SEQUENT

DYNIX/ptx® Command Quick Reference

DYNIX/ptx® Command Quick Reference

1003-51133-00
Sequent Computer Systems, Inc.

About this Document

Organization

This document is organized into the following sections:

— Command Summary

— Alphabetical Command Reference

— **vi** Editor Commands

— Shell Reference

Audience

This document is intended for users who are new to the DYNIX/ptx operating system. It covers the more common user commands and the more common options for those commands. More information about each command may be found in the *DYNIX/ptx User's Guide*, Section 1 of the *DYNIX/ptx Reference Manual*, and the online man pages.

Notation

The following notational conventions are used in this document:

— The names of commands are shown in **large bold** or **bold** font.

— The names of files, directories, and user accounts are shown in *italics*.

— The examples show commands, command flags, filenames and directory names in `this font`. You provide your own file and directory names.

— The examples show system output (the result of performing a command) in `this font`.

Command Summary

This section lists the commands by the function the commands perform. Some commands perform a specific action on a file or directory which you specify. Other commands control the user environment and user processes.

Online Help

man	Display online man pages.

File Operations

cat, more, pg, view	Display text files on the screen.
cp	Create a copy of a file.
mv	Move or rename a file.
rm	Remove (delete) a file.
file	Determine file type.
cat	Concatenate several files.
find	Locate a specific file.
>	Redirect output, create a file.
ln	Create a link to a file.
od	Display binary files on the screen.

Directory Operations

ls	List the contents of a directory.
mkdir	Create a directory.
rmdir, rm	Remove a directory.
cd, chdir	Change the current working directory.
pwd	Display the name of the current working directory.
mv	Rename a directory.
dircmp	Compare the contents of two directories.
ln	Create a link to a directory.

Text Processing

vi, ex, ed, edit	Edit text files.
pg, more, view	View text files.
tail	View the end of a text file.
spell	Check spelling.
grep	Search for specific text in many files.
diff	Compare two files line by line.
diff3	Compare three files line by line.
wc	Count lines, words, and characters.
sort	Sort lines alphabetically or numerically.
cut	Remove text in columns.
paste	Reassemble text columns.
nl, pr	Insert line and page numbering.

Printing

lp	Print a text file.
lpstat	Show status of the line printer queue.
cancel	Cancel a print job.

Communications

mail, mailx	Read and send messages.
write	Two-way communications program.
ct, cu	Connect to a remote system.
mesg	Turn message notification on and off.
uname	Get the name of the host system.

Shell Commands and Process Control *

Ctrl-Z	Suspend the current job (C, K only).
Ctrl-C	Interrupt (stop) the current job.
Ctrl-D	End of file, or logout.
history	Display commands previously executed (C, K only).
alias	Rename commands and build new ones (C, K only).
jobs, ps	Display a job status report (C, K only).
fg	Bring a background process to the foreground (C, K only).
bg	Make a suspended process run in the background (C, K only).
kill	Terminate or end a process.
source	Read shell commands from a file (C only).
.	Read shell commands from a file (B, K only).

* The letters C, K, and B in parentheses indicate that the command is only available in the C shell (**csh**), the Korn shell (**ksh**), or the Bourne shell (**sh**), respectively.

File and Directory Access

chmod	Change file and directory access permissions.
newgrp	Switch from current group to another group.
chgrp	Change group ownership of a file or directory.
groups	Determine the groups you belong to.
umask	Read/set the default file and directory permissions.

Miscellaneous Commands

bc	Calculator.
cal	Print a calendar.
calendar	Send mail to remind you of appointments.
date	Display the current date and time.
factor	Display the prime factors of a number.
passwd	Change your login password.
quota	Display disk usage and limits.
stty	Display and set terminal parameters.
tee	Pipe output to two places.
who	Show who is logged in.

Alphabetical Command Reference

This section shows the most common DYNIX/ptx system commands and shell commands. A brief explanation appears to the right of the command name. Below the explanation there are examples of the command in use, complete with directory names, filenames, and options. For a complete list of command syntax and options, refer to the online man pages (the **man** command) or section 1 of the *DYNIX/ptx Reference Manual.*

Some commands are part of one of the available operating system command interpreters (shells). In these cases, the shell name appears in parenthesis, for example: *(C and Korn shell command.)*

General Command Syntax

When the command interpreter is ready to accept a command, it displays a *prompt* character at the left side of the screen. This character usually is a dollar sign (Bourne and Korn shell) or a percent sign (C shell).

Most DYNIX/ptx commands follow the following format:

— The command

— Option flags (optional)

— Arguments (usually file or directory names)

— Possibly file redirection or a pipe

To execute the command, press Return .

Commands, options, and arguments are case sensitive, and most commands are lowercase. Most options consist of a dash (–) followed by one or more letters. Arguments usually consist of filenames or directory names. Usually, the command, options, and arguments are separated by one or more spaces. Examples of redirection and pipes can be found in the *Shell Reference* section of this book and in the *User's Guide.*

---

alias Aliasing allows you to rename commands, abbreviate a long command, or create new commands to suit your needs.

Aliases typed at the command line are forgotten when you log out. Aliases can be placed in the startup files in a user's home directory. Aliases added to a startup file are in effect at the next login, or when the file is sourced (see **source**). Korn and C shells each have a different syntax. *(C and Korn shell command.)*

`alias`
By itself, **alias** displays all aliases that have been created.

`alias move mv` (C shell syntax)
`alias move=mv` (Korn shell syntax)
This provides a new name for the **mv** command. Whenever you type **move**, the **mv** command will be executed.

