

The UNIX File System Continued

This is the second chapter that discusses the UNIX file system and its associated commands; it complements the material discussed in Chapter 5. Chapter 8 presents more commands for manipulating files, including commands for copying files, moving files, and looking at the contents of a file. The chapter also explains the shell input/output redirection operators and file substitution metacharacters.

In This Chapter

8.1	FILE READING				
	8.1.1	The vi Editor Read-Only Version: The view Command			
	8.1.2	Reading Files: The pg Command			
	8.1.3	Specifying Page or Line Number			
8.2	SHELL	REDIRECTION			
	8.2.1	Output Redirection			
	8.2.2	Input Redirection			
	8.2.3	The cat Command Revisited			
8.3	ENHAN	CED FILE PRINTING			
	8.3.1	Practicing Linux Alternative Command Options			
8.4	FILE MA	ANIPULATION COMMANDS			
	8.4.1	Copying Files: The cp Command			
	8.4.2	Moving Files: The mv Command			
	8.4.3	Linking Files: The In Command			
	8.4.4	Counting Words: The wc Command			
8.5	FILENAME SUBSTITUTION				
	8.5.1	The ? Metacharacter			
	8.5.2	The * Metacharacter			
	8.5.3	The [] Metacharacters			
	8.5.4	Metacharacters and Hidden Files			
8.6	MORE	FILE MANIPULATION COMMANDS			
	8.6.1	Finding Files: The find Command			
	8.6.2	Displaying the Beginning of a File: The head Command			
	8.6.3	Displaying the End of a File: The tail Command			
	8.6.4	Selecting Portions of a File: The cut Command			
	8.6.5	Joining Files: The paste Command			
	8.6.6	Another Pager: The more Command			
	8.6.7	Linux Pager: The less Command			
8.7	11 XINU	TTERNALS: THE FILE SYSTEM			
	8.7.1	UNIX Disk Structure			
	8.7.2	Putting It Together			
COMMA	AND SUI	MMARY			

REVIEW EXERCISES

Terminal Session

8.1 FILE READING

Chapter 5 explained how you can use the vi editor or the **cat** command to read files. To refresh your memory, you can use vi with the read-only option to read files, or you can use **cat** to view a small file. Using the **cat** command to view a large file, [**Ctrl-s**] to stop screen output, and [**Ctrl-q**] to resume screen output is very inconvenient. Try the following examples to get the feel of it. Doing so will make you appreciate the other file reading commands that let you view files one screen at a time.



Assume that your working directory is david and you have a file (say, 20 pages long) called large_file in it.

- □ To read large_file using the vi editor, type vi -R large_file and press [Return]. This opens large_file for read only. Contents of large_file are displayed on the screen, and you can use vi commands to view the other pages.
- To read large_file using the cat command, type cat large_file and press [Return]. The contents of large_file are displayed and scroll before your eyes. You can use [Ctrl-s] and [Ctrl-q] to stop and resume scrolling.

8.1.1 The vi Editor Read-Only Version: The view Command

Some UNIX systems provide the vi editor version called **view** that you can use to read files. It is a good tool for reading large files because you can use all of the vi editor commands to facilitate the reading of different parts of your file. Viewing a file using the **view** command prevents you from saving your editing or changes to the file. You can use it only to read a file. Refer to Chapter 6 for usage examples.

8.1.2 Reading Files: The pg command

You can use the **pg** command to view files one screen at a time. A prompt sign (:) is produced at the bottom of the screen, and you press [**Return**] to continue viewing the rest of the file. The **pg** command shows EOF (end of the file) on the last line of the screen when it reaches the end of your file. You press [**Return**] at this point to get to the \$ prompt.

Using the **pg** command options gives you more control over the format and the way you want to view your file. Table 8.1 summarizes these options.



Unlike other commands' options, some pg options start with the plus sign (+).



The **pg** command is not available on all systems. You can check its availability by typing **pg**, or **man pg** (to look at the manual pages about the pg commands). In each case, some sort of message is displayed announcing that **pg** is not available.

Table 8.1
The pg Command Options

Option	Operation
-n	Does not require [Return] to complete the single-letter commands.
-s	Displays messages and prompts in reverse video.
-num	Sets the number of lines per screen to the integer <i>num</i> . The default value is 23 lines.
-pstr	Changes the prompt: (colon) to the string specified as str.
+line–num	Starts displaying the file from the line specified in <i>line-num</i> .
+/pattern	Starts viewing at the line containing the first occurrence of the specified <i>pattern</i> .



Assuming you have a file called large_file in your working directory, use the **pg** command to read it by doing the following:

- Type **pg large_file** and press [**Return**]. This is a simple way of looking at large_file one screen at a time.
- ☐ Type **pg -p Next +45 large_file** and press [**Return**] to view large_file starting from line 45 and show the prompt **Next** instead of the normal prompt : (colon).

Two options are used: **-p** to change the default prompt from : to **Next**, and +45 to start viewing from line 45.

☐ Type **pg -s +/hello large_file** and press [**Return**] to show prompt and other messages in reverse video, and start viewing from the first line that contains the word *hello*.

Two options are used: **-s** to show the prompt and other messages in reverse video, and **+/hello** to search for first occurrence of the word *hello*.

When the **pg** command displays the prompt sign: (or any other prompt if you have used the **-p** option), you can give commands to move forward or backward a specified number of pages or lines to view different parts of the text. Table 8.2 summarizes some of these commands.

Table 8.2
The pg Command Key Operators

Key	Operation
+ <i>n</i>	Advances n screens, where n is an integer number.
-n	Backs up n screens, where n is an integer number.
+ <i>n</i> l	Advances n lines, where n is an integer number.
-nl	Backs up n lines, where n is an integer number.
n	Goes to the n^{th} screen, where n is an integer number.

8.1.3 Specifying Page or Line Number

You can specify a page number or line number from the beginning of the file or relative to the current page number. Use unsigned integers to indicate that the reference is to the beginning of the file. For example, type **10** to go to page 10, or **60l** (that's *six*, *zero*, *lowercase letter l*) to go to line 60 in the file.

Use signed integers to indicate that the reference is relative to the current page. For example, type +10 to move forward 10 pages, or type -30l (*minus sign three, zero, low-ercase letter l*) to move backward 30 lines. If you type only + or - without any numbers, the command is interpreted as +1 or -1, respectively.



These operators are applicable only while you are viewing your file and pg is displaying the prompt sign.



If you use **pg** with the **-n** option, then you do not need to press [**Return**] for single-letter operators.

8.2 SHELL REDIRECTION

Among the most useful facilities that the shell provides are the *shell redirection operators*. Many UNIX commands take input from the standard input device and send the output to the standard output device. This is usually the default setting. Using the shell redirection operators, you can alter where a command gets its input and where it sends its output. The command's standard (default) input/output device is your terminal.

The shell redirection operators allow you to do the following things:

- Save the output of a process in a file.
- Use a file as input to a process.



- 1. A process is any executable program. This could be an appropriate shell command, an application program, or a program you have written.
- 2. The redirection operators are instructions to the shell and are not part of the command syntax. Accordingly, they can appear anywhere on the command line.
- 3. Redirection is temporary and effective only with the command using it.

8.2.1 Output Redirection

Output redirection allows you to store the output of a process in a file. Then you can edit it, print it, or use it as input to another process. The shell recognizes the greater than sign (>) and double greater than sign (>>) as output redirection operators.

The format is as follows:

command > filename

or

command >> filename

For example, to get the list of the filenames in your working directory, you use the **ls** command. Type **ls** and press [**Return**]. The shell default output device is your terminal screen (the standard output device). Consequently, you see the list of files on the screen. Suppose you want to save the **ls** command output (the list of filenames in the directory) in a file. One way to do that is to redirect the output of the **ls** command from the screen to a file as follows:

```
$1s > mydir.list [Return] . . . Redirect ls output to mydir.list.
```

This time the **ls** command output is not sent to the terminal screen, but is saved in a file called mydir.list. If you open the mydir.list file, you see the list of files.



- 1. If the specified filename already exists, then it is written over, and the contents of the existing file are lost.
- 2. If the specified filename does not exist, then the shell creates a new file.

The double greater than sign (>>) redirection operator works the same as the greater than sign (>) operator, except for the fact that it appends the output to the specified file. If you type **ls** >> **mydir.list**, the shell adds the filenames in the working directory to the end of the file called mydir.list.



- 1. If the specified file does not exist, then the shell creates a new file to save the output in it.
- 2. If the specified filename does exist, then the shell adds the output to the end of the file, and the previous contents of the file remain intact.

The following command sequences show more examples of the output redirection operators.



To obtain a hard copy of the filenames in your home directory, do the following:

```
$ cd [Return]
                    . . . . . . . . . Change to your HOME directory.
$ 1s -C [Return] . . . . . . . List filenames in the david
                                     directory in column format. You
                                     have two files:
myfirst
            yourlast
$ ls -C > mydir.list [Return] . . Save the output in mydir.list.
$_ . . . . . . . . . . . . . . . Done. The prompt is back.
$ cat mydir.list [Return] . . . . Check what you have in
                                     mydir.list.
            yourlast
myfirst
$ lp mydir.list [Return] . . . Print the list.
request id is lp1-8056 (1 file)
$_ . . . . . . . . . . . . . . . Ready for the next command.
```



To append the list of the users on the system to mydir.list, do the following:



To save the current year's calendar in this_year, print it, and then remove it from the file, do the following:

```
$ cal > this_year [Return] . . . Save output of cal in this_year.
$ lp this_year [Return] . . . . Print it.
request id lp1-6889 (1 file)
$ rm this_year [Return] . . . . Remove it.
$_ . . . . . . . . . . . . . . . . . . Ready for the next command.
```

8.2.2 Input Redirection

The input redirection operator allows you to issue commands or run programs that get their input from a specified file. The shell recognizes the less than sign (<) and the double less than sign (<<) as the input redirection operators. The format is as follows:

```
command < filename
```

or

```
command << word
```

For example, to send mail to another user, you use the **mailx** command (**mailx** is discussed in Chapter 10) and type **mailx daniel** < **memo**. This command tells the computer to send mail to the user called *daniel* (his user id). The input to **mailx** is not coming from the standard input device, your terminal, but from the file called **memo**. Thus, the input redirection operator (<) is used to indicate where the input comes from.



