chapter 4

TOP TEN MOST USEFUL UNIX COMMANDS

In this chapter we discuss the following commands:

Unix Command	DOS Command	Description
cd	CD	change to another directory
clear	CLS	clear the screen
cp	COPY	copy a file
grep	FIND	search for a pattern in a file
lpr	PRINT	print a file
ls	DIR	list the files in a directory
mkdir	MKDIR	create a directory
more	MORE	display a file one screenful at a time
mv	RENAME	rename a file
pwd	CHDIR	print the name of the current working directory
rm	DEL	delete a file
rmdir	RMDIR	delete a directory

4.1 Introduction

There are a number of fundamental commands in Unix that every user needs to learn right away. In Chapter 2 we covered the most important commands relating to accounts: **login**, **logout**, and **passwd**; in Chapter 3 we covered the most important commands relating to the Unix Reference Manual: **apropos**, **man**, and **whatis**.

In this chapter we cover a number of very important commands relating to files and directories. These commands handle many of the routine functions that a typical user performs on a daily basis. We describe the most common features of

each of these commands here. Later in the book, we will delve into more specialized uses of the commands. Having mastered these ten[1] commands, you will be able to work on a Unix system and perform essential functions such as

- list the files in a directory.
- display command output one screenful at a time.
- change to another directory.
- search for a pattern in a file.
- print a file.
- create a directory to store files in.
- copy a file.
- rename a file.
- delete a file.
- delete a directory.
- clear the screen.

We cover the "top ten" commands in the order you will probably encounter them.

4.2 Listing Your Files—ls Command

The **ls** command name is an abbreviation for "list." It is used to list your directories and files. Suppose that you have just logged onto a Unix system. You are now in a position to do some useful work. Initially, you will encounter the Unix prompt

```
%
```

When you log into your account, the system by default places you in an area known as your *home directory*. This initial file space contains some very important files and is your home base from which you will begin working. (On some operating systems, directories are called *folders*.)

In the Unix operating system, directories and files are organized into a tree-like hierarchy. Figure 4.1 depicts such a sample structure graphically. It is worth going over some basic definitions about *trees* since the terminology associated with them has been adopted by Unix, and we make use of it throughout this book.

In Figure 4.1 the circles represent *nodes* and the lines between the nodes represent *edges*. The tree is oriented so the part highest up on the page is referred to as the *top*; the other end of the tree is referred to as the *bottom*. The node at the top of the tree, in this case A, is called the *root*. If there is an edge between two

1. All right, twelve.

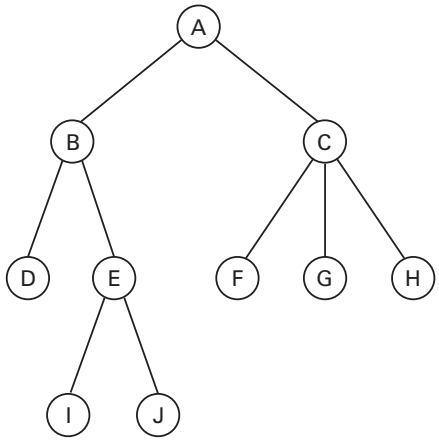

Figure 4.1—A Tree

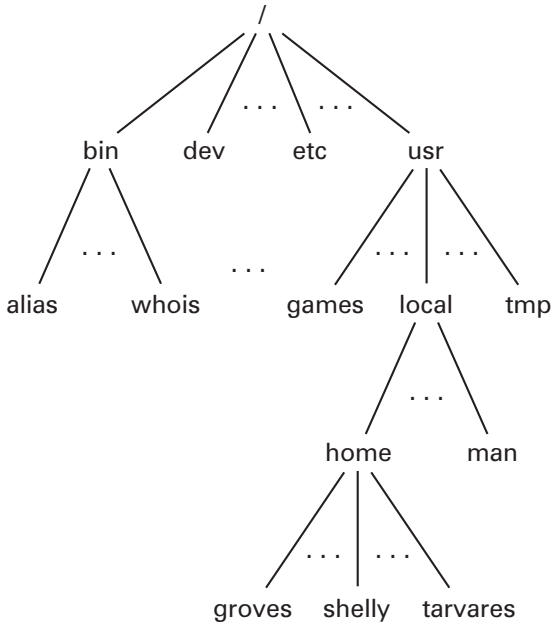

Figure 4.2—A Simplified Representation of the Unix File System

nodes, we say the nodes are *adjacent*. For example, nodes C and F are adjacent whereas nodes F and G are not. The *children* of a node are the nodes adjacent to it and also below it in the picture. For example, D and E are the two children of B. A *parent* of a node is the node that is above it and adjacent to it. For example,

C is the parent of F, G, and H. A *path* is a sequence of adjacent nodes in the tree. For example, A-C-F is a path, whereas D-A-C-H is not. We are usually interested in paths that go down the tree.

In Figure 4.2 we depict a sample Unix file space in the form of a tree. We do not include circles for nodes in the tree, but instead just write in a directory or file name to represent a node. In the figure only **alias** and **whois** represent files; the other nodes represent directories. The directory at the top of the tree is known as the *root directory* and is represented by the / (forward slash) character.