`alias rm 'rm -i'` (C shell syntax)
`alias rm='rm -i'` (Korn shell syntax)
This redefines the way the **rm** command is performed. Whenever the **rm** command is typed, the system will use **rm –i**. (The **–i** option causes *rm* to ask permission before overwriting an existing file or directory.)

`alias ll 'ls -CalF'` (C shell syntax)
`alias ll='ls -CalF'` (Korn shell syntax)
This creates a new command called **ll**. When you type **ll**, the **ls** command is executed with options **–C**, **–a**, **–l**, and **–F**.

---

bc A basic calculator. Enter an expression. The answer appears after you press Return. Type **quit** to exit.

```
bc
12*12+53
197
```

---

bg Restarts a suspended process and runs it in the background. A process is suspended by pressing Ctrl-Z. After the job has finished, the system displays a message at the next opportunity, usually the next time you press Return. *(C and Korn shell command.)*

---

cal Displays a calendar for any month and year. When a year is specified, all digits must be given. **cal 90** prints the calendar for the year 90, not 1990!

`cal`
With no arguments, **cal** prints out a calendar for the current month:

```
   February 1990
 S  M Tu  W Th  F  S
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28
```

`cal 7 1776`
This prints out a calendar for July 1776:

```
   July 1776
 S  M Tu  W Th  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

`cal 1990 > year1990`
This generates a calendar for the year 1990 consisting of 12 monthly calendars like the ones above grouped 3 by 4. The result is directed to the text file *year1990.*

---

calendar Consults the file named *calendar* in your home directory; any line with today's or tomorrow's date is displayed on the screen.

The *calendar* file is an ordinary text file. Date formats such as `Dec 19`, `December 19`, and `12/19` will all be recognized; upper or lower case may be used. Date formats having the day before the month will not be recognized.

---

cancel Cancels a printer job in the printer queue. Individual print jobs can be specified by the print job ID (obtained by executing **lpstat**). Or, all your own jobs on a specific printer can be canceled by specifying the printer. You can only cancel your own print jobs.

`cancel laser2`
Cancels your print jobs on printer *laser2*.

---

cat Displays the contents of a file (or files) on the screen. If several filenames are given, they are displayed in the order given. If no input filename is given, **cat** reads from the terminal (standard input) until it reaches an end-of-file character Ctrl-D; this character must appear on its own line.

`cat memo_1`
The contents of the file *memo_1* are displayed on the screen.

`cat memo_1 memo_2 memo_3 > memo_all`
The files *memo_1*, *memo_2*, and *memo_3* will be read and the result will be placed in *memo_all*; this combines the three files into one and leaves the original three files intact.

`cat > newfile`
This creates a text file without using a text editor. Since an input file was not specified, **cat** reads from the terminal (standard input) until it reaches a Ctrl-D appearing alone on a line. Whatever was typed is stored in the file *newfile*.

---

cd Change to a new directory, or your home directory. The new directory becomes the default directory for all actions performed on files.

```
cd
```
By itself, **cd** always returns you to your home directory.

```
cd starship
```
Change to the directory *starship*. The directory can be given as a path relative to the current directory (as shown) or as an absolute path.

```
cd ..
```
Change to the parent of the current directory (a special case of a relative path).

---

chdir Identical to the **cd** command. *(C shell only.)*

---

chgrp Change the group ownership of a file or directory. The first option is the desired group name; subsequent arguments are the files or directories to be changed. The **ls –l** command shows the group ownership of files.

---

chmod Change mode. Change the access permissions associated with a file or directory. Three types of access permission are granted or denied for each file and directory: *read*, *write*, and *execute*.

Access Permissions

	For files:	For directories:
Read	Read the file contents (**cat**, **pg**)	List the directory contents (**ls**)
Write	Modify, write to, or remove the file (**vi**, **rm**)	Create or delete files in the directory
Execute	Execute a file as a command (shell scripts)	Access files in the directory (**cd** to the directory)

The **chmod** command is always followed by a mode argument; this specifies what to do. The mode is followed by one or more file or directory names. The mode argument can take one of two formats, symbolic or octal .

Symbolic Notation

The mode can be broken into *who*, *change*, and *what* parts.

	The *who* part:
u	User (you)
g	Group: members of your group
o	Other: those not in your group
a	All: same as specifying **ugo**

	The *change* part:
+	Add permission
-	Deny or remove permission
=	Set permission, removes previous settings

	The *what* part:
r	Read permission
w	Write permission
x	Execute permission

`chmod g+w schedule`
Allows members of your group to write in the file *schedule*. Permissions for *user* and *other* remain unchanged.

`chmod go-rw schedule`
Removes *read* and *write* permission for *group* and *other* users; execute permission, if any, is not changed.

`chmod ugo+r schedule`
Gives *read* permission to all users.

`chmod a+r schedule`
Gives *read* permission to all users.

`chmod go= schedule`
Removes all permissions from *group* and *other* users for the file *schedule*; leaves existing *user* permissions intact.

Octal Notation

An octal mode consists of three digits: these digits represent the *user*, *group*, and *other* permissions.

Octal Digit	**Permissions Granted**	**Resulting Directory Entry**
0	no permissions	`---`
1	execute only	`--x`
2	write only	`-w-`
3	write & execute (2 + 1)	`-wx`
4	read only	`r--`
5	read & execute (4 + 1)	`r-x`
6	read & write (4 + 2)	`rw-`
7	read, write, & execute (4 + 2 + 1)	`rwx`

`chmod 700 Mail`
Gives the *user* read, write and execute permission, and removes permissions from *group*

and *other* users.

`chmod 644 memories`
Gives the *user* read and write permission for the file *memories*, and gives members of *group* and *others* permission to read only. Any prior privileges are removed.

---

cp Copy a file; leave the original file intact. Copies are made in the current directory if another directory is not specified.

The following examples assume that the current directory is */user1/starship/draft* and there is also another directory called */user1/starship/letters*.

`cp outline outline.new`
Makes a copy of the file *outline*. The new copy is called *outline.new* and it is placed in the current directory. If *outline.new* already exists in the current directory, it is overwritten by the copy.

`cp outline ..`
Copies the file *outline* to the directory */user1/starship*. The new copy will also be named *outline* ; if the file *outline* exists in that directory it is overwritten by the copy.

`cp outline ../letters`
Since *letters* is a directory, this creates a copy of the file *outline* in that directory; if the file *outline* exists in that directory it is overwritten by the copy.

`cp outline ../letters/plan`
This is similar to the example above, but instead of keeping the same name as the original, the copy is named *plan*. If the file *plan* exists in that directory it is overwritten by the copy.

`cp * ../letters`
This copies all the files in the current directory (*draft*) into the *letters* directory. Any file that has the same name in the *letters* directory is overwritten by the copy.

---

ct

Dials a telephone modem line and connects to a remote terminal. The command is followed by various options, and the telephone number is last. If several phone numbers are given, **ct** will dial each number in succession until one answers. The default modem speed is 1200 baud.

Valid characters for the telephone number are the digits 0 thru 9, asterisk (*), number (#). A – causes a four second delay after area code; an = causes a wait for a secondary dial tone.

