Farewell to UNIX

Now that we have covered the UNIX basics and you know about shell programming, it is time to add a few flourishes. The commands discussed in this chapter include disk commands, file manipulation commands, and spelling commands. You have already learned quite a few file manipulation commands, and the new commands in this chapter supplement your previous knowledge. The chapter concludes with a few security and system administration commands to help you understand more about the UNIX system and give you a more comfortable feeling about your system.

In This Chapter

14.1 DISK SPACE

There is a limit to the number of files you can store on a disk or in a file system. That limit depends on two things:

- The total amount of storage space available
- The amount of space set aside for i-nodes

An i-node number (discussed fully in Chapter 8) is assigned to each file in the system; these numbers are kept in the i-node list. An i-node contains specific file information, such as its location on the disk, its size, and so on.

14.1.1 Finding Available Disk Space: The df Command

You can use the **df** (disk free) command to find the total amount of disk space or the space available on a specified file system. If you do not specify a particular file system on the command line, then the **df** command reports the free space for all file systems.

Find the total amount of disk space available:

```
$ df [Return] . . . . . . . . . No name is specified.
/     (/dev/dsk/c0d0s0).   14534 blocks     2965  i-nodes
/usr  (/dev/dsk/c0d0s2).   203028 blocks    51007  i-nodes
$_ . . . . . . . . . . . . . . . . Ready for the next command.
```

The output shows that this system has two file systems. The first value is the number of free blocks, and the second is the number of free i-nodes. Each block is usually a 512-byte block; some systems use 1024-byte blocks for this report.

-t Option Using the **-t** option makes the **df** include the total number of blocks in the file system in the output.

Invoke the **df** command with the **-t** option:

```
$ df -t [Return] . . . . . . . . Use the -t option.
/      (/dev/dsk/c0d0s0).     14534  blocks   2965  i-nodes
                      total:  31552  blocks   3936  i-nodes
/usr   (/dev/dsk/c0d0s2).    203028  blocks  51007  i-nodes
                      total: 539136  blocks  65488  i-nodes
$_ . . . . . . . . . . . . . . . . Prompt returns.
```

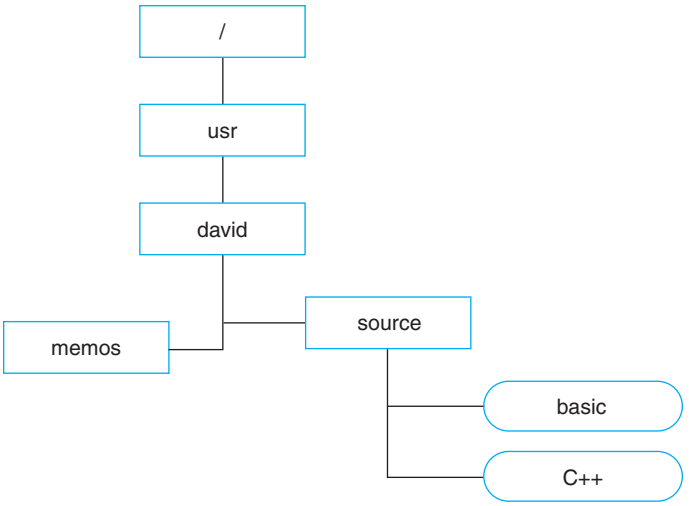
1. *Use the* **man** *command to obtain the list of available options for the* **df** *command.*
2. *For Linux users, use the* **--help** *option to see the help page.*
3. *You may want to place the* **df** *command in your* `.profile` *file so you get a report as soon as you log in.*

14.1.2 Summarizing Disk Usage: The du Command

You can use the **du** (disk usage) command to obtain a report that includes each directory in the file system, the number of blocks used by the files in that directory, and its subdirectories. This command is useful when you want to know how the space on a file system is being used.

Figure 14.1 shows the directory structure that is used to demonstrate the **du** command examples.

Figure 14.1
Directory Structure for the **du** Command Examples



Obtain a disk usage report on the current directory and its subdirectories, assuming that your current directory is `david`.

```
$ du
4       ./memos
12      ./source
20      .
$_
```

The current directory is indicated by the dot (`.`). Here `./memos` occupies 4 blocks, `./source` occupies 12 blocks, and dot (`.`) is 20 blocks.

The **du** *command can be used to obtain reports on any directory structure.*

du Options

Table 14.1 summarizes the **du** command options. Use the **man** command to obtain a detailed list of the available options for the **du** command.

Table 14.1
The **du** Command Options

Option		Operation
UNIX	Linux Alternative	
-a	**--all**	Display the size of directories and files.
-b	**--bytes**	Displays the size of directories and files in bytes.
-s	**--summarize**	Displays only the total blocks for the specified directory; subdirectories are not listed.
	--help	Displays a usage message.
	--version	Displays the version information.

-a Option The **-a** option displays the space used by each file in the specified directory, as well as the directory.

Assuming your current directory is `david`, the following command line shows the usage of the **du** command with **-a** option:

```
$ du -a
4       ./memos
4       ./source/basic
4       ./source/c++
12      ./source
20      .
$_
```

Here, the size of directories (`./memos, ./source, .`) are shown, plus the size of the files (`./source/basic and ./source/c++`).

-b Option The **-b** option displays the space used by each file in bytes instead of the default, which is in blocks.

Assuming your current directory is `david`, the following command line shows the usage of the **du** command with **-b** option.

```
$ du -b
4096    ./memos
12288   ./source
20480   .
$_
```

Here, the size of directories (`./memos, ./source, .`) are shown in bytes.

-s Option The **-s** option displays the total size of the directories or files in blocks. For example, the following command line shows the size of the current directory in blocks:

```
$ du -s [Return]. . . . . Show the current directory size in blocks.
20 . . . . . The current directory (.) is 20 blocks.
$_ . . . . . Prompt is back.
```

Assuming your current directory is `david`, the following command line shows the usage of the **du** command with **-s** and **-b** options:

```
$ du -bs ./source
12288        ./source
$_
```

Here, the size of the source directory (`./source`) is shown in bytes.

Practicing Linux Alternative Command Options

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

```
$ du --all [Return] . . . . . Same as du -a command.
$ du --byte [Return]  . . . . Same as du -b command.
$ du --summarize [Return]. . Same as du -s command.
$ du --help [Return]  . . . . Display the help page.
$ du --version [Return] . . . Display the version information.
```

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

14.2 MORE UNIX COMMANDS

This section adds to your command repertoire and introduces new commands that help you to manipulate certain aspects of the UNIX environment, create banners, and obtain necessary information about the status of your programs.

1. *As was suggested many times in other chapters and sections of this book, use the* **man** *command to learn more about each command.*
2. *For Linux users, use the* **--help** *to get a help page and use the* **--version** *to get the version information for a specified command.*

14.2.1 Displaying Banners: The banner Command

You can use the **banner** command to produce output in large letters. It displays its arguments (10 characters or less) on the standard output, one argument per line. This is useful for creating banners, signs, report titles, and so on.

Make a happy birthday banner.

```
$ banner happy birthday ¦ lp [Return] . . . . Make a banner and send it to the printer.
$_ . . . . . . . . . . . . . . . . . . . . . . Return to the prompt.
```

The pipe (¦) routes the output to the printer. Each word (argument) is printed on a separate line. You can use quotation marks to make two or more words a single argument.

Make a GO HOME banner, with both words on the same line.

```
$ banner "GO HOME" ¦ lp [Return]
$_ . . . . . . . . . . . . . . . Return to the prompt.
```

14.2.2 Running Commands at a Later Time: The at Command

You use the **at** command to run a command or list of commands at a later time. This is useful if you want to run your programs when the computer is less busy, or when you want to send mail on a certain date. You can specify the time and date part of the command in various formats, and the syntax is quite flexible.

On any given UNIX system, there may be restrictions on who can use the at command. The system administrator can limit the access to the at command to only a few users. You can try it; if you are not authorized, the following message is displayed:

```
at: You are not authorized to use at. Sorry.
```

1. *You specify on the command line the time and date when you want your command to be carried out.*
2. *You do not have to be logged in when the commands are scheduled to run.*

Specifying the Time

If the time part of the **at** command is one or two digits (HH format), then it is interpreted as the time in hours. Thus, 04 and 4 both mean four o'clock in the morning. If the time part is four digits (HHMM format), then it is interpreted as time and minutes. Thus 0811 is 08:11. You can also specify the time by typing the word *noon*, *midnight*, or *now* in the time part of the command.

UNIX assumes a 24-hour clock, unless you specify that the a.m. or p.m. suffix should be used. Accordingly, 2011 is 08:11 p.m.