Try using the **cat** command with the input redirection operator.

☐ Type cat < mydir.list and press [Return] to display the contents of mydir.list, UNIX responds:

```
myfirst yourlast
```

Using the **cat** command with the input redirection operator gives you the same result as the **cat** command with the filename as an argument (cat mydir.list). There are other commands that work in the same manner. If you specify a filename on the command line, the command takes its input from the specified filename. If you do not specify any argument, the command takes its input from the default input device (your keyboard), and if you use the input redirection operator to specify where the input comes from, then the command takes its input from the file you specify.

The redirection operator (<<) is used mostly in script files (shell programs) to provide standard input to other commands.

8.2.3 The cat Command Revisited

Now that you know about the shell redirection capabilities, we can explore the **cat** command in more detail. The **cat** command was introduced in Chapter 5, where it was used to show the contents of a small files on the screen. However, the **cat** command can be used for many things other than displaying files.

To refresh your memory, let's look at the following command lines:

```
$ cat myfirst [Return] . . . Display myfirst.
$ cat -n myfirst [Return] . . . Display myfirst with line numbers.
```

```
$ cat -n myfirst
    1 The vi history
    2 The vi editor is an interactive text editor that is supported
    3 by most of the UNIX operating systems.
$_
```



The **cat -n** command is used throughout this book to display content of the files. It is convenient to have the line numbers when one or more lines are referenced in the text.

Creating Files

By using the **cat** command with the output redirection operator (>), you can create a file. For example, if you wanted to create a file called myfirst, you would type **cat** > **myfirst**. This command means that the output of the **cat** command is to be redirected from the standard output device (your terminal) to a file called myfirst. The input comes from the standard input device—your keyboard. In other words, you type the text, and **cat** saves it in the myfirst file. You signal the end of the file by pressing [**Ctrl-d**].

This feature of the **cat** command is useful for creating small files quickly. Of course, you can also use it to create long files, but you must be a very accurate typist because after you press [**Return**] you cannot edit the text you have typed. The following command sequences show how to create a file using the **cat** command.

Try using the cat command with the output redirection operator to create a file.





- 1. If myfirst does not exist in your working directory, then cat creates it.
- 2. If myfirst already exists in your working directory, then cat overwrites it. The contents of the old myfirst are lost.
- 3. If you do not want to overwrite a file, use the (>>) operator.



Try appending text to the end of myfirst in your current directory.

\$ cat >> myfirst [Return] . . . Append to a file called myfirst.
_ Type your text.

However, for now, we have to suffer and read all these boring UNIX books.

[Ctrl-d] Signal the end of the input text.
\$_ Back to the prompt.
\$ cat myfirst [Return] Display the contents of myfirst.
I wish there was a better way to learn
UNIX. Something like having a daily UNIX pill.
However, for now, we have to suffer and read all these boring UNIX books.



- 1. If myfirst does not exist in your working directory, then cat creates the file.
- 2. If myfirst exists, then **cat** appends the input text to the end of the existing file. In this example, your one line of input was appended to myfirst. Thus three lines of text are displayed.

Copying Files

You can use the **cat** command with the output redirection operator to copy files from one place to another. The following command sequences show how this capability of the **cat** command works.



Try copying myfirst in the david directory to another file called myfirst.copy.

<pre>\$ cd [Return]</pre>	Make sure you are in your
	HOME directory.
<pre>\$ cat myfirst > myfirst.copy [</pre>	Return] . Copy myfirst to
	myfirst.copy.
\$	Ready for the next
	command.



The input to the **cat** command is myfirst file, and the output from the **cat** command (the contents of the myfirst) is saved in myfirst.copy.



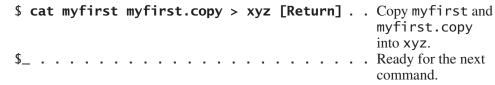
Now copy myfirst in the david directory to the source directory, and call it myfirst.copy:

<pre>\$ cat myfirst > source/myfirst.copy</pre>	[Return]	
<pre>\$ ls source/myfirst.copy [Return] .</pre>		 myfirst.copy and place it in the source directory. Check to see if it has been copied.
myfirst.copy \$		 Yes, myfirst.copy is in the source directory.



Because you are in your HOME directory, the pathname to the file myfirst.copy in the source directory must be specified in both cat and ls commands.

Next, use the **cat** command to copy two files into a third file.





- 1. The previous contents of the xyz file, if any, are lost.
- 2. There is a space between each file name in the command line.

Appending Files

You can use the **cat** command with the output redirection append operator (>>) to add a number of files together into a new file.



Append two files to the end of the third file.

\$	cat	myf	first	m	yfi	rs	t.c	ору	>>	xyz	[R	et	urn].	Append myfirst
															and myfirst.
															copy to the end
															of the file called
															XYZ.
\$_															Prompt is
															displayed.



- 1. Using the (>>) redirection operator saves the previous contents of xyz, if any, and the two files are added to the end of xyz.
- 2. You may have more than two filenames in the command line, but they must be separated by a space.
- 3. The files are appended to the specified output file in the same sequence in which the input files are specified.

8.3 ENHANCED FILE PRINTING

The **lp** command sends your file to the printer as it is; it does not change the appearance or format of your file. You can, however, improve the appearance of the output by formatting it—for example, adding page numbers, page headings, and double-spaced lines to a document before sending it to the printer or viewing it on the screen.

You use the **pr** command to format a file before printing or viewing it. The **pr** command with no options formats the specified file into pages 66 lines long. It puts a five-line heading at the top of the page that consists of two blank lines, one line of information about the specified file, and two more blank lines. The information line includes the current date and time, the name of the specified file, and the page number. The **pr** command also produces five blank lines at the end of each page.



Try using the **pr** command to format myfirst:

Type **pr myfirst** and press [**Return**] to format a file called myfirst. (See Figure 8.1.)

Figure 8.1File Printed with Page Formatting: Five-Line Heading and Five-Line Blank Footing

```
[2 blank lines]

Nov 28 16:30 2001 myfirst Page 1

[2 blank lines]

The vi history
The vi editor is an interactive text editor which is supported by most of the UNIX operating systems. However, . . .

rest of the page . . .

[5 blank lines at bottom of page]
```

The **pr** command output is displayed on your terminal, the standard output device. But most of the time you will want to format files for a hard copy printout. One way to do that is to use the output redirection operator. There are other ways to send formatted files to the printer, such as using the pipe operator (+) (which is explained in Chapter 9). Let's save the formatted version of myfirst in another file and then print it.

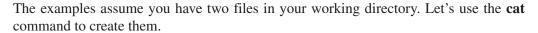
pr Options

90 [Return]

It is not enough to place five lines at the top and five lines at the bottom of each page and call it formatting. The **pr** options allow you to format a file's appearance with a little more sophistication. Table 8.3 summarizes the **pr** command options.

- 1. The -m or -columns is used to produce multicolumn output.
- 2. The -a option can only be used with the -columns option and not -m.

The following command sequences show the output of the ${\bf pr}$ command using different options.





```
$ cat > names [Return] . . . . . Create a file called names.
David [Return]
Daniel [Return]
Gabriel [Return]
Emma [Return]
[Ctrl-d]
$ cat > scores [Return] . . . . . Create a file called scores.
```

Table 8.3
The pr Command Options

Option		Operation
UNIX	Linux Alternative	
+page	pages=page	Starts displaying from the specified <i>page</i> . The default is page 1.
-columns	columns=columns	Displays output in the specified number of <i>columns</i> . The default is one column.
-a	across	Displays output in columns across (rather than down) the page, one line per column.
-d	double-space	Displays output double spaced.
-h string	header=string	Replaces the filename in the header with the specified <i>string</i> .
-l number	length= <i>number</i>	Sets the page length to the specified <i>number</i> of lines. The default is 66 lines.
-m	merge	Displays all the specified files in multiple columns.
-p		Pauses at the end of each page and sounds the bell.
-s character	separator=character	Separates columns with a single specified <i>character</i> . If character is not specified, then [Tab] is used.
-t	omit-header	Suppresses the five-line header and five-line trailer.
-w number	width=number	Sets line width to the specified number of characters. The default is 72.
	help	Displays help page and exits.
	version	Displays version information and exits.

```
100 [Return]
70 [Return]
85 [Return]
[Ctrl-d]
$_ . . . . . . . . . . . . . . . . . Ready for the next command.
```

Notice that most of the options for the **pr** command under UNIX can be used also for Linux. However, Linux provides some alternative options that do the same job and some additional options that can be used only under Linux.

For example, look at the following commands:

```
$ pr -a myfirst [Return] . . . . Works for UNIX and Linux.
$ pr --across myfirst [Return] . . . Works only for Linux.
$ pr --help [Return] . . . . . . Works only for Linux.
```



To show the names in column format, and change the heading to STUDENT LIST, type pr -2 -h "STUDENT LIST" names and press [Return].

[2 blank lines] Nov 28 2005 14:30 STUDENT LIST

[Page 1]

[2 blank lines]

David Daniel Gabrie1 Emma

[5 blank lines at bottom of page]

The -h option changes the heading, but if the specified string has embedded white space, then you must put the string in quotation marks.

However, your screen will scroll up and you will not see the entire file. One way to observe the output on the screen is to redirect the **pr** output to a file, and then use the vi editor or the **view** command (the read only version of vi) to see the formatted output.

- ☐ Type **pr myfirst > outfile** and press [**Return**]. You have saved the output of the **pr** command in outfile.
- Type view outfile and press [Return] to see the output generated by the pr command.