```
ct -h -s1200 9=503-5551212
```
First, an outside line is obtained by dialing 9; then (503)5551212 is dialed, with a pause after the area code. The baud rate is set by the **–s** option to 1200. The **–h** option specifies that **ct** should not disconnect the local terminal once the connection is established.

---

cu

Connect to a remote system. The system is specified either by the **uucp** system name or a telephone number. Once connected, the system allows a set of commands (called *tilde escapes*) to be executed from within the program. These commands will: hang up (end) the connection; transmit files to or from the remote end; change transmission protocol; perform debugging; transmit break signals; exit to a shell to execute commands.

~.
Disconnect, end the dial-up session.

~!
Escape to a shell on the local computer without dropping the connection. Type Ctrl-D to return.

~! *command*
Execute *command* on the local system.

~$*command*
Execute *command* on the local system and send the resulting output to the remote system.

~%take thatfile
Copy file *thatfile* (located on the remote system) to the local system. The new copy has the same name, *thatfile*.

~%put thisfile
Copy file *thisfile* (located on the local system) to the remote system. The remote copy has the same name, *thisfile*.

~%cd directory
Changes the current active directory on the local system. Note that if you use `~!cd directory` to do this, it won't work.

---

cut Cut columns out of a table or extract fields from each line of a text file. The material to be cut can be selected by specifying the starting and ending column, or it can be marked by a field separator. The tab character is the default field separator; data files may use some other character—such as space, tab, colon, or semicolon—to separate each record.

cut -c9-13 birthdays
The *birthdays* file is typed so that a date appears in the ninth through thirteenth character on each line. The **–c** option of **cut** accesses those columns in the data file.

cut -d: -f1,5 /etc/passwd
This example uses data in the system password file to generate a list of user logins and their full names. The **–d** option sets the field separator character to something other than tab; in this case, the colon character used by the password file. The **–f** option specifies a list of the fields that are to be printed out, in this case the first and fifth fields.

---

date Displays the current day, date, and time. A format string may be added after the **+** option to customize the date format.

date

Results in the following output:

```
Thu Dec 28 15:44:06 PST 1989
```

date +'%A, %B %e, %Y %H:%I'

Results in the following output:

```
Wednesday, February  7, 1990  17:05
```

This is an example of a format string: `%A` displays the day of the week, `%B` displays the month, and so on. More format options are listed in the reference manual. The format string must be quoted if it contains spaces or tabs.

---

diff Compares two text files line-by-line and reports differences to the screen. The command is silent about lines that are the same in both files. Lines which are unique to the first file are displayed with "<" in the left margin; lines which are unique to the second file are displayed with ">" in the left margin. Other differences are shown by a combination of both markings, "<" followed by "---" followed by ">".

For each difference it finds, the **diff** command also shows the **ed** editor commands that will transform the first file into the second file.

diff johnson mcdonough

```
3,6c3,6
< Mr. Ron Johnson
< Layton Printing
< 52 Hudson Street
< New York, N.Y.
---
> Mr. J.J. McDonough
> Ubu Press
> 37 Chico Place
> Springfield, N.J.
9c9
<Dear Mr. Johnson:
---
> Dear Mr. McDonough:
```

This compares two business letters in the files *johnson* and *mcdonough* and reports the difference; in this case the only difference is the inside address and salutation. The editing commands are `3,6c3,6` and `9c9`.

---

diff3 Compares three text files line-by-line and reports differences to the screen. Output is similar to the **diff** command. Text lines which differ across all three files are marked with `====`. Text lines which are different for only one file are marked `====1`, `====2`, or `====3`.

```
diff3 smith jones moore
====
1:4c
  Dear Mr. Smith:
2:4c
  Dear Mr. Jones:
3:4c
  Dear Miss Moore:
```

The `====` indicates that these three files each have a different salutation.

---

dircmp Compares two directories; reports files that are unique to either directory, and reports whether files having the same name have the same content or are different.