Suppose A and B are directories in a file space represented by a tree T. We say B is a *subdirectory* of A, if there is a path from A to B going down T. So, if all the nodes in Figure 4.1 represent directories, then B and C would be subdirectories of A. In addition, F, G, and H are subdirectories of C.

Some standard subdirectories under[2] the root directory on a large Unix system are **bin**, **dev**, **etc**, and **usr**. Unix users rely on the words "up" and "down" to indicate relative positions in a directory hierarchy. For example, **shelly** is down one from **home** and **home** is up one from **shelly**. That is, **home** is the parent directory of **shelly**. You will hear expressions such as "move down two directories" or "go to the parent directory."

When you initially log in to your account, by default you are placed in your home directory.

Starting from the root directory and proceeding through the tree until you reach your home directory results in a path to your home directory. For example, **usr-local-home-groves** is a path to Brian Groves' home directory. When the directory names you pass through are concatenated together, the result is a *pathname*. The pathname for Brian's home directory is

```
/usr/local/home/groves
```

Pathnames can be *full* or *relative*. A full pathname specifies a complete path through the directory structure starting at the root directory, whereas a relative pathname specifies a path relative to some starting position. For example, we have **/bin/whois** as a full pathname and **local/home/tarvares** as a relative pathname. We will explore the Unix file and directory structure further in Chapters 10–11.

The first forward slash in a pathname represents the root directory. Additional forward slashes in a pathname separate the names of subdirectories. Figure 4.3 shows John Tarvares' directory structure. We will use his account as a model to describe concepts throughout this chapter.

2. The word "under" is used because of the physical relationship shown in Figure 4.2.

ABBOTT, **classes**, **HTML**, and **misc** are child subdirectories of **tarvares**. The file **bud** is contained in the directory **ABBOTT**. There are three files in the directory **HTML**.

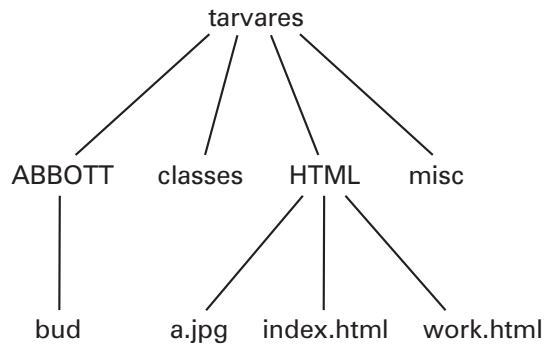

Figure 4.3—John Tarvares' Directory and File Structure

If John enters the command line

```
%ls
```

the directories in his home directory will be listed as shown below.

```
ABBOTT      classes
HTML        misc
```

The **ls** command lists files and directories contained in the directory where the **ls** command is executed from. When you first log in, this will be from your home directory. Later in this chapter we describe how to change directories. If you change to another directory, say a directory called **HTML**, and enter the **ls** command, the files in the directory **HTML** will be listed. In John's case this results in a listing of the files **a.jpg**, **index.html**, and **work.html**.

The *current working directory*, sometimes called the *working directory* for short, refers to the directory you are in. Unix provides the **pwd** command to display the full pathname of the current working directory, allowing you to find out which directory you are in. After moving up and down the directory tree a large number of times, it is easy to lose track of where you are.

If John enters the command

```
%pwd
```

from his home directory the result is

```
/usr/local/home/tarvares
```

assuming he is the same Tarvares as shown in Figure 4.2.

Next we describe two important flags to the **ls** command. As you have seen from the examples of **ls** presented above, by executing the **ls** command we learn only about what files and directories are in the current working directory. We do not actually get any details about the files themselves, such as when they were created or how many bytes they are. To obtain detailed information about the files, the **–l** flag may be used. The **–l** (the letter l, not the number 1) stands for "long" form. A sample output for the command

```
%ls -l
```

is shown in Figure 4.4. Notice that a lot of information about a file is displayed. We explain all this information in detail in Chapter 11. For now, remember that column 1 pertains to file permissions, column 5 to file size in number of bytes, and columns 6–8 to the last modification date of the file.

```
total 7060
drwx------  2  tweed  faculty      512  Aug  18   1998  SONGS
-rw-------  1  tweed  faculty     5204  Apr  28   1995  atrail.tex
-rw-------  1  tweed  faculty     4048  Jun  28   1996  comrades
-rw-------  1  tweed  faculty     1638  Aug   5   1994  ep15.tex
-rw-------  1  tweed  faculty     3292  Jul  19   1994  highs.tex
-rw-------  1  tweed  faculty     2557  Oct  29  13:43  iron.txt
-rw-------  1  tweed  faculty      299  Oct   1  13:08  labels.tex
-rw-------  1  tweed  faculty      312  Oct   1  13:10  labels2.tex
-rw-------  1  tweed  faculty      518  Feb   1  16:32  loop.aasu
-rw-------  1  tweed  faculty  3579380  Feb  24  18:46  bullalgs
drwx------  2  tweed  faculty     1024  Dec   6  21:11  recs
-rw-------  1  tweed  faculty     1869  Jun   8   1998  rent.sav
```

