]on-destructive

The **-s** flag takes two options at a time. Choose quick or thorough scan, and destructive or non-destructive scan.

Notes

This utility can only be used in single-user mode.

If a bad spot develops in the inode table or superblock, reinstallation is required.

Files

/etc/badtrk

Name

clock – The system real-time (time of day) clock.

Description

The **clock** file provides access to the battery-powered, real-time time of day clock. Reading this file returns the current time; writing to the file sets the current time. The time, 10 bytes long, has the following form:

MMddhhmmyy

where *MM* is the month, *dd* is the day, *hh* is the hour, *mm* is the minute, and *yy* is the last two digits of the year. For example, the time:

0826150385 is 15:03 on August 26, 1985.

Files

/dev/clock

See Also

setclock(M)

Notes

Not all computers have battery-powered real-time time of day clocks. Refer to your computer's hardware reference manual.

Name

console – System console device.

Description

The file **/dev/console** is the device used by the system administrator for system maintenance (single-user) operations. It is the **tty** to which the first default shell is attached.

The system *console* device can be either a terminal (a serial adapter device, **tty1a**) or a sytem keyboard display adapter monitor (**tty01**).

Many programs, such as the XENIX kernel, redirect error messages to **/dev/console**. Initially **/dev/console** is linked to **/dev/systty**.

Files

/dev/console

See Also

boot(HW), systty(M), tty(M)

Notes

/dev/console should not be enabled, users should log in using the display adapter (**tty01**) or the serial adapter device (**tty1a**).

Name

daemon.mn – Micnet mailer daemon

Syntax

/usr/lib/mail/daemon.mn [–ex]

Description

The mailer daemon performs the "backend" networking functions of the *mail, rcp*, and *remote* commands by establishing and servicing the serial communication link between computers in a Micnet network.

When invoked, the daemon creates multiple copies of itself, one copy for each serial line used in the network. Each copy opens the serial line, creates a startup message for the LOG file, and waits for a response from the daemon at the other end. The startup message lists the names of the machines to be connected, the serial line to be used, and the current date and time. If the daemon receives a correct response, it establishes the serial link and adds the message "first handshake complete" to the LOG file. If there is no response, the daemon waits indefinitely.

If invoked with the **–x** switch, the daemon records each transmission in the LOG file. A transmission entry shows the direction of the transmission (tx for transmit, rx for receive), the number of bytes transmitted, the elasped time for the transmission (in minutes and seconds), and the time of day of the transmission (in hours, minutes, and seconds). Each entry has the form:

direction byte_count elasped_time time_of_day

The daemon also records the date and time every hour. The date and time have the same format as described for the *date* command.

If invoked with the **–e** switch, the daemon records all transmission errors in the LOG file. An error entry shows the cause of the error preceded by the name of the daemon subroutine which detected the error.

The mailer daemon is normally invoked by the *start* option of the *netutil* command and is stopped by the *stop* option.

During the normal course of execution, the mailer daemon uses several files in the **/usr/spool/micnet/remote** directory. These files provide storage for LOG entries, commands issued by the *remote*(C) command, and a list of processes under daemon control.

Files

/usr/lib/mail/daemon.mn

/usr/spool/micnet/remote/*/LOG

/usr/spool/micnet/remote/*/mn

/usr/spool/micnet/remote/local/mn*

/usr/spool/micnet/remote/lock

/usr/spool/micnet/remote/pids

See Also

netutil(C)

Name

default – Default program information directory.

Description

The files in the directory **/etc/default** contain the default information used by system commands such as **backup**(C) and **remote**(C). Default information is any information required by the command that is not explicitly given when the command is invoked.

The directory may contain zero or more files. Each file corresponds to one or more commands. A command searches a file whenever it has been invoked without sufficient information. Each file contains zero or more entries which define the default information. Each entry has the form:

keyword

or

keyword=value

where *keyword* identifies the type of information available and *value* defines its value. Both *keyword* and *value* must consist of letters, digits, and punctuation. The exact spelling of a *keyword* and the appropriate *values* depend on the command and are described with the individual commands.

Any line in a file beginning with a number sign (#) is considered a comment and is ignored.

Files

/etc/default/archive

/etc/default/backup

/etc/default/boot

/etc/default/cron

/etc/default/dumpdir

/etc/default/filesys

/etc/default/login

/etc/default/lpd

/etc/default/mapchan

/etc/default/micnet

/etc/default/mkuser

/etc/default/msdos

/etc/default/passwd

/etc/default/restor

/etc/default/tar

See Also

archive(F), backup(C), boot(HW), cron(M), dos(C), dumpdir(C), filesys(F), login(M), lpr(C), mapchan(M), mapchan(F), micnet (M), mkuser(C), pwadmin(C), remote(C), restore(C), su(C), sysadmin(C), tar(C)

Note

Not all commands use **/etc/default** files. Please refer to the manual page for a specific command to determine if **/etc/default** files are used, and what information is specified.

Name

dial – Dials a modem.

Syntax

/usr/lib/uucp/dial *ttyname telno speed*
/usr/lib/uucp/dial -h *ttyname speed*

Description

/usr/lib/uucp/dial dials a modem attached to *ttyname*. Although the installed dialer program is a compiled "C" program, you can also create a shell script to perform dialing functions. The **-h** option is used to hangup the modem.

uucp(C) and **cu**(C) use **/usr/lib/uucp/dial.** Three dialer programs are distributed. **dialHA12** is for the Hayes® Smartmodem 1200 and 1200B. **dialHA24** is for the Hayes® Smartmodem 2400. **dialVA3450** is for the Racal-Vadic VA3450-Series dialers. The **dial** program in **/usr/lib/uucp** is liuked to **dialHA12.** Source for **dialHA12** is provided in the file **dialHA12.c.**

uucp(C) invokes *dial*, with a *ttyname*, *telno* (phone number), and *speed*. *dial* attempts to dial the phone number on the specified line at the given speed. When using the **dialHA12** or **dialHA24** *speed* can be a range of baud rates. The range is specified with the form:

> *lowrate – highrate*

where *lowrate* is the minimum acceptable connection baud rate and *highrate* is the maximum. The *dial* program returns the status of the attempt through the following dial return codes:

bit 0x80 = 1
: The connection attempt failed.

bit 0x10 = 1
: The line is also used for dial-in and the getty has been suspended. This bit is used to tell *uucp* and *cu* that they must call *dial* with the **-h** flag when they are finished using the line.

bits 0x0f =

If bit 0x80 is a 1, then these bits are the dialer error code:

0	general or unknown error code.
1	line is being used.
2	a signal has aborted the dialer.
3	dialer arguments are invalid.
4	the phone number is invalid.
5	the baud rate is invalid or the dialer could not connect at the requested baud rate.
6	can't open the line.
7	ioctl error on the line.
8	timeout waiting for connection.
9	no dialtone was detected.
10	unused.
11	unused.
12	unused.
13	phone is busy.
14	no carrier is detected.
15	remote system did not answer.

Error codes 12-15 are used to indicate that the problem is at the remote end and *uucp* and *cu* should not try another line.

If bit 0x80 is a 0, then these bits are used to indicate the actual connection baud rate. If 0, the baud rate is the same as the baud rate used to dial the phone number or the highest baud rate if a range was specified. Otherwise, these four bits are the CBAUD bits in the **struct termio c_flag** and the **struct sgttyb sg_ispeed** and **sg_ospeed** tty ioctl structures.

If the specified line has been configured as a dial-in/dial-out line, *dial* invokes *ungetty*(M) to suspend the getty, making it a dial-out line. When the dial-out session is finished, *dial* should be called with the **-h** option to restore the dial-in line. (See *ungetty*(M).) If *uucp* or *cu* abort abnormally, the line will still be enabled for dial-out. Use **who -a** to check the line status. If the serial line is still in a dial-out state, use **ungetty -r** to restore the dial-in line.

You can copy and modify the file **/usr/lib/uucp/dialHA12.c** to use a different modem. There is a makefile in **/usr/lib/uucp** which should be modified for the new dialer, and can be used to compile the new program.

If you create a *dial* program for another modem, send us the source. User generated *dial* programs will be considered for inclusion in future releases.

Files

/usr/lib/uucp/dial	Default dialer program used by uucp
/usr/lib/uucp/dialHA12.c	C source program for *dialHA12*
/usr/lib/uucp/dialVA3450	Racal Vadic 3450 dialer
/usr/lib/uucp/dialVA212	Racal Vadic VA212 dialer
/usr/lib/uucp/dialHA12	Hayes Smartmodem 1200/1200B dialer
/usr/lib/uucp/dialHA24	Hayes Smartmodem 2400 dialer
/usr/lib/uucp/makefile	Makefile to compile and link new *dial*

See Also

dial(S), uucp(C), uux(C), ungetty(M)

Notes

You must have the XENIX Development System installed in order to compile and install a new **dial** program.

Name

environ – The user environment.

Description

The user environment is a collection of information about a user, such as his login directory, mailbox, and terminal type. The environment is stored in special "environment variables," which can be assigned character values, such as names of files, directories, and terminals. These variables are automatically made available to programs and commands invoked by the user. The commands can then use the values to access the user's files and terminal.

The following is a short list of commonly used environment variables.

PATH — Defines the search path for the directories containing commands. The system searches these directories whenever a user types a command without giving a full pathname. The search path is one or more directory names separated by colons (:). Initially, PATH is set to :/bin:/usr/bin.

HOME — Names the user's login directory. Initially, HOME is set to the login directory given in the user's **passwd** file entry.

TERM — Defines the type of terminal being used. This information is used by commands such as *more*(C) which rely on information about the capabilities of the user's terminal. The variable may be set to any valid terminal name (see *terminals*(M)) directly or by using the *tset*(C) command.

TZ — Defines time zone information. This information is used by *date*(C) to display the appropriate time. The variable may have any value of the form:

xxx*nzzzs;* **start/***time*, **end/***time*

where **xxx** is standard local time zone abbreviation (1-9 characters), *n* is the standard time zone difference from GMT, and may be given as hh:mm:ss (hours:minutes:seconds), **zzz** is the summertime local time zone abbreviation of 1-9 characters (if any), *s* is the summertime time zone difference from GMT, and may be given as hh:mm:ss (hours:minutes:seconds), **start** and **end** specify the day to begin and end summertime based on one of four rules, and **time** is the time of day the change to or from summertime occurs. The rules for specifying **start** and **end** are:

J*n*	1 based Julian day *n*
n	0 based Julian day *n*
W*n.d*	*n*th day of week *d*
M*m.n.d*	*n*th day of week *d* in month *m*

For example:

EST5:00:00EDT4:00:00;M4.1.0/2:00:00,M10.5.0/2:00:00. Refer to the *tz*(M) manual page for more on *TZ*.

HZ
: Defines, with a numerical value, the number of clock interrupts per second. The value of this variable is dependent on the hardware, and configured in the file **etc/default/login**. If *HZ* is not defined, programs which depend on this hertz value, such as *prof*(CP) and *times*(S), will not run.

LANG
: Defines the language locale a user wishes to use. This variable can be queried by applications and utilities to determine how to display information, what language to use for messages, sorting order, and other language dependent functions.

The environment can be changed by assigning a new value to a variable. For Bourne shell, *sh*(C), an assignment has the following format:

```
name=value
```

For example, the assignment:

```
TERM=h29
```

sets the TERM variable to the value "h29". The new value can be "exported" to each subsequent invocation of a shell by exporting the variable with the *export* command (see *sh*(C)) or by using the *env*(C) command.

C-shell users make assignments using the **setenv** command. For example:

```
setenv TERM h29
```

For more information, see *csh(C)*.

A user may also add variables to the environment, but must be sure that the new names do not conflict with exported shell variables such as MAIL, PS1, PS2, and IFS. Placing assignments in the **.profile** file is a useful way to change the environment automatically before a session begins. C-shell users can place assignments in their **.cshrc** or **.login** files.

Note that the environment is made available to all programs as a string of arrays. Each string has the format:

name=value

where the *name* is the name of an exported variable and the *value* is the variable's current value. For programs started with a *exec*(S) call, the environment is available through the external pointer *environ*. For other programs, individual variables in environment are available through *getenv*(S) calls.

See Also

csh(C), env(C), exec(S), getenv(S), login(M), profile(M), sh(C), tz(M)

Name

error – Kernel error output device.

Description

System error messages are collected and made available to error logging daemons through the **/dev/error** device. **/dev/error** is a read-only device which returns one error per read and no EOF character. **/etc/rc** uses a utility to read messages from **/dev/error** and write them to the system error log file **/usr/adm/messages**:

```
/etc/logger /dev/error /usr/adm/messages &
```

Any process can read **/dev/error** or arrange to be signaled when errors are queued in **/dev/error.** The following *ioctl* causes the error device to signal the process with **SIGUSR1** when an error message is queued in **/dev/error.**

```
#include <signal.h>
#include <syserr.h>
...
int fd;
...
fd = open("/dev/error", O_RDONLY);
ioctl(fd, EMSG_SIG, SIGUSR1);
```

Before exiting, the process must return **/dev/error** to its normal state. Do this with the following *ioctl*:

```
...
ioctl(fd,EMSG_NOSIG, 0);
...
```

Panic error messages are not logged in **/dev/error.**

Files

/dev/error

See Also

messages(M)

Name

fixperm - Correct or initialize file permissions and ownership.

Syntax

fixperm [-cfilnsvwDS [-d *package*]] specfile

Description

For each line in the specification file **specfile**, *fixperm* makes the listed pathname conform to a specification. *fixperm* is typically used to configure a XENIX system upon installation. Non-superusers can only use *fixperm* with the -n, -f, -D or -I flags. To use any other flags, you must be superuser.

The specification file has the following format: Each non-blank line consists of either a comment or an item specification. A comment is any text from a pound sign "#" up to the end of the line. There is one item specification per line. User and group id numbers must be specified at the top of the specification file for each user and group mentioned in the file.

An item specification consists of a package specifier, a permission specification, owner and group specifications, the number of links on the file, the file name, and an optional volume number.

The package specifier is an arbitrary string which is the name of a package within a distribution set. A package is a set of files.

After the package specifier is a permission specification. The permission specification consists of a file type, followed by a numeric permission specification. The item specification is one of the following characters:

x Executable.

a Archive.

e Empty file (create if -c option given).

b Block device.

c Character device.

d Directory.

f Text file.

p Named pipe.

If the item specification is used as an upper-case letter, then the file associated with it is optional, and *fixperm* will not return an error message if it does not exist.

The numeric permission conforms to the scheme described in *chmod*(C). The owner and group permissions are in the third column separated by a slash: e.g.,: "bin/bin". The fourth column indicates the number of links. If there are links to the file, the next line contains the linked filename with no other information. The fifth column is a pathname. The pathname must be relative, i.e., not preceded by a slash "/". The sixth column is only used for special files, giving the major and minor device numbers, or volume numbers.

Options

The following options are available from the command line:

-c Create empty files and missing directories.

-d*package*
Process input lines beginning with given package specifier string (see above). For instance, -dBASE processes only items specified as belonging to the Basic utilities set. The default action is to process all lines.

