Using XENIX 3.2.0 and its documentation

26-6460

XENIX 3.2.0 is a release of System III XENIX. It runs programs compiled for 3.2.0 and earlier versions of System III and will also run many XENIX V7 programs.

The manual *Introducing your Tandy 6000* provides useful information on how to set up your computer. The *System Administrator's Guide to XENIX* describes how to install XENIX on your hard disk and how to add users to your system.

Also included are some individual manual pages. These pages were part of the *XENIX Development System.* They are now included with the core because the utilities they describe have been moved to the core, or the utilities are new. If you do not have the development system, you do not have the binder that these pages would normally be placed in. Simply leave these pages in the *Multiuser Operating System* binder. If you own the development system, then insert these pages in the *XENIX Reference* binder.

Radio Shack

A Division of Tandy Corporation

875-8126

(3.2.0)

Using tsh with 20-megabyte Disk Cartridges

700-3039 26-6460

If you use *tsh* and own a 20 megabyte Disk Cartridge System (25-4066), you need to edit */etc/default/tsh* before attempting to save or restore files to disk cartridges using *tsh*. For more information on editing a file, refer to chapter 11 of *the System Administrator's Guide to XENIX*.

In the file */etc/default/tsh*, replace the following line:

```
c20:        cd    20    20896      20-Megabyte Cartridge Disk
```

with this line:

```
c20:        cd    16    20896      20-Megabyte Cartridge Disk
```

If you do not use *tsh* or do not own a 20-megabyte disk cartridge system, you do not need to make this change.

Radio Shack
A Division of Tandy Corporation

875-8125
(3.2.0)

Using the Dial Program

You can configure your system to dial one of three types of modems. They are:

- Tandy®/Radio Shack® Modems
- D.C. Hayes™ Modem
- VADIC™ Modem

Please note, however, that VADIC Modem dialing is provided on an "as-is" basis. Neither Tandy Corporation nor Radio Shack guarantee its functionality.

Your XENIX system is configured to dial Tandy/Radio Shack's Modem II, DC-1200, and DC-2212. You do not need to make any changes if you are using one of these modems. However, if you should alter your system to dial another type of modem, then you must reconfigure the system if you plan to use a Tandy/Radio Shack modem. Instructions for setting the XENIX system back to a Tandy/Radio Shack modem are given at the end of this document.

Using a Hayes Modem

To set XENIX to dial a Hayes modem follow these steps:

1. Log in as root. Type the following commands:

```
cd /usr/lib/uucp <ENTER>
rm dial <ENTER>
ln dialHAYES dial <ENTER>
```

2. Next, you must edit the file L-devices. XENIX uses this file to know the available dial-out lines. The L-devices file contains a line for each dial-out line. Each line has 4 fields:

   Field 1 — Type of Connection
   - "ACU" for a dial-out line
   - "DIR" for a direct connection

   Field 2 — Line Device
   - /dev/tty01 for Serial Channel A
   - /dev/tty02 for Serial Channel B
   - /dev/tty04 through /dev/ttyxx for any additional serial channels

   Field 3 — Dialer Device
   - Set the same as Line Device.

Field 4 Baud Rate
Can be 300, 1200, or 2400.

Edit the L-devices file using any XENIX text editor. See your XENIX documentation for information on editors.

3. Remove any line that begins with "ACU." You'll add new lines for the dial-out modems. You do not need to add any lines for the dial-in modems.

4. Add new lines that refer to the appropriate serial channel and baud rate. For example, to set the dialer for Serial Channel A, use one of these lines:

ACU tty01 tty01 1200	for 1200 baud
ACU tty01 tty01 300	for 300 baud

If your modem operates at only 300 baud, do not add the line for 1200 baud.

5. Connect the modem to the appropriate serial channel on your computer.

6. Set the Hayes Modem switches as described next:

Switch	Set	Description
1	UP	This setting is preferred if your cable connects DTR (pin 20) between the modem and the computer.
	DOWN	This setting forces DTR on if DTR is not connected, such as with a 3-wire cable. With this setting, the modem may not hang up if the other end doesn't hang up when the session is over.
2	UP	This setting is required for automatic dialing. Results are sent as English words.
3	DOWN	This setting is required for automatic dialing. Results codes are sent.
4	DOWN	This setting does not allow echoing characters in the local command mode. It is not required, but it does prevent problems with the modem and computer echoing characters at each other on a dial-in line. This can cause a load on the system.
5	DOWN	Modem won't answer incoming calls.
	UP	Modem answers incoming calls. This setting does not affect automatic dialing. However, UP is required if the line will be used as a dial-in line.
6	UP	This setting causes the Carrier Detect Line to reflect the real state of the carrier. This allows the computer to log a user out if the connection is broken when they dial in. Your cable must allow for Carrier Detect (pin 8 from the modem) for this to work properly.

Switch	Set	Description
7	UP	This setting is used for single-line telephone installations. It can differ depending on your telephone line.
8	DOWN	This setting enables command recognition. It is required for automatic dialing.

You should proceed to the section "Dial-in VS Dial-out."

Using a VADIC Modem

Follow these steps to set up a VADIC Modem:

1. Log in as root. Type the following commands:

```
cd /usr/lib/uucp <ENTER>
rm dial <ENTER>
ln dialVADIC dial <ENTER>
```

2. Follow Steps 2 - 5 detailed in "Using a Hayes Modem."

You should now proceed to the section "Dial-in VS Dial-out."

Dial-in VS Dial-out

A serial communication line connected to a modem may be used as either a dial-in line or a dial-out line; but, not both. If you plan to ever use the modem as a dial-in line, check the entry for that serial line in the /etc/ttys file. For most modems the second character of the entry should be a '3' for 1200/300. What this means is that XENIX tries 1200 baud until a Modem Break is received. Then it tries 300 baud. There are other codes you can use instead. They are:

Code	Baud Rates
3	1200/300
5	300/1200
P	300/1200/2400
Q	1200/2400/300
R	2400/300/1200

To change a serial line from a dial-in line to a dial-out line, log in as root and type this command:

```
disable tty0x  <ENTER>
```

Replace *x* with the appropriate serial line number. You should wait at least 60 seconds after entering the above command before entering any other enable or disable command on any other communication line.

To change a serial line from a dial-out line to a dial-in line, log in as root and type this command:

```
/usr/lib/uucp/dial -h /dev/tty0x 0 1200 <ENTER>
enable tty0x  <ENTER>
```

Replace *x* with the appropriate serial line number. You should wait at least 60 seconds after entering the above command before entering any other enable or disable command on any other communication line.

Rebooting the System

You may at some time have to reboot the system, or turn the power on or off on the modem. If you wish to do this and want the modem to operate as a dial-in line, you must enter the following commands after rebooting. Log in as root and type:

```
disable tty0x <ENTER>
   wait at least 60 seconds, then type:
/usr/lib/uucp/dial -h /dev/tty0x 0 1200 <ENTER>
enable tty0x <ENTER>
```

Replace *x* with the appropriate serial line number. You should wait at least 60 seconds after entering the above command before entering any other enable or disable command on any other communication line.

Using cu.s3

This example illustrates using **cu.s3** with Serial Channel A at 1200 baud:

```
cu.s3 -s 1200 phone-number
```

You must use **cu.s3** with Hayes and VADIC modems. The **cu** command only dials Tandy/Radio Shack modems.

Modems

dialing Tandy/Radio Shack modems, follow

odems.

IX uses this file to know the available dialout
h dialout line. Each line has 4 fields:

on

el A
el B
ny additional serial channels

lev/cu 0 or /dev/cua0.1200 for Serial Channel A
00 or /dev/cua1.1200 for Serial Channel B
300 or /dev/cua4.1200 through /dev/cua*x* and
/cua*x*.1200 for any additional serial channels.

text editor. See your XENIX documentation for

" You'll add new lines for the dial-out modems.
l-in modems.

serial channel and baud rate. For example, to set the
se lines:

for 1200 baud
for 300 baud

do not add the line for 1200 baud.

801

5. Connect the modem to the appropriate serial channel on your computer.

You can now follow the intructions given in your XENIX documentation for using m

Thank You

Radio Shack,
A Division of Tandy Corporation
7/85

XENIX® 3.2

Core Upgrade

Index

Commands (C)

Alphabetized List

Commands, Systems Calls, Library Routines, and File Formats

Contents

Commands (C)

dirname	Delivers directory part of pathname.
disable	Turns off terminals.
du	Summarizes disk usage.
dumpdir	Prints the names of files on a backup archive.
echo	Echoes arguments.
ed	Invokes the text editor.
enable	Turns on terminals.
env	Sets environment for command execution.
ex	Invokes a text editor.
expr	Evaluates arguments as an expression.
factor, primes	Factor a number, generate large primes.
false	Returns with a nonzero exit value.
file	Determines file type.
find	Finds files.
finger	Finds information about users.
fixperm	Corrects file permissions and ownership.
format	Organizes floppy diskettes and disk cartridges into tracks and sectors.
fsck	Checks and repairs file systems.
getopt	Parses command options.
grep,egrep, fgrep	Searches a file for a pattern.
grpcheck	Checks group file.
haltsys	Closes out the file systems and halts the CPU.
hd	Displays files in hexadecimal format.
head	Prints the first few lines of a stream.
hello	Write to another user.
id	Prints user and group IDs and names.
idle	Logs idle users off.
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.
look	Finds lines in a sorted list.
lpq	Line printer spooler queue status display.
lpr	Sends files to the lineprinter queue for printing.
lprm	Removes an entry from the line printer queue.
ls	Gives information about contents of directories.
mail	Sends, reads or disposes of mail.
mesg	Permits or denies messages sent to a terminal.
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.
mv	Moves or renames files and directories.
ncheck	Generates names from inode numbers.
netutil	Administers the XENIX network.
newgrp	Logs user in to a new group.
news	Prints news items.

nice	Runs a command at a different priority.
nl	Adds line numbers 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.
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.
qps	Quick process status report (quick "ps")
quot	Summarizes file system ownership.
random	Generates a random number.
rcp	Copies files across XENIX systems.
red	Invokes a restricted version of *ed*(C).
remote	Executes commands on a remote XENIX system.
restore	Invokes incremental file system restorer.
rm,rmdir	Removes files or directories.
rmdir	Removes directories.
rmuser	Removes a user from the system.
rsh	Invokes a restricted shell (command interpreter).
sddate	Prints and sets backup dates.
sdiff	Compares files side-by-side.
sed	Invokes the stream editor.
setmnt	Establishes /etc/mnttab table.
settime	Changes the access and modification dates of files.
sh	Invokes the shell command interpreter.
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 super-user or another user.
sum	Calculates checksum and counts blocks in a file.
sync	Updates the super-block.
sysadmin	Performs file system backups and restores files.
tail	Delivers the last part of a file.
tar	Archives files.
tee	Creates a tee in a pipe.
test	Tests conditions.
touch	Updates access and modification times of a file.
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 current XENIX name.
uniq	Reports repeated lines in a file.
units	Converts units.
uuclean	Clean-up the uucp spool directory.
uucp, uulog	Copies files from XENIX to XENIX.
uunow	Initiate a uucp sequence now.
uusend	Send a file to a remote host.
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	Invokes a screen-oriented display editor.
vsh	Invokes the visual shell.
wait	Awaits completion of background processes.
wall	Writes to all users.
w	Displays who is logged on and what they are doing.
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.
xmail	Improved mail program.
yes	Prints string repeatedly.