Figure 4.4—Sample Output from an **ls –l** Command

To list *all* of the files and directories contained in a directory, you use the **–a** flag to the **ls** command. That is,

```
%ls -a
```

lists all files in the current working directory. Unix directories contain special *hidden files.* By using the **–a** flag to the **ls** command, you are able to view these files as well. We cover hidden files in Section 10.3. A sample output of the **ls –a** command is shown in Figure 4.5. Notice several hidden files are displayed, among others, **.**, **..**, **.cshrc**, **.login**, and **.netscape**. Every directory always contains the **.** and **..** hidden files. These refer to the current working directory (the directory itself) and the directory's parent directory.

```
.                       africa
..                      albany
.addressbook            february
.cshrc                  fish
.history                fritos
.hotjava                moneymatters
.login                  monkey
.logout                 zebra
.netscape
```

Figure 4.5—Sample Output from an **ls –a** Command

You may combine the different options to Unix commands. For example, the command

```
%ls -la
```

lists all material contained in the current working directory in the long form. This produces the same result as entering

```
%ls -al
```

The order of the specified options does not matter; they are both applied.

We will describe one final use of the **ls** command. There are times when you want to copy a file from another directory to the current working directory. You may remember the pathname of the directory the file is stored in, but you may not remember the name of the file itself. You can list the contents of this other directory by specifying its name as an argument to the **ls** command.

For example, suppose you are in the directory called

```
/export/local/home/riddle
```

and that you would like to copy a file from the directory

```
/export/local/home/messner/climbs/public
```

to your directory but cannot remember the name of the file. The command

```
%ls /export/local/home/messner/climbs/public
```

executed from your directory will provide you with a list of files in the other directory. Once you locate the name of the file, you can proceed to copy it using the technique described in Section 4.8.

You should execute the command line

```
%man ls
```

to learn more about the **ls** command.

Exercises

1. Execute the **ls** command from your home directory. How many files do you have? How many directories? Now execute the command **ls –a**. How many items are listed? How many hidden files are there?
2. How many options to the **ls** command are there on your system? Describe two interesting options different from those covered in this section.
3. What is the full pathname for your home directory?

4.3 Displaying a File—more Command

In Chapter 2 you first encountered the **more** command. The **more** command provides a convenient way to view the contents of a file one screenful at a time. For example, entering the command

```
%more index.html
```

displays one screenful of content of the file **index.html**. Hitting the **Spacebar** brings up the next screenful of text, and typing **q** "quits" the **more** command and brings you back to the Unix prompt. The **more** command only allows you to view the file. To alter the file's contents you need to use a text editor.

Try entering the command

```
%man man
```

on your system. The Unix Reference Manual documentation about the **man** command consists of more than one screenful of information. In the lower left of the screen, the **more** command tells you what percentage of the file has *already* been displayed. Thus, a display such as

```
---More---(13%)
```

indicates you have seen 13% of the file, so there is another 87% of the file that has yet to be displayed. The percentages are very helpful and let you make a mental note of where you are in the file. For example, you may recall that you had seen some important information that was 47% of the way through a file and then be able to easily return to the information. On most Unix systems, you can press **b** while viewing a file with **more** and you will be returned to the preceding screen of information or remain at the first screen if you have not moved forward in the file.

The **more** command also provides you with a mechanism for searching for a user-specified pattern of characters in a file. To execute a search in a file being displayed by **more**, you simply type **/** and then the pattern you are looking for. Suppose you were looking for the pattern **sailboat**. You would type

```
/sailboat
```

The **more** program would then search forward from where you currently are in the file and highlight the first occurrence of the word **sailboat** it found. If the pattern were not present in the file, **more** would indicate that the pattern was not found. To locate subsequent occurrences of a pattern you just entered, you need only type `/` and press **Enter**.

Another convenient feature of **more** is the **–s** option. This option tells **more** to squeeze consecutive blank lines into a single blank line. In this way additional information can be displayed on the screen. So, for example, to display the file **data.txt** with extra blank lines squeezed out of it, you would enter the command

```
%more -s data.txt
```

The **more** command has a number of other interesting features. You should execute the command line

```
%man more
```

to learn additional information about it.

Two other Unix programs for displaying files are **less** and **pg**. Some users prefer **less** over **more** because **less** allows you to scroll both down and up. The command name **less** was chosen sarcastically; in fact, **less** provides greater functionality than **more**. You can do a **man** on **less** and **pg** to find out how to use them and how they differ from **more**.

Exercises

1. Can the **more** command take several arguments? If so, what is the result?
2. Perform a **man more** command. How many times does the pattern "manual" occur in the **man page** for **more**? How many times does the pattern "Manual" occur? Is the searching done within the **more** command case sensitive?
3. Are the commands **less** and **pg** available on your system? Compare and contrast them with the **more** command.