Specifying the Date

The date part of the **at** command can be a day of the week such as Wednesday or Wed (three-letter abbreviation). It can be in month, day, and year format such as Aug. 10, 2005. It can also consist of the special word *today* or *tomorrow*. Here are some examples of legal **at** time and date formats:

```
$ at 1345 Wed [Return] . . . . . . . . . . . Run a job on Wednesday at 1:45 p.m.
$ at 0145 pm Wed [Return]. . . . . . . . . . Run a job on Wednesday at 1:45 p.m.
$ at 0925 am Sep 18 [Return] . . . . . . . . Run a job on September 18 at 9:25 a.m.
$ at 11:00 pm tomorrow [Return] . . . . . . . Run a job tomorrow at 11 p.m.
```

The following command sequences show how to use the **at** command and specify time and date in different formats.

Run a job at 4 a.m. the next day:

```
$ at 04 tomorrow [Return] . . . . . Specify the time and date.
sort BIG_FILE [Return]. . . . . Sort BIG_FILE.
[Ctrl-d] . . . . . Indicate the end of the input.
user=david......75969600.a......Wed Jan 26 14:32:00
$_. . . . . Ready for the next command.
```

The **sort** command will be invoked tomorrow morning at 4 o'clock. The job ID number is (75969600.a).

What does **at** do with the possible output produced by your command—in this case, the output of **sort**? If you do not specify an output file, the output is mailed to you, and you can find it in your mailbox.

Run a job using output redirection:

```
$ at 04 tomorrow [Return] . . . . . Specify the time and date.
sort BIG_FILE > BIG_SORT[Return] . . Save output in BIG_SORT.
[Ctrl-d] . . . . . Indicate the end of the
                                    command list.
user=david......75969600.a......Wed Jan 26 14:32:00
$_. . . . . Return to the prompt.
```

The output of the **sort** command is redirected to `BIG_SORT`.

You can use **cat**, **vi**, *or any of the pagination commands to look at the* `BIG_SORT` *file.*

Run a job using input redirection:

```
$ at noon Wed [Return]. . . . . At noon on Wednesday.
mailx david < memo [Return] . . . . . Mail to david the memo file.
[Ctrl-d] . . . . . End of the list of the
                                    commands.
user=david......75969600.a......Wed Jan 26 14:32:00 2005
$_. . . . . Prompt reappears.
```

This mails your memo to `david` at noon on Wednesday. A file called `memo` must be in your current directory before noon on Wednesday.

Execute the script file called `cmd_file` at 3:30 p.m. on Friday.

```
$ at 1530 Fri < cmd_file [Return]. . Input comes from cmd_file.
user=david......75969601.a......Fri Jan 28 14:32:00 2005
$_. . . . . You are ready for the next
                                    command.
```

at Options

Table 14.2 summarizes the **at** command options.

Table 14.2
The **at** Command Options

Option	Operation
-l	Lists all jobs that are submitted with **at**.
-m	Mails you a short message of confirmation at completion of the job.
-r	Removes the specified job numbers from the queue of jobs scheduled by a**t.**

Show the list of your jobs that are scheduled to run later (submitted to **at**):

```
$ at -l [Return] . . . . . List all jobs.
75969601.a.....at Fri Jan 26 14:32:00 2005
$_ . . . . . Prompt reappears.
```

Remove a job from the **at** job queue:

```
$ at -r 75969601.a [Return] . . . Remove the job.
$ at -l [Return] . . . . . Check the at queue.
$_ . . . . . Nothing is in the queue.
```

You must specify the job process ID you intend to remove from the queue. You can remove more than one job from the queue by specifying the jobs' process IDs on the command line, separated by a space.

14.2.3 Revealing the Command Type: The type Command

The **type** command is useful when you want to know more about a command. It shows whether the specified command is a shell program or a shell built-in command. The *built-in commands* are part of the shell, and no child process is created when any of them is invoked.

Let's look at some examples:

```
$ type pwd [Return] . . . . . Find more about the pwd command.
pwd is a shell built-in
$ type ls [Return] . . . . . And the ls command.
ls is /bin/ls
$ type cd [Return] . . . . . The cd command.
cd is a shell built-in
$_ . . . . . Prompt reappears.
```

14.2.4 Timing Programs: The time Command

You can use the **time** command to obtain information about the computer time your command uses. It reports the real time, user time, and system time. You type **time** followed by the command name you want to time.

Real time is the actual time (*elapsed time*) from the moment you enter the command until the command is finished. This includes I/O time, time spent waiting

for other users, and so on. The real time can be several times greater than the total CPU time.

User time is the CPU time dedicated to the execution of your command.

System time is the time spent executing UNIX kernel routines to service your command.

CPU time is the time in seconds and fractions of seconds that the CPU spends to execute your command.

Report the time it takes to sort `BIG_FILE`:

```
$ time sort BIG_FILE > BIG_FILE.SORT [Return]
60.8 real    11.4 user    4.6 sys
$ . . . . . . . . . . . . . . . . You are ready for the next command.
```

The reports show that the program will run for 60.8 seconds of real time, using 11.4 seconds of user time, and 4.6 seconds of system time, for a total of 16 seconds of the CPU time.

14.2.5 Reminder Service: The calendar Command

You can use the **calendar** command to remind yourself of your appointments and other things you want to do. To use this service, you must create a file called `calendar` in your `HOME` directory or the current directory. It displays those lines in the `calendar` file that contain today's or tomorrow's date. If your system is set up to run the **calendar** command automatically, then it sends you e-mail that contains appropriate lines from your `calendar` file. You can also run the **calendar** command at the $ prompt to display lines from your `calendar` file.

The date in each line can be expressed in various formats. Figure 14.2 shows an example of a `calendar` file, with each line containing a different format of the date string.

Figure 14.2

Example of a `calendar` File

```
$ cat calendar
1/20 Call David
Meet with your advisor on January 22.
1/22 You have a dental appointment.
The BIG MEETING is on Jan. 23.
3/21 Time to clean up your desk.
$_
```

Run the **calendar** command, assuming today is January 22:

```
$ calendar [Return]  . . . . . Run the command.
1/22 You have a dental appointment.
Meet with your advisor on January 22.
The BIG MEETING is on Jan. 23.
$_. . . . . . . . . . . . . . . . Prompt reappears.
```

Any line in the `calendar` file that contains the date 1/22 or 1/23 is displayed.

1. *You can place the* **calendar** *command in your* `.profile` *(startup file) so you are informed of your schedule as soon as you log in.*
2. *This command is not available on all systems. Check its availability in your system before placing the command in the* `.profile` *file.*

14.2.6 Detailed Information on Users: The finger command

The **who** command (introduced in Chapter 3) is used to obtain information about people on the system. You can use the **finger** command to obtain more detailed and informative reports about the users who are logged on. By default, the **finger** command displays information in a multicolumn format. Figure 14.3 is an example of output displayed by the **finger** command.

- The *Login* column shows the login name of the users.
- The *Name* column shows the full name of the users.
- The *TTY* column shows the device number of the users' terminals. The (*) before the terminal name (such as `*console`) indicates that sending messages to that terminal is blocked (see **mesg**, Chapter 10).
- The *Idle* column shows the elapsed time since each user last typed on the keyboard.
- The *When* column shows the time that each user logged in.
- The *Where* column shows the addresses of the users' terminals.

Figure 14.3
The **finger** Command

```
$ finger
Login       Name          TTY      Idle     When       Where
dan     Daniel Knee     *console   5:08   Mon 14:14
gabe    Gabriel Smart   *p0        1:33   Mon 17:15   205.130.70
emma    Emma Good       p2         8:21   Mon 11:02   205.130.71
$_
```

If **finger** is issued with a user's name as an argument, the information about the specified user is displayed regardless of whether he or she is logged in. Figure 14.4 shows information about a user called `gabe`.

Figure 14.4
The **finger** Command with an Argument

```
$ finger gabe
 Login name: gabe    (message off)       In real life: Gabriel Smart
 Directory: /home/students/gabe          shell:/bin/ksh
 On since July 30 18:45:38 On p0
 Mail last read Tue Jul 23 09:08:06 2005
 No Plan.
$_
```

The .plan and .project Files

The only curious thing here is the last line of output: "No Plan." The **finger** command displays the contents of two hidden files called `.plan` and `.project` in the HOME directory of the specified person. In this example, "No Plan." indicates that gabe has not prepared a `.plan` file. Assuming the following `.project` and `.plan` files exist in Gabe's HOME directory, Figure 14.5 shows the output of the **finger** command.

Figure 14.5
The **finger** Command Output Displaying `.project` and `.plan` Files

```
$ finger gabe
Login name: gabe          (message off)       In real life: Gabriel Smart
Directory: /home/students/gabe                shell:/bin/ksh
On since July 30 18:45:38 On p0
Mail last read Tue Jul 23 09:08:06 2005
Project: C11 Application Project
Plan:
Hi,
I am all smiles these days!
Gabe :-)
$_
```

The `.project` file in Gabe's home directory:

```
C++ Application Project
This project is assigned to the Gold Team.
For more information about this Project contact Gabe Smart.
```

The `.plan` file in Gabe's home directory:

```
Hi,
I am all smiles these days!
Gabe:-)
```

The **finger** *command displays the first line of the* `.project` *file, if it exists in the user's* HOME *directory* (`$HOME/.project`), *and the whole content of the* `.plan` *file, if it exists in the user's* HOME *directory* (`$HOME/.plan`).

finger Options

The **finger** command options mostly manipulate the format of the output. Table 14.3 shows some of these options.