If the view command is not available on your system, type vi -R (for read only) instead.

Display names in two columns across the page and suppress the header:

☐ Type **pr -2 -a -t names** and press [**Return**].

David Daniel Gabriel **Emma**

The difference between options -2 and -2 -a is the order in which the columns are arranged.



Show the files names and scores side by side:

☐ Type **pr -m -t names scores** and press [**Return**]

David	90
Daniel	100
Gabriel	70
Emma	85

The **-m** option shows the specified files side by side in the same order in which the filenames are specified in the command line.



Display names in 2 columns, separated by @ character, and omit header:

☐ Type pr -2 -s@ -a names and press [Return].

Notice the **-s**@ option caused the names to be separated by @.

David@Daniel Gabriel@Emma

8.3.1 Practicing Linux Alternative Command Options



Use the Linux alternative options for the **pr** command, as suggested in the following command lines. Notice some of the options, such as *pages*, are meaningful when applied to large files.

<pre>\$ prpages=2 large_file [Return]</pre>	1 2
	large_file
	starting from
	page 2.
<pre>\$ prcolumns=2 myfirst [Return]</pre>	
	in two columns.
<pre>\$ prdouble-space myfirst [Return]</pre>	1 2
	myfirst in
	double space.
<pre>\$ promit-headercolumns=2across names [Return]</pre>	. Display names
	across two
	columns with
	no header.
<pre>\$ promit-headercolumns=2separator=@ names [Return] .</pre>	1 2
	in two columns
	separated by
	@ sign.
\$ prhelp [Return]	. Display the help
	page.



Read the help page and familiarize yourself with other options available for the **pr** command.

8.4 FILE MANIPULATION COMMANDS

Some of the file manipulation commands were discussed in Chapter 5. From that discussion, you know how to create directories (using the **mkdir** command), create files (using the **vi** and **cat** commands), and delete files and directories (using the **rm** and **rmdir** commands). Now we will look at a few more commands to increase your knowledge

of file manipulation in UNIX. These commands are used to copy (cp), link (ln), and move (mv) files. The general format of these commands is as follows:

```
command source target
```

where *command* is any of the three commands, *source* is the name of the original file, and *target* is the name of the destination file.

8.4.1 Copying Files: The cp Command

The **cp** (copy) command is used to create a copy (duplicate) of a file. You can copy files from one directory to another, make a backup copy of a file, or just copy files for the fun of it!



Suppose you have a file called REPORT in your current directory and want to create a copy of it. To do so, you type **cp REPORT REPORT.COPY** and press [**Return**].

REPORT is the source file and REPORT. COPY is the target file. If you do not provide the correct pathname/filename for the source or target file, **cp** complains by showing a message similar to the following:

```
File cannot be copied onto itself O file(s) copied
```

Figure 8.2 shows your directory structure before and after application of the **cp** command.



If the target file already exists, then its contents are destroyed.

The following command sequences show how the **cp** command works.



Figure 8.2 An Example Application of the **cp** Command

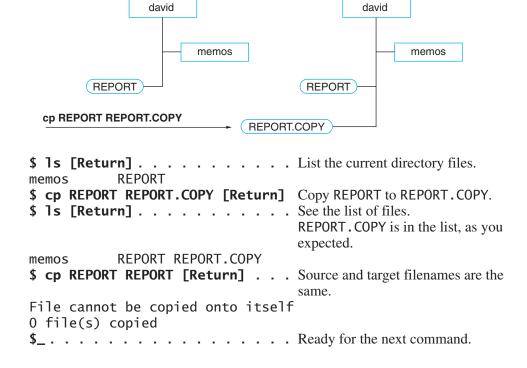


Figure 8.3 Another Example Application of the cp Command





Copy a file from your current directory to another directory. Figure 8.3 shows your directory structure before and after the **cp** command.

- **\$ cp REPORT memos [Return]** . . . Create a copy of REPORT in memos.
- \$ ls memos [Return] List files in memos directory, and REPORT is there, as you expected.

REPORT



When the target file is a directory name, then the source file is copied in the specified directory with the same filename as the source filename.

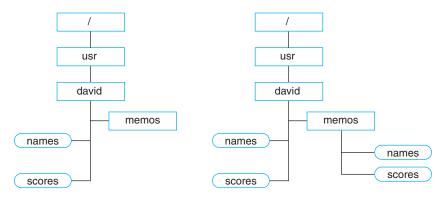
You can also copy multiple files to another directory. Suppose you have files called names and scores under your current directory (david) and you want to copy them to the memos directory. Figure 8.4 shows your directory structure before and after the **cp** command.

- \$ cp names scores memos [Return] . Copy names, scores in current directory to the memos directory.
- Ready for the next command.



- 1. The files called names and scores are in your current directory.
- 2. The filenames on the command line are separated by at least one space.
- 3. The last filename must be a directory name. In this case, memos is a directory name.

Figure 8.4 Using the **cp** Command to Copy Multiple Files to Another Directory



cp names score memos

Table 8.4
This cp Command Options

Option		Operation
UNIX	Linux Alternative	
-b	backup	Makes a backup of the specified file if file already exists.
-i	interactive	Asks for confirmation if the target file already exists.
-r	recursive	Copies directories to a new directory.
verbose		Explain what is being done.
	help	Displays the help page and exits.

cp Options

Table 8.4 summarizes the **cp** command options.

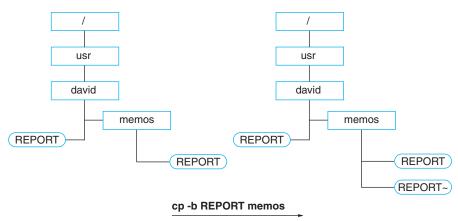
-b Option The **-b** (backup) option creates a backup copy of the file if the file you want to copy already exits in the target directory. This protects you from overwriting an existing file.



Use the **cp** command with the **-b** (backup) option. Assuming you already have a file called REPORT under the memos directory, a backup copy of REPORT will be created. This means the REPORT file under memos is not overwritten. Figure 8.5 shows your directory structure before and after the cp -b command.

Notice that REPORT is backed up and the backup filename is REPORT~.

Figure 8.5
Using the cp Command with the -b Option



-i Option The -i option protects you from overwriting an existing file. It asks for confirmation if the target file already exists. If your reply is *yes*, it copies the source file, overwriting the existing file. If your answer is *no*, then it quits, and your existing file remains intact.

-r Option It takes a long time and is a tedious job to copy files one by one if you have a long list of files to copy. You can use **cp** with the **-r** option to copy directories and all their contents into a new directory.



Try using the **cp** command with the **-i** option.

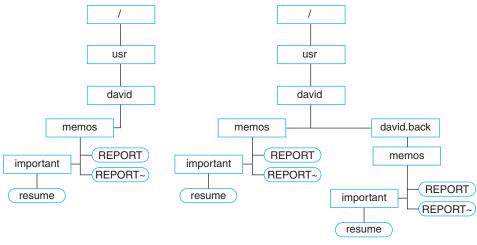


Suppose you have a directory called memos in your current directory and you want to copy all the files and subdirectories under memos to the david.bak directory. Figure 8.6 shows your directory structure before and after the **cp-r** command. Notice there is a sub directory called important under memos that contains a file called resume. Copy files and subdirectories in david to another directory called david.bak using the **-r** option.



- 1. If david.bak exists in your current directory, then files and directories in memos are copied into david.bak.
- 2. If david.bak does not exist in your current directory, then it is created and all the files and directories including memos itself are copied into david.bak. Now the pathname for files in memos under david.bak is ./david.bak/memos.

Figure 8.6
Using the cp Command with the -r Option



cp -r ./memos ./david.bak

Practicing Linux Alternative cp Options



Use the Linux alternative options for the **cp** command, as suggested in the following command lines:



Read the help display and familiarize yourself with other options available for the cp command in Linux.



Copy files using the **-verbose** or **-v** option.



Notice the feedback from the **-v** option that shows what file is copied and where it is copied.

8.4.2 Moving Files: The mv Command

You use the **mv** command to move a file from one place to another or to change the name of a file or a directory. For example, if you have a file called REPORT in your current directory and you want to change its name to REPORT.OLD, you type **mv REPORT REPORT.OLD** and press [**Return**].

Figure 8.7 shows your directory structure before and after application of the **mv** command to rename REPORT.



Move REPORT to the memos directory.

Figure 8.8 shows your directory structure after application of the **mv** command to move REPORT.

Figure 8.7
Using mv to Rename a File

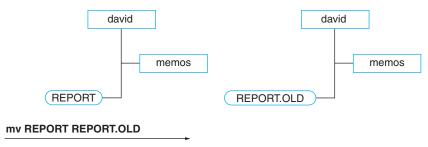
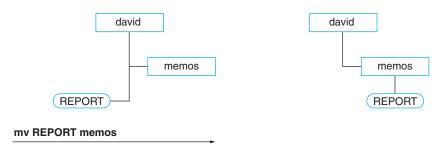


Figure 8.8
Using mv to Move a File



Both the **cp** and **mv** command accept more than two arguments, but the last argument must be a directory. For example, the command

copies xfile, yfile, and zfile to the directory called backup, provided that the backup directory exists.

mv Options

Table 8.5 shows some of the **mv** command options. The following command lines show the usage of the **mv** command options.

Table 8.5The mv Command Options

Option		Operation	
UNIX	Linux Alternative		
-b	backup	Makes backup of the specified file if file already exists.	
-i	interactive Asks for confirmation if the target file already exists.		
-f	force	Removes target file if file already exists and does not ask for confirmation.	
-v	verbose Explains what is being done.		
help		Displays the help page and exits.	
	version	Displays version information and exits.	



Use the **mv** command with the **-b** (backup) option. Assuming you already have a file called REPORT under the memos directory, a backup of REPORT will be created. This means the REPORT file under memos is not overwritten.

Notice that REPORT is backed up and the file name is REPORT~.