4.4 Changing Directory—cd Command

The Unix file system is arranged into a hierarchy of directories conveniently represented by a tree. As you organize your work, you will need to be able to navigate through the tree. To move to another directory, you use the **cd** command, short for "change directory." The **cd** command by itself with no arguments will place you in your home directory—regardless of which directory your current working directory is.

To verify a change of directory, you can use the **pwd** command. The output of the **pwd** command is the full pathname of the directory you are in. Suppose Jenny Shelly just logged into her account with userid **shelly**. Figure 4.2 illustrates the location of Jenny's file space graphically. If Jenny executed the command

```
%pwd
```

for output she would see

```
/usr/local/home/shelly
```

By specifying a pathname as the argument to the **cd** command, you can change to other directories. For example, consider John Tarvares' file space depicted in Figure 4.6. To transfer into his Web directory (**HTML**), John can enter

```
%cd /usr/local/home/tarvares/HTML
```

from any other directory in the Unix file system. Notice that John has specified a full pathname. Typing in a full pathname every time you want to change directories is time consuming; it is often more efficient to use a relative pathname. For example, from his home directory, **tavares**, John could have typed

```
%cd HTML
```

to achieve the same result. That is, relative to his home directory, the directory **HTML** is one level down. The **cd** command followed by an explicit directory name will take you to that directory if it is a child subdirectory of the current working directory. In other words, to move to a child directory called **childsubdirectory** from within its parent directory, you simply enter

```
%cd childsubdirectory
```

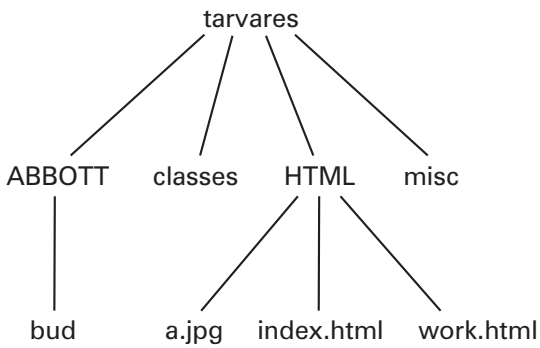

Figure 4.6—John Tarvares' File Space

To move up one level in a directory hierarchy, follow the **cd** command with two dots

```
%cd ..
```

The two dots represent the *parent directory* of where you are currently located. Using relative path names when navigating through the file structure can save a lot of typing time. To move from within his **HTML** directory to his **misc** directory, John can enter

```
%cd ../misc
```

Recall that Unix is case sensitive. Thus, it is important to type directory names exactly as they appear.

The tilde (~) symbol is used to refer to your home directory. For example,

```
~tarvares
```

is expanded automatically to

```
/usr/local/home/tarvares
```

The tilde character can prove very useful for moving around a directory structure. As an example, suppose your current working directory is

```
/usr/local/home/tarvares/brown/bags/computers
```

and you would like your current working directory to be

```
/usr/local/home/tarvares/pool/tables/balls
```

The command

```
%cd ~/pool/tables/balls
```

can be used to accomplish this change of directories. Contrast this with other methods for moving into this directory, which require considerably more typing.

You should execute the command line

```
%man cd
```

to learn more about the **cd** command.

Exercises

1. Consider the directory and file structure shown in Figure 4.7. Suppose you were initially located in the **food** directory. Provide the **cd** commands that require the least number of characters to be typed to perform the following tasks:
   a. move to the **omelette** directory
   b. from the **omelette** directory move to the **cereal** directory

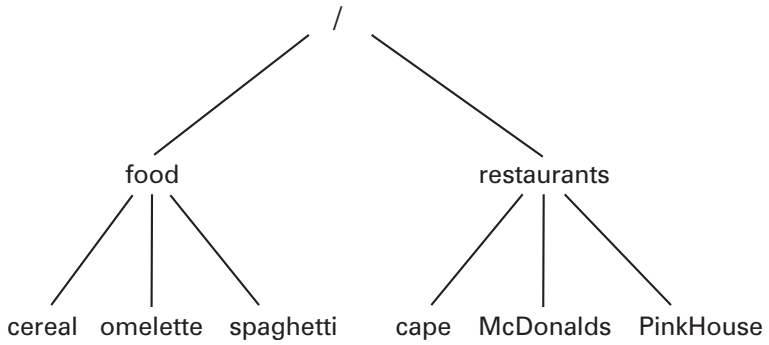

Figure 4.7—Directory and File Structure Used in the Exercises

c. from the **cereal** directory move to the **cape** directory
d. from the **cape** directory move to the **omelette** directory
e. return to the home directory

2. What is the effect of the command **cd .**?
3. What are the child subdirectories of the root directory on your system?
4. Are there any interesting flags to the **cd** command? If so, describe two of them.

4.5 Searching for a Pattern—grep Command

There will be many times when you want to search a file for a pattern. It would be convenient if you could do this without using a text editor. If you could perform such a search from the operating system, you would not have to go through the usual steps of opening the file with a text editor, using the editor's search facility, and then closing the file. Unix provides the **grep** command for searching a file for a pattern. In fact, **grep** allows you to search an entire directory of files or even an entire file system for a pattern. The **grep** command is a powerful search mechanism that provides a convenient notation, allowing you to specify complex patterns to search for.