-u*package*
Like -u, but processes items that are not part of the given package.

-f List files only on standard output. Does not modify target files.

-i Check only if the selected packages are installed. Return values are:

0:	package completely installed
4:	package not installed
5:	package partially installed

-l List files and directories on standard output. Does not modify target files.

-n Report errors only. Does not modify target files.

-D
List directories only on standard output. Does not modify target files.

-v Verbose, in particular, issues a complaint if executable files are word swapped, not fixed stack, not separate I and D, or not stripped.

-s Modify special device files in addition to the rest of the permlist.

-w Lists where (what volume) the specified files or directories are located.

-S Issues a complaint if files are not in x.out format.

The following two lines make a distribution and invoke *tar*(C) to archive only the files in base.perms on /dev/sample:

```
/etc/fixperm -f /etc/base.perms > list
tar cfF /dev/sample list
```

This example reports *BASE* package errors:

```
/etc/fixperm -nd BASE
```

Notes

Usually *fixperm* is only run by a shell script at installation.

See Also

custom (C)

Name

fsname- Prints or changes the name of a file system.

Syntax

fsname [-p] [-s name]

Description

The */etc/fsname* utility is used to print or change the name of a filesystem. The options are:

-p Select the "pack" name field instead of the filesystem name field.

-s *name* Changes the specified field in the superblock.

The default action is to print the name of the filesystem.

See Also

mkfs(C), ustat(S), filesystem (F)

Name

getty – Sets terminal type, modes, speed, and line discipline.

Syntax

/etc/getty [**-h**] [**-t** timeout] line [speed [type [linedisc]]]
/etc/getty -c file

Description

getty is a program that is invoked by *init*(M). It is the second process in the series, (*init-getty-login-shell*), that ultimately connects a user with the XENIX system. Initially *getty* displays the login message field for the entry it is using from **/etc/gettydefs**. *getty* reads the user's login name and invokes the *login*(M) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in **/etc/ttys** to which *getty* is to attach itself. *getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **-h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **-t** flag, plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one enters anything in the specified number of seconds. The optional second argument, *speed*, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by entering a BREAK character). The default *speed* is 300 baud. The optional third argument, *type*, is a character string describing to *getty* what type of terminal is connected to the line in question. *getty* understands the type **none**–any CRT or normal terminal unknown to the system. This is the default.

For terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, **LDISC0**.

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode will be used (awaken on every character), that echo will be suppressed, either parity allowed, that new-line characters will be converted to carriage return-line feed, and that tab expansion is performed on the standard output. It displays the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the BREAK key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(S)).

The user's name is scanned to see if it contains any lower-case alphabetic characters. *getty* suggests that the user use all lower-case characters. If the user uses upper case characters, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, the *login-program* from **/etc/gettydefs** is called with the user's name as an argument. Additional arguments may be entered after the login name. These are passed to the *login-program*. The default *login-program*, **/etc/login**, places them in the environment (see *login*(M)).

A check option is provided. When *getty* is invoked with the **-c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it displays the values of the various flags. See *ioctl*(S) to interpret the values. Note that some values are added to the flags automatically.

Files

/etc/gettydefs
/etc/ttys

See Also

init(M), login(M), ioctl(S), gettydefs(F), ttys(M)

Name

group – Format of the group file.

Description

group contains the following information for each group:

– Group name

– Encrypted password (optional)

– Numerical group ID

– Comma-separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a newline. If the password field is null, no password is demanded.

This file resides in directory **/etc**. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group IDs to names.

Files

/etc/group

See Also

newgrp(C), passwd(C), passwd(M)

Name

imagen.sbs, imagen.pbs, imagen.spp, imagen.remote – IMAGEN printer interface scripts.

Syntax

imagen.sbs request user title copies options files...
imagen.pbs request user title copies options files...
imagen.spp request user title copies options files...
imagen.remote request user title copies options files...

Description

The *imagen* scripts are the XENIX System V spooler interface programs for IMAGEN printers. They accept the following types of files for printing: *troff*(CT) input, *troff* output (C/A/T format), imPRESS format, DVI format (generally produced by), and straight text. The proper *imagen* interface script is normally installed using *lpadmin*(C).

imagen.sbs uses the "serial byte stream protocol" provided by *isbs*.

imagen.pbs uses the "parallel byte stream protocol" provided by *ipbs*.

imagen.spp uses the serial "sequence packet protocol" provided by *ips*(M).

imagen.remote sends the print job to a remote computer using either *uux*(C) or *remote*(C). The exact command to use is defined in the file **/usr/spool/lp/remote**, and the "printer" device defined by *lpadmin* (using the **−v** flag) should be **/dev/null**.

Recognized *lp*(C) *options* are:

−oli
The input files are in imPRESS format but with no document header.

−olp
The input files are text for line printing.

−olfi
The input files are in full imPRESS format with a document header.

−olt
The input files are *troff* input.

−oldvi
The input files are DVI format (output), to be filtered through *dviimp*(CT).

−olc
The input files are *troff* output (C/A/T format), to be filtered through *catimp*(CT).

−ot*flag*
Pass option *flag* to *troff*.

−oc*flag*
Pass option *flag* to *catimp*.

−ov*flag*
Pass option *flag* to *dviimp*.

−oi*flag*
flag is an IMAGEN printer control setting:

1 Print one page per sheet of paper.

2 Print two pages per sheet of paper.

O Print outlines around the page.

r Print pages opposite (reverse) of usual order.

c Do not collate pages of multiple copies.

R Print rules on pages (one every two lines).

J Suppress printing the job header (banner) page.

m Do not print detailed error messages on the banner page.

j Enables jam resistance measures. The default jam resistance action is controlled by the setting of **JAMPROOF** in the file **/etc/default/imagen**.

Not all control settings are meaningful for every IMAGEN printer **language**.

−ob
No banner information about the local user or host should be generated.

−oh*host*
The computer responsible for this job is *host*.

−ou *user*
The person responsible for this job is *user*.

All of the *imagen* interface scripts read **/etc/default/imagen** to obtain various default settings. The values obtained, and the default values, are:

JAMPROOF=no
Whether or not paper-jam resistance measures should be used. If such steps are taken, printing is usually slowed down.

The values of the default settings can be changed to reflect the local system configuration. If **/etc/default/imagen** does not exist or cannot be read, the above default values are used.

Files

/usr/bin/itroff
troff for an IMAGEN printer.

/usr/bin/catimp
Converts from *troff* C/A/T output to imPRESS format.

/usr/bin/dviimp
Converts from DVI to imPRESS format.

/usr/lib/ips
(*imagen.spp*) IMAGEN serial sequence packet protocol.

/usr/lib/isbs
(*imagen.sbs*) IMAGEN serial byte stream protocol.

/usr/lib/ipbs
(*imagen.pbs*) IMAGEN parallel byte stream protocol.

/usr/spool/lp/remote
(*imagen.remote*) Mapping from local printer name to *remote* or *uux* command. Each line is in the format:

printer: *command*

where *printer* is the name of the "local" IMAGEN printer, and *command* is either a *remote* or *uux* invocation of *lp* on another machine. The other machine must be configured so that a remote *lp* is allowed, and the local *command* should specify whatever options are necessary so that the input can be piped into it. Additional flags to *lp* are appended onto the end of *command* by *imagen.remote*. A typical *remote command* would be:

printer: **remote −** *machine* **lp −dimagen** and a typical *uux command* would be:

printer: **uux –** *machine*!**lp –dimagen**

See Also

catimp(CT), dviimp(CT), imprint(C), lp(C), lpadmin(C), ips(M), itroff(CT), remote(C), uux(C)

Author

IMAGEN Corporation.

Name

init, inir – Process control initialization.

Syntax

/etc/init
/etc/inir

Description

The *init* program is invoked as the last step of the boot procedure and as the first step in enabling terminals for user logins. *init* is one of three programs (*init*, *getty*(M), and *login*(M)) used to initialize a system for execution.

init creates a process for each terminal on which a user may log in. It begins by opening the console device, **/dev/console**, for reading and writing. It then invokes a shell which prompts for a password to start the system in "maintenance mode". If at this prompt an EOF is read, the system proceeds toward "multi-user mode". If the root pasword is entered, a shell is started and attached to the console. When this shell is terminated the system proceeds toward "multi-user mode".

If the system was automatically loaded at boot time, *init* will be passed a **-a** flag when it is started. *init* also passes this flag to the programs it runs so they may choose to behave differently under *autoboot*(M) conditions.

The user may *boot* and the filesystem may be dirty. In this case, *inir* prompts the user, asking whether to do an *fsck (C)* (See *fsck (C)* for more information.)

The user may *boot* and the filesystem may be clean. In this case, *init* reads commands from the **/etc/rc** file. This is followed by the "multi-user/rc" and the "getty/login" procedures as documented below.

"multi-user/rc" procedure: Once the filesystem is clean, the shell terminates, and *init* performs several steps to begin normal operation. It invokes a shell and reads the commands in the **/etc/rc** file. This command file performs housekeeping tasks such as removing temporary files, mounting file systems, and starting daemons. Then it reads the file **/etc/ttys** and forks several times to create a process for each terminal device in the file. Each line in the **/etc/ttys** lists the state of the line (0 for closed, 1 for open), the line mode, and the serial line (see *ttys*(M)). Each process opens the appropriate serial line for reading and writing, assigning the file descriptors 0, 1, and 2 to the line and establishing it as the standard input, output,

and error files. If the serial line is connected to a modem, the process delays opening the line until someone has dialed up and a carrier has been established on the line.

"getty/login" procedure: Once *init* has opened a line, it executes the *getty* program, passing the line mode as an argument. The *getty* program reads the user's name and invokes *login*(M) to complete the login process (see *getty*(M) for details). *init* waits until the user logs out by typing ASCII end-of-file (Ctrl-D) or by hanging up. It responds by waking up and removing the former user's login entry from the file **utmp**, which records current users, and makes a new entry in the file **wtmp**, which is a history of logins and logouts. Then the corresponding line is reopened and *getty* is reinvoked.

init has special responses to the hangup, interrupt, and quit signals. The hangup signal SIGHUP causes *init* to change the system from normal operation to maintenance mode. The interrupt signal SIGINT causes *init* to read the **ttys** file again to open any new lines and close lines that have been removed. The quit signal SIGQUIT causes *init* to disallow any further logins. In general, these signals have a significant effect on the system and should not be used by a inexperienced user. Instead, similar functions can be safely performed with the *enable*(C), *disable*(C), and *shutdown*(C) commands.

Files

/dev/tty*
/etc/utmp
/usr/adm/wtmp
/etc/default/boot
/etc/ttys
/etc/rc
/etc/gettydefs

See Also

autoboot(M), disable(C), enable(C), login(M), kill(C), sh(C), shutdown(C), ttys(M), getty(M), gettydefs(F)

Diagnostics

If seven or more *getty* processes are started on the same line in five minutes or less, *init* writes an error message to **/dev/console** and refuses to start another *getty* on that line for at least 30 minutes. If desired, *init* will try again immediately if a SIGINT is sent.

Name

install – Installation shell script.

Syntax

/etc/install [**device**]

Description

/etc/install is the *sh*(C) script used to install XENIX distribution (or application program) floppies. It performs the following tasks:

- Prompts for insertion of floppies.
- Extracts files using the *tar*(C) utility.
- Executes **/once/init.*** programs on each floppy after they have been extracted.
- Removes any **/once/init.*** programs when the installation is finished.

The optional argument to the command specifies the device used. The default device is **/dev/install** and is normally linked to **/dev/rfd0.**

Files

/etc/install

/once/init.*

Name

ips, isbs, ipbs – IMAGEN protocol handlers.

Syntax

/usr/lib/ips [options] [file]
/usr/lib/isbs [options] [file]
/usr/lib/isps [options] [file]

Description

ips, isbs, and *ipbs* are the lowest level of IMAGEN printer software. Each handles a different form of communications. They present a very similar view to higher-level software, allowing such software to be relatively independent of communications method.

ips sends files to the IMAGEN printer using the "sequence packet protocol" over RS-232C serial communication lines. This protocol provides for error detection, retransmission, status reporting, detection of unrecoverable errors, and printer usage accounting. The IMAGEN printer must be configured for "sequenced packet (V1)" at the desired baud rate.

isbs supports the "serial byte stream" communications method over RS-232C serial communication lines. This protocol provides for flow control, but no error detection or correction. The IMAGEN printer must be configured for the "serial byte stream" protocol at the desired baud rate, should use an "EOF" character of **0x04**, a "quote" character of **0x02**, an 8-bit-wide data path, XON/XOFF flow control, and should not ignore unprintable characters.

ipbs supports the "parallel byte stream" communications method, which is essentially similar to *isbs* but works on parallel rather than serial ports. The IMAGEN printer must be configured for the parallel interface, should use an "EOF" character of **0x04**, a "quote" character of **0x02**, and should not ignore unprintable characters.

Neither *isbs* nor *ipbs* provides printer-usage accounting. The following information is common to all these programs.

If no *file* name is given, the standard input is read.

The following *options* are recognized:

–D*string*
Imbed *string* into the document control language of the file being sent. If an unrecoverable failure occurs during transmission and the file can be resent, this string will be resent as well.

−a *stsfile*
Save the current printer status in file *stsfile*.

−i*device*
The IMAGEN printer is connected to XENIX special file *device*. The default *device* is the value of **DEVICE** in the file **/etc/default/imagen.**

−l*logfile*
Save information on the communications needed to send the input in the file *logfile*.

−r
The standard input is rewindable. Normally, the standard input is not assumed to be rewindable. Or, if an explicit *file* was named, that file is not rewindable. Normally, an explicitly named file is assumed to be rewindable.

−s
Regardless of what other indications may have been given, use the standard output as the printer.

−p*debug*
Sets various debugging values.

The following *options* are recognized by all three programs, but only used by *ips* and *isbs*; *ipbs* ignores them:

−o
The line characteristics need to be set up if the output is being sent to the standard output. Normally, the characteristics of the standard output are not changed. Or they do not need to be set up if an explicit device was given with the **−i** option. Normally, the characteristics of named devices are changed.

−B*speed*
The baud rate of the serial communications lines is *speed*. It is assumed that the IMAGEN printer is capable of running at the specified *speed*, and that it has been configured to do so. The default *speed* is the value of **SPEED** in the **/etc/default/imagen** file.

The following *options* are recognized by all three programs, but only used by *ips*; both *isbs* and *ipbs* ignore them:

−A*acctfile*
Turns on accounting, and places the accounting information in file *acctfile*. This file should be read by *imacct*(C). There should be a separate *acctfile* for each IMAGEN printer.

−n *user*

Names the user or account to be credited with this printing job. Since these programs are normally run by the spooling system, there is no reasonable default username. Therefore, if accounting is enabled by use of the **−A** option, this option should also be specified. If it is not, all printing is charged to user "???."

−h*host*

Names the machine to be credited with originating this printing job. If not specified, first *uname*(S) and then **/etc/systemid** are used to determine the name of the local system. If the name of the system cannot be determined, "LOCAL" is used.