Use the **mv** command with the **-i** (confirmation) option. Assuming you already have a file called REPORT under the memos directory, a confirmation prompt is displayed and you press [y]es [Return] or [n]o [Return] to signal your intention.



Use the **mv** command with the **-v** (verbose) option.

```
$ mv -v REPORT memos [Return] . . Move REPORT to memos.

REPORT -> memos/REPORT

$_ . . . . . . . . . . . . . . . . . Ready for the next command.
```



Practicing Linux Alternative mv Options

Use the Linux alternative options for the **mv** command, as suggested in the following command lines:

```
$ mv --backup REPORT memos [Return] . . . . Same as mv -b REPORT memos.
$ mv --interactive REPORT memos [Return] . . . Same as mv -i REPORT memos.
$ mv --verbose REPORT memos [Return] . . . . Same as mv -v REPORT memos.
$ mv --version [Return] . . . . . . . . . . . . Displays version information.
$ cp --help [Return] . . . . . . . . . . . . . . . . Displays the help page.
```



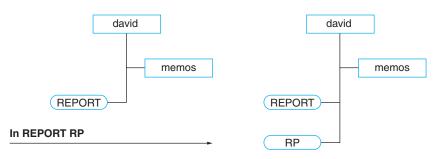
Read the help page and familiarize yourself with other options available for the **mv** command.

8.4.3 Linking Files: The In Command

You can use the **In** command to create new links (names) between an existing file and a new filename. This means you can create additional names for an existing file and refer to the same file with different names. For example, suppose you have a file called RE-PORT in your current directory, and you type **In REPORT RP** and press [**Return**]. This creates a file named RP in your current directory and links that name to REPORT. Now, REPORT and RP are two names for a single file. Figure 8.9 shows your directory structure before and after the **In** command application.

At first glance, this looks like the **cp** command, but it is not! The **cp** command physically copies the file into another place, and you have two separate files. Whatever

Figure 8.9Using the **In** Command to Link Filenames



changes you make in one are not reflected in the other. The **ln** command, however, just creates another filename for the same file; no new file is created. If you change anything in any of the linked files, the changes are there in the file regardless of the name you use to refer to it.



To experiment with the **ln** command, try the following command sequence:

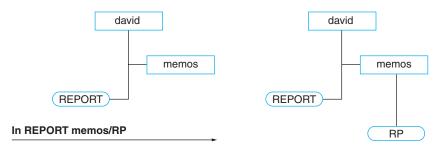
```
$ cat > xxx [Return] . . Create a file called xxx. Now type the
                             following line in the file:
                             Line 1:aaaaaa
$ [Ctrl-d]
             . . . . . . . Signal end of the input.
$ ln xxx yyy [Return] . Link yyy to xxx.
$ cat yyy [Return] . . . Display the contents of the xxx file, but use the
                             new filename yyy; the output is, as you
                             expected, the contents of xxx:
Line 1:aaaaaa
$ cat >> yyy [Return] . Append a line to the end of yyy, and type the
                             following line:
                             Line 2:bbbbbb
[Ctrl-d]..... Signal the end of your input.
$ cat yyy [Return] . . . Display the contents of yyy. You have two lines
                             in yyy, as you expected:
Line 1:aaaaaa
Line 2:bbbbbb
$ cat xxx [Return] . . . Display the contents of xxx. You have two lines
                             in it because xxx and yyy are actually the same
                             file:
Line 1:aaaaaa
Line 2:bbbbbb
$_ . . . . . . . . . . . Ready for the next command.
```

If you specify an existing directory name as the new filename, you can access the file in the specified directory without typing its pathname. For example, suppose you have a file called REPORT and a subdirectory called memos in your working directory, and you type **In REPORT memos** and press [**Return**]. Now you can access REPORT from the memos directory without specifying its pathname, in this case .../REPORT.

If you want to specify a different name for the file in a different directory, type **In REPORT memos/RP** and press [**Return**]. Now RP in the memos directory is linked to REPORT, and from memos you can use the filename RP to refer to REPORT.

Figure 8.10 shows your directory structure before and after application of the **ln** command.

Figure 8.10Using the **In** Command to Link Filenames to a Directory



Chapter 5 explained that the **ls -l** command lists the filenames in the current directory in a long format and that the second column in the long-format output shows the number of the links.



To list the files in david in long format, type ls -l and press [Return].

```
$ ls -l
total 2
drwx rw- --- 1 david student 32 Nov 28 12:30 memo
-rwx rw- --- 1 david student 155 Nov 18 11:30 REPORT
```



To link REPORT to RP and list the files using the **ls** command with the **-l** option to see the number of links, type **ln REPORT RP** and press [Return]. Then type **ls -l** and press [Return].

```
$ In REPORT RP
$ ls -l
total 3
drwx rw- --- 1 david student 32 Nov 28 12:30 memo
-rwx rw- --- 2 david student 155 Nov 18 11:30 REPORT
-rwx rw- --- 2 david student 155 Nov 18 11:30 RP
```



When you create a file, you also establish a link between the directory and the file. Thus, the link count for every file is at least one, and subsequent use of **ln** adds to the number of links.

Linux Alternative Options for the In Command



Use the Linux alternative options for the **ln** command, as suggested in the following command lines:

```
$ ln --version [Return] . Display version information.
$ ln --help [Return] . . . Display the help page.
```



Read the help page and familiarize yourself with other options available for the **ln** command.

Some Final Words

The three commands **cp**, **mv**, and **ln** all affect the filenames and work in a similar manner, but they are different commands and are used for different purposes:

- cp creates a new file.
- **mv** changes the filename or moves files from one place to another.
- In creates additional names (links) for an existing file.

8.4.4 Counting Words: The wc Command

You can use the **wc** command to find out the number of lines, words, or characters in a file or a list of specified files.

The following command sequences show the output of the **wc** command, assuming you have the myfirst file in your current directory.



First display the contents of myfirst. Then show the number of lines, words, and characters in it:

4 30 155 myfirst \$_.... Ready for the next command.

The first column shows the number of lines, the second column shows the number of words, and the third column shows the number of characters.



A word is considered a sequence of characters with no white space (space or tab character). Therefore, what? is one word, and what? counts as two words.

If no filename is specified, then **wc** gets its input from the standard input device (keyboard). You signal the end of the input by pressing **[Ctrl-d]**, and **wc** shows the results on the screen.



Use the **wc** command to get the count of input from the keyboard:

	Invoke the wc command with no filename. The sign is the prompt indicating that the shell is waiting for the rest of the
	command. Type the following text:
The wc command is useful to	find out how large your file is.
[Ctrl-d]	End your input; wc displays the output.
2 13 48	

You can specify more than one filename as the argument. In this case, the output shows one line of information for each file, and the last line shows the total counts.

\$_.... Ready for the next command.

Suppose you have two files in your current directory. The following command sequences show the output of the **wc** command specifying the two filenames as arguments.



Count the number of lines, words, and characters in the specified files:

<pre>\$ wc myfirst</pre>	yourfirst [Return]	. Show the counts of myfirst and yourfirst.
24 10 3 100 27 110	400 myfirst 400 yourfirst 800 total	

. Ready for the next command.

Table 8.6
This wc Command Options

	Option	Operation
UNIX	Linux Alternative	
-l	lines	Reports the number of lines.
-w	words	Reports the number of words.
-с	chars	Reports the number of characters.
	help	Displays the help page and exits.
	version	Displays version information and exits.

wc Options

You can use the wc command with options to get the number of lines, words, or characters only, or in any combination. Table 8.6 summarizes the wc command options.



- 1. When no option is specified, the default is all options (**-lwc**).
- 2. You can use any combination of the options.

The following command sequences show the use of the **wc** options.



Count the number of lines in myfirst:



Save the output of the **wc** command in a file and print it.



Linux Alternative Options for the wc Command

Use the Linux alternative options for the **wc** command, as suggested in the following command lines.:

```
$ wc --lines myfirst [Return] . . . . . Report only the number of lines.
$ wc --chars --lines myfirst [Return] . . Report the number of lines and characters in myfirst.
$ cp --help [Return] . . . . . . . . . . . Displays the help page.
```



Read the help page and familiarize yourself with other options available for the wc command in Linux.

8.5 FILENAME SUBSTITUTION

Most file manipulation commands require filenames as arguments. When you want to manipulate a number of files, for example, to transfer all files with a filename starting with letter a to another directory, typing all the filenames one by one is tiring and boring. The shell supports *file substitution*, which allows you to select those filenames that match a specified pattern. These patterns are created by specifying filenames that contain certain characters that have a special meaning to the shell. These special characters are called *metacharacters* (or *wild cards*). Table 8.7 summarizes the wild cards that can stand for one or more characters in a file name.

Table 8.7The Shell File Substitution Metacharacters

Character	Operation
?	Matches any single character.
*	Matches any string, including the empty string.
[list]	Matches any one of the characters specified in the <i>list</i> .
[!list]	Matches any one of the characters not specified in the <i>list</i> .



File substitution metacharacters (wild cards) can be used in any part—at the beginning, middle, or end—of the filename to create a search pattern.

For the terminal sessions in this section you need to create new files and directories. It is a good idea to create a directory for each chapter to be used for the terminal sessions and exercises of that chapter. Under the Chapter8 directory, create a subdirectory called 8.5. If you have not created a Chapter8 directory in your HOME directory, here is one way to do this:

```
$ cd [Return] . . . . . . . Change to the HOME directory.
$ mkdir Chapter8 [Return] . Create Chapter8 directory.
$ cd Chapter8 [Return] . . . Change current directory to Chapter8.
$ mkdir 8.5 [Return] . . . Create a directory called 8.5.
```

Now create the following files using **cat** command or vi editor. There is no need to type anything in the files, or just type in few words if you want. We need these files only for their names. Create the following six filenames:

```
report report1 report2 areport breport report32
```

8.5.1 The ? Metacharacter

The question mark (?) is a special character that the shell interprets as a single character substitution and expands the filename accordingly.