The word **grep** is an acronym for "global regular expression print." *Regular expressions* are an important concept in computer science. You can think of them as a means to compactly express patterns. We consider a couple of examples involving the use of the **grep** command.

Suppose you are residing in a directory that yields the following display when you execute an **ls** command:

```
annotation.tex     california.tex     game.c        words.favorites
buffalo.txt        denver.txt         golfing.txt
```

To search the file **words.favorites** for the word **zooks**, you enter the command

```
%grep zooks words.favorites
```

The output of this command is a display of all lines in the file **words.favorites** that contain the word **zooks**. In this case, the word appeared twice and the following was displayed as output:

```
yikes—an exclamation, also see zooks.
zooks—an exclamation, also see yikes.
```

If the word were not present in the file, you would have been returned to the Unix prompt without seeing any output.

This example illustrates that the first argument to the **grep** command is the pattern you are looking for, and the second argument specifies the file(s) to search for the pattern in. The pattern can be specified using a regular expression. We will look at various examples of specifying patterns below, and we will also consider several ways of specifying files to search in.

To search for the word **zooks** in the files **games.c** and **words.favorites** simultaneously, you could enter the following command:

```
%grep zooks games.c words.favorites
```

The output in this case is the same as before, since in our case the word **zooks** does not appear in the file **games.c**.

The **grep** command has been optimized and it searches very quickly even when you ask it to look in many large files at once. Suppose you had used the word **elephant** in one of the files in the directory under consideration but could not remember which file. You can use the **grep** command as follows to locate the desired file:

```
%grep elephant *
```

The asterisk (*****) serves as a wildcard. In this context it tells the **grep** command to search all files in the directory for the pattern **elephant**. That is, ***** means match all file names. The **grep** command will search all the files in the directory for the word **elephant**. The output of the command is the name of each file where the pattern was found followed by the lines in the file where the pattern occurred.

The *file extension* is the part of a file name occurring after the last period. For example, the file extension of **main.tex** is **tex**. If a file has an extension of **abc**, we refer to it as an **abc** file. For example, **main**.**tex** is called a **tex** (pronounced "tech") file.

Suppose you wanted to search only files with an extension of **tex** for the pattern **Peter the Great**. This could be accomplished by the following command:

```
%grep 'Peter the Great' *.tex
```

There are two things to notice about this command. Since the pattern we are looking for contains blank spaces, we put the pattern in single quotes. Otherwise, **grep** would interpret part of the pattern as a file name to search. This would result in an error message since there is no file in our directory called **the**.

The second thing to notice is how we were able to specify all files ending with the **tex** extension using ***.tex**. The star means "match any pattern" and the **.tex** means the file name must end with these four characters: **.**, **t**, **e**, and **x** in order.

If you executed the command **ls *.tex** on this directory, you would see the following output:

```
annotation.txt      california.tex
```

Thus, the **grep** command

```
%grep 'Peter the Great' *.tex
```

searches both **annotation.tex** and **california.tex** for the pattern **Peter the Great**.

The **grep** command provides notation for efficiently specifying patterns. We have already seen that the ***** means "match any pattern." The **.** is used to match any single character. The Unix regular expression

```
a.b.c
```

means match any pattern that consists of five characters, where the first character is an **a**, the third character is a **b**, the fifth character is a **c**, and characters two and four can be any single symbols. For example, the pattern **atbvc** meets these criteria as does the pattern **a$bZc**.

If you type in a character that has a special meaning to **grep**, for example, ***** or **.**, you should *escape* the character with the \ symbol. This tells **grep** that you want the character to be interpreted literally so that its special meaning is disregarded. For example, suppose you wanted to look for the pattern **a.b.c**. That is, you wanted to find the five characters **a**, **.**, **b**, **.**, and **c** in this order. The following command would search all files with a **txt** extension for this pattern:

```
%grep a\.b\.c *.txt
```

Notice that we have escaped the two dots so that they are matched exactly rather than telling the **grep** command to match any two characters.

The command line

```
%grep '[A-Z]\.' *.tex
```

finds all lines in all files with the extension **tex** that contain a capital letter followed by a period. You can specify a range of characters to **grep** by displaying them

in square brackets with a dash in between. A range of lowercase letters may be specified similarly. For example, **[d–g]** is used to match one of the characters **d**, **e**, **f**, and **g**.

There are many other useful ways of specifying patterns to **grep**. You should execute the command line

```
%man grep
```

to learn more about the **grep** command.