The following *option* is not supported, and the **−n** option should be used instead:

−u*uid*

The user identification number of the person to be credited with this printing job.

ips, *isbs*, and *ipbs* all read **/etc/default/imagen** to obtain various default settings. The values obtained, and the default values, are:

DEVICE=/dev/imagen

The name of the XENIX special file connected to the IMAGEN printer. This can be overridden with the **−i** or −s options.

SPEED=9600

The baud rate of the IMAGEN printer. This is meaningful only for *ips* and *isbs*, and can be overridden with the **−B** option.

The values of the default settings can be changed to reflect the local system configuration. If **/etc/default/imagen** does not exist or cannot be read, the above default values are used.

Files

/dev/imagen
Default name of the XENIX special file connected to the IMAGEN printer.

/dev/null
Default *stsfile*. See the **−a** option.

imagen.log
Default *logfile*. See the **−l** option.

See Also

imagen(M), ipr(C)

Author

IMAGEN Corporation.

Name

ld – Invokes the link editor.

Syntax

ld [*options*] *filename...*

Description

ld is the XENIX link editor. It creates an executable program by combining one or more object files and copying the executable result to the file **a.out**. The *filename* must name an object or library file. These names must have the ".o" (for object) or ".a" (for archive library) extensions. If more than one name is given, the names must be separated by one or more spaces. If errors occur while linking, *ld* displays an error message; the resulting **a.out** file is unexecutable.

ld concatenates the contents of the given object files in the order given in the command line. Library files in the command line are examined only if there are unresolved external references encountered from previous object files. Library files must be in *ranlib*(CP) format, that is, the first member must be named __.SYMDEF, which is a dictionary for the library. The library is searched iteratively to satisfy as many references as possible and only those routines that define unresolved external references are concatenated. Object and library files are processed at the point they are encountered in the argument list, so the order of files in the command line is important. In general, all object files should be given before library files. *ld* sets the entry point of the resulting program to the beginning of the first routine.

There are the following options:

–F*num*
: Sets the size of the program stack to *num* bytes. Default stack size if not given, is either variable stack or fixed stack of 2 Kbytes. See the *machine*(M) manual page for the default stack type for your system.

–i Creates separate instruction and data spaces for small model programs. When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and shared by all users executing the file.

–Ms
: Creates a small model program and checks for errors, such as fixup overflow. This option is reserved for object files compiled

or assembled using the small model configuration. This is the default model if no **−M** option is given.

−Mm
Creates middle model program and checks for errors. This option is reserved for object files compiled or assembled using the middle model configuration. This option implies **−i** .

−Ml
Creates a large model program and checks for errors. The option is reserved for object files compiled using the large model configuration. This option implies **−i** .

−o *name*
Sets the executable program filename to *name* instead of **a.out.**

ld should be invoked using the *cc*(CP) instead of invoking it directly. *Cc* invokes *ld* as the last step of compilation, providing all the necessary C-language support routines. Invoking *ld* directly is not recommended since failure to give command line arguments in the correct order can result in errors.

Files

/bin/ld

See Also

ar(CP), cc(CP), ld(CP), masm(CP), ranlib(CP)

Notes

The user must make sure that the most recent library versions have been processed with *ranlib*(CP) before linking. If this is not done, *ld* cannot create executable programs using these libraries.

Name

login - Gives access to the system.

Description

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It cannot be invoked except when a connection is first established, or after the previous user has logged out by sending an end-of-file (Ctrl-D) to his initial shell.

login prompts for your user name, and if appropriate, your password. Echoing is turned off (where possible) while entering your password, so it will not appear on the written record of the session.

If password aging has been invoked by the superuser on your behalf, your password may have expired. In this case, you will be shunted into *passwd*(C) to change it, after which you may attempt to log in again.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be returned to the "login:" prompt or silently disconnected from a dial-up line.

After a successful login, accounting files (*/etc/utmp* and */etc/wtmp*) are updated, you are told if you have any mail, and the start-up profile files (i.e., **/etc/profile** and **$HOME/.profile**), if any, are executed. See *profile*(M).

login checks **/etc/default/login** for **ULIMIT** (maximum file size in 512 byte blocks, default is 2,097,152), and for environment variables, such as **TZ** (time zone), **HZ** (hertz), and **ALTSHELL** (allows other than **sh** shell types). Other entries sometimes found in **/etc/default/login** are IDLEWEEKS, CONSOLE, and PASSREQ. IDLEWEEKS=*n*, where *n* is a number of weeks, works in conjunction with *pwadmin*(C). If a password has expired, you are prompted to choose a new one. If it has expired beyond IDLEWEEKS, the user is not allowed to log in, and must consult system administrator. The CONSOLE=**/dev/***???* entry means that root can only log in on the **/dev** listed. PASSREQ=YES, if set, forces you to select a password if you do not have one.

login initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh*(C)) according to specifications found in the **/etc/passwd** file. Argument 0 of the command interpreter is a dash (-) followed by the last component of the interpreter's pathname. The *environment* (see *environ*(M)) is initialized to:

HOME= your-login-directory

PATH=:/bin:/usr/bin

Initially, *umask* is set to octal 022 by *login*.

Files

/etc/utmp	Information on current logins
/etc/wtmp	History of logins since last multiuser
/usr/spool/mail/*your-name*	Mailbox for user *your-name*
/etc/motd	Message of the day
/etc/default/login	Default values for environment variables
/etc/passwd	Password file
/etc/profile	System profile
$HOME/.profile	Personal profile

See Also

environ(M), getty(M), machine(M), mail(C), newgrp(C), passwd(C), passwd(M), profile(M), su(C), sh(C), ulimit(S), umask(C), who(C).

Diagnostics

Login incorrect
The user name or the password is incorrect.

No shell, cannot open password file, no directory:
Your account has not been properly set up.

Your password has expired. Choose a new one.
Password aging is implemented and yours has expired.

Notes

Under System V, only the superuser may execute *login* from a shell. Hence, non-superusers must log out in order to log in as another user. Pre-system III *login*, if invoked from the command line while someone is logged on already, logs the current user out and logs in the new user. The current *login* nests, i.e., the current user is not logged out. Thus, it is somewhat like **su**(C), except that the new user's *.login* or *.profile* is run. Permissions and environment are those of the new user. When the new user logs out, the previous user is still running. This practice is not recommended, as nested logins can impair system performance.

As explained in *machine*(M), when setting **ULIMIT** in the **/etc/default/login** file on filesystems with 1024 byte blocks (see **machine**(M)), be sure to specify even numbers, as the ULIMIT variable accepts a number of 512–byte blocks. The default is 2,097,152 blocks, or 1 gigabyte. Use this variable to increase or decrease the maximum allowable file size.

Name

makekey – Generates an encryption key.

Syntax

/usr/lib/makekey

Description

makekey improves the usefulness of encryption schemes by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way that is intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first 8 input bytes (the *input key*) can be arbitrary ASCII characters. The last 2 input bytes (the *salt*) are best chosen from the set of digits, dot (.), slash (/), and uppercase and lowercase letters. The *salt* characters are repeated as the first 2 characters of the output. The remaining 11 output characters are chosen from the same set as the *salt* and constitute the *output key*.

The transformation performed is essentially the following: the *salt* is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as the key, a constant string is fed into the machine and recirculated. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

See Also

passwd(M)

Name

mapchan – Configure tty device mapping.

Syntax

mapchan [**-ans**] [**-f** *mapfile*] [*channels* ...]
mapchan [[**-o**] [**-d**]] [*channel*]
trchan [**-c**] *mapfile*

Description

mapchan configures the mapping of information input and output of XENIX. The *mapchan* utility is intended for users of vertical applications that employ languages other than English (non-7-bit-ASCII character sets).

mapchan interprets codes sent by peripheral devices, such as terminals, to the internal character set used by the XENIX system. *mapchan* can also map codes in the internal character set to other codes, for output to peripheral devices (such as terminals, printers, console screen, etc.). Note that some PC keyboard configuration may also be accomplished through the *mapkey*(M) utility.

mapchan has several uses: to map a *channel* (**-a** or **-s**); to unmap a *channel* (**-n** and optionally **-a**); or to display the map on a channel (optionally **-o**, **-d**, *channels*).

trchan performs mapping as a filter, using the same format of *mapfile* as *mapchan*. This allows a file consisting of one internal character set to be "translated" to another internal character set. *trchan* reads standard input, maps it, and writes to standard output. A *mapfile* must be given on the command line. Errors cause *trchan* to stop processing unless **-c** is specified.

mapchan with no options displays the map on the user's *channel*. The map displayed is suitable as input for *mapchan*.

The options are:

-a when used alone, sets all *channels* given in the default file (**/etc/default/mapchan**) with the specified map. When used with **-n**, it refers to all *channels* given in the default file. Super-user maps or unmaps all *channels*, other users map only *channels* they own. **-a** can not be used with **-d**, **-o**, or **-s**.

-c causes errors to be echoed on *stderr*, and processing is continued. Used with *trchan* only.

-d causes the mapping table currently in use on the given device, *channel*, to be displayed in decimal instead of the default hexadecimal. An ASCII version is displayed on standard output. This output is suitable as an input file to *mapchan* for another *channel*. Mapped values are displayed. Identical pairs are not output. **-d** can not be used with **-a**, **-f**, **-n**, **-o**, or **-s**.

-f causes the current *channel* or list of *channels* to be mapped with *mapfile*. **-f** can not be used with **-d**, **-n**, **-s**, or **-o**.

-n causes null mapping to be performed. All codes are input and output as received. Mapping is turned off for the user's *channel* or for other *channels*, if given. **-a** used with **-n** will turn mapping off on all *channels* given in the default file. This is the default mapping for all *channels* unless otherwise configured. **-n** can not be used with **-d**, **-f**, **-o**, or **-s**.

-o causes the mapping table currently in use on the given device, *channel*, to be displayed in octal instead of the default hexadecimal. An ASCII version is displayed on standard output. This output is suitable as an input file to *mapchan* for another port. Mapped values are displayed. Identical pairs are not output. **-o** can not be used with **-a**, **-d**, **-f**, **-n**, or **-s**.

-s sets the user's current *channel* with the *mapfile* given in the default file. **-s** can not be used with any other option.

The user must own the *channel* in order to map it. The super-user can map any channel. Read or write permission is required to display the map on a *channel*.

Each tty device *channel* (display adapter and video monitor on computer, parallel port, serial port, etc.) can have a different map. When XENIX boots, mapping is off for all *channels*.

mapchan is usually invoked in the **/etc/rc** file. This file is executed when the system enters multi-user mode and sets up the default mapping for the system. Users can invoke *mapchan* when they log in by including a *mapchan* command line in their **.profile** or **.login** file. In addition, users can remap their *channel* at any time by invoking *mapchan* from the command line. *Channels* not listed in the default file are not automatically mapped. *Channels* **tty02-06** are listed in the default file as distributed, using **stdrom** as the *mapfile*. *channels* are not changed on logout. Whatever mapping was in place for the last user remains in effect for the next user, unless they modify their **.profile** or **.login** file.

For example, the default file **/etc/default/mapchan** can contain:

tty02	stdrom
tty1a	
tty2a	fra/terminal
lp	ger/map78

The default directory containing *mapfiles* is **/usr/lib/mapchan**. Subdirectories are lower case three letter names that are language abbreviations. All the *mapfiles* for a given language or country are conveniently grouped. The default directory containing *channel* files is **/dev**. Full pathnames may be used for *channels* or *mapfiles*. If a *channel* has no entry, or the entry field is blank, no mapping is enabled on that *channel*. Additional *channels* added to the system, (for example, adding a serial or parallel port) are not automatically entered in the *mapchan* default file. If mapping is required, the system administrator must make the entries.

The format of the *mapfiles* is documented in the *mapchan*(F) manual page.

Using a Mapped *channel*

The input information is assumed to be 7- or 8-bit codes sent by the peripheral device. The device may make use of "dead" or "compose" keys to produce the codes. If the device does not have dead or compose keys, these keys can be simulated using *mapchan*.

One to one mapped characters are displayed when the key is pressed, and the mapped value is passed to the kernel.

Certain keys are designated as dead keys in the *mapfile*. Dead key sequences are two keystrokes that produce a single mapped value that is passed to the kernel. The dead key is usually a diacritical character, the second key is usually the letter being modified. For example, the sequence ´ *e* could be mapped to the ASCII value 0xE9, and display as é.

One key is designated as the compose key in the *mapfile*. Compose key sequences are composed of three keystrokes that produce a single mapped value that is passed to the kernel. The compose key is usually a seldom used character or ctrl-*letter* combination. The second key is usually the letter being modified. The third key may be another character being combined, or a diacritical character. For example, if '@' is the compose key, the sequence @ *c O* could be mapped to the ASCII value 0xA9, and display as ©.

Characters are not echoed to the screen during a dead or compose sequence. The mapped character is echoed and passed to the kernel once the sequence is correctly completed.

Characters are always put through the input map, even when part of dead or compose sequences. The character is then checked for the internal value. The value may also be mapped on output. This should be kept in mind when preparing map files.

The following conditions will cause an error during input:

- dead key followed by a dead key
- non-recognized (not defined in the *mapfile*) dead or compose sequence
- restarting a compose sequence before completion by pressing the compose key in the middle of a dead or compose sequence. This is an error, but a new compose sequence is initiated.

If the *mapfile* contains the keyword *beep*, a bell sounds when any of the above conditions occurs. In either case, the characters are not echoed to the screen, or passed to the kernel.

Character Sets

The internal character set used by XENIX is defined by the *mapfiles* used. By default, this is the 8-bit ASCII character set which is also known as the dpANS X3.4.2 and ISO/TC97/SC2 or ISO 8859 Level I character sets. It supports most of the Latin alphabet and can represent most European languages.

Several partial map files are provided as examples. They must be modified for use with specific peripheral devices. Consult your hardware manual for the codes needed to display the desired characters. Input maps for each of the following 7-bit national character sets are included:

Character Set	Standard Designation
UK 7-bit Std	(BS 4730)
French 7-bit Std	(NFZ 62-101 1982)
German 7-bit Std	(DIN 66003)
Canadian 7-bit English and French bilingual	(CSA Z243.4-1973)

In addition, *mapkey*(C) and corresponding *mapchan* mapfiles are provided for eight different countries, one file for computers with standard PC character ROMs, and one file for 8859 ROMs. 8-bit terminals need little, if any, mapping as long as the *stty*(C) settings are correct. The **/etc/gettydefs** file may require modification to allow logging in with the correct settings.

US 7-bit ASCII (ANSI X3.4) is used if no mapping is enabled on the *channel*.

Files

/etc/default/mapchan
/usr/lib/mapchan/map.stdrom
/usr/lib/mapchan/map.*

See Also

ascii(M), keyboard(HW), lp(C), lpadmin(C), mapchan(F), mapkey(M), parallel(HW), screen(HW), serial(HW), setkey(M), tty(M)

Notes

mapchan should be used with vertical applications only, as there are several XENIX utilities, including the *sh*(C) shell, which may not function as expected. 8-bit codes should not be used in filenames.