Name

adj_clock – Adjusts the system clock speed.

Syntax

adj_clock [-n *namelist*] [*+seconds /period* [s | m | h | d]]
adj_clock [-n *namelist*] [*-seconds /period* [s | m | h | d]]

Description

Adj_clock permits adjustment of the system clock speed. If the XENIX clock is slow, you may speed it up. If it is fast, you may slow it down.

For example, if the system administrator notices that the system clock ran slow by 5 seconds in 10 hours, they could correct the clock by typing:

adj_clock -5 /10

Adj_clock responds by setting and displaying the new clock correction factor.

You may add an 's', 'm', 'h', or 'd' at the end of the period specification to indicate seconds, minutes, hours, or days. The default is hours.

Only the superuser may run *adj-clock*.

Error Messages

These are some error messages which could result from *adj_clock*:

"adj_clock: You must be superuser to run this program"
The user was not logged in as root or did not use *su* to change to root.

"adj_clock: No namelist"
"adj_clock: Bad namelist"
The program was unable to find the needed symbols in /xenix, or the file specified in *namelist* if "-n" was specified.

"adj_clock: Can't open kmem"
The program was unable to open the file /dev/kmem.

"adj_clock: Can't find upage"
The program was unable to find required data in the system. This usually happens when /xenix is not the kernel with which the system was booted.

"adj_clock: Can't read adjustment"
The program was unable to read the old adjustment value.

"adj_clock: Can't write adjustment"
The program was unable to write the new adjustment value.

"adj_clock: [warning]: Minimum correction is +1 second every 18 hrs 12 min 15 sec"
The user gave an error rate that was so close to the system clock that it requires an adjustment of less than 1 second every 18 hours 12 minutes and 15 seconds.

"adj_clock: Can't correct for that large of an error"
"Maximum correction is +1 second every second"
The user gave an error rate so large that the system cannot correct it.

"adj_clock: Can't correct for that small of an error"
The user gave an error rate change so small that the system cannot correct it.

"adj_clock: Can't open xenix"
The program was unable to open the file /xenix.

"adj_clock: Can't read xenix"
The program was unable to read the file /xenix.

"adj_clock: Not enough data in /xenix"
The file /xenix did not match its namelist. This probably indicates a corrupted /xenix file.

"adj_clock: Can't seek to position"
The program cannot position to write the new correction value.

"adj_clock: Can't write data to xenix"
The program cannot write the new correction value.

"adj_clock: Missing argument to '-n' "
The user specified the '-n' option and did not specifiy a kernel name.

"adj_clock: Too many adjustment specifications"
The user specified more than one +/- seconds value on the command line.

"adj_clock: Bad adjustment specification"
The user gave an unintelligible +/- seconds specification.

"adj_clock: Too many period specifications"
The user specified more than one /period on the command line.

"adj_clock: Bad period specification"
The user gave an unintelligible /period specification.

"adj_clock [warning]: Unknown scalar"
The user gave an unknown scalar on the /period specification.

"adj_clock: You must specify both an adjustment and a period"
The user specified only an adjustment or a period.

"adj_clock: Unknown option "xxx" "
"Usage: adj_clock [-n kernel] [+ seconds | - seconds /period[s | m | h | d]] "
The user gave the program an unknown option.

Name

autoboot – Sets automatic reboot operation.

Syntax

autoboot [**–a** | **–ar** | **–n**] [*namelist*]

Description

The *autoboot* allows the system to be configured so the system automatically brings itself in multi-user mode if the user does not answer the boot prompt within 1 minute. *Autoboot* skips the time and date entry prompts, and automatically cleans the file system if required.

The *autoboot* option can sometimes bring up the system after a power failure. This depends mostly on whether the boot ROM can load and execute the boot track when the drive heads are not positioned near the boot track.

You may manually start an *autoboot* sequence by pressing the F1 key at the boot prompt, whether or not the autoboot option is turned on.

The *reboot* option reboots the system after it panics or shuts down with haltsys.

You may use one of the following three options:

–ar Turn on both the *autoboot* and *reboot* options. The system will automatically boot:

when it is reset
when it is shutdown with *haltsys*
when most panic messages occur

–a Turn on the *autoboot* option and turn off the *reboot* option. The system will only boot automatically when it is reset. *Haltsys* or panic messages will not cause a boot.

–n Turn off the *autoboot* and *reboot* options. The system will not restart itself unless requested to by the operator answering the boot prompt.

Namelist is the name of the XENIX kernel that the system is running. The default is /xenix.

Files

/etc/autoboot	autoboot program
/xenix	default namelist
/hdboot	non autoboot boot track
/hdbootr	autoboot boot track

Name

csh10 – Invokes an enhanced C shell with tenex-like file name and command completion.

Syntax

csh10 [-cefinstvVxX] [*arg* ...]

Description

Csh10 is an enhanced version of the Berkeley UNIX C shell *csh(1)*. It behaves like the C shell, except for two additional utilities:

1 Interactive file name, command, and user name recognition and completion.

2 Command, file, directory, or user list in the middle of a typed command.

A description of these features follows. For information on the other standard *csh* features, please see *csh(C)*.

File name completion

Typing in file names as arguments to commands has been simplified. It is no longer necessary to type in a complete name, only a unique abbreviation is necessary. Typing an escape to *csh10* will complete a file name, and echo the full name on the terminal. If the file name prefix typed in does not match any file name, *csh10* rings the terminal bell. The file name may be partially completed if the prefix matches several longer file names. If this is the case, *csh10* extends the name up to the ambiguous deviation, and rings the terminal bell.

Example

In the following example, assume the plus character "+" is where the user pressed the escape key. Assume that the current directory contained the following files:

DSC.OLD	bin	cmd	lib	memos
DSC.NEW	chaosnet	cmtest	mail	netnews
bench	class	dev	mbox	new

The command

% vi ch+

causes *csh10* to complete the command with the name *chaosnet*. If you type:

% vi D+

csh10 will extend the name to *DSC*, and ring the terminal bell to indicate partial completion.

File name completion works equally well when addressing other directories. *Csh10* also understands the tilde (~) convention for home directories in this context. For example:

% cd ~speech/data/fr+

does what one might expect. This may also be used to expand login names. For example:

% cd ~sysi+

This command performs a *cd* to the *sysinfo* home directory.

File/directory list

At any point in typing a command, you may request a list of available files, or a list of files which match your current specification. For example, while you are typing in:

% cd ~speech/data/bench/fritz/

you might want to find out what files or subdirectories are available in ~speech/data/bench/fritz. Pressing Control-D will list the available files, without eliminating the partial line that was previously typed in. *Csh10* lists the files in multicolumn format, sorted by columns. The list indicates directories with a trailing '/' and executable files with a trailing '*'. After printing the list, *csh10* reprints the partially entered command for you to complete.

Csh10 can also list files which have specific first letters in their names. In the example above, if you typed in:

% cd ~speech/data/hench/fr

and then pressed Control-D, *csh10* would list all files and subdirectories with names starting with 'fr' in ~speech/data/bench. Notice that the first example was a degenerate case with a null trailing file name. (The null string is a suffix of all strings.) Also, notice that a trailing slash is required to pass to a *new* subdirectory for both file name completion and listing. The degenerate case:

% ~^D

prints a full list of login names on the current system. In addition, *csh10* keeps track of your current working directory with the $cwd built-in variable.

Command name recognition

Command name recognition and completion works in the same manner as file name recognition and completion. *Csh10* uses the current value of the environment variable when searching for the command. For example:

% newa+

might expand to

% newaliases

while

% new^D

would list all commands along PATH that begin with "new".

As an option, if the shell variable *listpathnum* is set, then *csh10* prints a number next to each command on a ^D listing which indicates the index in PATH.

Csh10 does not use raw or cbreak mode. It works by temporarily setting the "additional" tty break character to ESC. This method does not add the overhead usually associated with programs that run in raw or cbreak mode.

If you select ESC as your default additional break character, *csh10* will be able to support recognition on typeahead.

Files

/usr/lib/csh.builtins/*	fake list of built-in commands

See Also

csh(1)

Notes

A ^D on a blank line (a degenerate case of the List option) logs you out. [SPACE][^D] lists all commands in the system along PATH.

Typing anything immediately after pressing ESC (before recognition expansion completes) will result in character juxtaposition or loss.

Terminal type is only examined the first time that you attempt to use recognition expansion.

Authors

Ken Greer, HP Labs; Mike Ellis, Fairchild, added command name recognition/completion.

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, then it also executes commands from the file .login there. In the normal case, the shell will then begin 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 an &. 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. This ! 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 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 typed 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, 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 typed in). 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 a 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 can be placed a sequence of modifiers, 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 do 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 side 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 into 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. Variable and command expansion may occur on strings enclosed in double quotes.

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.

Variable Substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. 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 thus 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 [...] matches any one of the characters enclosed. Within [...], 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, without any chance of error if 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 is 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 nonseekable 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 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. An \c causes the echo to complete without printing a newline. An \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 redirections occur 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. It 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 then 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 typing a CNTRL-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 says "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, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says "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, and real times 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 will be 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 inputs and outputs 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. Since 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
~/.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(F)

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

fixperm – Corrects file permissions and ownership.

Syntax

fixperm [**-c** | **-s**] specfile ...

Description

Fixperm makes the listed pathname conform to the specification. *Fixperm* does this for each line in the specification file (*specfile*). The specification file must be in the format described below. There are two available options in *fixperm*:

-s Alter the major/minor numbers of special files to match the specification when the file exists.

-c Perform the function of the -s option, and create any files that are in the specification but are not found on the disk. Files created with this option match the specification. Plain files are given zero length, directories are created empty.

If you do not select either option, *fixperm* leaves the major/minor numbers of existing special files alone. In addition, *fixperm* sets the *user id*, *group id*, *permissions*, *setuid bit*, *setgid bit*, and *sticky bit* to match the specification.

Only the superuser may run *fixperm*. *Fixperm* should not be *setuid*.

If the file type does not match the specification, *fixperm* does not process the file, but instead prints appropriate diagnostic messages. Valid file types include plain, directory, character special, and block special.