```
dircmp /u/jon/plan /u/jim/plan | pg
```

This compares the *plan* directories of two users, and pipes the result to a pager to make it easier to read. One page of the output has two columns: files that are unique to */u/tom/plan*, and files that are unique to */u/jim/plan*. Another page of output shows filenames that appear in both directories. If two files with the same name have the same contents, they are marked `same`, otherwise, they are marked `different`.

---

ed A text editor (line editor); **ed** is useful on teletype or printing terminals. Supports mark and return, cut and paste, search and replace with regular expressions. For more information, see Chapter 8 of the *User's Guide* .

---

edit A text editor. Invokes the **ex** text editor with several options set as shown here. These settings make **ex** easier for the first-time or casual user.

novice	ON
report	ON
showmode	ON
magic	OFF

```
edit textfile
```

This edits the file called *textfile*. If there is no file by this name in the current directory, it starts editing a new file by that name.

```
edit
```

This starts the editor on a blank file. You will need to name the file by writing the file before quitting.

---

factor Finds the prime factors of a number.

```
factor 84
2
2
3
7
```

---

fg Foreground. Reactivates a job that was suspended by Ctrl-Z or placed in the background by **bg**. *(C and Korn shell command.)*

---

file Performs some tests on a file and reports back what type of file it is. Useful to find out if a file is a directory, executable, or text file.

```
file a.out
a.out:              executable not stripped
```

File *a.out* is a runable program; this type of file cannot be printed or viewed on the screen.

```
file hello.c
hello.c:            c program text
```

File *hello.c* is an ordinary text file containing a C language program.

```
file pagesizes
pagesizes:          ascii text
```

File *pagesizes* is an ordinary text file.

```
file check
check:              commands text
```

File *check* is an executable text file, probably a shell script file.

```
file starship
starship:           directory
```

starship is a directory.

---

find Locate any file with a particular name or whose name matches a specific pattern. **find** requires two or more arguments; first, the directory to start the search from; second, one or more of the *expressions* in the following table. Several search directories may be given.

The search is performed in the specified directory and all directories under it. For this reason, **find** may take a while if there are many directories to search.

You can display the names of files found by specifying the **–print** option. You can also specify that a command be performed for every file that matches the search criteria; braces { } may be used as part of the command argument to signify the name of the file.

Expression	Meaning
-name *filename*	Find file with the name *filename*.
-name '*pattern*'	Find files whose names match the pattern.
-mtime *n*	Find file that has been modified exactly *n* days ago.
-mtime -*n*	Find file that has been modified less than *n* days ago.
-mtime +*n*	Find file that has not been modified for at least *n* days.
-atime *n*	Similar to **-mtime**, but checks access time of a file (also accepts +*n* or -*n* days).
-print	Display the name of any file found by the above methods.
-exec *command* { } \;	Execute the *command* on the found file.
-ok *command* { } \;	Execute the *command* on the found file, with prompt.

```
find . -name plan -print
```
Look for all files named *plan* and display the full pathnames of those files wherever they are found. Only an exact match will be successful, so files named *PLAN* or *Plan* will not be found. The search starts in the current directory (.) and descends into all subdirectories.

```
find /u/jill -name 'memo*' -print
```
Look for all files whose names start with *memo* and display the names of those files on the screen. Files named *memo*, *memo1*, *memo.for.caj* will all be found. This search starts in directory

/u/jill and descends into all subdirectories.

```
find /u/joe -mtime -5 -print
```

Reports files which have been modified in the last four days. The search starts in the directory */u/joe* and descends into all subdirectories.

```
find /u/joe -atime +120 -print
```

This example reports files which have not been accessed for at least 120 days.

```
find . -name a.out -ok rm {}\;
```

This search starts in the current directory (.) and descends into all subdirectories. It will look for all files named *a.out* and delete them by executing the **rm** command. **find** will ask permission to execute the **rm** command for each file found. (The backslash and semicolon must appear at the end of the line.)

---

grep

Searches for text in a file. Lines which match the search pattern are displayed. By default, **grep** searches are case sensitive. **grep** can search for the pattern in several files.

grep requires two arguments. The first argument is a text string to search for; if this text contains spaces or special characters, the text should be surrounded by single quotes ('). The second (and subsequent) argument is the file or files to be searched.

```
grep automation johnson
and office automation software.
```

Search the file *johnson* for lines containing the word "automation." One line was found and displayed.

```
grep -i automation johnson
Automation Summary.
and office automation software.
```

The **–i** option makes the search case-insensitive.

---

groups Shows your group memberships.

---

history The shell (C and Korn shell only) will store your most recent commands. You can access this stored information to reexecute a command, display all the commands executed thus far, execute a new command using the same arguments you used in the previous command, and perform text editing (Korn shell only) or text substitution on the command line.

The Korn shell and C shell handle command history differently. See Chapter 4 of the *User's Guide* for more information.

---

jobs Displays the status of jobs that have been suspended or placed in the background.

The jobs are displayed in the order they were sent to the background. A + indicates the most recent job, a – indicates the previous job. `running` indicates the job is running in the background. `suspended` indicates the job has been suspended with Ctrl-Z ; it will not go away until it is either killed or allowed to run to completion. An **fg** command will bring the most recent job (+) to the foreground. *(C and Korn shell command.)*

```
jobs
[1]     Running     pr chap5 | lp &
[2] -   Running     grep machine spec* > mach &
[3] +   Running     pr chap1 | lp &
```

This shows that three jobs are running in the background. The most recent job is the third one. These **pr** and **grep** commands were all placed in the background by the **&** character (see the *Shell Reference* section).