The long format option displays a format similar to the single user report (Figure 14.4) for all the users.

14.2.7 Saving and Distributing Files: The tar Command

You can use the **tar** (*t*ape *ar*chiver) command to copy a set of files into a single file called `tarfile`. A `tarfile` is usually saved to a magnetic tape but it can be on any other media, such as a floppy disk. The **tar** command packs multiple files into a single

Table 14.3
The **finger** Command Options

Option	Operation
-b	Suppresses displaying the user's HOME directory and shell in a long form at display.
-f	Suppresses displaying the header in a nonlong format output.
-h	Suppresses displaying the `.project` file in a long format output.
-l	Forces long format output.
-p	Suppresses displaying the `.plan` file in a long format output.
-s	Forces short format output.

file in `tar-format` that can be later unpacked by **tar**. For example, to archive all the files in your current directory and subdirectories to a single `tarfile`, you type

```
$ tar -cvf /dev/ctape1 . [Return]
```

In this example, **tar** packs files from your current directory, which is represented by the dot **(.)** at the end of the command line, into a single `tarfile` and sends it to the device `/dev/ctape1`.

tar Options

Table 14.4 summarizes the common options for **tar**.

Table 14.4
The **tar** Command Options

Option		Operation
UNIX	**Linux Alternative**	
-c	**--create**	(**create**) Creates a new `tarfile`. Writing begins at the beginning of the `tarfile`.
-f	**--file**	(**file**) Uses the next argument as a place where the archive is to be placed.
-r	**--concatenate**	(**replace**) Writes a new archive at the end of the `tarfile`.
-t	**--list**	(**table of contents**) Lists the name of the files in the `tarfile`.
-x	**--extract** or **--get**	(**extract**) Extracts files from the `tarfile`.
-v	**--verbose**	(**verbose**) Provides additional information about the `tarfile` entries.
	--help	Displays a usage message.
	--version	Displays the version information.

The following command sequences show examples using **tar** to archive files.

Assume your current directory is called `projects` and that it contains two files named `head` and `tail` and a subdirectory called `tarfiles`. The following command archives the current directory in a `tarfile` called `projects.tar` in the `tarfiles` directory.

`$ tar -cvf ./tarfiles/projects.tar . [Return]` . . . Archive the current directory.

The option **c** is for creating a new `tarfile`, **v** is for listing everything as **tar** goes along, and **f** indicates the next argument is where the `tarfile` is placed.

```
$ tar -cvf ./tarfiles/projects.tar .
 a ./projects/                    0K
 a ./projects/head                3K
 a ./projects/tail                6k
$_
```

The **v** option provides additional information about the `tarfiles` entries. The output consists of three fields. The letter **a** is a function letter indicating archiving, followed by the pathname of each file and then its size.

The name of the `tarfile` *can be any valid UNIX filename. Here, the extension* `.tar` *is added for clarity and is not a necessity.*

The following screen shows the **tar** output without the **v** option. Notice that no information is provided.

```
$ tar -cf ./tarfiles/projects.tar .
$_
```

The following command sequences show examples using **tar** to retrieve files from a `tarfile`.

Assume you have a `tarfile` called `projects.tar` in a directory called `tarfiles`. The following command displays the table of contents of the `projects.tar` file:

`$ tar -tvf ./tarfiles/projects.tar [Return]` . . . List the table of contents in `projects.tar` file.

The option **t** is for listing a table of contents, **v** is for listing everything as **tar** goes along, and **f** indicates the next argument is where the `tarfile` is placed:

```
$ tar -tvf ./tarfiles/projects.tar
 drwx r-- r-- 2029/1005 0    Sep 3 10:33 2005 ./tarfiles/project/
 -rw- r-- r-- 2029/1005 2350 Sep 3 10:33 2005 ./tarfiles/project/head
 -rw- r-- r-- 2029/1005 5374 Sep 3 10:33 2005 ./tarfiles/project/tail
$_
```

The output is similar to the format produced by the **-l** *option of the* **ls** *command.*

The columns have the following meanings:

Column 1	Column 2	Column 3	Column 4	Column 5
-rw- r-- r--	2029/1005	2350	Sep 3 10:33 2005	./tarfiles/project/head
Access permission to `head`	User ID/ Group ID of `head`	Size of `head` in bytes	Modification date of `head`	Name of the file, `head`

Now, if you want to extract (retrieve) all files from a `tarfile`, you type

```
$ tar -xvf ./tarfiles/projects.tar [Return] . . . .
```
Extract all files from `projects.tar` into current directory.

The option **x** indicates you want **tar** to extract files from the `projects.tar` file. The **v** option causes a verbose output:

```
$ tar -xvf ./tarfiles/projects.tar .
  x ./projects, 0 bytes, 0 tape blocks
  x ./projects/head, 2350 bytes, 5 tape blocks
  x ./projects/tail, 5374 bytes, 11 tape blocks
$_
```

Notice it provides the size of each file in bytes and tape blocks.

You also can retrieve certain files from the backup medium by specifying the file names on the command line, such as

```
$ tar -xvf ./tarfiles/projects.tar tail [Return]. .
```
Retrieve `tail` from `projects.tar` into current directory.

```
$ tar -xvf ./tarfiles/projects.tar tail
  x ./projects/tail, 5374 bytes, 11 tape blocks
$_
```

Practicing Linux Alternative Command Options

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

```
$ tar --create --verbose --file ./tarfiles/projects.tar . [Return]  .
```
Same as **tar -cvf** command.

```
$ tar --create --file ./tarfiles/projects.tar . [Return]. . . . . .
```
Same as **tar -cf** command.

```
$ tar --list --verbose --file ./tarfiles/projects.tar . [Return]. . .
```
Same as **tvf** command.

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

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

14.3 SPELLING ERROR CORRECTION

You can use the **spell** command to check the spelling of the words in your documents. The **spell** command compares the words in a specified file against a dictionary file. It displays the words that are not found in the dictionary file. You can specify more than one file, but when no file is specified, **spell** gets its input from the default input device, your keyboard. Let's look at some examples.

Run the **spell** command without specifying any filename:

```
$ spell [Return] . . . . . No argument is specified.
lookin goood [Return] . . . . . Input is from the keyboard.
[Ctrl-d] . . . . . This is the end of input.
lookin
goood
$_ . . . . . And the prompt appears.
```

You signal the end of your input by pressing **[Ctrl-d]** at the beginning of a blank line. The **spell** command does not suggest correct spelling; it just displays the suspected words. The output is one word per line.

The **spell** command is case sensitive: it is happy with *David* but complains about *david.*

Assuming you have a file called `my_doc`, check its spelling and save the output in another file:

```
$ spell my_doc > bad_words [Return] . Spell check my_doc.
$_ . . . . . Prompt.
```

The **spell** command output is redirected to the `bad_words` file. You can use the **cat** command to look at this file.

You can specify more than one file as the **spell** command argument. The specified file names are separated by at least one space.

Assuming you are in the vi editor, invoke the spelling checker:

```
: !spell [Return] . Invoke the spelling checker.
pervious . . . . . Type the word whose spelling you are in doubt about.
[Ctrl-d] . . . . . Indicate end of the input.
pervious . . . . . Misspelled word.
[Hit return to continue]
```

You can invoke any command from within vi by pressing **!** *at the colon prompt followed by the name of the command (see Chapter 12).*

Table 14.5
The **spell** Command Options

Option	Operation
-b	Checks British spelling.
-v	Displays words that are not in the spelling list and their derivation.
-x	Displays plausible stems for each word being checked.

14.3.1 spell Options

Table 14.5 shows the **spell** command options. Explanations and examples for each option follow.

-b Option This option makes the **spell** command check your file with British spelling. Words like *colour*, *centre*, and *programme* are accepted.

-v Option This option shows all the words that are not literally in the spelling list and their plausible derivations. The derivations are indicated by the plus sign.

Run the **spell** command with the **-v** option. The input is from the keyboard:

```
$ spell -v [Return] . . . . Input is expected from the keyboard.
appointment looking worked preprogrammed [Return]
[Ctrl-d] . . . . . . . . . . Signal the end of your input.
+ ment        appointment
+ ing         looking
+ ed          worked
+ pre         preprogrammed
$_. . . . . . . . . . . . . . . Return to the prompt.
```

-x Option This option displays plausible stems of each word until a matching word is found, or the list is exhausted. The stems are prefixed by the equal sign.