Specification File Format

The specification file format uses three (with an optional fourth) data fields per line. The fields are:

1 Permissions

2 Owner

3 Pathname

4 Device numbers.

If a file is not a plain file, the first character of the line must contain a file type code. The codes are:

b Block special

c Character special

d Directory

The rest of first field contains the file mode, including *setuid*, *setgid* and *sticky* bits. The second field is the *uid* in decimal, unless the *uid* is different from the *gid*. If the *uid* is different than the *gid*, the second field contains the *uid* (decimal), followed by a slash, followed by the *gid* (decimal). The third field contains the relative pathname. If the file is a block or character special file then the fourth field is required. The fourth field contains the major and minor numbers in decimal, separated by a slash.

Example:

Permissions	Owner	Pathname	Device-numbers
d755	3	bin	
711	3	bin/awk	
4711	0/3	bin/.mail	
700	0/3	bin/disable	
711	3	bin/ed	
d755	2	dev	
c222	0	dev/console	0/0
b666	2	dev/fd0	1/0
c666	2	dev/rfd0	5/0
c222	0	dev/lp	6/128
644	3	etc/default/backup	

See Also

mknod(C), chown(C), chgrp(C)

Diagnostics

Messages resulting from any of the following are directed to the standard error output:

Poorly formatted specification file
Incomplete system call
Unknown command line argument
Attempted use without being superuser

Notes

If a file is listed twice in the specificaiton file, *fixperm* uses the last valid entry. This can happen if the file has links and the specification file incorrectly edited.

Name

format – Organizes floppy diskettes and disk cartridges into tracks and sectors.

Syntax

format [**-qy**][**-ds** | **-dd**] [**-i** *interleave*] *specialfile*

Description

Format formats two types of media: floppy disks and disk cartridges, based on the *specialfile* specified. By default, *format* performs a read-verify if a floppy diskette is formatted. If the **-q** option is selected and a floppy diskette is being formatted, no read-verification is performed, allowing the diskette to be formatted more quickly.

The *specialfile* must be the name of the I/O device special file that corresponds to the floppy disk drive or disk cartridge drive.

The *-i* option is only used with cartridges and allows you to specify the interleave used to format them with. You may use the following values for *interleave*:

10-meg cartridges
1, 2, 4, 8, 16, 32

20-meg cartridges
1, 2, 3, 4, 5, 7, 8, 9, 11, 13, 15, 16, 17, 19, 21, 23, 25, 27, 29, 31, 32

The *-d* option is only used when formatting floppies and allows you to specify the density to format the floppy with. Using *-d s* formats the floppy in single density with 128 byte sectors. Using *-d d* formats the disk as a XENIX disk (double density, 512 byte sectors).

For example, the command

format fd0

formats a diskette in floppy drive 0. The command

format cd0

formats the disk cartridge in cartridge drive 0 (if installed).

Formatting a floppy disk or disk cartridge destroys the data on it. By default, *format* checks to see whether data is stored on the media before formatting it. If data is found, *format* displays the message

Warning: *device* contains data. Overwrite? (y/n):

If you still wish to overwrite the data, press 'Y' and press RETURN. Otherwise, press 'N' and press RETURN. This prompt is not displayed if you select the **-y** option.

Note that the floppy disk or disk cartridge must be in the appropriate drive before invoking the command.

Files

/bin/format

Notes

An error message appears on the console if you run *format* on media that has never been formatted before. This is normal and the format continues.

Format only formats the diskette in the drive the system unit. Keep this in mind if you are working from a terminal.

The disk cartridge uses one track for the boot track. During the formatting procedure, information describing the size and shape of the formatted media is writen to the cartridge. Each cartridge contains spare areas that are used to maintain a constant-sized area for data storage. If these extra areas are all used, then the cartridge will no longer format.

A 10-megabyte cartridge has 9,760 blocks available for use by **mkfs**, **tar**, or **restore**. A 20-megabyte cartridge has 20,880 blocks available.

Never remove a floppy diskette from its drive until the drive light goes out.

Name

hello – Write to another user

Syntax

hello *user* [*ttyname*]

Description

Hello copies lines from your terminal to that of another user. When first called, *hello* sends the message: Hello from *yourname yourttyname* ... The recipient of the message should *hello* back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *hello* writes "bye now" on the other terminal and exits.

If you want to write to a user who is logged at more than one terminal, use the *ttyname* argument to indicate the appropriate terminal name.

Permission to *hello* may be denied or granted by use of the *mesg* command which by default allows writing. Certain commands, such as *nroff* and *pr* disallow messages in order to prevent messy output.

If you try to write to a user who has messages disabled, *hello* sends mail to the user saying that a *hello* was attempted by you. This also happens if the user is not currently logged in. The following protocol is suggested for using *hello*: when you first write to another user, wait for him to write back before starting to send a message. Each party should end their message with a distinctive signal, such as "o" for "over" so the other knows to reply. Another suggestion is the use of "oo" for "over and out" when conversation is about to be terminated.

Files

/etc/utmp
/bin/sh

See Also

mesg(C), who(C), mail(C)

Name

idle – Logs idle users off.

Syntax

idle [-d]

Description

Idle finds users that have been logged in and idle for more than a specified period of time, and logs them out. *Idle* is usually invoked by *cron*, the clock daemon.

If idle is invoked with the '-d' flag, it only reports user idle times, and the names of any processes that idle users are running. In this case, *idle* only reports on idle users and processes, and does not take any action. You can use this to see whom *idle* would log off if run without the '-d' option.

The time allowed, and other items are set in the file /etc/default/idle. All time values are in minutes. The items that you can control are:

NULLFILE The list of programs that are not considered work. The user is considered idle if running nothing but these programs.

LOGFILE Where to record logouts. When set, a line is recorded here for each user that *idle* logs out.

C_LIMIT The normal idle time limit. This determines how long *idle* will wait before logging out an idle user.

CD_LIMIT The normal idle time limit for dial-in lines. A line is considered a dial-in line if it starts with ''ttyd''. For example, ttyd0 would be considered a dial-in line.

CB_LIMIT The normal time limit for work programs. A work program is any program that is not listed in NULLFILE.

CDB_LIMIT The normal time limit for work programs invoked through a dial-up line. A work program is any program that is not listed in NULLFILE.

CHECK_INTERVAL This tells *idle* how often *cron* is to run it. For example, if you want *idle* to run every 10 minutes, put a "10" here, and put the line:

0,10,20,30,40,50 * * * * /usr/lib/idle

into /usr/lib/crontab.

Error Messages

"idle: Unknown argument"
The user gave *idle* an argument other than '-d'.

"idle: You need to run this program as root"
The user tried to run *idle* to log users off, and was not logged in as the superuser.

"idle: Can't open /etc/utmp"
The program was unable to open the login information file /etc/utmp.

"idle: Can't stat /dev/tty??"
The program was unable to find the idle time of the specified terminal.

"idle: Can't run "ps";"
The program was unable to execute the system utility *ps*.

"idle: Can't open /dev/tty??"
The program was unable to open the specified terminal to warn the user.

"WARNING fron Idle Daemon at *time*"
"If you do not type something in about *nn* minutes, you will be logged off."
This is the message that is used to warn the idle user.

Name

lc – Lists directory contents in columns.

Syntax

lc [**–1ACFRabcdfgilmnqrstux**] *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 the user is not root, this option displays all files that begin with "." (except "." and ".." themselves). If the user is root, this option does not display the files that begin with ".".

–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; usually "." and ".." are 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.

–o The same as **–l,** except that the group is not printed.

–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.

–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 printed.

Files

/etc/passwd	To get user IDs for ''lc –l''
/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.

Name

lpq – line printer spooler queue status display.

Syntax

lpq

Description

Lpq prints the line printer queue. Each entry in the queue is printed showing the owner of the queue entry, an identification number, the size of the entry in characters, and the file which is to be printed. The *id* is useful for removing a specific entry from the printer queue using *lprm* (C).

Files

/usr/spool/lpd/* spool area

See Also

lprm(C), lpr(C)

Name

lprm – Removes an entry from the line printer queue.

Syntax

lprm [*id...*] [*filename*] [*owner*]

Description

Lprm removes an entry from the line printer queue. The *id*, *filename*, or *owner* should be the same as *lpq*(C) reports. *Lprm* will remove all of the appropriate files. *Lprm* will print the *id* of each file removed from the queue.

Files

/usr/spool/lpd/*	spool area
/usr/lib/lpd	printer daemon

See Also

lpq(C), lpr(C)

Name

ls – Gives information about contents of directories.

Syntax

ls [**–logtasdrucif**] 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 several options:

–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.

–o The same as **–l**, except that the group is not printed.

–g The same as **–l**, except that the owner is not printed.

–t Sorts by time of last modification (latest first) instead of by name.

–a Lists all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.

–s Gives size in (512 byte) blocks, including indirect blocks, for each entry.

–d If argument is a directory, lists only its name; often used with **–l** to get the status of a directory.

–r Reverses the order of sort to get reverse alphabetic or oldest first, as appropriate.

–u Uses time of last access instead of last modification for sorting (with the **–t** option) and/or printing (with the **–l** option).

–c Uses time of last modification of the inode (mode, etc.) instead of last modification of the file for sorting (**–t**) and/or printing (**–l**).

–i For each file, prints the inode number in the first column of the report.

–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

See Also

chmod(C), find(C), l(C), lc(C)

Notes

Newline and tab are considered printing characters in filenames.

All switches must be given as one argument. Thus "ls –lsg" is legal, but "ls –l –s –g" is not.

Name

mknod – Builds special files.

Syntax

/etc/mknod name [**c**] [**b**] major minor

/etc/mknod name **p**

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.

Mknod can also be used to create named pipes with the **p** option.

Only the super-user can use the first form of the syntax.

See Also

mknod(S)

Name

mount – Mounts a file structure.

Syntax

/etc/mount [special-device directory [**–r**]]

/etc/umount special-device

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.

The *mount* and *umount* commands maintain a table of mounted devices. If invoked with no arguments, for each special device *mount* prints the name of the device, 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 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*.

Files

/etc/mnttab Mount table

See Also

umount(C), mount(S), mnttab(F)

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/mnttah** from a non-shutdown stoppage) or no results (due to an empty mnttab from a *shutdown* stoppage).

When in multi-user mode 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 do not use a lock file to guarantee exclusive access to **/etc/mnttab**, and it is possible to access the file while its contents are being updated. Executing *mount* and/or *umount* commands in parallel can cause the file to be corrupted and may cause possible warning messages in subsequent operations. This is not a problem in practice since *mount* and *umount* are not frequent operations and conflicts can be avoided by not performing mounts and/or umounts as background tasks.

For the purpose of system security, this command is available only to the super user.

Name

qps – Quick process status report (quick "ps")

Syntax

qps

Description

Qps is a fast version of the ps(C) program. *Qps* prints information about all active processes.

The *qps* program does not have any options.