```
        jobs -l
[1]     15325  Running    pr chap5 | lp &
[2]  -  15330  Running    grep machine spec* > mach &
[3]  +  15349  Running    pr chap1 | lp &
```

The **–l** option brings up the process ID (PID) for each job.

---

kill Kills or stops a job that has been suspended or is running in the background. The job to be killed is specified by its process ID number (PID), or by a percent sign (%) followed by the job number. The PID can be obtained three ways: record the number that appears on the screen when the job is placed in the background; use the **ps** command; or, use the **jobs –l** command (see **jobs** and **ps**). *(C and Korn shell command.)*

```
kill 15325 15330
```
Kills two jobs, process IDs 15325 and 15330.

```
kill %1 %2
```
Kills two jobs, the first and second jobs operating in the background (see **jobs**).

---

ln **ln** creates a link to a file, which in effect creates a new directory entry for that file. Links are pointers that associate a filename with the location of the actual file on the disk.

Links allow a single copy of a file to appear in several directories. The advantage of this is that several users can have access to a file, each in their own directory, without making separate copies of the file.

---

lp This command prints a file or files. The named files will be printed on the default system printer unless another printer is specified. The system will return information which identifies your print job; use this identification if you wish to use the **cancel** command or change your print request with the **lp** command. A specific printer is specified by **-d** followed immediately by the name of the printer.

Various options allow you to: change previous print parameters; print multiple copies; set a particular line length and page length; select special forms paper for printing; select fonts, character pitch, or line spacing; and set other options available at your site.

`lp -dim4 schedule`
This sends the file *schedule* to the printer. The printer destination in this case is **im4**. The **lp** command reports back the following information:

```
request id is im4-106 (1 file)
```

The print job identification, `im4-106`, is a combination of the destination printer name and a unique number to distinguish this job from other jobs on the printer.

---

lpstat This command reports the status of the line printer queue. When several jobs are waiting to be printed, they are stored in the queue; the print job at the top of the queue list is the one currently being printed. This command can also be used to determine the name of the default system printer used by the **lp** command.

`lp -dim4 schedule; lpstat`
This command line actually contains two commands: the **lp** command, followed by a **lpstat** to check the line printer queue. The output of the **lpstat** command indicates that the print job is indeed on the line printer queue:

```
im4-106  clinel  570  Feb 20 21:23 on im4
```

The print job ID is `im4-106`, the person submitting the print request is *clinel*, the file is 570 bytes long, and it is being printed on printer `im4`.

---

ls

List the contents of a directory or directories. **ls** shows a list of file and subdirectory names. By default, the names are sorted in alphabetical order. The options can be combined; some useful combined options include **ls –CF** and **ls –CalF**.

```
ls
```

By itself, **ls** shows the contents of the current directory. Files starting with a period will not be listed. Names of directories and executable files cannot be distinguished from conventional files (see **ls –F**).

```
ls starship
ls ..
ls /
```

With a directory argument, **ls** shows the contents of that directory. The directory name may be an absolute path or reference relative to the current directory. The examples show the files in: directory *starship* ; the parent of the current directory (..); and the system root directory (/).

```
ls -C
Mail    a.out    calendar    hello.c    whodo
```

Lists directory entries in two or more columns across the screen.

```
ls -a
```

Shows filenames that start with a period, including the current directory (.), and the parent directory (..), which normally are not shown.

```
ls -F
Mail/
a.out*
calendar
hello.c
whodo@
```

Appends / to all directory names, * to the names of executable files, and @ to symbolic links.

```
ls -m
Mail, a.out, calendar, hello.c, whodo
```

Lists many filenames on each line with each name separated by a comma and a space.

ls -r
Reverse the sorting order.

ls -l

```
total 66
drwxr-xr-x  2 joe  lt   512 Feb 21 12:36 Mail
-rwxr-xr-x  1 joe  lt 27955 Jan 18 10:35 a.out
-rw-r--r--  1 joe  lt   726 Feb 20 20:17 calendar
-rw-r--r--  1 joe  lt    64 Jan 18 10:35 hello.c
lrwxr-xr-x  1 joe  lt    10 Feb 21 12:36 whodo ->
                                         /etc/whodo
```

Long listing. Shows the file type, file and directory permissions, number of links, owner, group ownership, size in bytes, last modification date and time, and filename. The file type is coded by a single character in the leftmost column of the listing: `-` for regular files; `d` for directories; and `l` for symbolic links. Symbolic links also show `->` after the filename; the path to the actual file appears at the end of the line.

---

mail Sends and reads incoming mail messages. See Chapter 10 of the *User's Guide* for more information.

---

mailx Sends and reads incoming mail messages. See Chapter 10 of the *User's Guide* for more information.

---

man Displays man pages, which are an online version of the *Reference Manual.* Most man pages contain the following sections:

— NAME: the command name and a single-sentence description of what the command does.

— SYNOPSIS: a terse description of command syntax.

— DESCRIPTION: a detailed description of the command and options.

— SEE ALSO: cross reference to other man pages.

— FILES: related files.

— BUGS: known problems.

More information about **man** and other online resources can be found in Chapter 5 of the *User's Guide* .

`man vi`
Displays the man page for the **vi** editor. If there is more than one screen of text, the default pager **pg** will allow you to scroll through the text.

`man -k pattern`