Run the **spell** command with the **-x** option and input from the keyboard:

```
$ spell -x [Return] . . . . Input is expected from the keyboard.
appointment looking worked preprogrammed [Return]
[Ctrl-d] . . . . . . . . . . Signal the end of your input.
= appointment
= appoint
= looking
= looke
= look
= preprogrammed
= programmed
= worked
= worke
= work
$_. . . . . . . . . . . . . . . Return to the prompt.
```

14.3.2 Creating Your Own Spelling List

On most UNIX systems, you can create your own dictionary file that supplements the standard dictionary with additional words. For example, you can create a file that contains the correct spellings of the special words and terms that are specific to your site or project. Using the plus sign option, you specify your dictionary file on the command line. The **spell** command first checks its own dictionary and then checks the words against your file. You can use the vi editor to create your own spelling list.

1. *Each word in your dictionary file must be typed on one line.*
2. *Your dictionary file must be sorted alphabetically.*

Assuming you have created your own dictionary file called `my_own`, specify that file on the command line using the plus sign option:

```
$ spell +my_own BIG_FILE > MISSPELLED_WORDS [RETURN]
$_  . . . . . . . . . . . . . . . . . . . . Prompt reappears.
```

The `+my_own` *indicates you want to use your own dictionary file called* `my_own`. *The* **spell** *command lists the words that are not found in either of the two dictionary files, its own and yours. The output is redirected into the* `MISSPELLED_WORDS` *file.*

Scenario The default spelling list (dictionary) does not contain the shell commands or other specific UNIX words. Therefore, if you have words such as **grep** and **mkdir** in your text, they are displayed as suspicious words.

You can create a file similar to the one depicted in Figure 14.6, which contains the shell commands or UNIX words with correct spelling.

Figure 14.6
Example of a Dictionary File

```
$ cat U_DICTIONARY
grep
i-node
ls
mkdir
pwd
```

Run the **spell** command with and without your spelling list:

```
$ spell +U_DICTIONARY [Return]  . . .  Run with your spelling list.
grep  pwd  mkdir  is [Return] . . . .  This is your input.
$_  . . . . . . . . . . . . . . . . .  No spelling errors; the prompt
                                       is back.
$ spell [Return] . . . . . . . . . . .  Run without your spelling list.
grep  pwd  mkdir  ls  [Return]  . . .  This is your input, same as
                                       before.
grep
ls
mkdir
pwd
$_  . . . . . . . . . . . . . . . . . .  The prompt is back.
```

14.4 UNIX SECURITY

Information and computer time are valuable resources that require protection. System security is a very important part of multiuser systems. There are various aspects of system security to consider:

- Keeping unauthorized people from gaining access to the system
- Keeping an authorized user from tampering with the system files or other users' files
- Granting some users certain privileges

Security on the UNIX system is implemented by using simple commands, and the system security can be as lax or tight as you desire. Let's examine some of the available means to secure your system.

14.4.1 Password Security

All the information the system needs to know about each user is saved in a file called `/etc/passwd`. This file includes each user's password; however, it is encrypted, using an encoding method that makes the deciphering of the passwords a very difficult task. (Encryption is discussed in more detail later in this section.)

The `passwd` file contains one entry for each user. Each entry is a line that consists of seven fields, which are separated by colons. The following line shows the format of each line in the `passwd` file, followed by the explanation of each field in the line.

```
login-name:password:user-ID:group-ID:user-info:directory:program
```

login-name: This is your login name, the name you enter in response to the login prompt.

password: This is your encrypted password. In System V Release 4, the encrypted password is not stored in the `passwd` file; instead, it is stored in a file called `/etc/shadow` and the letter *x* is used as the placeholder for the `password` field in the `passwd` file.

You can print the `/etc/passwd` *file, but the* `/etc/shadow` *file is not readable by ordinary users.*

user-ID: This field contains the user ID number. The user ID is a unique number assigned to each user, and user ID 0 indicates the *superuser*.

group-ID: This field contains the group ID. The group ID identifies the user as a member of a group.

user-info: This field is used to further identify the user. Usually it contains the user's name.

directory: This field contains the absolute pathname of the `HOME` directory assigned to the user.

program: This field contains the program that is executed after the user logs in. Usually it is the shell program. If the program is not identified, `/usr/bin/sh` is assumed. You can change your login shell to `/usr/bin/chs` to log in to the C shell, or change it to any other program you wish to log in to.

Figure 14.7
Sample of the passwd File

```
$ cat /etc/passwd
root:x:0:1:admin:/:/usr/bin/sh
david:x:110:255:David Brown:/home/david:/usr/bin/sh
emma:x:120:255:Emma Redd:/home/emma:/usr/bin/sh
steve:x:130:255:Steve Fraser:/home/steve:/usr/bin/csh
$_
```

Figure 14.7 shows some sample entries in the `passwd` file. This file can be printed out by any user on the system.

14.4.2 File Security

File security limits access to the files. The UNIX system provides you with the commands to specify who can access a file and, once accessed, what type of operation can be done on the file.

The **chmod** command was discussed in Chapter 12; here we will explore the other way of setting file permission. When you use the **ls** command with the **-l** option (Chapter 5), you get your directory listing in full detail. Part of this detailed information is the pattern of **rwx** on each entry, which represents the file permission. The **chmod** command is used to change the file access mode (permission). For example, if you want to give execution permission on `myfile` to all users, you type the following:

```
$ chmod a=x myfile [Return]
```

However, you can specify the new mode as a three-digit number that is computed by adding together the numeric equivalents of the desired permission. Table 14.6 shows the numeric value assigned to each permission letter.

Table 14.6
The File Permission Numeric Equivalents

owner	group	others
r w x	r w x	r w x
4 2 1	4 2 1	4 2 1

Assuming you have a file called `mayflies` in your current directory, the following examples use the **chmod** command to change the file permission. You can use the **ls -l** command to verify the changes.

Change `mayflies` permission to allow **read**, **write**, and **execute** permission (access) to all users.

```
$ chmod 777 mayflies [Return]. .  Change the mayflies access mode.
$_ . . . . . . . . . . . . . . .  Done; the prompt is back.
```

The number 777 is the result of adding each column's (owner, group, or others) numeric values. Each of the digits represents one of the columns.

Change `mayflies` permission to allow the **read** access to all users, but allow **write** and **execute** access only to the owner.

```
$ chmod 744 mayflies [Return] . . . . Change the mayflies access
                                      mode.
$_ . . . . . . . . . . . . . . . . . . Done; the prompt is back.
```

The digit *7* grants all access to the owner. The first digit *4* grants only the **read** access to the group, and the second digit *4* grants only the **read** access to the others.

Grant all permissions to the owner and group, but only **execute** permission to the others.

```
$ chmod 771 myfile [Return] . . . . . Change the file permission.
$_ . . . . . . . . . . . . . . . . . . Done; the prompt is back.
```

The digit *1* grants only the **execute** permission to the others.

14.4.3 Directory Permission

Directories have permission modes that work in a similar manner to file permission modes. However, the directory access permissions have different meanings:

read: The read (**r**) permission in a directory means you can use the **ls** command to list the filenames.

write: The write (**w**) permission in a directory means that you can add and remove files from that directory.

execute: The execute (**x**) permission in a directory means you can use the **cd** command to change to that directory or use the directory name as part of a pathname.

Grant access permission to all users on the directory called `mybin`:

```
$ chmod 777 mybin [Return] . . . . . Change the access mode of a
                                     directory.
$_ . . . . . . . . . . . . . . . . . Done; prompt is back.
```

As with file permissions, each digit stands for one of the user groups (owner, group, and others). The digit *7* means granting write, read, and execute access to a particular group of users.

14.4.4 The Superuser

It is time to know who the *superuser* is, as some of the commands explained in this chapter might be available only to a superuser on your system. The *superuser* is a user who has privileged authority and is not restricted by the file permissions. The system administrator must be a superuser in order to perform administrative tasks such as establishing new accounts, changing passwords, and so on. Usually, the superuser logs in as **root** and can read, write to, or remove any file in the system, or even shut the system down. Superuser status is usually granted to the system administration personnel.

14.4.5 File Encryption: The crypt Command

The superuser can access any file regardless of file permissions. How can you protect your sensitive files from others (superuser or not)? UNIX provides the **crypt** command, which encrypts your file and makes it unreadable to others. The **crypt** command changes each character in your file in a reversible way, so you can obtain the original file later. The encoding mechanism relies on simple substitution. For example, the letter *A* in your file is changed to the symbol ~. The **crypt** command uses a key to scramble its standard input into an unreadable text that is sent to the standard output.

The **crypt** command is used for both encryption and decryption. In fact, for decryption of a file, you must provide the same key that you specified to its encryption. Let's look at some examples.

Encrypt a file called names in your current directory:

```
$ cat names [Return] . . . . . Display the content of names.
David   Emma   Daniel   Gabriel   Susan   Maria
$ crypt xyz < names > names.crypt [Return] . . Use xyz as the encryption key.
$ cat names.crypt [Return] . . . . . Display the content of names.crypt,
                                     the encrypted file.
<<güé^>>ZQX_∂å£ëÁÒ7ÂpgŒYñÅÍ[ÎrÂÆ
$_
$ rm names [Return] . . . . . Remove names.
$ crypt xyz < names.crypt > names [Return] . . Decode names.crypt.
$ cat names [Return] . . . . . Check the contents of names.
David   Emma   Daniel   Gabriel   Susan   Marie
$_ . . . . . The prompt is back.
```

The **xyz** is the key used to encrypt the file. The key is actually a password that you use to uncode your file later.