Qps prints information in columns. The column headings and their meanings are given below:

F 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 number. Process flag bits and their meanings are:

0x01	in core
0x02	system process
0x04	locked in core for physical I/O
0x08	being swapped
0x10	being traced by another process

S The state of the process:

S	sleeping
W	waiting
R	running
I	intermediate
Z	terminated
T	stopped

PID The process ID

PPID The ID of the parent process

TTY The controlling terminal for the process

UID The user ID of the process owner

Wchan The event for which the process is waiting or sleeping; "[run]" means that it is running

Command The name of the command

Name

shutdown – Terminates all processing.

Syntax

/etc/shutdown [[+] *minutes*] [**–q**] [**–h** | **–r** | **–su**] [**–n** *namelist*]

Description

Shutdown is part of the XENIX operating procedures. *Shutdown* terminates all currently running processes in an orderly and cautious manner. The *minutes* argument is the number of minutes before a shutdown will occur. The value must range from 0-30 inclusive. The default value is 0 (immediate shutdown). Only root can use shutdown, although it can be run from any terminal.

Shutdown has five options:

–q *Shutdown* shuts down the system as quickly as possible. A *Shutdown* does not issue a warning to users that the system is being shutdown. Do not specify a number of minutes when using –q.

You may use any **one** of the following three options. If you do not specify one, *shutdown* uses –h as the default. The three options are:

–h *Shutdown* halts the system.

–r *Shutdown* reboots the system. The system returns to the boot prompt. If *autoboot* is enabled, the system will automatically come up in multi-user mode if the boot prompt is not answered within one minute. If you are at the console and wish to speed up the boot process, press the F1 key. After pressing the F1 key, the system will boot XENIX and come up in multi-user mode with no further prompts. You may also boot normally by pressing ENTER, or entering the name of the XENIX kernel. If *autoboot* is disabled, the system will wait for the user at the console to answer the boot prompt.

–su *Shutdown* changes the system to single-user mode. The system changes to single-user mode, and shuts down multi-user mode.

You should use the –n option if the name of the kernel that you are booted on is not /xenix.

–n *Shutdown* uses a different pathname for the kernel. Type the pathname of the alternate kernel after –n.

If you do not specify any arguments when you run *shutdown*, the program will prompt you for them. If you run *shutdown* with at least one argument, the program will use defaults for the unspecified ones. The list of defaults is:

0 minutes (time until shutdown)

–h (halt system)

–n /xenix (standard kernel pathname)

No prompt messages

Shutdown operates in the following sequence:

1 *Shutdown* sends a broadcast message to all users, informing them of the pending shutdown.

2 *Shutdown* updates all file system super-blocks (see *sync*(*C*)) to ensure file system integrity.

3 *Shutdown* shuts down the system.

See Also

autoboot(C), sync(C), umount(C), wall(C)

Diagnostics

Device busy is the most common error diagnostic that will occur when running *shutdown*. This error occurs when *shutdown* cannot dismount a file system. See *umount(C)*.

Files

/etc/shutdown	shutdown program
/xenix	default kernel namelist

Notes

To cancel *shutdown*, press the BREAK key before the time to shut down expires. *Shutdown* ignores the BREAK key after the time expires.

Shutdown does not lock hard disk heads.

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**, **u**, **e**, and **r**. 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 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. Do not use the **u** option with a blocking factor other than 1 (see the b option).

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 type 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 print 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 taken from the appropriate line of /etc/default/tar, and the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above) or to explicitly specify that the archive is unblocked ("b 1"). 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. A lone dash (–) signifies that arguments will be taken from the standard input.

l Tells *tar* to print 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 printed.

m Tells *tar* to not restore the modification times. The modification time of the file will be 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 to be 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 during extraction. Any leading ''/''characters are removed from filenames. Arguments given should match the relative (rather than the absolute) pathnames of the files extracted.

Tar reads **/etc/default/tar** to obtain default values for the device, blocking factor and volume size. If no numeric key is specified on the command, *tar* will look for a line in the **default** file beginning with the string *"archive0="*. Following this pattern are 3 blank separated strings indicating the values for the device, blocking factor, and volume size, 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"
```

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 3 strings, but the first entry (specifying the device) is not significant. The default file **/etc/default/tar** must exist.

Examples

If the name of a floppy disk device is **/dev/fd1**, then a tar format file can be created on this device by typing:

```
assign /dev/fd1
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 then the contents of the directory are recursively archived. To print a listing of the archive, type:

```
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 typing:

```
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** in the examples.

Files

/etc/default/tar	Default devices, blocking and volume sizes
/tmp/tar*	

Diagnostics

Prints an error message about bad key characters and archive read/write errors.

Prints an error message if not enough memory is available to hold the link tables.

Will not function without a valid **/etc/default/tar**.

Notes

There is no way to ask for the *n*th occurrence of a file.

The **u** option can be slow.

A blocking factor should not be used with archives that are going to be updated. Either explicitly specify the b option with an argument of 1, or specify a device with a default blocking factor (in /etc/default/tar) of 1. If the archive is on a disk file, the **b** option should be specified as 1, or specify a device with a default blocking factor of 1, because updating an archive stored on disk can destroy it. In order to update (**r** or **u** option) a tar archive, do not use raw magtape and use the b option with a blocking factor of 1. This applies both when updating and when the archive was first created.

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.

Don't do:

```
tar xfF - -
```

This would imply taking two things from the standard input at the same time.

Name

vi – Invokes a screen-oriented display editor.

Syntax

vi [*–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").

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.

Options available on the vi command line:

–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 readonly 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*. 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. This allows editing of the buffer without immediately destroying the contents of the original file.

When you invoke vi with a single filename argument, the named file is copied to a temporary editing buffer. When the file is written out, the temporary file is written back to the named 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 escape to 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. These keys are used to edit, delimit, or abort commands and command lines.

ESC Used to return to vi command mode, cancel partially formed commands.

RETURN Terminates ex commands when in ex escape mode. Also used to start a new line 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.

CNTRL-U Moves the cursor back to the first character of the insertion, and restarts insertion.

CNTRL-V Removes the special significance of the next typed character. Use CNTRL-V to insert control characters. Line feed and CNTRL-J cannot be inserted in the text except as newline characters. CNTRL-Q and CNTRL-S are trapped by the operating system before they are interpreted by vi, so they too cannot be inserted as text.

CNTRL-W Moves the cursor back to the first character of the last inserted word.

CNTRL-T During an insertion, with the *autoindent* option set and at the beginning of the current line, typing this character will insert *shiftwidth* whitespace.

CNTRL-@ If typed 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 CNTRL-@ cannot be part of a file, even if quoted.

Invoking and Exiting Vi

To enter vi type:

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 typed. 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 overriden 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 arrow keys (if available on your terminal), the "h" "j", "k" , and "l" cursor keys, and SPACE, BKSP, CNTRL-N, and CNTRL-P. These three sets of keys perform identical functions.

Forward Space – l, SPACE, or -->

Syntax: **l**
SPACE
-->

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 <--

Syntax: **h**
BKSP
<--

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, CNTRL-N, and LF

Syntax: **+**
RETURN

Function: Moves the cursor down to the beginning of the next line.

Syntax: **j**
CNTRL-N
LF
(down arrow)

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, CNTRL-P, and –

Syntax: **k**
CNTRL-P
(up arrow)

Function: Moves the cursor up one line, remaining in the same column. If a count is given then the cursor is moved *count* lines.

Syntax: **–**

Function: Moves the cursor up to the beginning of the previous line. If a count is given then 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, then 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 CNTRL-G.

Column – |

Syntax: [*column*]**|**

Function: Moves the cursor to the column in the current line given by *column*. If no *column* is given then 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 non-whitespace 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 non-whitespace characters. If the cursor is already within a word, then 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 non-whitespace characters. If the cursor is already within a word, then 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 non-whitespace 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 (CNTRL-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 upper left comer of screen. Use this command to quickly move to the top of the screen. If an *offset* is given, then 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, then 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*

VI (C)

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, then the cursor is moved to previous context. If a single character is given after either quotation mark, then 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 – CNTRL-U and CNTRL-D

Syntax: [*size*]**CNTRL-U**
[*size*]**CNTRL-D**

Function: Scrolls the screen up a half window (CNTRL-U) or down a half window (CNTRL-D). If *size* is given, then the scroll is *size* number of lines. This value is remembered for all later scrolling commands.

Scroll – CNTRL-F and CNTRL-B

Syntax: **CNTRL-F**
CNTRL-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 – CNTRL-G

Syntax: **BELL**
CNTRL-G

Function: Prints 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*, then the number of lines displayed is equal to *size*. If a preceding *linenumber* is given, then the given line is placed at the top of the screen. If the last argument is a RETURN, then the current line is placed at the top of the screen. If the last argument is a period (.), then the current line is placed in the middle of the screen. If the last argument is a minus sign (–), then the current line is placed at the bottom of the screen.

Redraw – CNTRL-L

Syntax: **CNTRL-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, just 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 letter "d" 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, then 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 multiline command, then 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, discussed below:

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, then successive **u** commands print out 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, then *text* substitutes for count number of characters or lines depending on whether the command is **s** or **S**, respectively.

Filter – !

Syntax: **!***cursor-movement cmdRETURN*

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, then 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 such as put, yank, or delete, use a quotation mark. For example, to yank a line into the buffer named *a*, type:

"ayy

To put this text back into the file, type:

"ap

If you delete text into the buffer named *A* rather than *a*, then 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, then 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, then text is yanked into the unnamed buffer. If an uppercase *letter* is used, then 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

/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 to *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 below:

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 abort 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 invoked 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 typing the **vi** command from ex.