```
awk (1)       - pattern scanning and processing
                language
egrep (1)     - search a file for a pattern
                using full regular expressions
grep (1)      - search a file for a pattern
nawk (1)      - pattern scanning and processing
                language
regex (1F)    - match patterns against a string
```

The **-k** option performs a keyword search for the specified word(s). In this case, subject lines containing the word "pattern" are displayed. The number in parentheses is the *Reference Manual* section number.

`man 2 write`
Specifies section 2 of the *Reference Manual* . Displays the man page for the **write**(2) function call, which is not the same man page obtained by typing **`man write`** .

---

mesg Some commands (such as **write**) will write a message directly on the screen of another user. Each user has control over whether or not such messages will appear on the screen. The **mesg** command is used to allow or disable messages, and also allows you to check the current **mesg** setting.

`mesg`
By itself, the **mesg** command reports whether messaging is turned on (**y**) or off (**n**).

`mesg y`
Allows messages to appear on the screen at any time.

`mesg n`
Prevents messages from appearing on the screen.

---

mkdir Makes a new directory under the current directory. You must give a new directory name with this command. (Remove directories with **rmdir**; list directory contents with **ls**.)

`mkdir starship`
Creates a new directory under the current directory.

---

more Displays a text file or files: move forward; search forward or backward for text; jump to the next or previous file. If the file is longer than one screen, the word `--more--` appears at the bottom of the screen, and you can issue the following commands:

h	help, shows a list of **more** commands
Space	displays the next screenful of text
Return	displays the next line of text
q	quits **more**

`more outline`
This allows you to look at the *outline* file one screen at a time.

`cal 1990 | more`
This illustrates how the output of one command, **cal**, can be piped to the **more** command for viewing.

---

mv Move a file (or files) to a new directory, rename a file or directory.

```
mv filename directory
```
Moves the file *filename* from the current directory to the directory specified.

```
mv *.old oldfiles
```
Moves all files ending with the prefix *.old* to a directory named *oldfiles*. The directory *oldfiles* must already exist; **mv** will not create it.

```
mv oldname newname
```
Renames the file *oldname* to *newname*. If *oldname* is a directory, renames the directory.

---

newgrp Change from one group to another. At login, you belong to a default group; you may change to another group with the **newgrp** command. After changing to a new group, all files and directories you create will have the new group listed as group owner.

---

nl Number the lines of a text file. The numbers appear at the left margin and are separated from the text by a tab.

```
nl chap1 > chap1.n
```
Line numbers are placed at the beginning of each non-blank line. The result is placed in the file *chap1.n* .

```
nl chap1 -ba | lp
```
All lines are numbered, including blank lines. The result is sent to the printer.

```
nl chap1 -s':  ' | lp
```
Changes the separator character from a tab to a colon followed by two spaces. The result is sent to the printer.

---

passwd	Change your login password after you have logged in. The system will not echo (display) your password on the screen as you type it. You will be asked to type a new password and then to retype it.
paste	Merge several text files into one file; can be used to combine several tables into one large table. The first lines of each file will be joined into one long line; then the second line, and so on.

pg

Display a text file or files: move forward or backward; search forward or backward for text; jump to the next or previous file. You can issue the following commands at the colon prompt:

h Return	help, shows a list of **pg** commands
q Return	quits **pg**
Return	displays the next screenful of text

```
pg outline
```
This allows you to look at the *pg* file one screen at a time.

```
cal 1990 | pg
```
This illustrates how the output of one command, **cal**, can be piped to the **pg** command for viewing.

pr

Paginate a file. Divides a text file into pages with 66 lines; each page has a five-line header that shows the page number, current date, time, and filename. The output is displayed on the screen. Numerous options allow you to merge files, and to control the header, page length, line length, and so on.

ps

Displays information about the processes (jobs) being handled by the system. Your login is one process; usually it will appear as the command sh, csh, or ksh in the table. If processes are running in the background or are suspended, they are shown too. By default, **ps** shows processes belonging to you, and each process's identification number:

PID	Process identification number
TTY	Name of the controlling terminal
TIME	Computer execution (run) time
COMMAND	The command and options that are being executed

Many options are available to display more detailed information about each process, and information about all processes on the system.

---

pwd Displays the current working directory. When you log in, you are placed in your home directory. If you change directories with the **cd** command and forget what directory you are in, this command tells you where you are.

---

quota Checks quotas set for disk space and the total number of files; reports if you are over either limit.

---

rm Remove a file.

```
rm -i s*
shipments: ? n
schedule: ? y
```

The –i option will cause a prompt to appear before each file is removed. The answer y will remove the file. This is useful when many files are specified by a wild card (*); the confirmation prevents files from being lost accidentally.

```
rm -r starship
```

The recursive option –r removes all of the files in the directory *starship* . If any directories exist under *starship*, then *those* directories and their files are also removed. Finally, the *starship* directory is removed. This is a powerful option; it can delete all of your files and directories. For security, use **–ri**.

---

rmdir Removes the specified directory. If there are files in that directory, it cannot be removed until all files are removed.

```
rmdir starship
```
Removes the directory *starship*, provided that it is empty.

---

sort Sorts a file or files, and merges several sorted files into one sorted file. The default **sort** of one file will reorder each line of the file alphabetically. The ASCII sorting sequence is used; digits are placed first, followed by uppercase alphabetic, then lowercase. Various options will: change the default sorting order; sort in numeric order instead of alphabetic; ignore case; ignore punctuation; and ignore leading spaces and tabs.

```
sort glossary
```
This sorts the *glossary* file and displays the result on the screen.

```
sort -o glossary glossary
```
The **–o** option will sort the *glossary* file and overwrite the original file with the result. (If you use file redirection to do this, the source file will be lost!)

---

source Reads the specified file and executes each line as if it were a command typed at the system command line. The **source** command is a C shell command. The same command in Bourne and Korn shells is the period (.) followed by the filename. Frequently this is used if changes are made to a user startup file; the changes take effect in the current shell if the file is sourced.