Try the following command sequences to see how the ? metacharacter works:

The shell expands the filename report? to the filename report followed by exactly one character, any character. Thus, the filenames report1 and report2 are the only two files that match the pattern.

The shell expands the filename report?? to the filename report followed by exactly two characters, any characters. Thus, the filename report32 is the only file that matches the pattern.

```
$ 1s -C ?report [Return] . . . Put ? at the beginning of the filename. areport breport $_ . . . . . . . . . . . . . . . Ready for the next command.
```

The shell expands the filename ?report to the filename report preceded by exactly one character, any character. Thus the filenames are port and breport are the only two files that match the pattern.

8.5.2 The * Metacharacter

The asterisk (*) is a special character that the shell interprets as any number of characters (including zero characters) of substitution in a filename, and expands the filename accordingly.



Try the following command sequences to see how the * metacharacter works:

The shell expands the filename report* to the filenames report followed by any number of characters, any characters. Thus the file names are port and breport are the only two filenames that do not match the pattern. The * wild card includes zero characters following the specified pattern. Thus, the file name report followed by no character matches the pattern and is displayed.

The shell expands the filename *report to the filename report preceded by any number of characters, any characters. Thus the filenames report, are port, and breport are the only files that match the pattern. Remember, the * wild card includes zero characters following the specified pattern. Therefore, the filename report preceded by no character matches the pattern and is displayed.

The shell expands the file name $\mathbf{r}^*\mathbf{2}$ to the filename \mathbf{r} followed any number of characters, but the last character of the filename must be the character 2, referring to any file that starts with r and ends with 2.

8.5.3 The [] Metacharacters

The open and close brackets are special characters that surround a string of characters. The shell interprets this string of characters as filenames that contain the specified characters, and expands the filenames accordingly.

Using the ! before the specified string of characters (but within the brackets) causes the shell to display the filenames that do *not* contain the characters in the string at the specified position.



Experiment with the bracket metacharacters by doing the following:

 \square To list all the filenames that start with a or b, type **ls** -C [ab]* and press [Return], UNIX responds:

```
areport breport $_ . . . . . . Ready for the next command.
```

The shell expands the filename [ab]* to the filename a or b followed by any number of characters, any characters. Thus, areport and breport are the only two filenames that match the specified pattern.

☐ To list all the filenames that do *not* start with *a* or *b*, type **ls -C** [!ab]* and press [Return], UNIX responds:

```
report report1 report2 report32
```



You can use the [] metacharacters to specify a range of characters or digits. For example, [5-9] means the digits 5, 6, 7, 8, or 9; and [a-z] means all the lowercase letters of the alphabet.



The following command sequence shows the use of brackets with a specified alphabet or digits range:

☐ Type **ls** *[1-32] and press **[Return]** to list all the filenames that end with the digits 1 to 32, UNIX responds:

8.5.4 Metacharacters and Hidden Files

To use the metacharacters for displaying hidden files—filenames starting with . (dot)—you must explicitly have the . (dot) as part of the specified pattern.



To list all the invisible (hidden) files, type $\operatorname{ls-C}$ * and press <code>[Return]</code>, UNIX responds:

.exrc .profile

The shell expands the filename .* to the filename . (dot) followed by any number of characters, any characters. Thus, only the hidden files are displayed.



The pattern is .* and there is no space between the dot and the asterisk.

Wild cards are not limited in use to only the **ls** command. You can use wild cards with other commands that need filename arguments.



To experiment with some more examples of filename substitution, try the following:

\$ rm *.* [Return]	Delete all the files with filenames that contain at least one dot.
<pre>\$ rm report? [Return]</pre>	Delete all filenames that begin with <i>report</i> and end with only one character, any character.
\$ cp * backup [Return]	Copy all files from the current directory to the backup directory.
<pre>\$ mv file[1-4] memos [Return]</pre>	Move file1, file2, file3, and file4, indicated by the range [1–4], to the memos directory.
<pre>\$ rm report* [Return]</pre>	Delete all the filenames that start with the string <i>report</i> .
\$	Ready for the next command.



In the last example, there is no space between the string **report** and the asterisk wild card. The **rm report*** command deletes all the files with filenames that begin with **report.**

```
$ rm report * [Return] . . . . . Delete all files.
$_. . . . . . . . . . . . . . . . Ready for the next command.
```



In the preceding example, there is a space between the string *report* and the asterisk wild card. This space can have disastrous consequences. The command rm report * is interpreted as "delete a file called report, and then delete all the other files." In other words, all files in the current directory are deleted.

8.6 MORE FILE MANIPULATION COMMANDS

The following commands facilitate the search for a specific file in a crowded hierarchy of directories and display a specified portion of a file for a quick look.

8.6.1 Finding Files: The find Command

You can use the **find** command to locate files that match a given set of criteria in a hierarchy of directories. The criterion may be a filename or a specified property of a file (such as its modification date, size, or type). You can also direct the command to remove, print, or otherwise act on the file. The **find** command is a very useful and important command that is not used to its full potential. Maybe its unusual command format is discouraging.

The format of the **find** command is different from that of the other UNIX commands. Let's look at its syntax:

find pathname search options action option

pathname indicates the directory name from which **find** begins its search, and it then continues down to the subdirectories and their subdirectories and so on. This process of branching search is called a *recursive search*. The *search options* part identifies which file you are interested in, and the *action option* part tells what to do with the file once it is found.

Let's look at a simple example:

\$ find . -print [Return]

This command displays the names of all the files in the specified directory and all its subdirectories.

- 1. The specified directory is indicated by a dot, meaning the current directory.
- 2. The options after the pathname always start with a hyphen (-). The action option part indicates what to do with the files. In this case, **-print** means to display them.



Don't forget the -print action key. Without it, find does not display any pathnames.

find Options

Table 8.8 shows a partial list of the find options and explanations of the options.

-name Option You use the search option to find a file by its name. You type -name followed by the desired filename. The filename can be a simple filename or you can use the shell wild card substitution: [], ?, and *. If you use these special characters, place the filename in single quotation marks. Let's look at some examples. The $\pm n$ notation in Table 8.8 is a decimal number that can be specified as +n (meaning more than n) or -n (meaning less than n) or n (meaning exactly n).



Find some files by name.

Table 8.8The **find** Command Search Options

Operator	Description	
-name filename	Finds files with the given filename.	
-size ±n	Finds files of the size n .	
-type file type	Finds files of the specified type.	
-atime ±n	Finds files that were accessed n days ago.	
-mtime ±n	Finds files that were modified n days ago.	
-newer filename	Finds files that were modified more recently than <i>filename</i> .	



- 1. In all of the preceding commands, the directory name is the current directory. The . (dot) represents the current directory.
- 2. The **-name** option identifies the filename. The wild card may be used to generate the filename.
- *3. The action part* **-print** *is used to display the name of the file found.*

-size $\pm n$ Option You use this search option to find a file by its size in blocks. You type -size followed by the number of blocks indicating the size of the file to be checked. The plus or minus sign before the number of blocks indicates greater than or less than, respectively. Let's look at some examples.



Find some files by size:

find . -name "*.c" -size 20 -print [Return] . . Find files that are exactly 20 blocks large.

This command finds all files that have filenames that end with .c and that are exactly 20 blocks large.

- \$ find . -name "*.c" -size +20 -print [Return] . . Find files that are larger than 20 blocks.
- \$ find . -name "*.c" -size -20 -print [Return] . . Find files that are smaller than 20 blocks.

-type Option You use this search option to find a file by its type. You type *-type* followed by a letter specifying the file type. The *file types* are as follows:

- b: a block special file (such as your disk)
- c: a character special file (such as your terminal)
- d: a directory file (such as your directories)
- f: an ordinary file (such as your files)



Find files using the *file type* option:

find \$HOME -type f -print [Return] . . Use the -type option.

This command finds all the ordinary files in your HOME directory and displays their pathnames. Recall, \$HOME holds the absolute pathname to your HOME directory.

-atime Option You use this search option to find a file by its last access date. You type -atime followed by the number of days since the file was last accessed. The plus or minus sign before the number of days indicates greater than or less than, respectively. Let's look at some examples.



Find files by their last access times:

\$ find . -atime 10 -print [Return] . . . Find and display files last accessed exactly 10 days ago.

The previous command displays the names of the files that have not been read for exactly 10 days.

- \$ find . -atime -10 -print [Return] . . . Find and display files last accessed less than 10 days ago.
- \$ find . -atime +10 -print [Return] . . . Find and display files last accessed more than 10 days ago.

-mtime Option You use this search option to find a file by its last modification date. You type **-mtime** followed by the number of days since the file was last modified. The plus or minus sign before the number of days indicates greater than or less than, respectively. Let's look at some examples.



Find files by their last modification times:

\$ find . -mtime 10 -print [Return] . . Find and display files last modified exactly 10 days ago.

The previous command displays the names of the files that are exactly 10 days old.

- \$ find . -mtime -10 -print [Return] . . Find and display files last modified less than 10 days ago.
- **find . -mtime +10 -print [Return] . .** Find and display files last modfied more than 10 days ago.

-newer Option You use this search option to find a file modified more recently than a file with a specified filename.



Let's look at an example:

find . -newer first.c -print [Return] Find and display files modified more recently than first.c was.

Action Options

The action options tell **find** what to do with files once they are found. Table 8.9 summarizes the three action options.

Table 8.9The **find** Command Action Options

Operator	Description	
-print	Prints the pathname for every file found.	
-exec command\;	Lets you give <i>commands</i> to be applied to the files.	
-ok <i>command</i> \;	Asks for confirmation before applying the command.	

-print Option The **-print** action option displays the pathnames of the files found that match the specified criterion.

Find pathnames to files called first.c, starting from the HOME directory.



The output shows there are three instances of the file first.c in David's hierarchy of directories.