Some foreign keyboards and display devices do not contain characters commonly used by XENIX command shells and the C programming language. It is not recommended that these devices be used for system administration tasks.

Printers can be mapped, output only, and can either be sent 8-bit codes or one-to-many character strings using *mapchan*. Line printer spooler interface scripts can be used (setuid *root*) to change the output map on the printer when different maps are required (as in changing print wheels to display a different character set). See *lp*(C) and *lpadmin*(C) for information on installing and administering interface scripts.

Not all terminals or printers can display all the characters that can be represented using this utility. Refer to the device's hardware manual for information on the capabilities of the peripheral device.

WARNING: Use of *mapfiles* that specify a different "internal" character set per-channel, or a set other than the 8-bit ASCII set supplied by default can cause strange side effects. It is especially important to retain the 7-bit ASCII portion of the character set (see *ascii*(M)). XENIX utilities and many applications assume these values.

Media transported between machines with different internal code set mappings may not be portable as no mapping is performed on block devices, such as tape and floppy drives. However, *trchan* with an appropriate *mapfile* can be used to "translate" from one internal character set to another.

Do not set ISTRIP (see *stty*(C)) when using *mapchan*. This option causes the eighth bit to be stripped before mapping occurs.

Name

mapkey, mapscrn, mapstr, convkey – Configure monitor screen mapping.

Syntax

mapkey [**−dox**][*datafile*]
mapscrn [**−d**][*datafile*]
mapstr [**−d**][*datafile*]
convkey [**in** [**out**]]

Description

mapscrn configures the output mapping of the monitor screen on which it is invoked. *mapkey* and *mapstr* configure the mapping of the keyboard and string keys (eg. function keys) of the monitor (and multiscreens if present). *mapkey* can only be run by the super-user.

mapstr functions on a per-screen basis. Mapping strings on one screen does not affect any other screen.

If a file name is given on the argument line the respective mapping table is configured from the contents of the input file. If no file is given, the default files in **/usr/lib/keyboard** and **/usr/lib/console** is used. The **-d** option causes the mapping table to be read from the kernel instead of written and an ASCII version to be displayed on the standard output. The format of the output is suitable for input files to *mapscrn*, *mapkey*, or *mapstr* . Non-super-users can run *mapkey* and *mapstr* when the **-d** option is given.

With the **-o** or **-x** options, *mapkey* displays the mapping table in octal or hexadecimal.

convkey translates an old-style mapkey file into the current format. If *in* or *out* are missing, they default to *stdin* or *stdout*.

Files

/usr/lib/keyboard/keys
/usr/lib/keyboard/keys.gr
/usr/lib/keyboard/keys.uk
/usr/lib/keyboard/strings
/usr/lib/console/screens.gr
/usr/lib/console/screens.uk
/usr/lib/console/screens.usa

Notes

There is no way to specify that the map utilities read their configuration tables from standard input.

See Also

keyboard(HW), screen(HW), setkey(C)

Name

mem, kmem – Memory image file.

Description

The **mem** file provides access to the computer's physical memory. All byte-addresses in the file are interpreted as memory addresses. Thus, memory locations can be examined in the same way as individual bytes in a file. Note that accessing a nonexistent location causes an error.

The **kmem** file is the same as **mem** except that it corresponds to kernel virtual memory rather than physical memory.

In rare cases, the **mem** and **kmem** files may be used to write to memory and memory-mapped devices. Such patching is not intended for the naive user and may lead to a system crash if not conducted properly. Patching device registers is likely to lead to unexpected results if the device has read-only or write-only bits.

Files

/dev/mem

/dev/kmem

Name

messages – Description of system console messages.

Description

This section describes the various system messages which may appear on the system console. The messages are categorized as follows:

Fatal
Recovery is impossible.

System inconsistency
A contradictory situation exists in the kernel.

Abnormal
A probably legitimate but extreme situation exists.

Hardware
Indicates a hardware problem.

Fatal system messages begin with "panic:" and indicate hardware problems or kernel inconsistencies that are too severe for continued operation. After displaying a fatal message, the system will stop. Rebooting is required.

System inconsistency messages indicate problems usually traceable to hardware malfunction, such as memory failure. These messages rarely occur since associated hardware problems are generally detected before such an inconsistency can occur.

Abnormal messages represent kernel operation problems, such as the overflow of critical tables. It takes extreme situations to bring these problems about, so they should never occur in normal system use. However, in some cases you can raise the limit that causes the error message. In the message list below, limits for applicable messages are given. Use the *configure*(C) utility to make the necessary changes.

Hardware messages normally specify the device, *dev*, that caused the error. Each message gives a device specification of the form *nn/mm* where *nn* is the major number of the device, and *mm* is its minor number. The command pipeline

ls -l /dev | grep *nn* | grep *mm*

may be used to list the name of the device associated with the given major and minor numbers.

System Messages

** ABNORMAL System Shutdown **

This message appears when errors occur during system shutdown. It is usually accompanied by other system messages. *System inconsistency, fatal.*

bad block on dev *nn/mm*

A nonexistent disk block was found on, or is being inserted in, the structure's free list. *System inconsistency.*

bad count on dev *nn/mm*

A structural inconsistency in the superblock of a file system. The system attempts a repair, but this message will probably be followed by more complaints about this file system. *System inconsistency.*

Bad free count on dev *nn/mm*

A structural inconsistency in the superblock of a file system. The system attempts a repair, but this message will probably be followed by more complaints about this file system. *System inconsistency.*

error on dev *name* (*nn/mm*)

This is the way that most device driver diagnostic messages start. The message will indicate the specific driver and complaint. The *name* is a word identifying the device.

iaddress > 2^24

This indicates an attempted reference to an illegal block number, one so large that it could only occur on a file system larger than 8 billion bytes. *Abnormal.*

Inode table overflow

Each open file requires an inode entry to be kept in memory. When this table overflows, the specific request (usually *open*(S) or *creat*(S)) is refused. Although not fatal to the system, this event may damage the operation of various spoolers, daemons, the mailer, and other important utilities. Abnormal results and missing data files are a common result. Use *configure*(C) to raise the number of inodes. *Abnormal.*

interrupt from unknown device, vec=*xxxx*

The CPU received an interrupt via a supposedly unused vector. This message is followed by "panic: unknown interrupt." Typically, this event comes about when a hardware failure miscomputes the vector of a valid interrupt. *Hardware..*

no file

There are too many open files. The system has run out of entries in its "open file" table. The warnings given for the message "inode table overflow" apply here. Use *configure*(C) to raise the number of files. *Abnormal.*

no space on dev *nn/mm*

This message means that the specified file system has run out of free blocks. Although not normally as serious, the warnings discussed for "inode table overflow" apply: often user programs are written casually and ignore the error code returned when they tried to write to the disk; this results in missing data and "holes" in data files. The system administrator should keep close watch on the amount of free disk space and take steps to avoid this situation. *Abnormal.*

** Normal System Shutdown **

This message appears when the system has been shutdown properly. It indicates that the machine may now be rebooted or powered down.

Out of inodes on dev *nn/mm*

The indicated file system has run out of free inodes. The number of inodes available on a file system is determined when the file system is created (using *mkfs*(C)). The default number is quite generous; this message should be very rare. The only recourse is to remove some worthless files from that file system, or dump the entire system to a backup device, run *mkfs*(C) with more inodes specified, and restore the files from backup. *Abnormal.*

out of text

When programs linked with the *ld* **−i** or **−n** switch are run, a table entry is made so that only one copy of the pure text will be in memory even if there are multiple copies of the program running. This message appears when this table is full. The system refuses to run the program which caused the overflow. Note that there is only one entry in this table for each different pure text program. Multiple copies of one program will not require multiple table entries. Each "sticky" program (see *chmod*(C)) requires a permanent entry in this table; non-sticky pure text programs require an entry only when there is at least one copy being executed. Use *configure*(C) to raise the number of text segments. *Abnormal.*

panic: bad 287 int

Attempted execution of a real mode 287 instruction. *System incosistency, fatal.*

panic: blkdev

An internal disk I/O request, already verified as valid, is discovered to be refering to a nonexistent disk. *System inconsistency, fatal.*

panic: devtab

An internal disk I/O request, already verified as valid, is discovered to be refering to a nonexistent disk. *System inconsistency, fatal.*

panic: iinit

The super-block of the root file system could not be read. This message occurs only at boot time. *Hardware, fatal.*

panic: IO err in swap

A fatal I/O error occurred while reading or writing the swap area. *Hardware, fatal.*

panic: memory failure - parity error

A hardware memory failure trap has been taken. *System inconsistency, fatal.*

panic: memory management failure

An error occurred during memory management operations. *System inconsistency, fatal.*

panic: no fs

A file system descriptor has disappeared from its table. *System inconsistency, fatal.*

panic: no imt

A mounted file system has disappeared from the mount table. *System inconsistency, fatal.*

panic: no procs

Each user is limited in the amount of simultaneous processes he can have; an attempt to create a new process when none is available or when the user's limit is exceeded and refused. That is an occasional event and produces no console messages; this panic occurs when the kernel has certified that a free process table entry is available and can't find one when it goes to get it. *System inconsistency, fatal.*

panic: Out of swap

There is insufficient space on the swap disk to hold a task. The system refuses to create tasks when it feels there is insufficient disk space, but it is possible to create situations to fool this mechanism. *Abnormal, fatal.*

panic: general protection trap

General protection trap taken in kernel. *System inconsistency, fatal.*

panic: segment not present

An attempt has been made to access an invalid segment. It may also indicate the segment-not-present trap has been taken in the kernel. *System inconsistency, fatal.*

panic: Timeout table overflow

The timeout table is full. Timeout requests are generated by device drivers, there should usually be room for one entry per system serial line plus ten more for other usages. Use *configure*(C) to raise the number of timeout table entries.

panic: Trap in system

The CPU has generated an illegal instruction trap while executing kernel or device driver code. This message is preceded with an information dump describing the trap. *System inconsistency, fatal.*

panic: Invalid TSS

Internal tables have become corrupted. *System inconsistency, fatal.*

panic: unknown interrupt

The CPU received an interrupt via a supposedly unused vector. Typically, this event comes about when a hardware failure miscomputes the vector of a valid interrupt. *Hardware, fatal.*

proc on q

The system attempts to queue a process already on the process ready-to-run queue. *System inconsistency, fatal.*

Trap *type*

This message precedes a "panic:" message. The *type* is the trap number given by the processor. The message is followed by a dump of registers. *System inconsistency, fatal.*

Notes

Not all messages appear on all machines. Some messages are processor dependent.

Name

micnet – The Micnet default commands file.

Description

The **micnet** file lists the system commands that may be executed through the *remote* command. The file is required for each system in a Micnet network. Whenever a *remote* command is received through the network, the Micnet programs search the **micnet** file for the system command specified with the *remote* command. If found, the command is executed. Otherwise, the command is ignored and an error message is returned to the system which issued the *remote* command.

The file may contain one or more lines. If all commands may be executed, only the line

```
executeall
```

is required in the file. Otherwise, the commands must be listed individually. A line that defines an individual command has the form:

```
command=commandpath
```

Command is the command name to be specified in a *remote* command. **Commandpath** is the full pathname of the command on the specified system. The equal sign (=) separates the command and commandpath. For example, the line:

```
cat=/bin/cat
```

defines the command name *cat* (used in the *remote* command) to refer to the system command *cat* in the **/bin** directory.

When *executeall* is set, commands are sought in a series of default directories. Initially, the directories are **/bin** and **/usr/bin.** The default directories can be explicitly defined in the file by including a line of the form:

```
execpath=PATH=directory[:directory]...
```

The first part of the line, *execpath=PATH=*, is required. Each **directory** must be a valid pathname. The colon is required to separate directories. For example, the line:

execpath=PATH=/bin:/usr/bin:/usr/bobf/bin

sets the default directories to **/bin, /usr/bin,** and **/usr/bobf/bin.**

Files

/etc/default/micnet

See Also

aliases(M), netutil(C), systemid(M), top(M)

Notes

The **rcp** command cannot be executed from a remote system unless the **micnet** file contains either *executeall*, or the line

rcp=/usr/bin/rcp

Name

multiscreen – Multiple screens (device files)

Syntax

alt–Fn
alt–ctrl–Fn
alt–shift–Fn

Description

With the *multiscreen* feature, a user can access up to ten different "screens," each corresponding to a separate device file. Each screen can be viewed one at a time through the **primary monitor** video display.

The number of screens on a system depends upon the amount of memory in the computer. The system displays the number of enabled screens during the boot process.

Access

To see the next consecutive screen, enter:

Ctrl–PrtSc

To move to any screen from any other screen, enter:

alt–F*n* or alt–ctrl–F*n* or alt–shift–F*n*

where *n* is the number of one of the "F" function keys on the primary monitor keyboard. For example:

alt–F2

selects **tty02**, and all output in that device's screen buffer is displayed on the monitor screen.

Files

/dev/tty[01–10]	multiscreen devices (number available depends on system memory)

See Also

keyboard(HW), screen(HW), serial(HW), stty(C)

Notes

Any system error messages are normally output on the **console** device file (**/dev/console**). When an error message is output, the video display reverts to the **console** device file, and the message is displayed on the screen. The **console** device is the only teletype device open during the system boot sequence and when in single user, or system maintenance mode.

Limitations to the number of multiscreens available on a system does not affect the number of serial lines or devices available. See *serial*(M) for information on available serial devices.

Note that the keystrokes given here are the default for XENIX, but your keyboard may be different. If so, see *keyboard*(M) for the appropriate substitutes. Also, any key can be programmed to generate the screen switching sequences by using the mapkey utility.

Name

null – The null file.

Description

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

Files

/dev/null

Name

passwd – The password file.

Description

Passwd contains the following information for each user:

–Login name

–Encrypted password

–Numerical user ID

–Numerical group ID

–Comment

–Initial working directory

–Program to use as shell

Refer to *finger*(C) for information in the required format of the comment field for *finger*(C) to display the information. Each user is separated from the next by a newline. If the password field is null, no password is demanded; if the shell field is null, *sh*(C) is used.

This file resides in the directory **/etc**. Because the passwords are encrypted, the file has general read permission and can be used, for example, to map numerical user IDs to names.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (**.**, **/**, **0–9**, **A–Z**, **a–z**), except when the password is null, in which case the encrypted password is also null. Password aging is in effect for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced by the super-user.) The first character of the age denotes the maximum number of weeks for which a password is valid. A user who attempts to log in after his password has expired will be forced to supply a new one. The next character denotes the minimum period in weeks which must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) The first and second characters must have numerical values in the range 0–63, where the dot (.) is equal to 0 and lowercase *z* is equal to 63. If the numerical value of both characters is 0, the user will be forced to change his password the next time he logs in. If the second character is greater than the first, only the super-user will be able to change the password.

Files

/etc/passwd

See Also

login(M), passwd(C), a64l(S), getpwent(S), group(M), pwadmin(C).