```
source .cshrc
```
For C shell users, this causes additions made to the *.cshrc* file to take effect.

---

spell Checks spelling in the specified file. Words not found in the dictionary are displayed on the screen.

```
spell chap.1 | pg
```
Check the spelling of file *chap.1*. This example pipes the output of **spell** to the pager **pg**; this prevents a long list of words from scrolling off the screen.

---

stty Displays and sets terminal options such as the data transmission rate and special control keys like backspace. This command frequently appears in each user's start-up file.

```
stty
new tty, speed 9600 baud; tabs crt
decctlq
erase = ^H
```
By itself, **stty** shows the current settings. Each terminal type will show different settings.

---

tail Prints the last part of a file on the screen.

```
tail conference
```
Prints the last 10 lines (default) of the file *conference.*

```
tail +20 conference
```
Displays all the lines after line 20 in the file *conference.*

```
tail -20 conference
```
Displays the last 20 lines of the file *conference.*

---

tee This is an extension to the pipe (|). A **tee** is like a pipe but it sends the output of one command to two places (instead of one). It sends output to the next command on the command line (like a pipe) and also to the display screen. This is useful for watching what is going through a pipeline, to monitor the progress of commands in the pipeline, or to debug a pipeline.

---

umask Reports or changes the *default* file permission value. This default value establishes the access permissions at the time that a file or directory is created. This command is normally placed in a user's *.profile* or *.login* file. (Use the **chmod** command to change the permissions of an existing file or directory.)

By itself, the **umask** command reports three digits that are the *permissions mask* for *user*, *group*, and *other*. The meaning of each digit is shown in the table below. (Note that the digits of the **umask** permission mask have the inverse of the meaning they have for **chmod** octal notation.)

Octal Digit	**Permissions Granted**	**Resulting Directory Entry**
7	no permissions	`---`
6	execute only	`--x`
5	write only	`-w-`
4	write & execute	`-wx`
3	read only	`r--`
2	read & execute	`r-x`
1	read & write	`rw-`
0	read, write, & execute	`rwx`

```
umask
22
```

This reports a **umask** value of `022` (leading zeros are not printed). With this setting, newly created directory will have the following permissions: (0) read, write, and execute permission for the *user*; (2) read and execute permission for members of the same *group*; (2) read and execute permission for all *other* users.

`umask 077`
This sets the **umask** value to `077` for the current login session. This setting will deny all permissions to *group* and *other* and retains all permissions for *user*.

---

uname This command reports your system name. The system name can be used by others to send you mail from a remote system. System names are set up by the system administrator.

---

vi The visual text editor. Uses many of the same commands as the **ed** editor, and is designed for display terminals. Supports mark and return, cut and paste, search and replace with regular expressions. For more information, see the ***vi** Command Reference* in this book and Chapter 9 of the *User's Guide* .

---

view View a file. This invokes the visual text editor, **vi**, with the **–R** flag set (read only). This mean the editor can be used to view a file without the risk of changing or overwriting the file. The powerful search and scroll capabilities of **vi** are available.

---

wc Word count. Displays the number of lines, words, and characters in each file you specify.

```
wc draft7.mm
311     1513    10291 draft7.mm
```

The file *draft7.mm* contains 311 lines, 1,513 words, and 10,291 characters.

`wc -l draft7.mm`
Reports only the number of lines.

wc -w draft7.mm
Reports only the number of words.

wc -c draft7.mm
Reports only the number of characters.

---

who Displays user information. For each person currently on the system, it shows the login name, the terminal line, and the date and time they logged in.

---

write Copies text from your terminal to the display of another user. Your message is sent one line at a time; a line is not sent until you press Return. The recipient may write back by responding with the **write** command. When you are done, type Ctrl-D at the start of a line.

```
write jmrs
There is a meeting at noon today.
Return
Ctrl-D
```
This will flash the message on *jmrs*'s screen, including your login name and the date.

Ctrl-Z Suspends the current job. *(C and Korn shell command.)*

---

vi Editor Commands

When you start **vi**, you are in *command mode*. Use one of the *insert* commands to start the *insert mode*. Type the new text, then press the Esc key to return to command mode.

Insert/Overtype Text

R	Overtype mode
r	Replace current character with next character typed
a	Insert after cursor
A	Insert at end of line
i	Insert at cursor
I	Insert before first character
o	Insert after current line
O	Insert before current line

Delete Text

dd	Delete current line
x	Delete current character
D	Delete to end of line
dG	Delete everything from current line to the end of the file

Change Text

cc	Change current line
C	Change to end of line
s	Change character
cw	Change to next word
cb	Change from beginning of word
ce	Change to end of word
c)	Change to end of sentence

After issuing a *change* command, a **$** appears at the position affected by the editing operation. The text you type will replace the text between the current cursor position and the **$**. Type the new text, then press the Esc key to return to command mode.

Read, Write, and Quit Commands

:w	Write the file
:w *file*	Write file to a new file named *file*
:wq	Write file and quit
:q	Quit if no changes
:q!	Quit without writing
:r *file*	Read *file* and insert at current position
:!*command*	Execute any operating system command, return to document when done
!!*command*	Execute operating system command, replace current line with the output of the command

Basic Cursor Motion

Return	Beginning of next line down
+	Beginning of next line down
–	Beginning of next line up
0 (zero)	Beginning of current line
$	End of current line
h	Previous character
j	Next line down
k	Next line up
l	Next character
w	Next word
b	Previous word
e	End of word

Screen Control

Ctrl-U	Scroll up partial screen
Ctrl-D	Scroll down partial screen
Ctrl-B	Scroll up full screen
Ctrl-F	Scroll down full screen
Ctrl-L	Redraw screen

Text Search & Replace

/*string*	Forward search for *string*
?*string*	Backward search for *string*
n	Repeat last search
N	Repeat last search in reverse direction
:s/*string1***/***string2***/**	Replace *first* occurrence of *string1* on the current line with *string2*
:s/*string1***/***string2***/g**	Replace *all* occurrences of *string1* on the current line with *string2*
:1,$s/*string1***/***string2***/g**	Replace *string1* with *string2* throughout the entire file

Text Mark & Return

m*c*	Mark current location with a single letter *c* (*c* must be lowercase)
'*c*	Return to the location marked by letter *c*

Marks allow you to set a marker any place in the text and return to the same location.

Miscellaneous

u	Undo previous command
U	Restore current line
Y	Yank current line into buffer
P	Put buffer before current line
p	Put buffer after current line
xp	Transpose two characters
:f	Display file and line information
:*number*	Go to line *number*

Shell Reference

More information about shell commands can be found in the *DYNIX/ptx User's Guide* and the following **man** pages: **sh**(1) for Bourne shell, **csh**(1) for C shell, and **ksh**(1) for the Korn shell.

File and Directory Names

The following characters should not be used in filenames or directory names. These characters have a special meaning to some or all of the shell programs (the command interpreters).