Exercises

1. Write **grep** expressions to search the file named **computers** for the following patterns:
   a. the word **personal**
   b. the phrase **personal computer**
   c. the phrase **Personal Computer**
2. On many Unix systems there is a file called **/usr/words/dict**. This file contains a long list of words used by spell-checking programs. Write a **grep** expression to search this file for any words that contain all the vowels in consecutive order. That is, you are looking for words that contain the letters **a**, **e**, **i**, **o**, and **u** in this order. There can be other letters interspersed between the vowels. What words did you find?
3. You will need to read the **man page** for **grep** to complete this exercise. Write **grep** expressions to search all files in a directory for the following patterns:
   a. any line that begins with a capital letter
   b. any line that ends with a capital letter
   c. a pattern consisting of three vowels in a row
   d. the pattern **'a(b)..&*'**, where the quotes *are* part of the pattern you want to find

4.6 Printing—lpr Command

It is very important to be able to print files from a computer system. Unix provides the command **lpr** for this purpose. The command name **lpr** is an abbreviation for "line printer." On many systems a default printer will be set up for you to use. That is, if you send a file to the printer, the default printer will be the physical printer that actually outputs your file. If a default printer has been set up, you can print the file **banner** from your current working directory using the command line

```
%lpr banner
```

Suppose you want to print the same file to a printer named **laser** rather than to the default printer. You would enter the following command line:

```
%lpr -Plaser banner
```

On some systems you would enter

```
%lp -D laser banner
```

or

```
%lp -dlaser banner
```

The **–d** and **–D** flags stand for "destination." The argument following these flags tells the system the name of the printer on which to print your file. You will need to check to see if your system uses **lpr** or **lp**, or some other print command.

It is important to send the correct file types to the printer. If you send the wrong type of file, the output may be nonsense; you could waste a lot of paper; or you could jam the printer. Today many printers handle *plain text* and *PostScript* files. You should try to learn what formats your local printer can handle.

Many printers do not handle **dvi** files properly. The LaTeX documentation preparation system that we cover in Appendix H generates **dvi** files as output. It is not a good idea to send a **dvi** file directly to a printer. In Appendix H we will explain how to print **dvi** files properly.

There are many options to the **lpr** command that we have not covered. You should execute the command line

```
%man lpr
```

to learn more about it. When you do, you will notice many related commands such as **lprm** and **lpq**. We cover these commands in Chapter 9.

On many systems you have to acquire some local knowledge to be able to print effectively. For example, you will need to obtain the names of the local printers, information about how to process various types of files, information about printing quotas, and which print commands are available. This information is usually posted near the printers or online. Other users or the system administrator will usually be happy to share printing information with you.

Exercises

1. On your Unix system, what is the command line for printing the file **homework** on the default printer? Is more than one printer available to you? What would the command line be for sending the file **homework** to a printer named **laserwriter**?

2. You will need to read the **man page** for **lpr** or **lp** to complete this exercise. How do you print five copies of a document without repeating the print command five times?
3. What does the word *duplex* mean? Can you print in a duplex style? If so, what is the command for doing this?
4. Are there printing quotas on your system? If so, describe them and explain how they are enforced.

4.7 Creating a Directory—mkdir Command

In order to properly organize your work, you will want to be able to create subdirectories. To create a subdirectory, you use the **mkdir** command. The command name is an abbreviation for "make directory." You supply the subdirectory name as an argument to the **mkdir** command, and a subdirectory will be made in the current working directory. For example, to create a subdirectory in the working directory called **datafiles**, you enter the command line

```
%mkdir datafiles
```

You can check that the directory **datafiles** was created by performing an **ls** command. To begin working in the directory, you can execute a **cd datafiles** command.

Directories can be nested so you could create subdirectories of subdirectories of subdirectories, and so on. In practice, personal subdirectories that are more than five or six levels deep become cumbersome.

If you are working in a directory that has a growing number of files, say 20 or more, you may want to think about organizing some of the files into a subdirectory. It is a good idea to have a number of subdirectories set up in your home directory. When you log in, you can then switch to the directory where you want to work.

You should execute the command line

```
%man mkdir
```

to learn more about the **mkdir** command.

Exercises

1. Create a subdirectory called **test** in your home directory. Move into the directory. Can you create another directory called **test** inside of the original directory **test**? Try it and **cd** to the latest **test**. What is the result of executing a **pwd** command?

2. Can you create a subdirectory called **VERYLONGNAMEDIRECTORY**?
3. A man named Walter has 300 files in his home directory and no subdirectories. What are some of the problems Walter faces when trying to locate one of his files?
4. For this problem do not count hidden directories. How many directories in total would you have if you created directories five levels deep and had five subdirectories (not including hidden directories) in every directory?

4.8 Copying a File—cp Command

There are many times when you will want to copy a file. For example,

- when you want to create a local backup version of a file.
- when you want to create a duplicate version of a file for test purposes.
- when you want a local copy of a file so you can edit it.
- when you are beginning work on a new file and have a similar one that serves as a good starting point.

The **cp** command is used to copy a file. The command name **cp** is an abbreviation for "copy." When you copy a file, you simply create a distinct exact duplicate of the file. This is different from renaming a file. We cover renaming files in Section 4.9.

Warning: If you tell Unix to overwrite an existing file using the **cp** command, it will. Be careful not to destroy the contents of a file you want to keep by accidently overwriting it with the **cp** command.