Using input/output redirection, the input to the command is `names` and the output is stored in `names.crypt.` When the file is encrypted, the input to the command is `names.crypt` and the output is stored in `names` when the file is decrypted by issuing the **crypt** command again.

Usually, you remove the original copy of a file after file encryption and leave only the encrypted version, making the information in the file accessible only to someone who knows the encryption key.

Encrypt the `names` file without specifying the encryption key on the command line:

```
$ crypt < names > names.crypt [Return] . . . . . No encryption key is specified.
Key . . . . . The prompt for entering the key
                                     appears.
$_ . . . . . Return to the prompt.
```

If you do not specify the encryption key on the command line, then **crypt** *prompts you to enter one. The key that you enter is not echoed, and this method is preferred over typing the key on the command line.*

If you type the wrong code while entering the password (key) the first time when you encrypt a file, you will not be able to decrypt the file later. As a safety check, you may want to try decrypting the encrypted file before you remove the original.

14.5 USING FTP

The *File Transfer Protocol* (FTP) is one of the most frequently used services available on your system. You can use the FTP commands to transfer files from one system to another. Files of any type can be transferred, although you may have to specify whether the file is an ASCII or a binary file.

1. *FTP (file transfer protocol) is not just the name of the protocol; it is also the name of a program or command.*
2. *FTP is a popular way to share information over the Internet.*

You issue the FTP command by typing **ftp**, followed by the address of another site, and then pressing the **[Return]** key.

```
$ ftp server2 [Return]  . .  Issue the FTP command
```

You are prompted to enter the user name and password. That means you must have the login permission on the site called *server2*.

```
Username: daniel . . . . . Enter user name, in this case daniel.
Password:  . . . . . Enter password (not echoed)
```

If you have a login name on *server2*, you can transfer files from your system to *server2*. FTP provides a set of commands that allows you to transfer files from one system to another, list directories, and copy files in either direction.

14.5.1 FTP Basics

FTP works as a client/server process. You give the command **ftp** using a remote address, such as the following:

```
$ ftp server2 [Return]  . .  Issue the ftp command to contact server2.
```

In this case, *server2* is the name of the remote site you want to contact. The **ftp** command immediately attempts to establish a connection to the FTP server that you specified on the command line. In this case, it is the FTP server on *server2*.

When you issue the **ftp**, the FTP running on your system is a client to an FTP process that acts as a server on *server2*. You issue commands to the FTP process at *server2*, and it responds appropriately.

In an FTP session, your system is referred to as **local-host***, and the system you are connected to is referred to as* **remote-host**.

You can use the FTP interactive mode to connect to the remote site. If you do not specify the name of the FTP server, **ftp** enters its interactive mode and prompts you for further instructions. For example

```
$ ftp [Return]  . .  The remote site to be contacted is not specified.
$ ftp> . . . . . ftp enters into interactive mode and displays its prompt.
```

To see a list of all **ftp** commands, type **?** and press **[Return]** during an FTP session. For example

```
ftp> . . . . . Prompt indicating an FTP session is in progress.
ftp> ?[Return]  . .  List the commands.
```

To terminate the current FTP session and return to the prompt, type **bye** as follows:

ftp> bye[Return] . . . . . Terminating and aborting FTP session.

Or you can use the **quit** command as follows:

ftp> quit[Return] . . . . . Terminating and aborting FTP session.

In both cases, the end of the session message is displayed:

```
221 Goodbye.
```

Figure 14.8 shows the list of commands that is displayed when the **?** command is used in a **FTP** session.

Figure 14.8

List of the **ftp** Commands

```
ftp> ?
Commands may be abbreviated. Commands are:
   !        cr             macdef     proxy          send
   $        delete         mdelete    sendport       status
account     debug          mdir       put            struct
append      dir            mget       pwd            sunique
ascii       disconnect     mkdir      quit           tenex
bell        form           mls        quote          trace
binary      get            mode       recv           type
bye         glob           mput       remotehelp     user
case        hash           nmap       rename         verbose
cd          help           ntrans     reset             ?
cdup        lcd            open       rmdir
close       ls             prompt     runique
ftp>
```

You probably need only a few of these commands for most of your FTP work. The following tables list and categorize most of the **ftp** commands.

FTP Connection Commands

The FTP connection related commands are used to establish connection to the remote site and terminate the connection at the end of the FTP session. Table 14.7 summarizes these commands.

FTP File Transfer Commands

The FTP file transfer-related commands are used to transfer files between the local and remote hosts or vice versa. The two most common modes of transfer are ASCII and binary. By default, the transfer mode is set to **ascii**. You can transfer any text (ASCII) file using **ascii** mode. However, binary files such as compiled and executable files must be transferred using the **binary** mode. Table 14.8 summarizes these commands.

FTP File and Directory Commands

The FTP file and directory-related commands are used to manage and manipulate files in an FTP session. These commands are similar to the UNIX file and directory commands. Table 14.9 summarizes these commands.

Table 14.7
FTP Access Commands

Command	Description
open ***remote-host name***	Opens a connection to the FTP server on the specified host. It prompts you to enter the user name and password to log in on the remote host.
close	Closes current open connection and returns to the local FTP command. At this point you may issue the **open** command for a different remote host.
quit (bye)	Closes the current FTP session with the remote server and exits **ftp**. That is, it returns to UNIX shell level.

Table 14.8
FTP File Transfer Commands

Command	Description
ascii	Sets the file transfer mode to ASCII. This is the default file type.
binary	Sets the file transfer mode to binary.
bell	Sounds a bell when file transfer is completed.
get ***remote-filename [local-filename]***	Copies a single file from the remote to the local host. If no local filename is specified, the copy has the same name on the local host.
mget ***remote-filenames***	Copies multiple files from the remote to the local host.
put ***local-filename [remote-filename]***	Copies a single file from the local to the remote host. If no remote filename is specified, the copy has the same name on the remote host.
mput ***local-filenames***	Copies multiple files from the local to the remote host.

Table 14.9
FTP Files and Directory Commands

Command	Description
cd ***remote-directory-name***	Changes the current directory on the remote host to the specified directory.
lcd ***local-directory-name***	Changes the currect directory on the local host to the specified directory.
dir	Lists the current directory on the remote host.
pwd	Prints the name of the current directory on the remote host.
mkdir ***remote-directory-name***	Makes a new directory on the remote host. Typically, you must have permission to do this.
delete ***remote-filename***	Deletes a single specified file on the remote host.
mdelete ***remote-filenames***	Deletes multiple files on the remote host.

Miscellaneous Commands

The (**?**) command provides help about **ftp** commands. The (**!**) is the shell escape command. It is used to perform commands on the local host. The **hash** command provides feedback messages in file transfer. This command is particularly useful when you are transferring large files. Table 14.10 summarizes these commands.

Table 14.10
Miscellaneous FTP Commands

Command	Description
? or **help**	Displays an informative message about the meaning of the specified command. If no argument is given, a list of the known commands is displayed.
!	Switches to escape shell mode.
hash	Displays hash sign (#) as feedback for each data block transferred. The size of a data block is 8192 bytes.

The following FTP sessions are examples to show the use of the **ftp** commands.

Opening an FTP Session

The **ftp open** command is used to begin a session with a remote host. The following command sequences shows how to begin an FTP session with remote host called *server2*.

```
$ ftp [Return] . . . . . Issue the ftp command.
ftp> . . . . . ftp prompt is displayed.
ftp> open [Return] . . . . . Issue the open command.
(to) . . . . . ftp open prompt is displayed.
(to) server2 [Return] . . . . . Enter the remote host name.
Responses from FTP server on the remote host (server2)
Connected to server2
220 server2 FTP server (UNIX(r) System V Release 4.0) ready.
Name (server2:gabe): . . . . . ftp prompt is displayed.
Name (server2:gabe): gabe [Return] . . Enter your login name on the remote host.
331 password requires for gabe
password: . . . . . ftp password prompt is displayed.
password: [Return] . . . . . Enter the password (not displayed).
230 User gabe logged in.
ftp> bye [Return] . . . . . End of the FTP session.
221 Goodbye . . . . . ftp end of session message.
```

You can also specify the remote host name on the command line. For example, to begin an FTP session with the remote host called *server2*, you type

```
$ ftp server2 [Return] . . . . . Remote host name is specified on the command
                                 line.
Connected to server2.
220 server2 FTP server (UNIX(r) System V Release 4.0) ready.
```

Similar to the previous FTP session examples, **ftp** prompts you to enter your login name and password on the remote host before the FTP session is established.

The following command sequences show how to use the help and a few other **ftp** commands. It is assumed you have started an FTP session with a remote host called *server2*.