Name

profile – Sets up an environment at login time.

Description

The optional file, **.profile**, permits automatic execution of commands whenever a user logs in. The file is generally used to personalize a user's work environment by setting exported environment variables and terminal mode (see **environ**(C)).

When a user logs in, the user's login shell looks for **.profile** in the login directory. If found, the shell executes the commands in the file before beginning the session. The commands in the file must have the same format as if they were entered at the keyboard. Any line beginning with the number sign (#) is considered a comment and is ignored. The following is an example of a typical file:

```
# Tell me when new mail comes in
MAIL=/usr/mail/myname
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
```

Note that the file **/etc/profile** is a system-wide profile that, if it exists, is executed for every user before the user's **.profile** is executed.

Files

$HOME/.profile
/etc/profile

See Also

env(C), login(M), mail(C), sh(C), stty(C), su(C), environ(M)

Name

setclock – Sets the system real-time (time of day) clock.

setclock [*time*]

Description

The **setclock** file sets the battery-powered, real-time time of day clock to the given *time*. If *time* is not given, the current contents of the battery-powered clock are displayed. The *time* must be a combination of digits with the form:

MMddhhmmyy

where *MM* is the month, *dd* is the day, *hh* is the hour, *mm* is the minute, and *yy* is the last two digits of the year. If *yy* is not given, it is taken from the current system time. For example, the command:

082615035

sets the time of day clock to 15:03 on August 26, 1985.

Files

/etc/setclock

See Also

clock(M)

Notes

Not all computers have battery-powered real-time time of day clocks. Refer to your computer's hardware reference manual.

Name

sxt – Pseudo-device driver

Description

Sxt is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the *sxt* driver. *Sxt* acts as a discipline manipulating a *real tty* structure declared by a real device driver. The *sxt* driver is currently only used by the *shl*(C) command.

Virtual ttys are named **/dev/sxt???** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt??0** (channel 0) and then execute a SXTIOCLINK *ioctl* call to initiate the multiplexing.

Only one channel, the *controlling* channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of *ioctl*(S) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(M), with the addition of the following:

TIOCEXCL Set *exclusive use* mode: no further opens are permitted until the file has been closed.

TIOCNXCL Reset *exclusive use* mode: further opens are once again permitted.

The second group are directives to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

- EINVAL The argument is out of range.
- ENOTTY The command was not issued from a real tty.
- ENXIO *linesw* is not configured with *sxt*.
- EBUSY An SXTIOCLINK command has already been issued for this real *tty*.

	ENOMEM	There is no system memory available for allocating the virtual tty structures.
	EBADF	Channel 0 was not opened before this call.

SXTIOCSWTCH — Set the controlling channel. Possible errors include:

EINVAL	An invalid channel number was given.
EPERM	The command was not executed from channel 0.

SXTIOCWF — Cause a channel to wait until it is the controlling channel. This command will return the error, *EINVAL*, if an invalid channel number is given.

SXTIOCUBLK — Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error *EINVAL* will be returned if an invalid number or channel 0 is given.

SXTIOCSTAT — Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error *EFAULT* will be returned if the structure cannot be written.

SXTIOCTRACE — Enable tracing. Tracing information is written to the console. This command has no effect if tracing is not configured.

SXTIOCNOTRACE — Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt??[0-7]	virtual tty devices
/usr/include/sys/sxt.h	driver specific definitions

SEE ALSO

shl(C), stty(C), ioctl(S), open(S), termio(M)

Name

systemid – The Micnet system identification file.

Description

The **systemid** file contains the machine and site names for a system in a Micnet network. A *machine name* identifies a system and distinguishes it from other systems in the same network. A *site name* identifies the network to which a system belongs and distinguishes the network from other networks in the same chain.

The **systemid** file may contain a *site name* and up to four different *machine names.* The file has the form:

[site-name]
[machine-name1]
[machine-name2]
[machine-name3]
[machine-name4]

The file must contain at least one machine name. The other machine names are optional, serving as alternate names for the same machine. The file must contain a site name if more than one machine name is given or if the network is connected to another through a uucp link. The site name, when given, must be on the first line.

Each name can have up to eight letters and numbers but must always begin with a letter. There is never more than one name to a line. A line beginning with a pound sign (#) is considered a comment line and is ignored.

The Micnet network requires one **systemid** file on each system in a network with each file containing a unique set of machine names. If the network is connected to another network through a uucp link, each file in the network must contain the same site name.

The **systemid** file is used primarily during resolution of aliases. When aliases contain site and/or machine names, the name is compared with the names in the file and removed if there is a match. If there is no match, the alias (and associated message, file, or command) is passed on to the specified site or machine for further processing.

Files

/etc/systemid

See Also

aliases(M), netutil(C), top(M)

Name

systty – System maintenance device.

Description

The file **/dev/systty** is the device on which system error messages are displayed. The actual physical device accessed via **/dev/systty** is selected during boot, and is typically the device used to control the bootup procedure. The default physical device **/dev/systty** is determined by *boot*(HW) when the system is brought up.

Initially **/dev/console** is linked to **/dev/systty**.

Files

/dev/systty

See Also

boot(HW), console(M)

Name

termcap – Terminal capability data base.

Description

The file **/etc/termcap** is a data base describing terminals. This data base is used by commands such as *vi*(C), *vsh*(C), Lyrix®, Multiplan™ and sub–routine packages such as *curses*(S). Terminals are described in *termcap* by giving a set of capabilities and by describing how operations are performed. Padding requirements and initialization sequences are included in *termcap*.

Entries in *termcap* consist of a number of fields separated by colons ':'. The first entry for each terminal gives the names that are known for the terminal, separated by vertical bars (|). For compatibility with older systems the first name is always 2 characters long. The second name given is the most common abbreviation for the terminal and the name used by *vi (C)* and *ex*(C). The last name given should be a long name fully identifying the terminal. Only the last name can contain blanks for readability.

Capabilities (including XENIX Extensions)

The following is a list of the capabilities that can be defined for a given terminal. In this list, (P) indicates padding can be specified, and (P*) indicates that padding can be based on the number of lines affected. The capability type and padding fields are described in detail in the following section "Types of Capabilities."

The codes beginning with uppercase letters (except for CC) indicate XENIX extensions. They are included in addition to the standard entries and are used by one or more application programs. As with the standard entries, not all modes are supported by all applications or terminals. Some of these entries refer to specific terminal output capabilities (such as GS for "graphics start"). Others describe character sequences sent by keys that appear on a keyboard (such as PU for PageUp key). There are also entries that are used to attribute special meanings to other keys (or combinations of keys) for use in a particular software program. Some of the XENIX extension capabilities have a similar function to standard capabilities. They are used to redefine specific keys (such as using function keys as arrow keys). The extension capabilities are included in the **/etc/termcap** file, as they are required for some XENIX utilities (such as *vsh*(C)). The more commonly used extension capabilities are described in more detail in the section "XENIX Extensions."

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not ^**H**
bs	bool		Terminal can backspace with ^**H**
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
CF	str		Cursor off
ch	str	(P)	Like cm but horizontal motion only, line stays same
CL	str		Sent by CHAR LEFT key
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
CO	str		Cursor on
cr	str	(P*)	Carriage return, (default ^**M**)
cs	str	(P)	Change scrolling region (vt100), like **cm**
cv	str	(P)	Like **ch** but vertical only.
CW	str		Sent by CHANGE WINDOW key
da	bool		Display may be retained above
DA	bool		Delete attribute string
db	bool		Display may be retained below
dB	num		Number of millisec of **bs** delay needed
dC	num		Number of millisec of **cr** delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of **ff** delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of **nl** delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give `:ei=:' if **ic**
EN	str		Sent by END key
eo	bool		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default ^**L**)
G1	str		Upper−right (1st quadrant) corner character
G2	str		Upper−left (2nd quadrant) corner character

Name	Type	Pad?	Description
G3	str		Lower-left (3rd quadrant) corner character
G4	str		Lower-right (4th quadrant) corner character
GC	str		Center graphics character (similar to "+")
GD	str		Down-tick character
GE	str		Graphics mode end
GG	num		Number of chars taken by GS and GE
GH	str		Horizontal bar character
GL	str		Left-tick character
GR	str		Right-tick character
GS	str		Graphics mode start
GU	str		Up-tick character
GV	str		Vertical bar character
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
HM	str		Sent by HOME key (if not **kh**)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	str		Insert mode (enter); give ':im=' if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by 'other' function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of 'keypad transmit' mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of 'other' keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in 'keypad transmit' mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on 'other' function keys
LD	str		Sent by line delete key
LF	str		Sent by line feed key
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by **vi** version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor
MP	str		Multiplan initialization string
MR	str		Multiplan reset string
ms	bool		Will scroll in stand-out mode
mu	str		Memory unlock (turn off memory lock)

Name	Type	Pad?	Description
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but doesn't scroll
NU	str		Sent by NEXT UNLOCKED CELL key
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
PD	str		Sent by PAGE DOWN key
pt	bool		Has hardware tabs (may need to be set with is)
PU	str		Sent by PAGE UP key
RC	str		Sent by RECALC key
RF	str		Sent by TOGGLE REFERENCE key
RT	str		Sent by RETURN key
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by **so** or **se**
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by **us** or **ue**
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
UP	str		Sent by up-arrow key (alternate to ku)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
WL	str		Sent by WORD LEFT key
WR	str		Sent by WORD RIGHT key
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like **ce** \r \n (Delta Data)
xs	bool		Standard out not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic **so** char (Teleray 1061)

A Sample Entry

The following entry describes the Concept–100, and is among the more complex entries in the *termcap* file. (This particular Concept entry is outdated, and is used as an example only.)

```
c1 |c100 |concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:\
        :cm=\Ea%+ %+ :co#80:dc=16\E^A:dl=3*\E^B:\
        :ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue over to multiple lines by giving a backslash (\) as the last character of a line. Empty fields can be included for readability between the last field on a line and the first field on the next. Capabilities in *termcap* are of three types: Boolean capabilities, which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence that can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has 'automatic margins' (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. The description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co**, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the rest of the string is sent to provide this delay. The delay can be either a integer, e.g., '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a ****, and the characters ^ and **** may be given as **\^** and ****. If it is necessary to place a colon (:) in a capability, it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability, it must be encoded as **\200**. The routines that deal with *termcap* use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it. To test a new terminal description, you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in **/etc/termcap.** TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor.

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, it should have the **am** capability. If the terminal can clear its screen, this is given by the **cl** string capability. If the terminal can backspace, it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H** in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over), it should have the **os** capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on (i.e., **am**).

These capabilities suffice to describe hardcopy and 'glass-tty' terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as:

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability. This capability uses *printf*(S) like escapes (such as **%x**) in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, its arguments are the line and then the column to which motion is desired, and the **%** encodings have the following meanings:

%d	replaced by line/column position, 0 origin
%2	like %2d - 2 digit field
%3	like %3d - 3 digit field
%.	like *printf*(S) %c
%+x	adds *x* to value, then %.
%>xy	if value > x adds y, no output
%r	reverses order of line and column, no output
%i	increments line/column position (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output
%D	Reverse coding (x-2*(x%16)), no output (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is 'cm=6\E&%r%2c%2Y'. The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, 'cm=^T%.%.'. Terminals that use '%.' need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** **^D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus 'cm=\E=%+ %+ '.

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, it should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen), this can be given as **ho**; similarly, a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, the sequence should be given as **ce**. If the terminal can clear from the current position to the end of the display, the sequence should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, the sequence should be given as **al**. Note that this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line on which the cursor rests, the sequence should be given as **dl**. This is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, the sequence can be given as **sb**, but **al** can suffice. If the terminal can retain display memory above, the **da** capability should be given, and if display memory can be retained below, then **db** should be given. These let the editor know that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to the insert/delete character that can be described using *termcap*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and entering text separated by cursor motions. Enter 'abc def', using local cursor motions (not spaces) between the 'abc' and the 'def'. Then position the cursor before the 'abc' and put the terminal in insert mode. If entering characters causes the rest of the line to shift rigidly and characters to fall off the end, your terminal does not distinguish between blanks and untyped positions. If the 'abc' shifts over to the 'def' which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for 'insert null'. No known terminals have an insert mode, not falling into one of these two classes.

The editor can handle both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Specify **im** as the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Specify **ei** as the sequence to leave insert mode (specify this with an empty value if you also gave **im** an empty value). Now specify **ic** as any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not support **ic**, terminals that send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode, these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as reverse video, blinking, or underlining – half bright is not usually an acceptable 'standout' mode unless the terminal is in reverse video mode constantly), the preferred mode is reverse video by itself. It is acceptable, if the code to change into or out of standout mode leaves one, or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do. Although it may confuse some programs slightly, it cannot be helped.

Codes to begin underlining and end underlining can be given as **us**, and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, the sequence can be given as **uc**. (If the underline code does not move the cursor to the right, specify the code followed by a nondestructive space.)

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), the sequence can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex*, the sequence can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed), even though it does not overstrike, you should give the capability ul. If overstrikes are erasable with a blank, this should be indicated by specifying **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not to transmit, enter these codes as **ks** and **ke**. Otherwise, the keypad is assumed always to transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl**, **kr**, **ku**, **kd**, and **kh**. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, ':ko=cl,ll,sf,sb:', which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete, but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl**, **kr**, **ku**, **kd**, and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the Mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than ^I to tab, the sequence can be given as **ta**.

Terminals that do not allow '~' characters to be displayed (such as Hazeltines), should indicate **hz**. Datamedia terminals that echo carriage-return-linefeed for carriage return, and then ignore a following linefeed, should indicate **nc**. Early Concept terminals, that ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **xx**.

If the leading character for commands to the terminal (normally the escape character) can be set by the software, specify the command character(s) with the capability **CC**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** is displayed before **if**. This is useful where **if** is **/usr/lib/tabset/std** , but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability, **tc**, can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since *termlib* routines search the entry from left to right, and since the **tc** capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be cancelled with **xx@** where xx is the capability. For example:

```
hn |2621nl:ks@:ke@:tc=2621:
```

This defines a 2621nl that does not have the **ks** or **ke** capabilities, and does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

XENIX Extensions

Capabilities This table lists the (previously listed) XENIX extensions to the termcap capabilities. It shows which codes generate information input from the keyboard to the program reading the keyboard and which codes generate information output from the program to the screen.

Name	Input/Output	Description
CF	str	Cursor off
CL	str	Sent by CHAR LEFT key
CO	str	Cursor on
CW	str	Sent by CHANGE WINDOW key
DA	bool	Delete attribute string
EN	str	Sent by END key
G1	str	Upper–right (1st quadrant) corner character
G2	str	Upper–left (2nd quadrant) corner character
G3	str	Lower–left (3rd quadrant) corner character
G4	str	Lower–right (4th quadrant) corner character
GC	str	Center graphics character (similar to +)
GD	str	Down–tick character
GE	str	Graphics mode end
GG	num	Number of chars taken by GS and GE
GH	str	Horizontal bar character
GL	str	Left–tick character
GR	str	Right–tick character
GS	str	Graphics mode start
GU	str	Up–tick character
GV	str	Vertical bar character
HM	str	Sent by HOME key (if not **kh**)
MP	str	Multiplan initialization string
MR	str	Multiplan reset string
NU	str	Sent by NEXT UNLOCKED CELL key
PD	str	Sent by PAGE DOWN key
PU	str	Sent by PAGE UP key
RC	str	Sent by RECALC key
RF	str	Sent by TOGGLE REFERENCE key
RT	str	Sent by RETURN key
UP	str	Sent by up–arrow key (alternate to ku)
WL	str	Sent by WORD LEFT key
WR	str	Sent by WORD RIGHT key

Cursor motion Some application programs make use of special editing codes. **CR** and **CL** move the cursor one character right and left respectively. **WR** and **WL** move the cursor one word right and left respectively. **CW** changes windows, when they are used in the program.

Some application programs turn off the cursor. This is accomplished using **CF** for cursor off and **CO** to turn it back on.

Graphic mode. If the terminal has graphics capabilities, this mode can be turned on and off with the **GS** and **GE** codes. Some terminals generate graphics characters from all keys when in graphics mode (such as the Visual 50). The other **G** codes specify particular graphics characters accessed by escape sequences. These characters are available on some terminals as alternate graphics character sets (not as a bit-map graphic mode). The **vt100** has access to this kind of alternate graphics character set, but not to a bit-map graphic mode.

Files

/etc/termcap File containing terminal descriptions

See Also

ex(C), curses(S), termcap(S), tset(C), vi(C), more(C), screen(HW)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

ex(C) allows only 256 characters for string capabilities, and the routines in *termcap(S)* do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, vs, and **ve** entries are specific to the *vi*(C) program.

Not all programs support all entries. There are entries that are not supported by any program.

XENIX termcap extensions are explained in detail in the software application documentation.

Refer to the *screen*(HW) manual page, for a description of the character sequences used by the monitor device on your specific XENIX System.

Name

terminals – List of supported terminals.

Description

The following list, derived from the file **/etc/termcap**, shows the terminal name (suitable for use as a TERM shell variable), and a short descsription of the terminal. The advice in *termcap*(M) will assist users in creating termcap entries for terminals not currently supported.

Name	Terminal
2621	hp 2621
2621wl	hp 2621 with labels
3045	Datamedia 3045a
4025	Tektronix 4024/4025/4027
4025-17	Tek 4025 17 line window
4025-17ws	Tek 4025 17 line window in workspace
4025ex	Tek 4025
5425	AT&T teletype
5425-w	AT&T teletype with 132 columns
8001	ISC8001
912b	Televideo 912b
925	Televideo 925
TWO	Altos Computer Systems II
a980	adds consul 980
aa	Ann Arbor
aaa	Ann Arbor ambassador/48 lines
aaadb	Ann Arbor ambassador 48/destructive backspace
act5s	skinny act5
adds	adds viewpoint
adm11	lsi adm11
adm12	lsi adm12
adm31	lsi adm31
adm3a	lsi adm3a
adm3a19.2	lsi adm3a at 19,200 baud
adm42	lsi adm42
ampex	Ampex dialogue
ansi	XENIX standard termcap entry for personal computers
ansic	Standard termcap entry for personal computers with color monitors.
b26	Burroughs ansi monitor with 29 lines
bh3m	Beehive IIIm
c100	Concept 100
c1004p	c100 w/4 pages
c100rv	c100 rev video
c100rv4p	c100 w/4 pages
c100rv4pna	c100 with no arrows
c100rvs	slow reverse Concept 100

c3102	Cromemco 3102
cci	cci 4574
cdc456	dc
cdc456tst	dc456tst
cie467	C. Itoh 467 color Graphics terminal
cie414	C. Itoh 414 Graphics terminal
cit80	C. Itoh 80
d132	Datagraphix 132a
datapoint	Datapoint 3360
delta	Delta data 5000
digilog	Digilog 333
dm1520	Datamedia 1520
dm1521	Datamedia 1521
dm2500	Datamedia 2500
dm3025	Datamedia 3025a
dt80	Datamedia dt80/1
dt80132	Datamedia dt80/1 in 132 char mode
du	dialup
dumb	unknown
ep40	Execuport 4000
ep48	Execuport 4080
esp925	Esprit 925
espHAZ	Esprit in Hazeltine mode
exidy	Exidy 2500
fox	Perkin elmer 1100
free100	Freedom 100
free110	Freedom 110
h1500	Hazeltine 1500
h1510	Hazeltine 1510
h1520	Hazeltine 1520
h19	Heathkit h19
hp	hp 264x series
ibm3101	IBM 3101-10
intext	ISC modified owl 1200
lisa	Apple Lisa XENIX monitor display (black on white)
macterm	Apple Macintosh terminal emulator in vt100 mode
microb	Micro bee series
microterm	Microterm act iv
microterm5	Microterm act v
mime	Microterm mime1
mime2a	Microterm mime2a (emulating an enhanced vt52)
mime2as	Microterm mime2a (emulating an enhanced soroc iq120)
mime3a	mime1 emulating 3a
mime3ax	mime1 emulating enhanced 3a
mimehb	half bright mime1
nabu	Nabu terminal
ot80	Onyx 80
owl	Perkin elmer 1200
pixel	Pixel terminal
pt1500	Convergent Technologies pt1500
qvt101	Qume vt101
qvt103	Qume vt103

qvt104	Qume vt104
qvt108	Qume vt108
qvt109	Qume vt109
qvt201	Qume vt201
qvt202	Qume vt202
regent	adds regent series
regent100	adds regent 100
regent25	adds regent 25
regent25a	adds regent 25a
rx303	Rexon 303
sb1	Beehive super bee
sb2	fixed superbee
sk8620	Seiko 8620
soroc	Soroc 120
sun	Sun Microsystems Workstation monitor
superbeeic	bee with insert char
t1061	Teleray 1061
t3700	dumb Teleray 3700
t3800	Teleray 3800 series
tek	Tektronix 4012
tek4014	Tektronix 4014
tek4014sm	Tektronix 4014 in small font
tek4023	Tektronix 4023
ti931	Texas Instruments 931
tvi910	televideo 910
tvi910+	televideo 910 PLUS
tvi912	Televideo 912
tvi950	Televideo 950
v55	Visual 55 emulation of vt52
v50	Visual 50 emulation of vt52
vi200	Visual 200
vi50	Visual's emulation of adds viewpoint
vi55	Visual using ADDS emulation
vt100	DEC vt100
vt100n	vt100 w/no init
vt100s	DEC vt100 132 cols 14 lines
vt100w	DEC vt100 132 cols
vt52	DEC vt52
vtz	Zilog vtz
wy50	Wyse 50
wy50vb	Wyse 50 with visible bell
wy50w	Wyse 50 with 132 columns
wy75	Wyse 75
wy75ap	Wyse 75 with applications and cursor keypad modes
wy75x	Wyse 75 with 132 columns
wy100	wyse 100
zen30	zentec 30

Files

/etc/termcap

See Also

tset(C), environ(M), termcap(M)

Name

terminfo – Terminal capability data base

Syntax

/usr/lib/terminfo/*/*

Description

terminfo is a data base describing terminals, used, *e.g.*, by *terminfo*(S). Terminals are described in *terminfo* by a set of capabilities that they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in *terminfo*.

Entries in *terminfo* consist of a number of fields separated by commas ','. White space after each ',' is ignored. The first entry for each terminal gives the various names that are known for the terminal. Each of these entries is separated by '|'. The first name given is the most common abbreviation for the terminal, (referred to as the "root name") the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should be in lower case and contain no blanks; the last name can contain upper case and blanks for readability.

Terminal names (except for the last entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, "hp2621". This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, a vt-100 in 132 column mode would be vt100-w. The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-w
-am	With auto margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-*n*	Number of lines on the screen	aaa-60
-na	No arrow keys (leave them in local)	c100-na
-*n*p	Number of pages of memory	c100-4p
-rv	Reverse video	c100-rv

In the following table, the "variable" is the name by which the programmer (using the *terminfo* library) accesses the capability. The "capname" is the short name used in the text of the database, and is used by a person updating the database. The "i.code" is the two letter internal code used in the compiled database, and always corresponds to the **termcap**(M) capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

(P)
indicates that padding may be specified

(G)
indicates that the string is passed through *tparm* with*parms* as given (#*i*).

(*)
indicates that padding may be based on the number of lines affected

(#*i*)
indicates the *i*th parameter.

(†) Not present in all versions of *termcap*.

Variable Booleans:	Cap-name	I. Code	Description
auto_left_margin,	bw	bw	cub1 wraps from column 0 to last column
auto_right_margin,	am	am	Terminal has automatic margins
beehive_glitch,	xsb	xb	Beehive (f1=escape, f2=ctrl C)
ceol_standout_glitch,	xhp	xs	Standout not erased by overwriting (hp)
eat_newline_glitch,	xenl	xn	Newline ignored after 80 cols (Concept)
erase_overstrike,	eo	eo	Can erase overstrikes with a blank
generic_type,	gn	gn	Generic line type (e.g., dialup, switch).
hard_copy,	hc	hc	Hardcopy terminal
has_meta_key,	km	km	Has a meta key (shift, sets parity bit)
has_status_line,	hs	hs	Has extra "status line"
insert_null_glitch,	in	in	Insert mode distinguishes nulls
memory_above,	da	da	Display may be retained above the screen
memory_below,	db	db	Display may be retained below the screen
move_insert_mode,	mir	mi	Safe to move while in insert mode
move_standout_mode,	msgr	ms	Safe to move in standout modes
over_strike,	os	os	Terminal overstrikes

status_line_esc_ok,	eslok	es	Escape can be used on the status line
teleray_glitch,	xt	xt	Tabs ruin, magic so char (Teleray 1061)
tilde_glitch,	hz	hz	Hazeltine; can not print ~'s
transparent_underline,	ul	ul	Underline character overstrikes
xon_xoff,	xon	xo	Terminal uses XON/XOFF handshaking

Numbers:

columns,	cols	co	Number of columns in a line
init_tabs,	it	it	Tabs initially every # spaces
lines,	lines	li	Number of lines on screen or page
lines_of_memory,	lm	lm	Lines of memory if > lines. 0 means varies
magic_cookie_glitch,	xmc	sg	Number of blank chars left by smso or rmso
padding_baud_rate,	pb	pb	Lowest baud where cr/nl padding is needed
virtual_terminal,	vt	vt	Virtual terminal number (UNIX system)
width_status_line,	wsl	ws	No. columns in status line

Strings:

back_tab,	cbt	bt	Back tab (P)
bell,	bel	bl	Audible signal (bell) (P)
carriage_return,	cr	cr	Carriage return (P*)
change_scroll_region,	csr	cs	Change to lines #1 through #2 (vt-100) (PG)
clear_all_tabs,	tbc	ct	Clear all tab stops (P)
clear_screen,	clear	cl	Clear screen and home cursor (P*)
clr_eol,	el	ce	Clear to end of line (P)
clr_eos,	ed	cd	Clear to end of display (P*)
column_address,	hpa	ch	Set cursor column (PG)
command_character,	cmdch	CC	Term. settable cmd char in prototype
cursor_address,	cup	cm	Screen rel. cursor motion row #1 col #2 (PG)
cursor_down,	cud1	do	Down one line
cursor_home,	home	ho	Home cursor (if no cup)
cursor_invisible,	civis	vi	Make cursor invisible
cursor_left,	cub1	le	Move cursor left one space
cursor_mem_address,	mrcup	CM†	Memory relative cursor addressing
cursor_normal,	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right,	cuf1	nd	Non-destructive space (cursor right)
cursor_to_ll,	ll	ll	Last line, first column (if no cup)
cursor_up,	cuu1	up	Upline (cursor up)
cursor_visible,	cvvis	vs	Make cursor very visible
delete_character,	dch1	dc	Delete character (P*)
delete_line,	dl1	dl	Delete line (P*)

dis_status_line,	dsl	ds	Disable status line
down_half_line,	hd	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode,	smacs	as	Start alternate character set (P)
enter_blink_mode,	blink	mb	Turn on blinking
enter_bold_mode,	bold	md	Turn on bold (extra bright) mode
enter_ca_mode,	smcup	ti	String to begin programs that use cup
enter_delete_mode,	smdc	dm	Delete mode (enter)
enter_dim_mode,	dim	mh	Turn on half-bright mode
enter_insert_mode,	smir	im	Insert mode (enter);
enter_protected_mode,	prot	mp	Turn on protected mode
enter_reverse_mode,	rev	mr	Turn on reverse video mode
enter_secure_mode,	invis	mk	Turn on blank mode (chars invisible)
enter_standout_mode,	smso	so	Begin stand out mode
enter_underline_mode,	smul	us	Start underscore mode
erase_chars	ech	ec	Erase #1 characters (PG)
exit_alt_charset_mode,	rmacs	ae	End alternate character set (P)
exit_attribute_mode,	sgr0	me	Turn off all attributes
exit_ca_mode,	rmcup	te	String to end programs that use cup
exit_delete_mode,	rmdc	ed	End delete mode
exit_insert_mode,	rmir	ei	End insert mode
exit_standout_mode,	rmso	se	End stand out mode
exit_underline_mode,	rmul	ue	End underscore mode
flash_screen,	flash	vb	Visible bell (may not move cursor)
form_feed,	ff	ff	Hardcopy terminal page eject (P*)
from_status_line,	fsl	fs	Return from status line
init_1string,	is1	i1	Terminal initialization string
init_2string,	is2	i2	Terminal initialization string
init_3string,	is3	i3	Terminal initialization string
init_file,	if	if	Name of file containing is
insert_character,	ich1	ic	Insert character (P)
insert_line,	il1	al	Add new blank line (P*)
insert_padding,	ip	ip	Insert pad after character inserted (p*)
key_backspace,	kbs	kb	Sent by backspace key
key_catab,	ktbc	ka	Sent by clear-all-tabs key
key_clear,	kclr	kC†	Sent by clear screen or erase key
key_ctab,	kctab	kt	Sent by clear-tab key
key_dc,	kdch1	kD†	Sent by delete character key
key_dl,	kdl1	kL†	Sent by delete line key
key_down,	kcud1	kd	Sent by terminal down arrow key
key_eic,	krmir	kM†	Sent by rmir or smir in insert mode
key_eol,	kel	kE†	Sent by clear-to-end-of-line key
key_eos,	ked	kS†	Sent by clear-to-end-of-screen key
key_f0,	kf0	k0	Sent by function key f0
key_f1,	kf1	k1	Sent by function key f1
key_f10,	kf10	ka	Sent by function key f10

key_f2,	kf2	k2	Sent by function key f2
key_f3,	kf3	k3	Sent by function key f3
key_f4,	kf4	k4	Sent by function key f4
key_f5,	kf5	k5	Sent by function key f5
key_f6,	kf6	k6	Sent by function key f6
key_f7,	kf7	k7	Sent by function key f7
key_f8,	kf8	k8	Sent by function key f8
key_f9,	kf9	k9	Sent by function key f9
key_home,	khome	kh	Sent by home key
key_ic,	kich1	kI	Sent by ins char/enter ins mode key
key_il,	kil1	kA†	Sent by insert line
key_left,	kcub1	kl	Sent by terminal left arrow key
key_ll,	kll	kH†	Sent by home-down key
key_npage,	knp	kN†	Sent by next-page key
key_ppage,	kpp	kP†	Sent by previous-page key
key_right,	kcuf1	kr	Sent by terminal right arrow key
key_sf,	kind	kF†	Sent by scroll-forward/down key
key_sr,	kri	kR†	Sent by scroll-backward/up key
key_stab,	khts	kT†	Sent by set-tab key
key_up,	kcuu1	ku	Sent by terminal up arrow key
keypad_local,	rmkx	ke	Out of "keypad transmit" mode
keypad_xmit,	smkx	ks	Put terminal in "keypad transmit" mode
lab_f0,	lf0	l0	Labels on function key f0 if not f0
lab_f1,	lf1	l1	Labels on function key f1 if not f1
lab_f10,	lf10	la	Labels on function key f10 if not f10
lab_f2,	lf2	l2	Labels on function key f2 if not f2
lab_f3,	lf3	l3	Labels on function key f3 if not f3
lab_f4,	lf4	l4	Labels on function key f4 if not f4
lab_f5,	lf5	l5	Labels on function key f5 if not f5
lab_f6,	lf6	l6	Labels on function key f6 if not f6
lab_f7,	lf7	l7	Labels on function key f7 if not f7
lab_f8,	lf8	l8	Labels on function key f8 if not f8
lab_f9,	lf9	l9	Labels on function key f9 if not f9
meta_on,	smm	mm	Turn on "meta mode" (8th bit)
meta_off,	rmm	mo	Turn off "meta mode"
newline,	nel	nw	Newline (behaves like cr followed by lf)
pad_char,	pad	pc	Pad character (rather than null)
parm_dch,	dch	DC†	Delete #1 chars (PG*)
parm_delete_line,	dl	DL†	Delete #1 lines (PG*)
parm_down_cursor,	cud	DO†	Move cursor down #1 lines (PG*)
parm_ich,	ich	IC†	Insert #1 blank chars (PG*)
parm_index,	indn	SF†	Scroll forward #1 lines (PG)
parm_insert_line,	il	AL†	Add #1 new blank lines (PG*)
parm_left_cursor,	cub	LE†	Move cursor left #1 spaces (PG)
parm_right_cursor,	cuf	RI†	Move cursor right #1 spaces (PG*)
parm_rindex,	rin	SR†	Scroll backward #1 lines (PG)
parm_up_cursor,	cuu	UP†	Move cursor up #1 lines (PG*)

pkey_key,	pfkey	pk	Prog funct key #1 to type string #2
pkey_local,	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit,	pfx	px	Prog funct key #1 to xmit string #2
print_screen,	mc0	ps	Print contents of the screen
prtr_off,	mc4	pf	Turn off the printer
prtr_on,	mc5	po	Turn on the printer
repeat_char,	rep	rp	Repeat char #1 #2 times. (PG*)
reset_1string,	rs1	r1	Reset terminal completely to sane modes
reset_2string,	rs2	r2	Reset terminal completely to sane modes
reset_3string,	rs3	r3	Reset terminal completely to sane modes
reset_file,	rf	rf	Name of file containing reset string
restore_cursor,	rc	rc	Restore cursor to position of last sc
row_address,	vpa	cv	Vertical position absolute (set row) (PG)
save_cursor,	sc	sc	Save cursor position (P)
scroll_forward,	ind	sf	Scroll text up (P)
scroll_reverse,	ri	sr	Scroll text down (P)
set_attributes,	sgr	sa	Define the video attributes (PG9)
set_tab,	hts	st	Set a tab in all rows, current column
set_window,	wind	wi	Current window is lines #1-#2 cols #3-#4
tab,	ht	ta	Tab to next 8 space hardware tab stop
to_status_line,	tsl	ts	Go to status line, column #1
underline_char,	uc	uc	Underscore one char and move past it
up_half_line,	hu	hu	Half-line up (reverse 1/2 linefeed)
init_prog,	iprog	iP	Path name of program for init
key_a1,	ka1	K1†	Upper left of keypad
key_a3,	ka3	K3†	Upper right of keypad
key_b2,	kb2	K2†	Center of keypad
key_c1,	kc1	K4†	Lower left of keypad
key_c3,	kc3	K5†	Lower right of keypad
prtr_non,	mc5p	pO†	Turn on the printer for #1 bytes

A Sample Entry

The following entry, which describes the Concept–100, is among the more complex entries in the *terminfo* file.