Ex Commands

Typing 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 invoked 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 a RETURN or an ESC. We shall use a RETURN in our examples. Command entry is terminated by typing an 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" will print the tenth line in the buffer while "move 5" will move 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, such as

```
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 prints 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*, then the trailing slash (*/*) or question mark (?) may be omitted. If *pattern* is omitted or explicitly empty, then 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 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, then 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 printed, 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 — Prints the current filename, whether it has been modified since the last **write** command, whether it is readonly, 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 the form "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 prints 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 printed 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".

CNTRL-^

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*, printing 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, then text will be 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 prints

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 then 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 then 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 issues a warning message if the file has changed since the last **w** command was issued, and does not quit. Vi will also issue 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 give a **w** command. If you wish to discard them, use 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* This variant overrides checking of the **w** command so that you can write to any file that the system allows.

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 the serious user of vi.

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, then each line matching *pattern* is printed.

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 printed on the screen.

g/s1/s//s2/gc This command asks you to confirm that you want to make each substitution of the string "s1" with the string "s2". If you type a "y" then 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, it 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, then all instances on a line are substituted. If the *confirm* indication character "c" appears, then before each substitution the line to be substituted is printed on the screen with the string to be substituted marked with caret (^) characters. By typing a "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, type a CNTRL-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, then a diagnostic is printed before the command is executed as a warning. A single exclamation (!) is printed 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 only in insert mode, while characters mapped with **map** work only in command mode.

nu — Prints each specified line preceded by its buffer line number. The current line is left at the last line printed. 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 don't know how to save your work.

= — Prints 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. Except when you use **preserve** you will be notified by mail when a file is saved.

set *argument* — With no arguments, **set** prints those options whose values have been changed from their defaults;

with the argument *all* it prints all of the option values.

Giving an option name followed by a question mark (?) causes the current value of that option to be printed. The ''?'' is unnecessary unless the option is Boolean valued. 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, then the previous *label* is used.

Similarly, if you type only a CNTRL-], vi searches for the word immediately after the cursor as a tag. This is equivalent to typing '':tag'', this word, and then a RETURN.

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*.

The first thing that must be done before you can use vi, is to set the terminal type so that vi understands 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, you can type CNTRL-D. The tab stops going backward are defined as multiples of the *shiftwidth* option. You cannot backspace over the indent, except by typing a CNTRL-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 CNTRL-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 CNTRL-D repositions the cursor at the beginning but without retaining the previous indent. *Autoindent* doesn't happen in global commands.

autoprint ap default: ap

Causes the current line to be printed 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 of many commands 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 CNTRL-^ (switch files) or CNTRL-] (tag go to) command.

beautify, bf default: nobeautify

Causes all control characters except tab, new line 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 this directory is not writable, then 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 command, instead of like **&.**

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 the system expands tabs.

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 will be displayed unambiguously, showing tabs and end-of-lines.

magic default: magic

If *nomagic* is set, the number of regular expression metacharacters is greatly reduced, with only caret (^) 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 printing 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, which have potentially far reaching scope, the net change in the number of lines in the buffer is presented at the end of the command, subject to this same threshold. 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 CNTRL-D is received from a terminal input in command mode, and the number of lines printed 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 CNTRL-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 *tabstop* boundaries for the purposes of display.

taglength, tl default: tl=0
The first *taglength* 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 thus 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 escaped (i.e., preceded) by a backslash (\) if they are to be treated as ordinary characters.

^ At the beginning of a pattern this forces the match to succeed only at the beginning of a line.

$ At the end of a regular expression this 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 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:

250,000 lines in a file

1024 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

Notes

The */usr/lib/ex3.7preserve* program is 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.

Name

w – display who is on and what they are doing

Synopsis

w [**–u**] [**–k** *kernel*] [*user*...]

Description

W prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the number of jobs, or processes running.

The fields output are: the users login name, the name of the tty the user is on (also shows if messages are disabled by displaying a ''*'' in front of the tty name), the time of day the user logged on, the number of minutes since the user last typed anything, the number of jobs the user is running, and the name of the current process.

The **–u** flag displays only the header line. The **–k** flag allows specifying an alternate kernel (for when the system is not booted on /xenix).

If any *user* names are included, the output will be restricted to those users.

Files

/etc/utmp	Who's logged on
/dev/procs	What they're doing
/dev/kmem	How long the system has been up

See Also

who(C), finger(C), ps(C), qps(C)

Notes

The notion of the ''current process'' is muddy. The current algorithm is ''the last descendant of the highest numbered process on the terminal''. This fails, for example, when the user is running a program in background and is sitting idle at the shell prompt. (In cases where no process can be found, *w* prints a ''–''.)

Background processes are not shown, even though they can account for much of the load on the system.

Name

xmail – Improved mail program.

Syntax

xmail [*options*] [*user ...*]

Description

Xmail is an improved version of the mail program. it lets you use any editor available on the system for creating mail. *Xmail* maintains a local working mailbox for later reference of past messages and new mail.

If you wish to read your mail, type:
xmail [*options*]

The available options are:

-d Overrides the initialization file .xmailrc "maildir=" option

-q Sets quiet mode when you are not sending mail. In quiet mode, *xmail* does not display the messages in your file when it starts up or when you enter a new mailbox. This is the same as the "set -showmsgs" option in .xmailrc.

When used while sending mail, the **-q** option sets "quick mode. When "quick" mode is set, *xmail* does not automatically enter the editor, regardless of how the "editmail" option is set. Instead, *xmail* prompts for "Subject" and other information without invoking the editor.

-R Makes the mailfile a read only file. That is, you cannot get new mail, delete messages, or edit messages. When **-R** is used, *xmail* does not make a copy of the mailbox and therefore loads the mailbox faster. This option is useful if you only wish to look at the mailbox.

-f *mailfile* Opens *mailfile* as the mailbox instead of the default mailbox.

-m Opens a *mailbox* file in the user's mail directory instead of in the current working directory (used with the -f option).

-s *subject* Sets the subject line to *subject* when sending mail.

RC Files

When you invoke xmail, it reads the file "/usr/lib/xmail.rc". *Xmail* gets the initialization options for the system from this file. Next, *xmail* tries to open the .xmailrc in your home directory. From this file, *xmail* gets the options that you set. You can set or change these options temporarily by using the *xmail* "o" command. You can also set them permanently by editing the .xmailrc file.

The options are:

alias *mailname username* [*username...*]
Creates a user alias. This sets *mailname* to expand to one or more user addresses.

autodelete *userid* [*userid...*]
Instructs *xmail* to delete new mail from the specified user(s) when you exit your mailbox.

full_name=*name*

full_name=OFF
Allows you to specify your full name for xmail to use when it is sending network mail. when xmail sends network mail, it adds this line to the message:

From: *system!user* (*Full Name Here*)

Use the fullname option to change the name shown or to specify OFF if you don't want your full name used.

ignore userid [userid ...]
Instructs xmail to automatically delete any mail coming from a user whose login name is *userid*. To display the current ignore list, type the following at the mail> prompt:

o show ignores

mailbox=*name*
Changes the name of the default working mailbox from *.mbox* to *name*. *Xmail* opens name in your home directory.

maildir=*directory*
Changes the current working directory when you run *xmail* for the duration of the mail session. *Xmail -d* will override this option.

record_file=*filename*
record_file=OFF
Tells xmail to save a copy of all mail that you send out in *filename*. This option acts independently of autocc. This option differs from the autocc option in that instead of sending the message through the system mailer, it is copied into the specified file.

savembox=*name*
Change the name of the default save mailbox from mbox to *name*.

set [-]*option*
Sets specific parameters for *xmail*. Possible options are given next.

Note: Specify the minus sign (–) to turn the option off.

autocc
Tells xmail to automatically send you a carbon copy (Cc:) of all mail that you send.

autoprint
Displays the next message automatically when you delete or save the current message.

banner
Uses a banner saying *"user's* Mail" when printing mail.

clear_hdr
Clear the screen when displaying message headers with the header command (h), or when displaying option settings.

confirm
Sets *xmail* to require a RETURN on commands that do not accept parameters such as the new mail command (n). When confirm is set On, *xmail* only accepts RETURN and the KILL and ERASE characters when you select such a command.

deletesave
Automatically deletes messages once you have saved them with the save command.

editmail
Automatically invokes the editor when sending mail. If you turn off this option, *xmail* prompts you for header information and then for the message itself (with a > prompt for each line).

getnewmail
Automatically gets new mail when you start *xmail* in your default working mailbox.

ignore_cr
Tells *xmail* to ignore the RETURN key when prompting for a command. If this option is off, RETURN acts the same as the = command: it reads the next message.

includetext
Includes the text of the message to which you are replying in your message. This option only works when the editmail option is turned ON, as otherwise you are not allowed to edit the message before you add your own text.

indent=string
Indents forwarded mail and the included text of reply mail with **string**. The default is no indention.

mesg
Lets you allow or disallow messages from other users during the mail session.

prompt_nl
Controls the newline in front of xmail's prompts. Some find extraneous newlines annoying.

showdelete
Display deleted messages when showing message headers.

showlines
Tells *xmail* to display the number of lines and bytes when displaying message headers. *Xmail* uses the format *lines/bytes* when displaying the size.

showmsgs
Displays messages currently in your mailbox when you start up *xmail*, or enter a new mailbox.

strict
Does not allow lines in the header block which *xmail* does not recognize. If this option is off, you can place anything in the header block. To edit the header block, you must also invoke the editmail option.

verbose
Displays messages indicating *xmail's* progress while loading a mailbox, updating a mailbox, or performing other functions.

setenv VAR *value* [*value* ...]
Sets the environment variable VAR to the value(s) given.

signfile=*signaturefile*
signfile=OFF
Tells *xmail* to add the text found in signaturefile to your message template before you reach the editor, or where to find your signature when using *sendmail's* put signature command (p). If you specify OFF, *xmail* does not include a signature file.

sysalias *name user* [*user* ...]
Creates a system alias, which is invisible to the user (not shown by the A command). This is normally

used by the system administrator to create a system-wide alias in the /usr/lib/xmail.rc file. To display the system aliases, type at the mail> prompt:

o show sysaliases

umask *value*
Sets the umask while running xmail to the requested value while running *xmail*.

unalias *name*
Removes *name* from *xmail's* alias list.

Defaults For *Xmail* Option Settings:
maildir=.
mailbox=~/.mbox
savembox=~/mbox
record_file=OFF
signfile=OFF
set prompt_nl showmsgs -getnewmail -autodelete
set mesg -showlines
set -ignore_cr -autocc verbose -showdelete banner
set -autoprint -confirm strict -includetext
set -showlines

Xmail Commands

When you startup *xmail*, it tells you which version number, and how many messages are in the mailbox you have selected or in your default mailbox if you didn't select one. *Xmail* also shows whether or not you have received new mail.

Xmail then displays its prompt. There is a set of commands which you can enter at this prompt.

NOTE
The term *msg#-list* refers to a set of messages which you can specify at this prompt. You may specify them in any of the following forms:

number	single message
number number	list of messages
number-number	range of messages
.	for current message
$	for last message
*	all messages in file

For example: d 1 2 4-7 .-11 14-$

The *xmail* commands are:

a *name list* Adds a user alias to *xmail's* list.

A Display *xmail's* current alias list.

c Checks system mailbox for new mail.

d *msg#-list* Delete message(s) from the mailbox.

D Delete the current message from the mailbox. This is the same as d ENTER.

e ***msg#-list***	Lets you edit the specified message.
E	Lets you edit the current message. This is the same as e ENTER.
f ***msg#-list user***	Forward a message to *user*.
F [-p] ***str [file]***	Find a string in the specified file. *Xmail* looks for files in the current directory. Specify the **-p** option if you want the output duplicated by the printer. You can specify a list of files by separating each with a space. Specify "." to indicate the current working mailbox (default).
g [-R] ***mailfile***	Goes to the specified *mailfile* after updating the current mailbox. Specify the **-R** option if you want read only access to mailfile.
G [-R] ***mailfile***	Goes to the specified *mailfile* but does not update the current mailbox. Specify the **-R** option if you want read only access to *mailfile*.
h ***msg#-list***	Shows a header for each message specified. If you specify only one number, *xmail* displays a screenful of headers starting at that message number. If you omit *msg#-list*, *xmail* displays the next screenful of messages.
H	Shows a header for the last screenful of messages
l	Displays the contents of the current directory.
m [-s ***'subj']*** ***user***	Sends a message to the specified user(s) by first invoking the editor. If you specify **-s** with a subject enclosed in quotes, *xmail* will insert the subject into the header block.
n	Gets new mail from the system mailbox.
o ***option***	Lets you enter a .xmailrc option.
O	Displays the current .xmailrc option settings.
p ***msg#-list***	Prints the specified message(s) on the system printer.
P	Prints the current message on the system printer. This is the same as **p** ENTER.
q	Exits *xmail* and updates the mailbox.
Q	Exits *xmail* but does not update the mailbox. Any new mail is still saved.
r ***msg#-list***	Sends a reply to specified message(s).
R	Sends a reply to the current message. This is the same as **r** ENTER.
s ***msg#-list file***	Saves the specified message(s) to *file*. If you omit *file*, *xmail* saves the file to the default save file (usually ~/mbox).
S	Saves the current message(s) to *file* (usually ~/mbox).
t ***msg#-list***	Types (displays) the specified message(s). You can omit **"t"** and simply type the message number.
T	Types (displays) the current message. This is the same as **t** ENTER.
u ***msg#-list***	Undelete message(s) in the mailbox.

v	Display the version number of *xmail*.
–	Displays the previous message.
+ or =	Displays the next message.
ENTER	Displays the next message if the ignore_cr option is turned off.
!	Starts up a shell.
!*command*	Runs a shell command and returns to *xmail*.
.	Displays a header line for the current message and the current mailbox name.
?	Display xmail's help screen.

Sending Mail

After exiting the editor when sending mail, xmail displays a sendmail> prompt. At this prompt, you can enter one of the following commands:

a *file*	Add the contents of file to the message.
c	Continue adding text to the message.
d	Display the message.
e	Re-edit the message.
p	Add the signature file to the message.
q	Quit.
s	Send the message.
?	Display help screen.

The following commands are only available from "sendmail>" if you have entered *xmail* and have read in a mailbox:

h	Display message headers.
r *msg#-list*	Read the specified messages into text.
.	Display the current message header.

The */etc/usemap* File

When sending mail, *xmail* looks at the address. If it is a uucp network address in the form *system!user*, *xmail* attempts to open the file /etc/usemap. If the file exists, *xmail* tries to find a uucp path for *system*. If the path exists, *xmail* substitutes that path for *system*. If there is not an entry for *system*, *xmail* displays a warning message. If /etc/usemap does not exist, *xmail* leaves the address alone.

Xmail expects /etc/usemap to contain a sorted list of system names followed by at least one space, then the path to that system with a %s for the user name. For example:

```
flakey    doc!flakey!%s
mcp       sneezy!eagle!mcp!%s
trsvax    trsvax!%s
```

Special Features Available While Sending Mail

Xmail has several special features available when sending mail. You may send a copy of your message to a shell command or save a copy of the message from the "To:", "Cc:" or "Bcc:" lines. You may also specify a mailing list file from there. Here is an example of how you use the features:

% xmail everybody
One moment please...

Xmail enters the editor. The following is a sample screen:

To: everybody *others
Subject:
Return-receipt-to: me
Cc: *yetothers |/pr | lpr/ +everybody
Bcc: theboss

Notice the "Cc:" line. The *yetothers tells *xmail* to open the file "yetothers" and send the message to every person in the file. The |/ pr | lpr/ tells *xmail* to write a copy of the message to the command "pr | lpr". (You can use any delimiter you wish, as the character after the "|" is taken as a delimiter). The +everybody appends a copy of the message to the file "everybody." Here is a list of "Cc:" line options:

* Reads a file and sends mail to the list of users contained in the file.

\+ Appends the message to the specified file.

| Writes a copy of the message to a command.

The "Bcc:" line (blind-carbon-copy) sends a copy of the messages to anyone listed, but the "Bcc:" line itself is not included in the message.

The "Return-receipt-to:" line is an instruction to the receiving mailer to send mail back to the sender acknowledging the receipt of the message. In this case, an acknowledgement will be sent to the user "me".

Also available is the "Retum-receipt-from:" line. This lets you specify from whom you wish to receive a return receipt. This is useful when you are sending mail to a group of people but only want a return receipt from one or just a few.

"Return-receipt-to:" and "Return-receipt-from:" are mutually exclusive.

Notes

The /usr/lib/xmail_recover program is used to restore new mail that had not been saved as a result of a system crash. This is called by /etc/rc when the system is booted.

Files

/usr/lib/xmail	mail program	/usr/spool/mail/*	post office
/usr/lib/xmail.cmd.hlp	commands help text	~/.mbox	old mail
/usr/lib/xmail.opt.hlp	options help text	/usr/lib/xmail.rc	system-wide xmail options
/usr/lib/xmail.set.hlp	"set" help text	~/.xmailrc	user's xmail options
/usr/lib/xmail_recover	*xmail* recovery program	/tmp/xm.#	xmail temporary files
/usr/lib/xmlock	*xmail* system mailbox locking program		

Contents

Miscellaneous **(M)**

Name

acu – Modem Auto-dialer interface.

Description

The Automatic Call Unit interface is provided by the files */dev/cua[01]* which are device entries, and by their corresponding driver which simulates a standard DN dialer. To place an outgoing call one forks a sub-process trying to open */dev/cul0* and then opens the corresponding file */dev/cua0* file and writes a number on it. The driver translates the call to proper format for the Automatic Dial Module.

The codes for the phone numbers are the same as in the DN interface:

0-9 dial 0-9
– delay 4 seconds
< end-of-number

The entire telephone number must be presented in a single *write* system call. The phone number must have an end-of-number code.

It is also required that the provided line devices (/dev/cul[01]) be used and not the tty devices (/dev/tty[12]) when dialing out (as with the cu or uucp commands).

The minor number of the line device (eg, /dev/cul?) is the minor number of the serial line that the modem with automatic call capabilities is connected to. The minor number for the dial device (eg, /dev/cua?) is a little different. It has the high bit ON to indicate an ACU device (0x80). The lower 5 bits of its minor number correspond with the minor number of the corresponding serial device. Bits 6 and 7 are set to indicate what speed you would like to dial at. When set to 0 (00, both bits OFF), this indicates that the kernel should figure out what speed the modem is set at and dial at that speed. This is mostly useful for modems which have an external speed switch. If they are set to 1 (01), this indicates that the kernel should dial out at 300 baud unconditionally. When they are 2 (10), the kernel dials out at 1200 baud unconditionally. This way, you can select a speed on modems which auto-baud.

For example, a modem on /dev/tty01 (serial channel A) would have a dialer minor number of 129 (= 1 | 0x80) and a line minor number of 1.

Files

/dev/cua0	virtual dialer #0 (uses tty01)
/dev/cua1	virtual dialer #1 (uses tty02)
/dev/cul0	the line which is connected to dialer 0
/dev/cul1	the line which is connected to dialer 1

See Also

cu(C), uucp(C)

Notes

Currently, only the following units are supported:

Radio Shack Modem II
DC-2212
DC-1200 (with an Automatic Call Unit installed)
DCM-7

Name

cfg – Configures /xenix

Syntax

cfg [–n xenix] [*Tables*] [*Misc*]

Tables may be '–d' or one or more of the following:

buffers=*n* inodes=*n* files=*n* locks=*n* procs=*n* clists=*n*

n above may be 'a' or an integer greater than 0.

Misc may be one or more of the following:

[maxprocs=*p*] [maxmem=*m*] [nodename=*name*]
[swapdev=a l swapdev=*mm,nn* swplo=*dd* nswap=*nn*]

Description

Cfg allows you to configure the size of various system tables and parameters. By default, the size of these tables is determined by how much memory is installed in your machine:

Mem size	Buffers	Inodes	Files	Locks	Procs	Clists
512k	25k	100	100	200	60	200
1.0 meg	50k	100	100	200	60	200
1.5 meg	100k	120	120	240	70	200
2.0 meg	150k	140	140	280	80	200
2.5 meg	150k	160	160	320	90	200
3.0 meg	150k	200	200	400	100	200

Note: It is not currently possible to install more than 1 megabyte of memory.

Cfg allows you to select the size of any of these tables. For example:

cfg buffers=200

sets the number of system buffers to 200. *Cfg* will respond:

/xenix is now configured as follows:

Buffers	200
Inodes	AutoConfig
Files	AutoConfig
Locks	AutoConfig
Procs	AutoConfig
Clists	AutoConfig
Maxprocs	15
Maxmem	256k
Nodename	""
Swapdev	AutoConfig

This indicates that /xenix will reserve 200 buffers when it boots next. Because each system buffer has 512 bytes, the kernel will reserve 100k of memory for system buffer use. To set /xenix back to automatic buffering, type:

cfg buffers=a

Cfg will respond:

/xenix is now configured as follows:

Buffers	AutoConfig
Inodes	AutoConfig
Files	AutoConfig
Locks	AutoConfig
Procs	AutoConfig
Clists	AutoConfig
Maxprocs	15
Maxmem	256k
Nodename	""
Swapdev	AutoConfig

This indicates that /xenix will automatically select the number of buffers based on memory size.

The 'nodename' option allows you to change the name of your system that is returned by the 'uname' system call [see *uname*(S)].

The '-d' option resets all options except 'nodename' and 'swapdev' to their default values, which are shown in the example above.

As with the configuration disk in earlier versions of 68000/XENIX, it is assumed that the system administrator has the technical understanding of the relationships between the various parameters and their effects on system performance. Some incorrect configurations can destroy data.

Cfg may only be run by the superuser.

Diagnostics

cfg: You must be superuser to run this program
The user was not logged in as root, nor was the *su* command used to change to root.

cfg: No namelist
The program was unable to find the required symbols in /xenix.

cfg: cfg not supported in that xenix
The user tried to run *cfg* on an older version of xenix.

cfg: Can't open xenix
The program was unable to open the file /xenix.

cfg: Can't read xenix
The program was unable to read the file /xenix.

cfg: Can't write xenix
The program was unable to write the new configuration.

cfg: Not enough data in /xenix
The file /xenix did not match its namelist. This may indicate a corrupted /xenix file.

cfg: Can't seek for read
The program can't position to read the old configuration.

cfg: Can't seek for write
The program can't reposition to write the new configuration.

cfg: Please type a number of *items* from *min* to *max*
The user specified a number of *items* less than *min* or greater than *max*.

cfg: Please type a number for *item* from *min* to *max*
The user specified a number for *item* less than *min* or greater than *max*.