Using the bell Command

```
ftp> help bell [Return] . . . Display description of the bell command.
bell    beep when command completed
ftp>_. . . . . . . . . . . . . ftp prompt is displayed.
ftp> bell [Return]  . . . . . Display the bell mode.
Bell mode on.
ftp> bell [Return]  . . . . . Turn the bell off.
Bell mode off.
ftp>_. . . . . . . . . . . . . ftp prompt is displayed.
```

Using the hash Command

The **hash** command is used to get feedback when a block of file is transferred. The block size is 8192 bytes. Hash signs, such as those shown on the following line, will be displayed at completion of each block transferred.

```
#####
```

The hash mode is usually set before transferring large files:

```
ftp> help hash [Return] . . . Display description of the hash command.
hash    toggle printing '#' for each buffer transferred
ftp> _ . . . . . . . . . . . . ftp prompt is displayed.
ftp> hash [Return]  . . . . . Set the hash mode.
Hash mark printing on (8192 bytes/hash mark).hash
ftp> hash [Return]  . . . . . Turn the hash off.
Hash mark printing off.
ftp> _ . . . . . . . . . . . . Display the bell mode.
```

Using the File Transfer Mode Commands

```
ftp> help bin [Return]  . . . Display description of the bin command.
binary    set binary transfer type
ftp> bin[Return]. . . . . . . Change to the file transfer binary mode.
200 Type set to I.
ftp> ascii [Return] . . . . . Change to the file transfer ASCII mode.
200 Type set to A.
ftp>_. . . . . . . . . . . . . ftp prompt is displayed.
```

Getting a File from Another System

Suppose you have started an FTP session with a remote host called *server2* and the remote host has a file named `Notes` in a directory named `Memos`. The following command

sequences show how to retrieve `Notes` from the remote host. As you issue the commands, you receive some feedback from the server, indicating that it is carrying out your orders.

The **cd** command is used to change to the directory named `Memos` on the remote host:

```
ftp> cd Memos [Return] . . . . . Change to the Memos directory.
ftp>
```

The **dir** command is used to list the current directory on the remote host (*server2*), to see if the file is there:

```
ftp> dir [Return]. . . . . List the filenames
-rwx --x --x    1 tuser other  3346 Aug      9 11:51 Notes
ftp>
```

Or you can use the **ls** command:

```
ftp> ls [Return] . . . . . List the filenames.
ftp> get [Return]. . . . . Issue a get command.
(remote) file . . . . . Prompt to enter the filename is
                        displayed.
(remote-file) Notes [Return]. . . Enter the filename on the remote
                                  host.
(local-file)  . . . . . Prompt to enter the filename is
                        displayed.
(local-file) Notes [Return]  . . . Enter the filename on the local
                                   host.
```

The remote host transfers the file to your current directory on the local host, and you receive feedback about how many bytes were sent and the rate of transfer:

```
200 PORT command successful.
150 ASCII data connection for Notes
(192.246.52.166,37175).
226 Transfer complete.
ftp>_
```

Miscellaneous Commands

As was mentioned, the **?** command is used to get the list of **ftp** commands. You can also use the **?** for obtaining help for an **ftp** command by specifying a command as the first argument to the **?**. For example:

```
ftp> ? close [Return]  . . . . . Help for the close command.
Close    Terminate FTP session.
```

The shell escape command (**!**) is used to start a subshell on the local host. This is very useful if you want to perform some operations on the local host while you are connected to a remote FTP server. After you are finished working on the local host, exit the subshell to return to your FTP session. The following command sequences demonstrate the **!** usage.

Assuming you are in an FTP session, the following command invokes a subshell:

```
ftp> ![Return]  . . . . . Issue a shell escape command.
$_ . . . . . Shell prompt issued by the local
                        host.
```

You are now in your current directory on the local host and any commands you use are invoked on the local host. You can use any of the available UNIX commands, and the commands are applied to the files on the local host. For example:

```
$ cd [Return] . . . . . .  Change to the HOME directory on
                           the local host.
$_ . . . . . .  Shell prompt issued by the local
                host.
```

This changes your current directory on the local host to be your HOME directory. You type **exit** to terminate the subshell and return to your FTP session:

```
$ exit [Return] . . . . . .  Terminate the shell and return to the
                             FTP session.
ftp>_ . . . . . .  Back to the FTP session.
```

14.5.2 Anonymous FTP

Anonymous FTP uses the `ftp` command and a special restricted account named *anonymous* as the remote host. Using anonymous FTP, you can access and share many files such as documents, source code, and executable files on the Internet. There is a special login for **ftp** that allows you to anonymously access files on a remote host. The following steps describes the process that is similar to using any FTP:

1. You need to know the address or host name of a site that allows anonymous FTP.
2. You use *anonymous* as the user name.
3. You enter any nonempty string for the password. Some systems do password validation on anonymous logins. In this case, you give your e-mail address as your password.

Once you are logged in as *anonymous*, you are granted limited access to the anonymous `ftp` subdirectory on the remote host. All of the commands described in this section can be used.

Usually, FTP sites place publicly accessible files in the directory `/public`. Some sites have a directory called `incoming` somewhere in the `ftp` subdirectory where you place your files.

Anonymous FTP Session

The following command sequence is an example of an FTP anonymous session. It shows the FTP commands and the FTP feedback messages.

Let us assume the anonymous FTP site is `ftp.xyz.net`, and the file you want to get is `pub/services/example.tar`:

```
$ftp .xyz.net [Return] . . . . . .  Start an FTP session with the remote
                                     site connected to ftp.xyz.net
220 ftp.xyz.net FTP server (Version 1.1 Nov 17 2005) ready.
Name (ftp.xyz.net: xyz): anonymous [Return]  . .  Specify anonymous as the user
                                                  name.
331 Guest login ok, send ident as password.
Password: [Return] . . . . . .  Give your e-mail address as the
                                password (not displayed).
230 Guest login ok, access restrictions apply.
```

```
ftb> cd/pub/services [Return] . . . . . Change to desired directory.
250 CWD command successful.
ftp> get examples.tar [Return]. . . . . Get the file.
200 PORT command successful.
266 Transfer complete.
```

1. *Note that many anonymous FTP sites contain a file named* README*, which contains information about available files. To read that file or any text file without retrieving it, you type*

   ```
   ftp> get README/pg [Return]
   ```

2. *Once you have successfully logged in as anonymous, you can use all of the FTP commands described in this section.*

   The **help** command is used to obtain a one-line description of the **get** command:

   ```
   ftp> help get [Return] . . . . . Help for the get command.
   get    receive file
   ftp>
   ```

   The **get** command is used to retrieve a file named `Notes` from the remote host:

   ```
   ftp> get [Return] . . . . . Issue the get command.
   (remote-file) . . . . . Prompt to enter the filename.
   (remote-file) Notes [Return] . . . . . Enter the filename on the
                                          remote host.
   (local-file) . . . . . Prompt to enter the filename.
   (local-file) Notes.txt [Return] . . . Enter the filename on the
                                         local host.
   ```

   The remote host transfers the file to your current directory on the local host. Information about how many bytes transferred and the rate of transfer is displayed:

   ```
   200 PORT command successful.
   150 ASCII data connection for Notes.txt (192.246.52.166,
   37176) (0 bytes).
   226 ASCII Transfer complete.
   ```

   You can enter the filenames on the command line, such as

   ```
   ftp> get Notes Notes.txt [Return] . . Retrieve Notes.
   ```

   Again, the remote host transfers `Notes` to your current directory on the local host. The name of the file on the local host is `Notes.txt`.

Transferring a File to Another System

Transferring a file from your system (local host) to another system (remote host) is done with the ftp command **put**. Suppose you have started an FTP session with a remote host called *server2* and you want to transfer a file named `november.rpt` from the directory `Reports`, a subdirectory of your current directory. The following command sequences show how to transfer `november.rpt` from the local host. As you issue the commands, you receive some responses from the server, indicating that it is carrying out your orders.

To transfer a file named `november.rpt` from the directory `Reports`, a subdirectory of your current directory, start an FTP session on the remote system and follow these steps.

The **lcd** command is used to change the directory on your system (local host):

```
ftp> lcd reports [Return]  . . . Change directory.
```

The **help** command is used to obtain a one-line description of the **put** command:

```
ftp> help put [Return] . . . . . Help for put command.
put    send one file
```

The **put** command is used to transfer the file `november.rpt` from your system to the remote host:

```
ftp> put [Return]  . . . . . . . . . . . . . . . . . Issue the put command.
(local - file) . . . . . . . . . . . . . . . . . . . Prompt to enter the filename.
(local - file) november.rpt [Return] . . . . . . . Enter the filename on the local host.
(remote - file)  . . . . . . . . . . . . . . . . . . Prompt to enter the filename.
(remote - file) november.rpt [Return]  . . . . . Enter the filename on the remote host.
```

The local host transfers the file called `november.rpt` to your current directory on the remote host and the following information is displayed:

```
200 PORT command successful.
150 ASCII data connection for november.rpt (137.161.110.77,1386).
226 Transfer complete.
ftp: 19456 bytes sent in 0.00Seconds 19456000.00Kbytes/sec.
ftp>_
```

14.6 WORKING WITH COMPRESSED FILES

Large files and software packages are often compressed to save space at anonymous FTP archives. These files are usually made available in *tar-format* with file extension such as `.tar.Z.`, and must be retrieved in binary mode. You must first uncompress the file at your site and then use **tar** to recover the original file.