-exec Option The **-exec** action option lets you give a command to be applied to the found files. You type **-exec** followed by the specified command, a space, backslash, and then a semicolon. You can use a set of braces ({}) to represent the name of the files found. An example will help to clarify the sequence.



Find and delete all instances of the first.c file that are 90 days old:

The search starts from your current directory (represented by .) and is continued through the hierarchy of the directories. The **find** command and the **-exec rm** command locate and the remove instances of first.c that are 90 days old.

Two search options are in effect: the **-name** and the **-mtime** options. That means **find** is seeking files that simultaneously satisfy these two search categories.



The command is comprised of many parts, and its syntax is peculiar:

- 1. The -exec option followed by the command (in this case rm)
- 2. A set of braces ({}) followed by a space
- 3. A backslash (\) followed by a semicolon

All instances of first.c are deleted. No warning or feedback message is displayed. When you see the \$ prompt, the job is done.

-ok Option The -ok action option is just like the -exec option, except that it asks for your confirmation before applying the command to the file.



Find and delete all instances of the first.c file, but ask for confirmation before deleting any file:

If a file, for example first.c, satisfies the criterion, then the following prompt is displayed:

```
<rm ... ./source/first.c> ?
```

If you reply with [Y] or [y], the command is executed (in this case, file.c is deleted); otherwise, your file remains intact.



You can also use the logical operators **or**, **and**, and **not** to combine the search options. The search starts from your current directory, and it is continued through the hierarchy of the directories.



It is highly recommended to use the **man** command to display the manual pages for the **find** command and its many options.

\$ man find [Return] Display man pages for find.

Under Linux, you can use the --help option to display the help page:

find --help [Return] . . . Displays the help page for find.

8.6.2 Displaying the Beginning of a File: The head Command

You can use the **head** command to display the beginning of a specified file. The **head** command gives you a quick way to check the first few lines of a file. For example, to display the first part of the file called MEMO in your current directory, you type the following:

\$ head MEMO [Return]

By default, head shows the first 10 lines of the specified file. You can override the default value by specifying the number of lines. For example, to display the first five lines of the file MEMO, you type the following:

\$ head -5 MEMO [Return]



The specified number of line must be a positive integer number.

You can specify more than one filename on the command line. For example, to show the first five lines of the files called MyFile, YourFile, and OurFile, you type the following:

\$ head MyFile YourFile OurFile [Return]

When more than one file is specified, the start of each file will look like:

```
==> filename <==
```



The following command sequences demonstrate the use of the **head** command.

<pre>\$ head -5 *File [Return]</pre>	Display the first five lines of all files in the current
\$ head * [Return]	directory ending with <i>File</i> Display the first 10 lines of all files in the current
\$ head -15 MEMO [Return]	directory Display the first 15 lines of the MEMO file.

head Options

Table 8.10 lists some of the **head** command options. For more detail, use the **man** command to read the man pages.

Table 8.10
The head Command Options

	Option	Operation
UNIX	Linux Alternative	
-l	lines	Counts by lines. This is the default option.
-c	chars=num	Counts by characters.
	help	Displays the help page and exits.
	version	Displays the version information and exits.

8.6.3 Displaying the End of a File: The tail Command

You can use the **tail** command to display the last part (the tail end) of a specified file. The **tail** command gives you a quick way to check the contents of a file. For example, to display the last part of the file called MEMO in your current directory, you type the following:

\$ tail MEMO [Return]

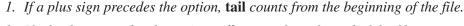
By default, **tail** shows the last 10 lines of the specified file. You can override the default value by using one of the available options.

tail Options

Table 8.11 lists some of the **tail** command options, which are similar to the options for the **head** command. When more than one filename is specified on the command line, each file starts with a header line similar to the following line:

==> filename <==

For more detail, use the **man** command and read the manual pages.



- 2. If a hyphen precedes the option, tail counts from the end of the file.
- 3. If a number precedes the option, tail uses that number instead of the default 10 lines count.

Table 8.11
The tail Command Options

	Option	Operation
UNIX	Linux Alternative	
-l	lines	Counts by lines. This is the default option.
-c	chars=num	Counts by characters.
	help	Displays the help page and exits.
	version	Displays the version information and exits.





The following command sequences demonstrate the use of the **tail** command:

```
$ tail MEMO [Return] . . . . Display the last 10 lines (no option).
$ tail 11 MEMO [Return] . . . Display the last 11 lines (11).
$ tail -4 MEMO [Return] . . . Display the last 4 lines (-4).
$ tail -10c MEMO [Return] . . . Display the last 10 characters (-10c).
$ tail +50 MEMO [Return] . . . Skip 50 lines from the beginning of the file MEMO and display the rest of the file.
```



You can specify only one filename as an argument with the tail command.

Practicing Linux Alternative tail Options



Use the Linux alternative options for the **tail** command, as suggested in the following command lines:

```
$ tail --chars=10 MEMO [Return] . Same as tail -10c MEMO.
$ tail --lines=5 MEMO [Return] . . . Same as tail -10l MEMO.
$ tail --version [Return] . . . . Display version information.
$ tail --help [Return] . . . . . Display the help page.
```

8.6.4 Selecting Portions of a File: The cut Command

You can use the **cut** command to "cut out" specific columns or fields from files. Many files are collections of records, with each record consisting of several fields. You might be interested in some of the fields or columns contained in a file. Figure 8.11 shows an example of such a file called **phones**. Each record in this file consists of five fields, and each field is separated by a space or a tab.

Figure 8.11 A File Called phones

```
$ cat phones
David Back (909) 999999 dave@xyz.edu
Daniel Knee (808) 888888 dan@xyz.edu
Gabe Smart (707) 777777 gabe@xyz.edu
$
```

cut Options

Let's suppose you have a file called phones that contains names, phone numbers, and E-mail addresses, similar to the file shown in Figure 8.11. Table 8.12 summarizes the **cut** command options. The following command sequences show the usage of **cut** and some of its options.

For the terminal sessions in this section, you need to create the phones file. Under the Chapter8 directory, create a subdirectory called 8.6. This is only a suggestion and you can create this file under any directory you wish. Now, use the vi or cat command to create the phones file similar to the file shown in Figure 8.11.

Table 8.12
The cut Command Options

	Option	Operation
UNIX	Linux Alternative	
-f	fields	Specifies the field position.
-c	characters	Specifies the character position.
-d	delimiter	Specifies the field separator (delimiter) character.
	help	Displays the help page and exits.
	version	Displays the version information and exits.

-f Option Following the **-f** option, a list of the fields is specified. Fields are assumed to be separated in the file by a delimiter character (the default is the tab character.) For example, **-f 1** indicates field one and **-f 1,7** indicates fields one and seven.



Use the **cut** command with the **-f** option to display the *first* field from the **phones** file.

\$ cut -f 1 phones [Return] . . . Show the first field in the phones file.
David Back
Daniel Knee
Gabe Smart
\$_ And the prompt appears.



Remember, the default field delimiter is the tab character.

Use the **cut** command with the **-f** option to display the *first* and *third* fields from the phones file.

-c Option Following the -c option, the character positions are specified. For example, -c 1-10 indicates the first 10 characters of each line.

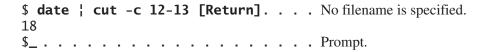


Display the first four characters of each line from the phones file.

\$ cut -c 1-4 phones [Return] . . Show columns 1 to 4.
Davi
Dani
Gabe
\$



Use the **cut** command with no filename:





- 1. The is a pipe operator that is used to pass the **date** output as input to the **cut** command. The pipe operator is explained in Chapter 9.
- 2. The output of the date command (date string) is passed to the cut command. The cut command displays the characters in positions 12 and 13 of the date string, which happen to be the hour field.

-d Option The character following **-d** is the field delimiter. The default character is tab. The space character or other characters with special meaning must be enclosed in double quotation marks. The delimiter is the character that separates the fields in your file.



Show one field using a space character as the field separator (delimiter):



If the field separator is a space character, make sure it is enclosed in quotation marks.



Practicing Linux Alternative cut Options

Use the Linux alternative options for the **cut** command, as suggested in the following command lines:



Read the help page and familiarize yourself with other options available for the cut command.

8.6.5 Joining Files: The paste Command

You can use the **paste** command to join files together line by line, or you can create new files by pasting together fields from two or more files. Figure 8.12 shows the output of the **paste** command if you have the two files called first and last in your current directory.

paste Options

Table 8.13 lists the **paste** command options. For the terminal sessions in this section, you need to create two files called first and last. Use the vi editor or the **cat** command to create these files, similar to the files in Figure 8.12. Remember, you only type in the names in the files, not the command line and the prompt sign.

Figure 8.12

The **paste** Command

```
$ cat first
David
Daniel
Gabriel
$ cat last
Back
Knee
Smart
$ paste first last
David
               Back
Daniel
               Knee
Gabriel
               Smart
$_
```

Table 8.13

The **paste** Command Options

Option		Operation
UNIX	Linux Alternative	
-d	delimiters	Specifies the field separator (delimiter) character.
	help	Displays the help page and exits.
	version	Displays the version information and exits.

-d Option The **-d** (delimiter) option specifies a specific delimiter character. The default is the tab character.



Use the **paste** command, indicating the : (colon) as the field separator.

```
$ paste -d : first last [Return] . . Use : as the field separator.
David:Back
Daniel:Knee
Gabriel:Smart
$_. . . . . . . . . . . . . . . . . The prompt is back.
```



Use the **paste** command, indicating the space character as the field separator.



If the delimiter character is the space character, make sure to enclose it in quotation marks.



Practicing Linux Alternative paste Options

Use the Linux alternative options for the **paste** command, as suggested in the following command lines:

8.6.6 Another Pager: The more Command

The **more** command is another pager provided for your convenience. Like **pg**, you can use **more** to browse or page through a text file. It pauses after each page (screen), and then it displays the word *More* and the percentage of the characters displayed so far at the bottom of the screen.

```
More-(11\%)
```



For screen continuity, **more** provides two lines of overlap between screens.

more Options

[Spacebar] Pressing the [Spacebar] moves the cursor ahead one screen.