cfg: Missing major number for 'swapdev'
cfg: Missing minor number for 'swapdev'
The user gave an incorrect 'swapdev' specification.

cfg: You must specify swapdev, swplo, and nswap together
The user gave one or more of swapdev, swplo, and nswap without specifying all of them.

cfg: Too many characters in nodename
The user specified a nodename which was greater than 8 characters long.

cfg: Illegal character in nodename
The user specified a nodename which had a character other than a-z, 0-9, '-', and '_' in it.

cfg: Unknown option
The user gave *cfg* an unknown option.

cfg: Missing argument for –n
The user gave the '–n' option without another argument.

cfg: Too many *item* change specifications
The user specified more than one of 'a' or a number of *item*.

cfg: You must specify "–d" alone.
The user gave the '–d' option with another change request.

See Also

configuration(M)

Name

console – The system console terminal interface

Description

The standard Tandy 6000 supports three terminals. One of these terminals is the system console. The system console communicates with the processor through a special interface, instead of a serial interface. Because the system console does not use a serial interface, *console* ignores requests to change line speeds.

The console has a "screen-saver" feature which helps preserve the monitor in the Tandy 6000. The console automatically darkens the console display after 20 minutes of inactivity. This feature is similar to the one found in Tandy DT-100 terminals. The display reappears when you press a key, or when XENIX sends new data to the screen. The SHIFT CTRL CAPS LOCK and REPEAT do not turn the screen back on when pressed by themselves.

Two keystrokes control the screen-saver feature. Pressing CTRL = turns the screen back on if it was off. This keystroke combination is only used by XENIX to turn the console screen back on. Pressing the CTRL = combination will not send data to any programs which read the console keyboard. Pressing CTRL . toggles the screen-saver feature on and off. When screen-saver is enabled with CTRL . the screen goes dark immediately as a confirmation, instead of waiting 20 minutes.

Following is a list of the video and keyboard control codes:

Video control codes

Value(s)	ID	Effect(s)
00h-06h	Ignored	Undefined – No effect
07h	Bell	Sounds the console beeper on systems that have a beeper.
08h	BS	Backspaces without wrapping backward
09h	HT	Performs a Modulo 8 horizontal tab
0Ah	LF	Performs a line feed without a carriage return
0Bh	Ignored	Undefined – No effect
0Ch	FF	Clears the screen and homes the cursor
0Dh	CR	Performs a carriage return without a line feed
0Eh–1Ah	Ignored	Undefined – No effect
1Bh	ESC	Starts an ESCape sequence (defined as follows):
	ESC A	Cursor up
	ESC B	Cursor down
	ESC C	Cursor right

Video control codes (continued)

Value(s)	ID	Effect(s)
	ESC D	Cursor left
	ESC E	Clears the screen and homes the cursor
	ESC F	Undefined – No effect
	ESC G	Undefined – No effect
	ESC H	Homes the cursor
	ESC I	Undefined – No effect
	ESC J	Clears to the end of the screen
	ESC K	Clears to the end of the line
	ESC L	Inserts a line
	ESC M	Deletes a line
	ESC N	Undefined – No effect
	ESC O	Undefined – No effect
	ESC P	Inserts a character
	ESC Q	Deletes a character
	ESC R	Starts an ESC R sequence
	ESC R @	Cancels reverse video
	ESC R D	Starts reverse video
	ESC R C	Turns the cursor ON
	ESC R c	Turns the cursor OFF
	ESC R G	Starts low resolution graphics mode (codes 40h–5Fh are displayed as graphics characters 00h–1Fh)
	ESC R g	Cancels low resolution graphics mode
	ESC S	Undefined – No effect
	ESC T	Undefined – No effect
	ESC U	Undefined – No effect
	ESC V	Undefined – No effect

Video control codes (continued)

Value(s)	ID	Effect(s)
	ESC W	Undefined – No effect
	ESC X	Undefined – No effect
	ESC Y <Row+20h><Col+20h>	Starts a cursor positioning sequence
	ESC Z	Undefined – No effect
	ESC [	Starts an ESC [sequence
	ESC [? 3 3 h	Turns the cursor blink ON
	ESC [? 3 3 l	Turns the cursor blink OFF
	ESC [_ <SPACE> q	Changes the cursor to an underline
	ESC [<DEL> <SPACE> q	Changes the cursor to a block
	ESC ^	Enables the screen-saver feature
	ESC _	Disables the screen-saver feature
	ESC ` <minutes+20h>	Changes the screen-saver timing
1Ch–1Fh	Ignored	Undefined – No effect
20h–7Fh	ASCII	Prints a standard ASCII character on the console display

Note: Values enclosed within "<>" represent single characters.

Keyboard codes

Key	Code Sequence Produced	
HOLD	13h/11h	Sends alternating DC3 <^S> and DC1 <^Q> codes
BREAK	03h	ETX ^C
↑	1Bh 41h	ESC A
↓	1Bh 42h	ESC B
→	1Bh 43h	ESC C
←	1Bh 44h	ESC D
CTRL BACK SPACE	7Fh	DEL ^?
CTRL 0	7Ch	< | >

Keyboard codes (continued)

Key	Code Sequence Produced	
CTRL 1	7Ch	|
CTRL 2	00h	NULL
CTRL 3	1Dh	GS ^]
CTRL 4	1Eh	RS ^^
CTRL 5	1F	US ^_
CTRL 6	7E	~
CTRL 7	1Ch	FS ^\
CTRL 9	5Ch	\
CTRL /	5Ch	\
CTRL '	60h	`
CTRL =	None	Turns on the console display if it was turned off by the screen-saver feature
CTRL .	None	Toggles the screen-saver feature ON and OFF

Files

/dev/console

See Also

graphics(M), screen(M)

Name

getty – Sets terminal mode.

Syntax

/etc/getty char [login]

Description

Getty automatically adapts a terminal's serial line to allow proper communication between the terminal and the system. It is one of three programs (*init*(M), *getty*(M), and *login*(C)) used by the system to enable a terminal and allow user logins.

Getty is initially called by *init* which passes a single character argument *char* (init reads the argument from the **ttys** file). It then writes a "login:" message, indicating the user may log in on the machine.

The optional *login* argument is a login name to use. If this argument is present, the terminal line is set to the first speed specified by *char*, and call login with that name. *Getty* does not prompt the user. The *login* argument is normally specified in the inittab file (see *init*(M)), and the login name usually does not have a password. See *init*(M) for more details. If the user types a name and terminates it with a newline (ASCII LF) or carriage return (ASCII CR), *getty* scans the name for uppercase alphabetic characters. If only uppercase characters are found, *getty* adapts the system to map all subsequent lowercase characters into the corresponding uppercase characters. Furthermore, if the name terminates with a carriage return character, *getty* sets the terminal's serial line mode to CRMOD (see *ioctl*(S)).

If, instead, the user presses the BREAK key, *getty* writes the login message again. It also changes the serial line speed if *char* is one of those which cause "cycling" as described below. This allows the system to adapt to terminals with various line speeds.

After a name has been typed and scanned, *getty* passes it to *login*(C) which asks for the user's password and completes the login process.

The *char* argument may be any one of the following:

– Intended for an on-line Teletype model 33, for example, an operator's console.

0 Cycles through 300-1200-150-110 baud. Useful as a default for dialup lines accessed by a variety of terminals.

1 Optimized for a 150-baud Teletype model 37.

2 Intended for an on-line 9600 baud terminal that requires delays, for example, the Textronix 4104.

3 Starts at 1200 baud, cycles to 300 and back. Useful with 212 datasets where most terminals run at 1200 speed.

4 Useful for an on-line console DECwriter (LA36).

5 Same as 3 above, but starts at 300.

6 Intended for machine-to-machine (such as over a network) logins at 2400 baud.

7 Useful for an on-line 4800 baud terminal.

8 Useful for an on-line 1200 baud terminal.

9 Intended for an on-line 9600 baud terminal that does not require delays.

The following types are intended for general-purpose on-line terminals (unlike the specialized settings above), and differ only in the baud rate:

a 50 baud

b 75 baud

c 110 baud

d 134.5 baud, usually with 2 stop bits.

e 150 baud

f 200 baud

g 300 baud

h 600 baud

i 1200 baud

j 1800 baud

k 2400 baud

l 4800 baud

m 9600 baud

P Autosequencing 300–1200–2400 baud dial-up line.

Q Autosequencing 1200–2400–300 baud dial-up line.

R Autosequencing 2400–300–1200 baud dial-up line.

Name

graphics – The high-resolution graphics card interface.

Syntax