```
/ * ? [ ] ~ { } ; | ( )
\ ' " < > $ : ^ ` # &
```

A filename that begins with a period (.) will not show in a conventional **ls** directory listing; these filenames do appear if the **ls -a** command is used. Special system filenames and directories often begin with a period, for instance *.login*, *.profile*, *.cshrc*, *.logout*, and *.newsrc*.

A filename should not start with a dash (–), but dashes may occur inside filenames. Commands will interpret the leading dash as a command option or flag, instead of a file, and the command will fail or deliver unexpected results.

File Redirection and Pipes

<	**Input redirection** A command reads its input from a specified file `mailx ted < report` This redirects the *report* file to the **mailx** command. This command will mail *report* to user *ted*.
>	**Output redirection** Output of a command is redirected to a file (previous file contents are lost). `spell letter > spellerror` The list of misspelled words in the file *letter* is placed in a new file called *spellerror* ; if *spellerror* already exists, it is overwritten.

>> **Append output redirection**
Output of a command is appended to the end of a file (previous file contents are retained).

`cat summer >> spring`
Appends the contents of the file *summer* to the file *spring*; the file *summer* is unchanged.

| **Pipe**
Causes the output of one command to become the input of another command. The pipe appears between commands; execution occurs from the leftmost command to the rightmost on the command line.

`date | cut -c12-19`
The output of this command states that it is 4:00 in the afternoon:

```
    16:00:06
```

The output of the **date** command is fed to the **cut** command. The **date** command would normally report:

```
    Wed Feb  7 16:00:09 PST 1990
               ^^^^^^^^
```

Here, the **cut** command filters out all but columns 12 to 19, leaving the time as the output.

`ls | grep old | wc -l`
Any number of commands may be strung together. This example reports the number of files in the current directory that have the word *old* in the filename. **ls** gets all the filenames and places one name on each line; **grep** searches this list of filenames for the text string `old` and reports one name per line; the **wc -l** command reports the number of output lines from the **grep** search.

Special Characters

Shell documentation refers to some of these characters as *metacharacters*.

/ **Slash**
Separates directory names from other directory and filenames. When used alone, signifies the root directory.

***** **Asterisk** (splat)
Matches zero, one, or more of any character in a file or directory name.

`spell abc*`
Checks spelling on all files in the current directory that start with the text string *abc*. Thus, the files *abc8*, *abc9*, *abc10*, *abc11*, and *abc.glossary* will all be checked.

? **Question mark**
Matches exactly one character in a file or directory name.

`spell abc?`
Checks spelling on files starting with the word *abc* followed by a single character. With the files named above, checks spelling of *abc8* and *abc9*.

`spell abc??`
Checks spelling on files starting with the word *abc* followed by two characters. With the files named above, checks spelling of *abc10* and *abc11*.

[*ab*] **Match set**
Matches any characters that are in the brackets, such as [fFwWsS] for upper and lowercase f, w and s.

[*a–z*] **Match Range**
Matches a single character that is in the range from *a* to *z*, such as [a–z] for lower case alphabetic, [A–Z] for

upper case alphabetic, [0–9] for digits, and [a-zA-Z] for combined upper and lower case.

; **Command separator**
Several commands may be specified on one line if they are separated by a semicolon.

```
cd ..; ls
```

This command line does two things: change to a new directory, then list the contents of the directory.

~ **Home directory**
(C and Korn shell only.) Tilde represents your home directory when used alone in place of a filename or path; it represents the home directory of any user when it precedes the name of their home directory.

```
ls ~
```

List the contents of your own home directory (works no matter what directory you are in at the time).

```
vi ~/schedule
```

Starts the **vi** editor on the *schedule* file in your home directory, no matter where you are.

```
cd ~janelle
```

Change to the home directory of another user, *janelle*.

& **Background command**
Placing a **&** at the end of the command line causes a command to run in the background. The system reports the process number and returns a prompt. Output directed to *stderr* or *stdout* will appear on the screen unless it is redirected.

```
spell encyclopedia > spellerror &
```

This results in the following output:

```
[1] 14757
$
```

The spell checking is performed as a background task, process number 14757 in this case, and the system prompt returns.

– **Command options** and **standard input**
The dash signals the start of a command option. Some commands allow the dash to signify *read from standard input.*

1003-51133-00