The **cp** command typically takes two arguments. The first argument is the name of the file you want to copy and the second argument is the name of the copy. If you want to make a copy of the file **important.notes** called **NOTES**, you enter the command line

```
%cp important.notes NOTES
```

This command copies the file **important.notes** to the file **NOTES**. If the file **NOTES** previously existed, it is overwritten with the contents of **important.notes**. The original contents of **NOTES** is lost. If the file **NOTES** did not exist, it is created and has the same contents as **important.notes**. To verify that the file was copied, you can execute the **ls –l** command and notice that both files exist and have the same size.

There are times when you want to copy all files from one directory to another. Suppose you want to copy all files from the current working directory to its subdirectory called **BACKUP**. The following command line accomplishes this task:

```
%cp * BACKUP/.
```

Regardless of how many files there are in the current working directory, this simple command copies them all to the directory called **BACKUP** and preserves their names. The ***** means "match all file names in this directory." The first part of the second argument tells the system the copies of the files are to be put in the (already existing) directory called **BACKUP**; the **.** tells the system that each file is to be given the same name that it had originally; the **/** is needed to separate the directory name **BACKUP** from the **.**.

The usage of the **cp** command can be displayed by typing **cp** without any arguments. You should execute the command line

```
%man cp
```

to learn more about the **cp** command.

Exercises

1. What is the command for making a copy of the file called **equipment**?
2. Describe two interesting options to the **cp** command.
3. Suppose you want to copy all files from a directory called **SYSTEM** to a directory two levels up called **Test**. What command line could you use to achieve this?
4. Create a test file called **junk**. What happens if you try to copy **junk** to itself?
5. Is there a command for copying an entire directory hierarchy that is multiple levels deep? If so, describe it.
6. What is a command to copy all files with a **txt** file extension from the current working directory to a child subdirectory called **Text**?

4.9 Renaming a File—mv Command

There will be many times when you want to rename a file. For example,

- when you copy a file from the Web or a friend, and decide you have a better name for it.
- when the contents of a file changes significantly.
- when you realize a different descriptive name is more appropriate.
- when you want to conduct a series of tests using a file and so decide to give it a very short name to save typing time.

The **mv** command is used to rename a file. The command name **mv** is an abbreviation for "move." When you rename a file, you simply change the name

of the file. The contents of the file are not altered. This is a different process from copying a file. We covered copying files in Section 4.8.

Warning: If you tell Unix to overwrite an existing file using the **mv** command, it will. Be careful not to destroy the contents of a file you want to keep by accidently overwriting it with the **mv** command.

The **mv** command typically takes two arguments. The first argument is the name of the file you want to rename and the second argument is its new name. To rename the file **black.shoes** to **brown.shoes**, you enter the following command line:

```
%mv black.shoes brown.shoes
```

After entering this command, the file **black.shoes** no longer exists, and the file **brown.shoes** contains the exact same content that the file **black.shoes** used to contain.

There are times when you want to move all files from one directory to another. Suppose you want to relocate all files from the current working directory to its child subdirectory called **VERSION-2**. The following command line accomplishes this task:

```
%mv * VERSION-2/.
```

Regardless of how many files there are in the current working directory, this command moves them all to the directory called **VERSION-2** and preserves their names. The ***** means "match all file names in this directory." The first part of the second argument tells the system the files are to be moved to the (already existing) directory called **VERSION-2**; the **.** tells the system that each file is to be given the same name it had originally; the **/** is needed to separate the directory name **VERSION-2** from the **.**.

The usage of the **mv** command can be displayed by typing **mv** without any arguments. You should execute the command line

```
%man mv
```

to learn more about the **mv** command.

Exercises

1. What is the command for changing the name of a file called **glasses** to **wine.glasses**?
2. Create a test file called **foo**. What happens if you try to rename a file that does not exist to **foo**?
3. Describe two interesting options to the **mv** command.

4. Suppose you want to move all files from a directory called **SEWING** to a directory three levels up called **CHORES**. What command line could you use to achieve this?
5. Create a test file called **junk**. What happens if you try to rename **junk** to **junk**?
6. How could you use the **mv** command to delete all but one file in a directory?

4.10 Deleting a File—rm Command

There will be many times when you want to delete a file. For example,

- the file is no longer needed.
- you are running low on disk space.
- you copied it to another file and now only want to keep the new version.

The **rm** command is used to delete a file. The command name **rm** is an abbreviation for "remove." When you remove a file, you delete it.

Warning: If you tell Unix to delete a file using the **rm** command, it will. Be careful not to delete a file you want to keep. Once you have deleted a file, you cannot get it back.

To delete the file **velvet** from the current working directory, you enter the following command line:

```
%rm velvet
```

If you execute an **ls** command after deleting a file, the file you deleted will no longer be listed. There is no undelete command, so you cannot undo a mistake. There is also no "recycle bin" from which you can retrieve the file; the file is really gone.

Sometimes you may want to delete all files that end in a certain file extension. The following command line would delete all files in the current working directory whose file extension is **dvi**:

```
%rm *.dvi
```

The following command would delete all **dvi** files, **log** files, and **aux** files:

```
%rm *.dvi *.log *.aux
```

As this example illustrates, the **rm** command can take several arguments. *Note:* The arguments are separated by spaces, not commas.