[Return] Key Pressing the [Return] key scrolls ahead one line.

Q or **q** Key Pressing the [**q**] or [**Q**] key exits the **more** command.

Table 8.14 lists other options you can use when you issue the **more** command.

For the following terminal sessions, you need a large file to be able to practice all the options available for the **more** command. An easy way to create a large file is to use the **man** and redirection commands, instead of using vi to create a large file by typing pages and pages of text. Following is the command line to create a large file:

\$ man who > who [Return] . . Save who description in a file called who

Table 8.14The **more** Command Options

Option		Operation
UNIX	Linux Alternative	
-lines	-num <i>lines</i>	Displays the indicated number of lines per screen.
+line-number		Starts up at <i>line-number</i> .
+/pattern		Starts two lines above the line containing the <i>pattern</i> .
-c	-р	Clears the screen before displaying each page instead of scrolling. This is sometimes faster.
-d		Displays the prompt [Hit space to continue, Del to abort].
	help	Displays the help page and exits.

In this case, the description of the **who** command is redirected to a file called **who**. The choice of the **who** command and the name of the file are both arbitrary, and you can choose any command from **man** and redirect it to a file of any name you choose. But by using the example command line, you now have a file called **who** that is few pages long.



The following examples show the use of the **more** command line options:

```
* more -10 who [Return] . . . Display the who file with 10 lines per screen.
```

The previous command sets the number of lines per screen to 10 rather than the default that is number of lines in the terminal screen less two lines, usually 22 lines.

```
$ more +100 who [Return] . . . Start at line 100.
```

The previous command displays the who file one screenful at a time, starting at line number 100.

```
$ more +/User who [Return] . . . Start with the word User.
```

The previous command displays the who file one screenful at a time, starting two lines above the line containing the word User.

```
$ more -cd who [Return] . . . Display who in clear mode and . . . . . . . . . . . . . . . . display prompt.
```

As with other commands, more than one option can be used on a command line. In the previous example, the -c (clear screen) and -d (display prompt) options are used to display the who file.

While using the **more** command, the following keys can be used to further control the paging process.



Assuming you have issued the following command line:

```
$ more +100 -cd2 who [Return]
```

your screen would be similar to the following screen:

```
field is the name of the program executed by init as found in /sbin/inittab. The state, line, and
```

```
--More--(42%)[Hit space to continue, Del to abort]
```

The prompt line is the result of using the **-d** option, and the screen size is two lines, as was set by the **-2** option.

☐ Press the = key to see the current line.

At this point, the number of current line (101) is displayed, and your screen will be similar to the following:

field is the name of the program executed by init as found in /sbin/inittab. The state, line, and 101

☐ Press **h** to obtain a list of the available options.

At this point, the help screen is displayed and your screen will be similar to the following screen. To save space, a partial list of the commands is shown.

At this point, you can press the [Spacebar] to display the next screen of text, or press the [Return] key to display the next line of text. You use the [q] or [Q] key to exit from more.

8.6.7 Linux Pager: The less Command

The **less** command is yet another pager that is provided under Linux. This command is a program similar to **more**, but it allows backward movement in the file as well as forward movement. Also, **less** starts up faster with large input files than text editors like vi do. The **less** commands are based on both **more** and vi. The **less** command is much more sophisticated pager than **more**. Please use the **man** command or the **-?** and **--help** options to view the many available options for the **less** command.

```
$ less -? [Return] . . . . Display usage and list of options.
$ less --help [Return] . . . Display usage and list of options.
$ man less [Return] . . . Display manual pages.
```

8.7 UNIX INTERNALS: THE FILE SYSTEM

How does the UNIX file system keep track of your files? How does it know the location of your files on the disk? From your point of view, you create directories to organize your disk space, and directories and files have filenames to identify them. This hierarchical structure of directories and files is a logical view of the file system. Internally, however, UNIX organizes the disk and keeps track of files in a different manner.

The UNIX file system associates every filename with a number called its *i-node number* and identifies each file by its i-node number. UNIX keeps all these i-node numbers in a list, appropriately called an *i-node list*. The list is saved on the UNIX disk.

8.7.1 UNIX Disk Structure

Under UNIX, a disk is a standard block device, and a UNIX disk is divided into four blocks (regions):

- Boot block
- Super block
- · i-node list block
- · Files and directories block

The Boot Block The boot block holds the boot program, a special program that is activated at the system boot time.

The Super Block The *super block* contains information about the disk itself. This information includes the following:

- · Total number of disk blocks
- · Number of free blocks
- · Block size in bytes
- Number of used blocks

The i-list Block The *i-list block* keeps the list of i-nodes. Each entry in this list is an i-node, a 64-byte storage area. The i-node of a regular file or a directory file contains the location of its disk block(s). The i-node of a special file contains the information that identifies the peripheral device. An i-node also contains other information, including the following:

- File access permission (read, write, and execute)
- Owner and group IDs
- File link count
- Time of the last file modification
- Time of the last file access
- · Location of blocks for each regular file and directory file
- Device identification number for special files



The i-nodes are numbered sequentially.

i-nodes and Directories

i-node 2 contains the location of the block(s) that contains the root directory (/). A UNIX directory contains the list of filenames and their associated i-node numbers. When you create a directory, it automatically creates two entries, one for the . (dot dot) or parent directory and one for the . (dot) or child directory.



Filenames are stored in directories and not in the i-nodes.

8.7.2

Putting It Together

When you log in, UNIX reads the root directory (i-node 2) to find your HOME directory and saves your HOME directory i-node number. When you change your

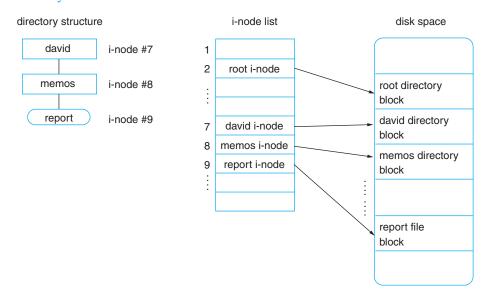
directory using **cd**, UNIX replaces this i-node number with the new directory's i-node number.

When you access a file using utilities or commands (such as **vi** or **cat**) or when a program opens a file, UNIX reads and searches the directory for the specified filename. There is an i-node associated with each filename that points to a specific i-node in the i-node list. UNIX uses your working directory i-node number to begin its search, or if you give a full pathname, it starts from the root directory, which always has the i-node 2.

Suppose your current directory is david, and you have a subdirectory called memos, and a file called report in memos, and you want to access report. UNIX starts searching from your current directory, david (with a known i-node number), and finds the filename memos and its i-node number. Next, it reads the memos i-node record from the i-node list. The memos i-node indicates the block that contains the memos directory.

Looking into the block that contains the filenames under memos, UNIX finds the report filename and its i-node number. UNIX repeats the preceding process, and reads the i-node record from the i-node list. Information in this record includes the location of the blocks on the disk that make the report file. (See Figure 8.13.)

Figure 8.13
The Directory Structure and i-node List



How do you determine what a file's i-node number is? You use the **ls** command with the **-i** option. For example, suppose your working directory is david and you have a subdirectory memos and a file called report in it.



List the filenames and their associated i-node numbers in your current directory.

```
$ ls -i
4311 memos
7446 report
$_
```



Make a copy of report, calling it report.old, and then show the i-node numbers.

```
$ cp report report.old
$ ls -i
4311     memos
7446     report
7431     report.old
$_
```

The new i-node number for report.old indicates that a new file has been created and a new i-node number is associated with it.



Move the report.old file to the memos directory, and then show the i-node numbers.

report.old is moved to the memos directory; its i-node number remains the same, but now it is associated with the memos directory.



Change the name of report.old in memos to report.sav.

The i-node number for report.sav remains the same as before; only the name associated with the i-node number is changed.



To link report to a new filename rpt (create another filename for report) and check the i-node changes after the two files are linked, do the following:

The i-node number for rpt, the new filename, is the same as report. Both the report i-node and the rpt i-node point to the same blocks that make up the report file.

COMMAND SUMMARY

The following commands and options have been discussed in this chapter.

CD

This command copies file(s) within the current directory or from one directory to another.

Option		Operation
UNIX	Linux Alternative	
-b	backup	Makes a backup of the specified file if file already exists.
-i	interactive	Asks for confirmation if the target file already exists.
-r	recursive	Copies directories to a new directory.
	verbose	Explain what is being done.
	help	Displays the help page and exits.

cut

This command is used to "cut out" specific columns or fields from a file.

Option		Operation
UNIX	Linux Alternative	
-f	fields	Specifies the field position.
-с	characters	Specifies the character position.
-d	delimiter	Specifies the field separator (delimiter) character.
	help	Displays the help page and exits.
	version	Displays the version information and exits.

find

This command locates files that match a given criterion in a hierarchy of directories. With the action options, you can instruct UNIX about what to do with the files once they are found.

Search Option	Description
-name filename	Finds files with the given filename.
size +n	Finds files with the size n .
-type file type	Finds files with the specified access mode.
-atime +n	Finds files that were accessed <i>n</i> days ago.
-mtime +n	Finds files that were modified n days ago.
-newer filename	Finds files that were modified more recently than <i>filename</i> .
Action Option	Description
-print	Prints the pathname for each file found.
-exec command\;	Lets you give <i>commands</i> to be applied to the files.
-ok <i>command</i> \;	Asks for confirmation before applying the command.

head

This command displays the first part of a specified file. This is a quick way to check the contents of a file. The number of lines to be displayed is an option, and more than one file can be specified on the command line.

Option		Operation
UNIX	Linux Alternative	
-l	lines	Counts by lines. This is the default option.
-c	chars=num	Counts by characters.
	help	Displays the help page and exits.
	version	Displays the version information and exits.

In

Creates links between an existing file and another filename or directory. It lets you have more than one name for a file.

more

This command displays files one screen at a time. This is useful for reading large files.