```
concept100 |c100| concept |c104 |c100-4p |concept 100,
     am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew,
     cols#80, cr=^M$<9>, cub1=^H, cud1=^J, cuf1=\E=,
     cup=\Ea%p1%' '%+%c%p2%' '%+%c,
     cuu1=\E;, cvvis=\EW, db, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
     ed=\E^C$<16*>, el=\E^U$<16>, eo, flash=\Ek$<20>\EK, ht=\t$<8>,
     il1=\E^R$<3*>, in, ind=^J, .ind=^J$<9>, ip=$<16*>,
     is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E,
     kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
     kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
     lines#24, mir, pb#9600, prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
     rev=\ED, rmcup=\Ev    $<6>\Ep\r\n, rmir=\E\200, rmkx=\Ex,
     rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\200,
     smcup=\EU\Ev  8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
     smul=\EG, tabs, ul, vt#8, xenl,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Comments lines begin with "#". Capabilities in *terminfo* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence that can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has *automatic margins* (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **cols**, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string valued capabilities, such as **el** (clear to end of line sequence) are given by the two-character code, an '=', and then a string ending at the next following ','. A delay in milliseconds may appear anywhere in such a capability, enclosed in $<..> brackets, as in **el**=\EK$<3>, and padding characters are supplied by *tputs* to provide this delay. The delay can be either a number, e.g., '20', or a number followed by an '*', i.e., '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of *lines* affected. This is always one unless the terminal has **xenl** and the software uses it.) When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. Both **\E** and **\e** map to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \l \r \t \b \f \s** give a newline, linefeed, return, tab, backspace, formfeed, and space. Other escapes include \^ for ^, \\ for \, \, for comma, \: for :, and \0 for null. (\0 will produce \200, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a ****.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above.

Preparing Descriptions

The most effective way to prepare a terminal description is to imitate the description of a similar terminal in *terminfo* and to build up a description gradually, using partial descriptions with *vi* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *terminfo* file to describe it or bugs in *vi*. To test easily a new terminal description you can set the environment variable TERMINFO to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/lib/terminfo**. To get the padding for insert line right (if the terminal manufacturer did not document it) a severe test is to edit a copy of **/etc/passwd** at 9600 baud, delete 16 or so lines from the middle of the screen, then hit the 'u' key several times quickly. If the terminal display is scrambled, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The *cols* numeric capability describes the number of columns on each line for the terminal. If the terminal is a CRT, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as TEKTRONIX 4010 series, as well as hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage return, control M.) If there is a code to produce an audible signal (bell, beep, etc) define this as **bel**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be defined as **cub1**. Similarly, codes to move to the right, up, and down should be defined as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over, for example, you would not normally use '**cuf1**= ' because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in *terminfo* are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin**, which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is when **bw** is given, in which case a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the *terminfo* file usually assumes that this is on; i.e., **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as:

```
33 |tty33 |tty |model 33 teletype,
bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM-3 is described as:

```
adm3 |3 |lsi adm3,
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,
ind=^J, lines#24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with *printf*(S) like escapes **%x** in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special % codes to manipulate it. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary.

The % encodings have the following meanings:

```
%%          outputs '%'
%d          print pop() as in printf
%2d         print pop() like %2d
%3d         print pop() like %3d
%02d
%03d        as in printf
%c          print pop() gives %c
%s          print pop() gives %s

%p[1-9]     push ith parm
%P[a-z]     set variable [a-z] to pop()
%g[a-z]     get variable [a-z] and push it
%'c'        char constant c
%{nn}       integer constant nn

%+ %- %* %/ %m
            arithmetic (%m is mod): push(pop() op pop())
%& %| %^    bit operations: push(pop() op pop())
%= %> %<    logical operations: push(pop() op pop())
%! %~       unary operations push(op pop())
%i          add 1 to first two parms (for ANSI terminals)

%? expr %t thenpart %e elsepart %;
            if-then-else, %e elsepart is optional.
            else-if's are possible ala Algol 68:
            %? c1 %t b1 %e c2 %t b2 %e c3 %t b3 %e c4 %t b4 %e %;
            ci are conditions, bi are bodies.
```

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use "%gx%{5}%-".

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cup** capability is cup=\E&%p2%2dc%p1%2dY$<6>.

The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, cup=^T%p1%c%p2%c. Terminals that use %c need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n ^D** and **\r**, as the system may change or discard them. (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cup=\E=%p1%' '%+%c%p2%' '%+%c. After sending '\E=', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two parameter sequence (as with the HP2645) and can be used in preference to **cup** . If there are parameterized local motions (e.g., move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the TEKTRONIX 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the \EH sequence on HP terminals cannot be used for **home**.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is positioned, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line on which the cursor is positioned, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** that take a single parameter and insert or delete that many lines can be given as **il** and **dl**. If the terminal has a settable scrolling region (like the vt-100) the command that sets this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, however, undefined after using this command. It is possible to get the effect of insert or delete line using this command – the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character that can be described using *terminfo*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the "abc" shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for insert null. While these are two logically separate attributes (one line vs. multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

terminfo can describe both terminals that have an insert mode, and terminals that send a simple sequence to open a blank position on the current line. To get into insert mode use the **smir** sequence. To leave insert mode use the **rmir** sequence. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n,* will repeat the effects of **ich1** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n,* to delete *n characters,* and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode,* representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the

TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc.**

Other capabilities to enter various highlighting modes include **blink** (blinking) **bold** (bold or extra bright) **dim** (dim or half-bright) **invis** (blanking or invisible text) **prot** (protected) **rev** (reverse video) sgr**0** (turn off *all* attribute modes) **smacs** (enter alternate character set mode) and **rmacs** (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist.

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **flash**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of both of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**.

This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the TEKTRONIX 4025, where **smcup** sets the command character to be the one used by *terminfo*.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1, kcuf1, kcuu1, kcud1,** and **khome** respectively. If there are function keys such as f0, f1, ..., f10, the codes they send can be given as **kf0, kf1, ..., kf10**. If these keys have labels other than the default f0 through f10, the labels can be given as **lf0, lf1, ..., lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1, ka3, kb2, kc1,** and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A "backtab" command that moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by the *tset*(C) command to determine whether to set the mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the terminfo description can assume that they are properly set.

Other capabilities include **is1**, **is2**, and **is3**, initialization strings for the terminal, **iprog**, the path name of a program to be run to initialize the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the terminfo description. They are normally sent to the terminal, by the *tset* program, each time the user logs in. They will be printed in the following order: **is1**; **is2**; setting tabs using **tbc** and **hts**; **if**; running the program **iprog**; and finally **is3**. Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1**, rs2, **rf**, and **rs3**, analogous to **is2** and **if**. Commands are normally placed in **rs2** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the vt-100 into 80-column mode would normally be part of **is2**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80 column mode.

If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is2** or **if**.

Delays

Certain capabilities control padding in the teletype driver. These are primarily needed by hard copy terminals, and are used by the *tset* program to set teletype modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** will cause the appropriate delay bits to be set in the teletype driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used.

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a vt-100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings to go to the beginning of the status line and to return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and fsl to get this effect.) The parameter **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to. If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, e.g., **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, tparm(repeat_char, 'x', 10) is the same as 'xxxxxxxxxx'.

If the terminal has a settable command character, such as the TEKTRONIX 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some XENIX systems: The environment is to be searched for a **CC** variable, and if found, all occurrences of the prototype character are replaced with the character in the environment variable.

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal.

If the terminal uses XON/XOFF handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as smm and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm**#0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the UNIX virtual terminal protocol, the terminal number can be given as **vt**.

Media copy strings that control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **mc5p** takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local; and **pfx** causes the string to be transmitted to the computer.

Glitches and Unusual Capabilities

Hazeltine terminals, which do not allow '~' characters to be displayed should indicate **hz**.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept and vt-100, should indicate **xenl**.

If **el** is required to get rid of standout (instead of merely writing normal text on top of it), **xhp** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a "magic cookie", that to erase standout mode it is instead necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the escape or control C characters, has **xsb**, indicating that the f1 key is used for escape and f2 for control C. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form *xx*.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be cancelled by placing **xx@** to the left of the capability definition, where xx is the capability. For example, the entry

2621-nl, smkx@, rmkx@, use=2621,

defines a 2621-nl that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

Files

/usr/lib/terminfo/?/*
files containing terminal descriptions compiled by *tic*(C)

See Also

terminfo(S), term(F), tic(C)

Notes

Neither *vi*, *tset*, nor any other XENIX command presently uses *terminfo*. It is intended that a full integration of *termcap* and *terminfo* will be provided in a future version of XENIX.

Name

termio – General terminal interface.

Description

All asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by *getty*(M) and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the "control terminal" for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(S). A process can break this association by changing its process group using *setpgrp*(S).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters can be entered at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a newline (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been entered. Also, no matter how many characters are requested in the read call, one line will be returned at most. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

Erase and kill processing is normally done during input. By default, a Ctrl-H or BACKSPACE erases the last character typed, except that it will not erase beyond the beginning of the line. By default, a Ctrl-U kills (deletes) the entire input line, and optionally outputs a newline character. Both these characters operate on a key-stroke basis, independent of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case, the escape character is not read. The erase and kill characters may be changed (see *stty*(C)).

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) Generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(S).

QUIT (Ctrl-\ or ASCII FS) Generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated, but a core image file (called **core**) will be created in the current working directory.

SWTCH (ASCII NUL) Is used by the job control facility, *shl*(C), to change the current layer to the control layer.

ERASE (Ctrl-H) Erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (Ctrl-U) Deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Ctrl-D or ASCII EOT) May be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL (ASCII LF) Is the normal line delimiter. It cannot be changed or escaped.

EOL (ASCII NUL) Is an additional line delimiter, like NL. It is not normally used.

STOP (Ctrl-S or ASCII DC3) Temporarily suspends output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Ctrl-Q or ASCII DC1) Resumes output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The START/STOP characters cannot be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding backslash (\) character, in which case no special function is carried out.

When the carrier signal from the dataset drops, a "hangup" signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for an end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as the previously typed characters have been entered. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds a given limit. When the queue has drained down to the given threshold, the program is resumed.