14.6.1 The compress and uncompress Commands

You can use the **compress** command to reduce the size of the file to save space. When you run a **compress** command on a file, your original file is replaced by a file with the same name but with the file extension `.Z`. The **compress** command is also used with the encryption command as security measures. In this case, the file is first compressed and then encrypted by the **crypt** command. The following command sequences show the use of the **compress** command.

Assume you have a file named `important` in your current directory:

```
$ compress important [Return] . Issue the command.
$ ls Important* [Return] . . . . List the files.
important.z . . . . . . . . . . . The file is compressed.
$_
```

You can use the **-v** option to make **compress** report the efficiency of its compression:

```
$compress -v important [Return] . . . . . Use the -v option.
important: compression: 49.18%—replaced with important.Z
$_
```

You use the **uncompress** command to recover the original file. This uncompresses the compressed file and removes the compressed file. Continuing with the previous example, assume you have the compressed file named `important.z` in your current directory:

```
$ uncompress important [Return] . . . . Uncompress the
                                        important.z file.
$_  . . . . . . . . . . . . . . . . . . Prompt is back.
```

14.7 14.7 THE telnet COMMAND

The **telnet** command allows you to log into a remote server and act as though you were physically connected to that machine and all its resources—for example, if the server has a powerful CPU or some special devices, such as a scanner, a CD storage device, or a CD-writer that you want to use. This is also helpful if you want to access your account on a UNIX server from your PC at home.

When you connect to a remote machine, you are prompted for your user name and password; once you are authenticated, a shell prompt appears and you are in. You can enter commands and interact with the remote system.

With **telnet**, you need to know only the *hostname* or IP address of the machine to which you are connecting. For example, if the hostname is *university.unc.edu,* then the following command attempts to connect to the remote host:

```
$ telnet university.unc.edu [Return] . . . Connect to university.unc.edu.
```

If **telnet** is invoked without a hostname, it enters command mode, indicted by its prompt, **telnet>**, and accepts and executes the commands. For example, the following command places the **telnet** into command mode:

```
$ telnet [return]  . . . . . . . . . . . Invoke telnet.
telnet>. . . . . . . . . . . . . . . . . Telnet command mode
                                        indicated by the telnet> prompt.
telnet> open university.unc.edu [Return]. Using open command.
```

Figure 14.9 shows a sample screen after the **telnet** command is executed and you are connected to the host. If you wait too long to login, the connection to the host will be lost and you will have to issue the telnet command again.

Figure 14.9
Login Sample Screen

```
Connected to unix3.university.edu.
ESCAPE character id '^]'.

************************************************************
System: unix3.university.edu (backup)
Note:   This system is a property of University. Any illegal
        access or activity will be reported to authority and
        prosecuted.

************************************************************
login:

Connection to host lost.
Press any key to continue...
```

Notice the ESCPE character is '^]'. This character (**Ctrl-]**) is used to switch from the remote host back to your telnet screen. Logging in with **telnet** is very similar to other formats of logging in. You must have a valid user name and password on the remote system.

```
Login: david [Return]  . . . . . .  Enter your login name,
                                    for example, david.
Password: xxxxxx [Return] . . . . . .  Enter your password and
                                       press the [Return] key.
```

You are in and you get the usual UNIX welcome massages and the **$** prompt similar to Figure 14.10. Now, you can use all the UNIX commands to navigate and operate the host system according to level of access you are granted.

Figure 14.10

Sample UNIX Login Screen

```
        Last login: Mon May 3 17:21:13 from 10.0.40.27

           ****** Welcome to the University ******

You have mail.
$_
```

The prime advantage of **telnet** *is that it allows you to work on other systems without being physically located near those systems.*

Remember, when you use telnet to log into a remote machine, you run a shell on that system. Make sure you have the configuration setup that is stored in your shell startup files (such as **.profile** *or* **.login***) on that system especially if you depend on command aliases or other aspects of your local system's configuration.*

Telnet from Windows

Windows includes a **telnet** application that works pretty much like the UNIX version. This is very useful if you have a Windows system on your desk and want to log into UNIX systems on a network. From the Start menu, choose the Run Command and then type in **telent** to launch the **telnet** program. See Figure 14.11. There are also more enhanced **telnet** programs for Windows available on the Internet.

The following command sequences show how to use the **telnet** command, log into the remote system, and use some other related commands.

After you are connected to the host and logged in, you can use the Escape character (**CTRL-]**) to switch from the remote host to the **telnet** screen on your workstation to check a few things. For example, you can check the status of the connection. The command **telnet> status** displays the status of the connection, either *connected* or *disconnected*. See Figure 14.12.

Figure 14.11

Sample Windows **telnet** Screen

```
Welcome to Microsoft Telnet Client

Escape Character is 'CTRL+]'

Microsoft Telnet>
```

Figure 14.12
Sample Screen for the **telnet> status** Command

```
telnet> status
Connected to unix3.university.edu
Negotiated term type is ANSI
telnet>_
```

Switching to the **telnet** screen does not disconnect you from the host, and the **[Return]** key will usually switch you back to the remote host screen.

The **telnet> display** command shows the display operating parameters. See Figure 14.13.

Figure 14.13
Partial Output of the **telnet> display** Command

```
telnet> display
won't map carriage return on output.
will recognize certain control characters.
won't turn on socket level debugging.
won't print hexadecimal representation of network traffic.
echo              [^E]
escape            [^]]
rlogin            [\377]
quit              [^\]
eof               [^D]
erase             [^H]
kill              [^U]
start             [^Q]
stop              [^S]
telnet>_
```

You can disconnect and end your telnet connection by using the **close** command:

`telnet>` **`close [Return]`** `.....`Close the telnet session (disconnect).

14.8 REMOTE COMPUTING

In addition to **telnet** and **ftp** commands, UNIX provides another set of commands, including **rcp**, **rsh**, and **rlogin**, which act much like **telnet**. Table 14.11 lists the commands, and the following sections explain how to use each of these commands.

Table 14.11
List of the **r** Commands.

Command	Operation
rcp	Remote copy program
rsh	Remote shell program
rlogin	Remote login program

14.8.1 The Remote Access Command: rpc

You can use the **rpc** command to copy files between systems. It looks and acts like cp, but you can copy files from other systems. The basic syntax is the same as for **cp**:

```
rcp source-file destination- file
```

However, the *source* or *destination* file is either a remote *filename* of the form *hostname:path* or a local *filename*. For example, the following command copies a remote file */usr/home/student/report* from host *unix3.university.edu* to your local directory.

```
$ rcp unix3.university.edu:/usr/home/student/report [Return]  . .
```
Using the rcp command.

The following command copies a local file named *report* to the directory on the *unix3.university.edu* system:

```
$ rcp report unix3.university.edu:/usr/home/student [Return]. . .
```
Using the rcp command.

1. *A colon* (**:**) *separates the remote hostname from the filename.*
2. *You can also copy files from one remote system to another remote system.*
3. *You must have a valid account and the same username on the remote host and local system.*

14.8.2 The Remote Access Command: rsh

You can use the **rsh** command to run a shell on the remote host to execute a command that you pass on the command line. For example, the following command sequences log you into the remote system named `unix3.university.edu` and execute the **who am I** using a shell on the remote host.

```
$ rsh unix3.university.edu "whoami" [Return]. .
```
login into `unix3.university.edu` and execute the whoami command

```
david . . . . . .
```
the results of the who am i command executed.

```
$ rsh unix3.university.edu [return]. . . . . .
```
Run rsh with no command.

```
$ exit [Return]. . . . . .
```
Close connection.

```
rlogin: connection closed
```

1. *Any data from standard input on your local machine will be passed as a standard input to the command on the remote machine.*
2. *Any output from the remote machine will be passed as standard output to the local machine.*
3. *Together this makes it appear as a remote command actually executes on your local system.*
4. *If you don't pass any command to rsh, it logs you in as if you used rlogin.*

14.8.3 The Remote Access Command: rlogin

You can use the **rlogin** command to log into a remote host. The **rlogin** command works much like **telnet**. However, it uses a different protocol. For example, the following

command sequence logs you into the remote system named `unix3.university.edu`. The **-l** option tells **rlogin** to log you in as a user named *david*. By default, the name of the user using **rlogin** is used.

```
$ rlogin -l david unix3university.edu [Return] . . . login into unix3.university.edu as david.
password: . . . . . . . . . . . . . . . . . . . . . . Connected. Enter your password.
$_ . . . . . . . . . . . . . . . . . . . . . . . . . . Prompt from remote host.
$ whoami [Return] . . . . . . . . . . . . . . . . . . You can run any command on this shell.
david. . . . . . . . . . . . . . . . . . . . . . . . . Output from whoami.
$_ . . . . . . . . . . . . . . . . . . . . . . . . . . Prompt from remote system.
$ exit [Return] . . . . . . . . . . . . . . . . . . . Close connection.
rlogin: connection closed
$_ . . . . . . . . . . . . . . . . . . . . . . . . . . Prompt from local system.
```

COMMAND SUMMARY

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

at

This command runs another command or a list of commands at a later time.