Warning: If you tell Unix to delete a group of files using the **rm** command, it will. Be careful not to delete files you want to keep. Be very careful when using the **rm** command with an argument involving *****. Once you have deleted a group of files, you cannot get them back.

Once in a while you will want to delete all files that begin with a certain pattern. For example, you may want to delete all files that have the first two letters **he**. However, before you decide to delete these files, it may be worthwhile to execute the command

```
%ls he*
```

in order to determine exactly which files will be deleted. Maybe you forgot that the file **help**, which you wanted to retain, was located in this directory. If you are sure you want to delete all files beginning with **he**, you can execute the command line

```
%rm he*
```

The **–i** option to the **rm** command asks you whether you are sure you want to delete a file before it is actually removed. The **i** stands for inquiry. The **–i** flag is recommended for beginning users, as it can prevent unwanted file deletions. Some users and system administrators redefine the command **rm** to be **rm –i**. This can be accomplished using the **alias** command, which we cover in Chapter 6, by executing the command line

```
%alias rm 'rm -i'
```

This way whenever an **rm** command is executed, the user has the option not to delete the file. Here is a concrete example:

```
%rm -i ponytail
rm: remove ponytail (yes/no)?
```

If you enter an **n**, the file is not deleted. To delete the file, simply enter a **y**.

You should execute the command line

```
%man rm
```

to learn more about the **rm** command.

Exercises

1. Write a command line to delete the files **a**, **a1**, and **a2**.
2. Describe two interesting options to the **rm** command.
3. How could you delete all the files in the current working directory?
4. How could you delete an entire hierarchy of files?
5. How could you delete all files that have a file extension of **bak**?

4.11 Deleting a Directory—rmdir Command

Once you begin creating files and directories, you will find yourself in a situation where you want to do some reorganization. In such a situation you may find that you want to delete a directory. The **rmdir** command, an abbreviation for "remove directory," deletes a directory. When you remove a directory, you delete the directory.

Warning: If you tell Unix to delete a directory using the **rmdir** command, it will, but only if the directory is empty. Be careful not to delete a directory you want to keep.

To delete the directory **cellphone** from its parent directory, you enter the command line

```
%rmdir cellphone
```

If the directory is empty, it will be deleted. Otherwise, you get a message such as

```
rmdir: directory "cellphone": Directory not empty
```

To delete a directory that is not empty, you can use the **–r** or **–R** flag. Before using this flag, make sure you really want to delete everything in the directory. Once you delete the material, it is irretrievable. Sometimes you will want to delete a directory and all of its subdirectories. The **–r** or **–R** flags can be used to recursively delete an entire directory hierarchy.

You should execute the command line

```
%man rmdir
```

to learn more about the **rmdir** command.

Exercises

1. Compare and contrast the two commands **mkdir** and **rmdir**.
2. Describe two interesting options to the **rmdir** command.
3. What command could you use to delete the entire file structure contained beneath and including the directory **GrayBeard**?
4. What happens if you try to delete a nonexistent directory?

4.12 Clearing the Screen—clear Command

Sometimes after performing a **man** command and then quitting **more** with **q**, your screen may have become cluttered. It might be difficult to separate the results of the next command you enter from those of the previous one. Since output on the screen simply scrolls up, you may sometimes find it desirable to start with a fresh screen. Unix provides the command **clear** for this purpose.

The command line

```
%man clear
```

yields a **NAME** section of

```
NAME
      clear - clear the terminal screen
```

We see that typing

```
%clear
```

clears the screen for us and displays the Unix prompt at the top of the screen. Suppose you perform an **ls** command and obtain the following result:

```
GENERALS        conclu.tex      normscite.sty   summary.tex
README          depth.tex       outline.tex     thesis.bib
abstract.tex    examples.tex    prelims.tex     thesis.tex
backgrd.tex     intro.tex       report.sty      uwthesis.sty
bib.tex         model.tex       slides          zoo.tex
breadth.tex     myalpha.bst     subgraph.tex
```

You then decide to delete the file **zoo.tex**, so you enter the command line

```
%rm zoo.tex
```

Now you want to verify that **zoo.tex** has been deleted, so you do another **ls**, resulting in the following display:

```
GENERALS        conclu.tex      normscite.sty   summary.tex
README          depth.tex       outline.tex     thesis.bib
abstract.tex    examples.tex    prelims.tex     thesis.tex
backgrd.tex     intro.tex       report.sty      uwthesis.sty
bib.tex         model.tex       slides
breadth.tex     myalpha.bst     subgraph.tex
```

If you continued to delete individual files in this fashion and then checked to see if they were gone, eventually the screen would become cluttered. At that point you might decide to use **clear**, as it will be easier to read your output on a fresh screen. In such circumstances, you will find the **clear** command very useful.

Exercises

1. Execute the command **man clear** on your system. How many screenfuls of information did you get? Are there any arguments to the **clear** command? Any related commands?
2. If you are using a GUI with your version of Unix, what is the effect of clearing the screen? Once you clear the screen, are you still able to retrieve its former contents?
3. Give two reasons why you might want to clear the screen.