```
# include <sys/graphics.h>
# include <sys/ioctl.h>
```

Description

Graphics is a seekable device interface for the Tandy 6000 high-resolution graphics board. The interface works like a 19200 byte long text file. Each byte corresponds to eight pixels on a scan line. The graphics board scans the pixels left to right, starting with the most significant bit in each byte. The graphics board provides 240 display lines with 640 pixels per line. Because each byte stores 8 pixels, each line contains 80 bytes.

These *ioctl* commands are available with the graphics interface:

TIOCEXCL Sets exclusive use on the device. This is the default mode when the device is first opened. No other users can open the device until it is closed, or until the current user invokes the TIOCNXCL *ioctl* command.

TIOCNXCL Releases exclusive use of the device. This permits other users to open and use the device while you have it open.

GIOCGETC Gets a copy of the graphics control word. The graphics control word controls various aspects of the operation of the graphics board. The control bits of this word are described below.

GIOCSETC Sets the graphics control word. This ioctl command uses a pointer to the new control word data.

GIOCCBIC Clears bits in the graphics control word. This ioctl command uses a pointer to the control word mask.

GIOCCBIS Sets bits in the graphics control word. This ioctl command uses a pointer to the control word mask.

You can set the following bits in the graphics control word:

GR_ENABLE Enables the graphics board. This makes the contents of the graphics memory appear on the console display. This is a default setting.

GR_WAITS Enables video memory wait periods. This reduces hashing when updating video memory. This is a default.

GR_INC_X Selects 'X' register increment mode when GR_X_RDCLK or GR_X_WRCLK is set. This is a default setting.

GR_DEC_X Selects 'X' register decrement mode when GR_X_RDCLK or GR_X_WRCLK is set.

GR_INC_Y Selects 'Y' register increment mode when GR_Y_RDCLK or GR_Y_WRCLK is set. This is a default setting.

GR_DEC_Y Selects 'Y' register decrement mode when GR_Y_RDCLK or GR_Y_WRCLK is set.

GR_X_RDCLK
Selects increment or decrement mode of the 'X' register after each read operation from graphics memory. This is a default setting.

GR_X_WRCLK Selects increment or decrement mode of the 'X' register after each write operation to graphics memory. This is a default setting.

GR_Y_RDCLK Selects increment or decrement mode of the 'Y' register after each read operation from graphics memory. This is a default setting.

GR_Y_WRCLK Selects increment or decrement mode of the 'Y' register after each write operation to graphics memory. This is a default setting.

GR_AUTOCLK Selects increment mode of the 'Y' register. The 'Y' register is incremented and the 'X' register is set to zero whenever the contents of the 'X' register reaches 80 in increment mode. This is a default setting.

GR_W_ASYNC Selects asynchronous write mode. The write(S) system call will return immediately instead of awaiting output completion. This enables an application to continue running as long as the write(S) system call buffer remains intact until completion of the next I/O operation. Your applications should use dual buffering when running in asynchronous mode. If you do not use dual buffering, the display output will be unpredictable.

Files

/dev/graphics

See Also

console(M), screen(M)

Name

init – Process control initialization.

Syntax

/etc/init

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 asks for a password to start the system in maintenance mode. The user may type the password or terminate the shell by typing ASCII end-of-file (CNTRL-D) at the console. If the shell terminates, *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 *init* 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.

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 (CNTRL-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 naive user. Instead, similar functions can be safely performed with the *enable*(C), *disable*(C), and *shutdown*(C) commands.

Init attempts to prevent thrashing caused by problems opening the line or having prompts echoed back to the computer. *Init* inserts a 20 second delay if *getty* terminates within 30 seconds after it was started, and repeats 40 times without ever running more than 2 minutes. If this occurs another 40 times, *init* will disable the line.

If *getty* runs more than 2 minutes, *init* assumes that the line is working properly. This can occur from one of two causes:

A user logged in on the line.

Getty
is waiting for a user to log in, and is not receiving a lot of characters.

Getty resets the timer count after it determines that the line is working properly.

If *init* disables a line, it sends a mail message similar to the one below to "root". The message indicates the time that the line was disabled and identifies which line was disabled.

From root Thu Jul 10 13:15:01 1986
Subject: Init: Thrashing problem on line tty04.
This line will be disabled.

Init will never automatically disable the console. However, if you repeatedly type CONTROL-D at the console, *init* will insert a 20 second delay. Leave the keyboard alone for 2 minutes and the delay-insertion will stop.

If you enable a thrashing line, it will remain enabled for at least 20 minutes. It may take longer than 20 minutes to be disabled. Because *init* inserts 20 second pauses on a thrashing line, the thrashing line will not load the system very much.

Init also inserts a 10 second delay after it fails to open a tty device.

The "ps -ef" command will show you which terminals are active. You will see a variety of processes in the "COMMAND" column. These processes may include terminals which are not logged in. Here is a list of sample messages and descriptions which may occur due to unlogged terminals:

init O 04	Line tty04 is enabled, but has no carrier detect signal. This is normal, and typical of a dial-in line that is not in use. If this occurs on a direct-wired terminal that is turned on and should be working but is not, the cable is probably incorrectly connected.
init N 04	Line tty04 is enabled, and is executing a 1 second pause before starting *getty*. This is normal, but does not occur frequently. The line is apparently not thrashing.
– 3 or – 9	These processes indicate that *getty* is running on a line. The terminal line can be found under the "TTY" column on the "ps -f" display. The number after the "–" is the speed code from the /etc/ttys file. The number may be either a number or letter. This is normal, and is typical of a direct-wired terminal that is not logged in.
init T 04	Line tty04 is enabled, and is pausing for 20 seconds before starting *getty* because the line appears to be thrashing. This indicates trouble with the line or with what is connected to the line.
init F 04	Line tty04 is enabled, but the system was unable to open line tty04. The system is waiting 10 seconds before trying again. This indicates trouble with the line which may include: No hardware for the line Enabling the line while an outgoing *uucp* or *cu* call was in progress on the same line The driver is hung This display does not indicate that something is echoing characters back to the computer.
/etc/init	Process 1 (see the "PID" column of the "ps -ef" display) should always show this. Other processes may display this briefly before changing to one of the other forms.

Init provides the ability to use a program other than /etc/getty for individual communication lines. This is desirable when you do not want the system to give users a "login:" prompt. It is also desirable when you do

not want to make the standard login available on a dial-up line.

You can specify an alternate *getty* program in the /etc/inittab file. Include one text line for each terminal that will use the alternate program. Each line contains four colon separated fields. The first field contains the character "0". The second field contains the terminal identifier. The terminal identifier is "co" for the console, or a two digit terminal number for other lines. The third field is empty. The fourth field contains the name of the alternate *getty* program for that terminal, and a optional space and argument. The program name and the optional argument must each be less than 32 characters long.

The standard /etc/getty may be invoked with a second argument consisting of a login name. *Getty* will automatically set up the line baud rate and other parameters. Then, *getty* invokes the appropriate login program without issuing any prompts. Normally, this login will not have a password, and a special program will be specified in the password file shell field. This enables the user to interact only with the specified program.

WARNING: Do not try to prevent the user from gaining access to a standard shell by entering something in .profile and then specifying /bin/sh as the shell. This method is not secure. The user will be able to access a standard shell.

Since /etc/getty does not issue any prompts or accept any input, it can not perform automatic baud rate switching. It will use the first baud rate code specified in the /etc/ttys file. Here is an example of a /etc/inittab file:

0:01::/etc/getty bbs
0:02::/etc/getty
0:05::/etc/mygetty xyzfoo

This will run /etc/getty on line 1, but will automatically log in as "bbs". The password file entry might look like this:

bbs::204:20:Bulletin Board System:/usr/bbs:/usr/bbs/bbsprog

Init runs /usr/bbs/bbsprog when a call comes in on modem line tty01. The program might be a simple shell script that gives the latest news. The program could also be a complicated message system that asks the user for identification and provides different levels of access for different users.

The second line in the example runs a standard *getty* on line tty02. This is the same thing that is done when /etc/inittab does not contain an entry for tty02.

The third line in the example runs the custom program /etc/mygetty on line tty05. The program is passed the line speed character for line tty05 as the first argument, and "xyzfoo" as the second argument. The program /etc/mygetty must be able to interpret "xyzfoo" and perform the appropriate actions.

When a program is invoked in place of *getty*, the first argument is the speed code. The speed code is typically 9 for hardwired lines, and 3 for modems. The second argument is the optional argument specified in /etc/inittab. If the optional argument is not specified, the second argument is a zero length string. File descriptors 0, 1, and 2 are open to the line, but the baud rate and other terminal parameters are completely uninitialized. Your *getty* program must be able to set up these parameters before using the line. If you want to include automatic baud rate switching, you must do it yourself.

The program will be running as root. You will probably want to set the current user and group IDs to something else. One way to do this is to invoke *login* with the name of a user without a password. Next, invoke a shell consisting of the applications program that you want to run. See the warning of the previous page about system security.

If you do not invoke *login*, you will probably want to make your program do the following:

Make an entry in /etc/utmp and /usr/adm/wtmp so that the *who* command can list the user. To do this, put an entry in the *n*th slot in /etc/utmp where *n* =

ttyslot(). Then add the record to the end of the /usr/adm/wtmp file. Your program may put whatever you like in the "user name" field. *Init* will make the logout records for you.

Set the user and group IDs to something other than root.

Set the HZ, TZ, TERM, HOME, and PATH enviromnent variables. HZ and TZ are typically in /etc/default/login. TERM is typically in /etc/ttytype, although your program may instead ask the user for TERM.

Invoke an application program.

You will probably want to make your application program do some of the following things:

Ignore or trap the interrupt and quit characters. If your program does not do this, an interrupt or quit character will cause a logout.

Trap, but do **not** ignore the hangup signal.

When your *getty* program (or alternate *getty* program) terminates, it will restart. *Init* measures the time from logout to logout. *Init* will disable the line if it repeatedly terminates within 30 seconds. If your program is running on a dial-in line, it will not be invoked until the modem detects a carrier. Programs that only output a short message and do not accept input from the user will execute repeatedly on hardwired lines. *Init* will consider the program thrashing and will disable the line.

Files

/dev/tty*

/etc/utmp

/usr/adm/wtmp

/etc/ttys

/etc/rc

/etc/inittab

See Also

disable(C), enable(C), login(M), kill(C), sh(C), shutdown(C), ttys(M), getty(M)

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.

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.*

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. Anomalous results and missing data files are a common result. *Abnormal.*

interrupt from unknown device, vec=*d*
panic: unknown interrupt
: This message indicates that an unknown, unused interrupt (number d) occured on the 68000. This usually indicates a 68000, or, more rarely, a Z80 hardware problem. *Hardware fatal.*

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. *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 **
...
[Z80 Control System Halted]
: 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; nonsticky pure text programs require an entry only when there is at least one copy being executed. *Abnormal.*

panic: blkdev
: An internal disk I/O request, already verified as valid, is discovered to be referring to a nonexistent disk. *System inconsistency, fatal.*

panic: devtab
: An internal disk I/O request, already verified as valid, is discovered to be referring 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 parity
A hardware memory failure trap has been taken. *Hardware or system inconsistency, fatal.*

panic: mmusub: chk
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 is 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 yet 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: 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.

Supervisor trap *d*
panic: trap in sys
The 68000 CPU has generated a TRAP number *d* while executing kernel or device driver code. This message is preceded with an information dump describing the trap. *Hardware or system inconsistency, fatal.*

panic: proc on q
The system attempts to queue a process already on the ready-to-run process queue. *System inconsistency, fatal.*

Warning: bad configuration, resetting to defaults.
The parameters specified with the "cfg" command require an unreasonable amount of memory be used for XENIX, and would leave the available user memory area too small to be usable. The system ignores table size parameters and uses the defaults. This message only occurs at boot time. This message may be caused by running a system with less memory than it ordinarily has. Abnormal.

Device Driver Messages

{Hard, Floppy, Cartridge} Drive n: <specific message>
These messages indicate a specific error, and include the following:

bad format
This means the user tried to read or write an unformatted or foreign disk.

Further I/O aborted until device closed
Device closed; error cleared

This means an error occurred, and the driver refused to do any more I/O until the program closed the device.

active disk changed

This message means that the user changed a floppy while XENIX was doing I/O on it.

drive not ready
active drive not ready

This means that the user tried to use a drive, and it was not ready. This can happen, for example, if the user tries to mount a floppy drive which has no disk in the drive.

drive write protected
active drive write protected

The user tried to write to a drive which was write-protected.

{Hard, Floppy, Cartridge} Drive N: {hard, soft} error HHHH while
{reading, writing, formatting} PLACE

An I/O error of some sort occurred. "hard" means the data was lost and "soft" means the data was written or read correctly.

PLACE will indicate where the error occured on the media and will include 'Cylinder', 'Head' or 'Side', and 'Sector'; or 'Block' numbers.

'HHHH' will be four digits of status information from the hardware that may be useful to technicians in case of hardware failure.

Name

screen – The console screen interface.

Description

The screen device is a seekable interface to the console video memory. The interface works like a 1920 byte long text file. Each byte corresponds to a character on the console screen. The console screen provides 24 lines, each with 80 columns.

There are two *ioctl* commands supported by the screen interface:

TIOCEXCL Sets exclusive use on the device. This is the default mode when the device is first opened. No other users can open the device until it is closed, or until the current user invokes the TIOCNXCL *ioctl* command.

TIOCNXCL Releases exclusive use of the device. This permits other users to open and use it while you have it open.

Files

/dev/screen

See Also

console(M), graphics(M)

Index

Miscellaneous (M)