Option		Operation
UNIX	Linux Alternative	
-lines	-num <i>lines</i>	Displays the indicated number of <i>lines</i> per screen.
+line-number		Starts up at <i>line-number</i> .
+/pattern		Starts two lines above the line containing the <i>pattern</i> .
-с	-р	Clears the screen before displaying each page instead of scrolling. This is sometimes faster.
-d		Displays the prompt [Hit space to continue, Del to abort].
	help	Displays the help page and exits.

mν

This command renames files or moves files from one location to another.

Option		Operation
UNIX	Linux Alternative	
-b	backup	Makes a backup of the specified file if file already exists.
-i	interactive	Asks for confirmation if the target file already exists.
-f	force	Removes target file if file already exists and does not ask for confirmation.
-v	verbose	Explains what is being done.
	help	Displays the help page and exits.
	version	Displays the version information and exits.

paste

This command is used to join files together line by line, or to create new files by pasting together fields from two or more files.

Option		Operation
UNIX	Linux Alternative	
-d	delimiters	Specifies the field separator (delimiter) character.
	help	Displays the help page and exits.
	version	Displays the version information and exits.

pg

This comand displays files one screen at a time. You can enter the options or other commands when pg shows the prompt sign.

Option	Operation	
-n	Does not require [Return] to complete the single-letter commands.	
-S	Displays messages and prompts in reverse video.	
-num	Sets the number of lines per screen to the integer <i>num</i> . The default value is 23 lines.	
-pstr	Changes the prompt: (colon) to the string specified as str.	
+line-num	num Starts displaying the file from the line specified in line–num.	
+lpattern	Starts viewing at the line containing the first occurrence of the specified <i>pattern</i> .	

The pg Command Key Operators

These keys are used when **pg** displays the prompt sign.

Key	Operation
+ <i>n</i>	Advances n screens, where n is an integer number.
-n	Backs up n screens, where n is an integer number.
+nl	Advances n lines, where n is an integer number.
-nl	Backs up n lines, where n is an integer number.
n	Goes to screen n , where n is an integer number.

pr

This command provides formatted files before printing or viewing it on the screen.

Option		Operation
UNIX	Linux Alternative	
+page	pages=page	Starts displaying from the specified <i>page</i> . The default is page 1.
-columns	columns=columns	Displays output in the specified number of <i>columns</i> . The default is one column.
-a	across	Displays output in columns across (rather than down) the page, one line per column.
-d	double-space	Displays output double-spaced.
-hstring	header=string	Replaces the filename in the header with the specified <i>string</i> .
-lnumber	length= <i>number</i>	Sets the page length to the specified <i>number</i> of lines. The default is 66 lines.
-m	merge	Displays all the specified files in multiple columns.
-p		Pauses at the end of each page and sounds the bell.
-character	separator=character	Separates columns with a single specified <i>character</i> . If character is not specified, then [Tab] is used.
-t	omit-header	Suppresses the five-line header and five-line trailer.
-w number	width=number	Sets line width to the specified number of characters. The default is 72.
	help	Displays help page and exits.
	version	Displays version information and exits.

wc

This command counts number of characters, words, or lines in the specified file.

Option		Operation
UNIX	Linux Alternative	
-l	lines	Reports the number of lines.
-w	words	Reports the number of words.
-с	chars	Reports the number of characters.
	help	Displays the help page and exits.
	version	Displays version information and exits.

tail

This command displays the last part (tail end) of a specified file. This is a quick way to check the contents of a file. The options give the flexibility to specify a desired part of the file.

Option		Operation
UNIX	Linux Alternative	
-l	lines	Counts by lines. This is the default option.
-c	chars=num	Counts by characters.
	help	Displays the help page and exits.
	version	Displays the version information and exits.

REVIEW EXERCISES

- 1. What are the symbols used for the redirection operators?
- 2. Explain input and output redirection.
- **3.** What are the commands used to read a file?
- **4.** What is the difference between moving (**mv**) a file and copying (**cp**) a file?
- **5.** What is the command to rename a file?
- **6.** Is it possible to have more than one name for a file?
- 7. What are the four regions (blocks) of a UNIX disk? Explain each part.
- **8.** What is an i-node number, and how is it used to locate a file?
- **9.** What is the i-node list, and what is the major information stored in each node?
- **10.** Which one of the following commands changes or creates an i-node number?
 - a. mv file1 file2
 - b. cp file1 file2
 - c. In file1 file2
- 11. What is the command to locate a file and remove it once it is found?
- 12. What is the command to look at the last 10 lines of a file?
- **13.** What is the command to select specified fields from a file?
- **14.** What is the command to put files together line by line?
- **15.** What is the command to read a file one screen at a time?
- **16.** What is the command line to list all the filenames that start with *start*?
- 17. What is the command line to list all the filenames that end with end?
- **18.** What is the command line to delete all the files that have *mid* in their filename?
- **19.** What is the command line to copy all the files that have filenames that start with the letter *A* or *a* from your current directory a directory to called Keep?

20. What is the command line to copy all the files that have filenames that start with the letters *A* or *a* and end with the letters *Z* or *z* from your current directory to a directory called Keep?

Match the commands shown in the left column with the explanations shown in the right.

- 1. wc xxx yyy
- a. Copy xxx to yyy.
- 2. **cp xxx yyy**
- b. Rename xxx to yyy.
- 3. ln xxx yyy
- c. Copy all filenames that begin with *file* and have exactly two characters (any characters) after it.
- 4. mv xxx yyy
- d. Delete all files in the current directory.

5. rm *

- e. Create another filename for xxx; call it yyy.
- 6. ls *[1-6]
- f. Display the contents of myfile.
- 7. cp file?? source
- g. Copy myfile to yyy.
- 8. pr -2 myfile
- h. Add all files that have exactly one character before the word file into one file called yyy.

9. **ls -i**

- i. Format myfile in two columns.
- 10. **pg myfile**
- j. List all files having filenames ending with digits 1 to 6.
- 11. cat myfile
- k. List the current directory filenames and their i-node numbers.
- 12. cat myfile > yyy
- 1. Create a file called yyy that contains the count of characters in file xxx.
- 13. **cat** ?file >> yyy
- m. View myfile one screen at a time.
- 14. find . -name "file*" -print
- n. Save the second field from the xyz file in a file called xxx.
- 15. find . -name xyz -size 20 -print
- o. Read zzz one screenful at a time.
- 16. cut f2 xyz > xxx
- p. Find all files that are called xyz and are exactly 20 blocks large.
- 17. more zzz
- q. Find all files whose name starts with *file*.



Terminal Session

In this terminal session, you practice the commands discussed in this chapter by creating directories and then manipulating files in the directories.

- 1. Create a directory called memos in your HOME directory.
- 2. Using the vi editor, create a file called myfile in your HOME directory.
- 3. Using the **cat** command, append myfile a few times to create a large file (say, 10 pages). Call this file large.
- **4.** Using the **pg** command and its options, view large on the screen.
- 5. Using the **pr** command and its options, format large and print it.
- **6.** Use **cp** command to copy all files in your HOME directory to the memos directory.

- 7. Use the **ln** command to create another name for large.
- 8. Using the mv command, change the name of large to large.old.
- **9.** Using the **mv** command, move large old to memos.
- **10.** Use the **ls** command and the **-i** and **-l** options to observe the changes in the i-node numbers and the number of links when you do the following commands:
 - a. Change to the memos directory.
 - **b.** Create another name for myfirst; call it MF.
 - c. Copy myfirst to myfirst.old.
 - **d.** List all files whose filenames start with my.
 - e. List all files that have the extension old.
 - f. Modify myfile. Look at the MF file; myfile modifications are also in the MF file.
 - **g.** Change to your HOME directory.
 - **h.** Delete all files in the memos directory that have the word *file* as part of their file names.
 - i. Delete the memos directory with all of the files remaining in it.
 - **j.** List your HOME directory.
 - **k.** Remove all the files you created in this session.
- 11. Show the last five lines of a file.
- **12.** Display the first five lines of a file.
- 13. Save the last 30 characters of a file in another file.
- **14.** Save the list of all the files that are seven days old, starting from your HOME directory.
- **15.** Find a file called passwd.
- **16.** Find a file called profile.

numbers

- **17.** Find all files that are exactly seven days old, starting from your HOME directory.
- 18. Find all files that are less than seven days old, starting from your HOME directory.
- **19.** Find all the files that are more than 10 days old and copy them to another directory.
- **20.** Create two files similar to those in Figures 8.11 and 8.12.
 - **a.** Use the **cut** command to cut out specific fields and columns.
 - **b.** Use the **paste** command to put the two files together.
- **21.** Use the **more** command to read a large file.
- **22.** Create two files called **numbers** and **characters** and type in lines of numbers and characters similar to the following:

characters

Hullibers	character 5
111111111111	AAAAAAAAA
2222222222	BBBBBBBBBB
33333333333	CCCCCCCCC
101010101010 11 11 11 11 11	KKKKKKKKKKK LLLLLLLLLLL

Make many lines of numbers so you can practice the **head** and **tail** commands and options.

- a. Show the first two lines of the numbers file.
- **b.** Show the first two lines of numbers and characters files.
- **c.** Show the last 10 lines of the numbers file.
- **d.** Show the last two lines of the numbers and characters files.
- **23.** Use the numbers and characters files to execute the following command lines:
 - **a.** Use the **paste** command to put together **numbers** and **characters**.
 - **b.** Use the **paste** command to show numbers and characters together and use the @ character as the field separator.
 - **c.** Use the **paste** command to show **numbers** and **characters** together using the @ character as the field separator. Save it in a file called **numbersANDcharacters**.
 - **d.** Use the **cut** command to show first five character of each line in numbers.
 - **e.** Show how many lines are in the numbersANDcharacters file.
 - **f.** Show how many lines, words, and characters are in the numbers, characters, and numbersANDcharacters files.