Several *ioctl*(S) system calls apply to terminal files. The primary calls use the following structure, defined in the file **<termio.h>**:

```
#defineNCC      8
struct  termio {
        unsigned short  c_iflag;    /* input modes */
        unsigned short  c_oflag;    /* output modes */
        unsigned short  c_cflag;    /* control modes */
        unsigned short  c_lflag;    /* local modes */
        char            c_line;     /* line discipline */
        unsigned char   c_cc[NCC];/* control chars */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	Ctrl-H
3	VKILL	Ctrl-U
4	VEOF/VMIN	EOT
5	VEOL/VTIME	NUL
6	Reserved	
7	SWTCH	NUL

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignores break condition
BRKINT	0000002	Signals interrupt on break
IGNPAR	0000004	Ignores characters with parity errors

PARMRK	0000010	Marks parity errors
INPCK	0000020	Enables input parity check
ISTRIP	0000040	Strips character
INLCR	0000100	Maps NL to CR on input
IGNCR	0000200	Ignores CR
ICRNL	0000400	Maps CR to NL on input
IUCLC	0001000	Maps uppercase to lowercase on input
IXON	0002000	Enables start/stop output control
IXANY	0004000	Enables any character to restart output
IXOFF	0010000	Enables start/stop input control

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise, if BRKINT is set the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the 3-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise, if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character will restart output which has been suspended.

If IXOFF is set, the system will transmit START characters when the input queue is nearly empty and STOP characters when nearly full.

The initial input control value is all bits clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocesses output
OLCUC	0000002	Maps lowercase to uppercase on output
ONLCR	0000004	Maps NL to CR-NL on output
OCRNL	0000010	Maps CR to NL on output
ONOCR	0000020	No CR output at column 0
ONLRET	0000040	NL performs CR function
OFILL	0000100	Uses fill characters for delay
OFDEL	0000200	Fills is DEL, else NUL
NLDLY	0000400	Selects newline delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Selects carriage return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Selects horizontal tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expands tabs to spaces
BSDLY	0020000	Selects backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Selects vertical tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Selects form feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to perform the carriage return function and the column pointer is set to 0 and the delays specified for CR will be used. Otherwise, the NL character is assumed to perform the linefeed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form feed or vertical tab delay is specified, it lasts for about 2 seconds.

Newline delay lasts about 0.10 seconds. If ONLRET is set, the carriage return delays are used instead of the newline delays. If OFILL is set, 2 fill characters will be transmitted.

Carriage return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits 2 fill characters, and type 2 transmits 4 fill characters.

Horizontal tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, 2 fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, 1 fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud

B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	External A
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Sends two stop bits, else one
CREAD	0000200	Enables receiver
PARENB	0000400	Parity enable
PARODD	0001000	Odd parity, else even
HUPCL	0002000	Hangs up on last close
CLOCAL	0004000	Local line, else dial-up
LOBLK	0010000	Block layer output

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Without this signal, the line is disconnected if it is connected through a modem. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, 2 stop bits are used, otherwise 1 stop bit. For example, at 110 baud, 2 stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. The data-terminal-ready and request-to-send signals are asserted, but incoming modem signals are ignored. If CLOCAL is not set, modem control is assumed. This means the data-terminal-ready and request-to-send signals are asserted. Also, the carrier-detect signal must be returned before communications can proceed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is B9600, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals
ICANON	0000002	Canonical input (erase and kill processing)
XCASE	0000004	Canonical upper/lower presentation
ECHO	0000010	Enables echo
ECHOE	0000020	Echoes erase character as BS-SP-BS
ECHOK	0000040	Echoes NL after kill character
ECHONL	0000100	Echoes NL
NOFLSH	0000200	Disables flush after interrupt or quit
XCLUDE	0100000	Exclusive use of the line

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus, these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least VMIN characters have been received or the timeout value VTIME has expired and at least one character has been input. This allows fast bursts of input to be read efficiently while still allowing single character input. The VMIN and VTIME values are stored in the position for the EOF and EOL characters respectively. VMIN and VTIME are interpreted as EOF and EOL if ICANON is set. Default VMIN and VTIME values are stored in the **/usr/include/sys/termio.h** file. To change these values, set ICANON to off and use *stty*(C) to change the MIN and TIME values as represented by EOF and EOL. The TIME value represents tenths of seconds.

If XCASE and ICANON are set, an uppercase letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

For:	Use:
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

For example, A is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters will not be done.

If XCLUDE is set, any subsequent attempt to open the TTY device using *open*(S) will fail for all users except the super-user. If the call fails, it returns EBUSY in *errno*. XCLUDE is useful for programs which must have exclusive use of a communications line. It is not intended for the line to the program's controlling terminal. XCLUDE must be cleared before the setting program terminates, otherwise subsequent attempts to open the device will fail.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(S) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA Gets the parameters associated with the terminal and stores them in the *termio* structure referenced by **arg**.

TCSETA Sets the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW Waits for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETAF Waits for the output to drain, then flushes the input queue and sets the new parameters.

Additional *ioctl*(S) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK Waits for the output to drain. If *arg* is 0, then sends a break (zero bits for 0.25 seconds).

TCXONC Starts/stops control. If *arg* is 0, suspends output; if 1, restarts suspended output.

TCFLSH If *arg* is 0, flushes the input queue; if 1, flushes the output queue; if 2, flushes both the input and output queues.

Files

/dev/tty

/dev/tty*

/dev/console

See Also

fork(S), ioctl(S), mapchan(F), mapchan(M), setgprp(S), signal(S), stty(C), tty(M)

Name

top, top.next – The Micnet topology files.

Description

These files contain the topology information for a Micnet network. The topology information describes how the individual systems in the network are connected, and what path a message must take from one system to reach another. Each file contains one or more lines of text. Each line of text defines a connection or a communication path.

The **top** file defines connections between systems. Each line lists the machine names of the connected systems, the serial lines used to make the connection, and the speed (baud rate) of transmission between the systems. Each line has the following format:

machine1 tty1a machine2 tty2a speed

machine1 and *machine2a* are the machine names of the respective systems (as given in the **systemid** files). The *ttys* are the device names (e.g., tty1a) of the connecting serial lines. The speed must be an acceptable baud rate (e.g., 110, 300, ..., 19200).

The **top.next** file contains information about how to reach a particular system from a given system. There may be several lines for each system in the network. Each line lists the machine name of a system, followed by the machine name of a system connected to it, followed by the machine names of all the systems that may be reached by going through the second system. Such a line has the form:

machine1 machine2 machine3 [machine4]...

The machine names must be the names of the respective systems (as given by the first machine name in the **systemid** files).

The *top.next* file must be present even if there are only two computers in the network. In such a case, the file must be empty.

In the **top** and **top.next** files, any line beginning with a number sign (#) is considered a comment, and is ignored.

Files

/usr/lib/mail/top

/usr/lib/mail/top.next

See Also

aliases(M), netutil(C), systemid(M), top(M)

Name

tty – Special terminal interface.

Description

The file **/dev/tty** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired, and when it is tiresome to find out what terminal is currently in use.

The general terminal interface is described in *termio*(M).

Files

/dev/tty
/dev/tty*

See Also

termio(M)

Name

ttys – Login terminals file.

Description

The **/etc/ttys** file contains a list of the device special files associated with possible login terminals, and defines which files are to be opened by the *init*(M) program on system start-up.

The file contains one or more entries of the form

state mode name

The *name* must be the filename of a device special file. Only the filename may be supplied, the path is assumed to be **/dev**. If *state* is "1", the file is enabled for logins; if "0", the file is disabled. The *mode* is used as an argument to the *getty*(M) program. It defines the line speed and type of device associated with the terminal. A list of arguments is provided in *getty*(M).

For example, the entry "1mtty02" means the serial line tty02 is to be opened for logging in at 9600 baud.

Files

/etc/ttys

See Also

disable(C), enable(C), getty(M), init(M), terminal(HW), terminals(M), tty(M)

Notes

The **/etc/ttys** file should only be edited when the system is in system maintenance mode. If it is edited when the system is in multiuser mode, the changes will not take effect until the system is rebooted, or until an *enable* or *disable* command is given. See the XENIX *Operations Guide*.

Name

TZ – Time zone environment variable.

Syntax

TZ=*sssn*[*ddd*[*m*][;*start*[/*time*],*end*[/*time*]]] ; export TZ

setenv TZ *sssn*[*ddd*[*m*][;*start*[/*time*],*end*[/*time*]]]

/etc/tz

Description

TZ is the shell environment variable for the time zone of the system and is set in the files **/etc/rc**, **/.profile**, and **/etc/default/login**.

The shell script **/etc/tz**, generally run during installation, prompts for the correct time zone and makes the changes in the appropriate files.

/etc/tz also prompts for the dates when time is shifted from standard to daylight time and back, and for the number of hours to shift (partial hours in the form of hh:mm:ss are acceptable).

Users living in a time zone different than that of the host machine may change *TZ* in their **$HOME/.profile** or **$HOME/.login** files.

TZ contains the following information:

(*sss*) One to nine letters designating the standard time zone.

(*n*) Number of hours past Greenwich mean time for the standard time (partial hours are valid e.g. 12:30:01). Positive hours are west of Greenwich, negative numbers are east of Greeenwich.

(*ddd*) One to nine letters designating the local daylight savings time (summer time) zone. If not present, summer time is assumed not to apply.

(*m*) Number of hours past Greenwich mean time for the summer time (partial hours are valid e.g. 11:30:01). Positive hours are west of Greenwich, negative numbers are east of Greeenwich. If *m* is not given, the distance to GMT during summer time is assumed to be one hour less than during standard time.

(*start*) The rule defining the day summer time begins. In the southern hemisphere, the ending day will be earlier in the year than the starting day.

(*end*) The rule defining the day summer time ends.

(*time*) The time of day the change to and from summer time occurs. The default is 02:00:00 local time.

The rules for defining the **start** and **end** of summer time are as follows:

J*n*	1 based Julian day n ($1 \leq n \leq 365$)*
n	0 based Julian day n ($0 \leq n \leq 364$)*
W*n.d*	nth ($1 \leq n \leq 53$)† day of week d ($0 \leq d \leq 6$)**
M*m.n.d*	nth ($1 \leq n \leq 5$)‡ day of week d in month m ($1 \leq m \leq 12$)**

* Leap days (February 29) are never counted; that is, February 28 (J59) is immediately followed by March 1 (J60) even in leap years.

** Sunday is the first day of the week (0). If d is omitted, Sunday is assumed.

† The 5th week of the month is always the last week containing day d, whether there are actually 4 or 5 weeks containing day d.

‡ The 53rd week of the year is always the last week containing day d, whether there are actually 52 or 53 weeks containing day d.

If **start** and **end** are omitted, current U.S. law is assumed.

For the simple expression of Eastern Standard/Daylight Time *TZ* is set as follows:

TZ=EST5EDT ; export TZ
(for *sh*(C) and *vsh*(C))

setenv TZ EST5EDT
(for *csh*(C))

The fully expressed *TZ* string for Eastern Standard/Daylight Time using the current U.S. law of changing to daylight saving time on the first Sunday in April, and back to standard time on the Sunday week in October at 2:00 a.m. local time would be:

TZ=EST05:00:00EDT04:00:00;M4.1.0/02:00:00,M10.5.0/02:00:00

To change the time zone for the entire system, run the shell script **/etc/tz** (as root) or use an editor to change the variable *TZ* in the files **/etc/rc**, **/.profile** and **/etc/default/login**. In **/etc/rc** the line changing the time zone (see the *sh* example above) must occur before the **/etc/asktime** command. The *TZ* variable in **/etc/default/login** causes the time zone to be set correctly on logging in and for programs such as *uucico*.

Files

/etc/rc
/etc/default/login
/etc/tz
$HOME/.profile
$HOME/.login

See Also

environ(M), date(C), ctime(S)

Notes

The *date*(C) automatically switches from Standard Time to Summer Time (Daylight Saving Time). Leap days are properly accounted for.

Changes to *TZ* are immediately effective, (i.e. if a process changes the *TZ* variable, the next call to a *ctime*(S) routine returns a value based on the new value of the variable).

Name

ungetty – Suspends/restarts a *getty* process.

Syntax

ungetty [**-r**, **-t**] *line*

Description

ungetty is used to suspend and restart a getty process on a serial line that has been set up as a dial-in/dial-out port. *dial*(M) invokes *ungetty* to toggle the port from dial-in to dial-out. (See */usr/lib/uucp/dial.c* for a sample program which uses *ungetty*.)

When *ungetty* is run on a serial line, it checks to see if the line is listed in */usr/lib/uucp/L-devices*. If so, it suspends the *getty* process on that line.

ungetty has the following options:

-r Restart a disabled line.

-t Test a line to see if it is enabled.

ungetty returns the following values when it is called with no options:

0 The line is not enabled.

1 Successful. The *getty* on that line is suspended.

2 Could not suspend the *getty*. This means that the line is being used by another program.

ungetty returns the following values when used with the **-t** and **-r** options:

0 **-r** or **-t**:
Line not enabled.

1 **-r**:
Successfully restarted.

-t:
Line needs restarting.

2 **-r** or **-t**:
Cannot restart getty.

See Also

cu(C), disable(C), enable(C), uucp(C), dial(M)

Name

utmp, wtmp – Formats of utmp and wtmp entries.

Syntax

#include <sys/types.h>
#include <utmp.h>

Description

These files, which hold user and accounting information for such commands as *who*(C), *write*(C), and *login*(M), have the following structure as defined by **<utmp.h>**:

```
#define    UTMP_FILE     "/etc/utmp"
#define    WTMP_FILE     "/etc/wtmp"
#define    ut_name       ut_user

struct utmp {
       char      ut_user[8];          /* User login name */
       char      ut_id[4];            /* usually line # */
       char      ut_line[12];         /* device name (console, lnxx) */
       short     ut_pid;              /* process id */
       short     ut_type;             /* type of entry */
       struct    exit_status {
          short      e_termination;   /* Process termination status */
          short      e_exit;          /* Process exit status */
       } ut_exit;                     /* The exit status of a process
                                          marked as DEAD_PROCESS. */
       time_t    ut_time;             /* time entry was made */
};

/*  Definitions for ut_type  */

#define EMPTY                0
#define RUN_LVL                  1
#define BOOT_TIME            2
#define OLD_TIME             3
#define NEW_TIME             4
#define INIT_PROCESS         5   /* Process spawned by "init" */
#define LOGIN_PROCESS        6   /* A "getty" process waiting for login */
#define USER_PROCESS         7   /* A user process */
#define DEAD_PROCESS         8
#define ACCOUNTING           9
#define UTMAXTYPE     ACCOUNTING /* Largest legal value of ut_type */
```

```
/* Special strings or formats used in the "ut_line" field when */
/* accounting for something other than a process */
/* No string for the ut_line field can be more than 11 chars + */
/* a NULL in length */
#define RUNLVL_MSG "run-level %c"
#define BOOT_MSG    "system boot"
#define OTIME_MSG   "old time"
#define NTIME_MSG   "new time"
```

Files

/usr/include/utmp.h
/etc/utmp
/etc/wtmp

See Also

getut(S), login(C), who(C), write(C)