Option	Operation
	Lists all jobs that are submitted with **at**.
-m	Mails you a short message of confirmation at completion of the job.
-r	Removes the specified job numbers from the queue of jobs scheduled by at.

banner

This command displays its argument, the specified string, in large letters.

calendar

This command is a reminder service and reads your schedule from the `calendar` file in the current directory.

compress

This command is used to compress the specified file, thus reducing the size of the file and saving space. The uncompress command is used to recover the original file and remove the compressed file.

crypt

This command is used to encrypt and decrypt a file. The command changes each character in your file in a reversible way, so you can obtain the original file later.

df

This command reports the total amount of the disk space or the space available on a specified file system.

du

This command summarizes the total space occupied by any directory, its subdirectories, or each file.

Option		Operation
UNIX	Linux Alternative	
-a	**--all**	Display the size of directories and files.
-b	**--bytes**	Displays the size of directories and files in bytes.
-s	**--summarize**	Displays only the total blocks for the specified directory; subdirectories are not listed.
	--help	Displays a usage message.
	--version	Displays the version information.

finger

This command displays detailed information on users.

Option	Operation
-b	Suppresses displaying of the user's `HOME` directory and shell in a long format display.
-f	Suppresses displaying the header in a nonlong format output.
-h	Suppresses displaying the `.project` file in a long format output.
-l	Forces long format output.
-p	Suppresses displaying the `.plan` file in a long format output.
-s	Forces short format output.

FTP

This command (utility) is used to transfer files from one system to another. Files of any type can be transferred, and you can specify whether the file is an ASCII or a binary file. You type **ftp** to start an FTP session.

FTP access commands

Command	Description
open ***remote-hostname***	Opens a connection to the FTP server on the specified host. It prompts you to enter the user name and password to log in on the remote host.
close	Closes current open connection and returns to the local FTP command. At this point you may issue the **open** command for a different remote host.
quit (bye)	Closes the current FTP session with the remote server and exits **ftp**. That is, it returns to UNIX shell level.

FTP file and directory commands

Command	Description
cd ***remote-directory-name***	Changes the current directory on the remote host to the specified directory.
lcd ***local-directory-name***	Changes the current directory on the local host to the specified directory.
dir	Lists the current directory on the remote host.
pwd	Prints the name of the current directory on the remote host.
mkdir ***remote-directory-name***	Makes a new directory on the remote host. Typically, you must have permission to do this.
delete ***remote-filename***	Deletes a single specified file on the remote host.
mdelete ***remote-filenames***	Deletes multiple files on the remote host.

FTP miscellaneous commands

Command	Description
? or **help**	Displays an informative message about the meaning of the specified command. If no argument is given, a list of the known commands is displayed.
!	Switches to escape shell mode.
hash	Displays hash sign (#) as feedback for each data block transferred. The size of a data block is 8192 bytes.

spell

This command checks the spelling of a specified document or words entered from the keyboard. It only displays the words not found in the spelling list and does not suggest a correct spelling.

Option	Operation
-b	Checks with British spelling.
-v	Displays the words that are not in the spelling list and their derivation.
-x	Displays plausible stems for each word being checked.

tar

This command is used to copy a set of files into a single file, called a `tarfile`. A `tarfile` is usually saved on a magnetic tape but it can be on any other media such as a floppy disk. It packs multiple files into a single file (in `tar-format`) that can be unpacked later by **tar**.

Option		
UNIX	**Linux Alternative**	
-c	**--create**	**(create)** Creates a new `tarfile`. Writing begins at the beginning of the `tarfile`.
-f	**--file**	**(file)** Uses the next argument as a place where the archive is to be placed.
-r	**--concatenate**	**(replace)** Writes a new archive at the end of the `tarfile`.
-t	**--list**	**(table of contents)** Lists the name of the files in the `tarfile`.
-x	**--extract** or **--get**	**(extract)** Extracts files from the `tarfile`.
-v	**--verbose**	**(verbose)** Provides additional information about the `tarfile` entries.
	--help	Displays a usage message.
	--version	Displays the version information.

time

This command provides information about the computer time your command uses. It reports the real time, user time, and system time required by a specified command.

type

This command gives more information about another command, such as whether the specified cmmand is a shell command or a shell built-in command.

The telnet Command

The **telnet** command allows you to log into a remote server using server resources such as such as a scanner, CD storage devices, or a CD-writer. Also, it allows you to access your account on a UNIX server from your system at home.

Remote Computing

UNIX provides a set of commands that allow you to log into a remote server and act as though you were physically connected to that server and all its resources.

Command	Operation
rcp	Remote copy program
rsh	Remote shell program
rlogin	Remote login program

REVIEW EXERCISES

1. Explain the UNIX system security. What are the ways to secure your file system?
2. Can the superuser delete your files?
3. Can the superuser read your encrypted files?
4. Is the **cd** command a built-in command? How do you find out?
5. Explain *elapsed time*, *user time*, and *system time*.
6. What is the command that reports the disk space?
7. Can you invoke the **spell** command within the vi editor? If so, how?
8. What is the command to use if you want to execute a program at a later time?
9. Do the access modes for files and directories mean the same thing?
10. What is the **tar** command used for?
11. What is a `tarfile`?
12. Can you use the **tar** command to archive files on media other than tapes?
13. What is the command to list the files in a `tarfile` called `save.tar`?
14. What is the **compress** command used for?
15. What does FTP stand for?
16. What is the command to open an FTP session?
17. What is the command to close an FTP session?
18. What is the command to reduce the file size?
19. What is the command to protect your files from others?
20. What is the command to check your spelling?

21. What is the command **telnet** used for?
22. How do you use the **telnet** command, and what do you have to know before using it?
23. What is the command to switch from **telnet** to your system? How do you return to **telent**?
24. While in **telnet**, what does the command *display* display?
25. What is the command to end **telnet**?
25. What is the command **rcp** used for?
26. What is the command for remote login?
27. What is the command for using the remote shell?
28. What is the command to end rlogin?

Terminal Session

Practice the following commands on your system.

1. Sort a file at 13:00 tomorrow.
2. Send mail to another user at 6 p.m. on Wednesday.
3. Use the **time** command on a few commands such as **sort**, **spell**, and so on, and observe the time reports.
4. Change the access mode of your directory to owner only **read**, **write**, and **execute**.
5. Check the spelling in one of your text files.
6. Create your own dictionary file and make the **spell** command use your dictionary file in addition to its own file.
7. Find out the available space on your disk.
8. Find out how many blocks are occupied by your `HOME` directory, its subdirectories, and the files in them.
9. Place the **du** and **df** commands in your `.profile` file. Observe the reports when you log in.
10. Encrypt a file (if you are authorized). Display the encrypted file on your terminal. Decrypt the file and display it again.
11. Make a banner that shows your initials on the screen.
12. Send your initials banner to the printer.
13. Can you make the system show your initials as soon as you log in?
14. Modify the `greetings` program (from Chapter 13) to show the greetings in large letters.
15. Make a `calendar` file in your `HOME` directory and type your schedule.
16. Use the **calendar** command to display your current schedule.
17. Place the **calendar** command in your `.profile` file. Observe the report when you log in.
18. Use the **tar** command to archive files in your current directory in another directory called `my_tar_files`.

19. Use the **tar** options, such as the **-t** and **-v** options, and observe the output.
20. Use the **tar x** option to extract a specified file from the `tarfile`.
21. Use the **compress** command to compress a large file. Observe the feedback message that displays the compression percentage. List the file and check the file extension.
22. Uncompress the file. Check that the compressed file is removed.
23. Open an FTP session, from your PC to the university computer (you usually can use the university computer from your PC). Do the following commands when connection is established:
    a. List the available commands.
    b. Get help for a specific command.
    c. Set **bell** and **verbose** to on mode.
    d. Set file transfer mode to ASCII mode.
    e. Make a new directory on the remote host.
    f. Delete a file from the remote host.
    g. Transfer a file from your PC to your account on the remote host.
    h. Transfer a file from the remote host to your PC.
24. Obtain the necessary information about remote access to another UNIX machine and try the following commands. (The necessary information is the hostname or IP address of the machine to which you are connecting.)
    a. Use the **telnet** command and login to your account.
    b. Use the **rcp** command and copy a file from remote server to your local machine.
    c. Use the **rlogin** command and login.