A few examples will help you see how *chmod* is used. To give the owner execute permission without changing any other permissions, you would use

\$ chmod u+x filename

Note that there are no spaces between u and +, or between + and x.

To remove read and write permissions from group members, you would use

\$ chmod g-rw filename

The following command will give everyone read permissions while removing any other permissions:

\$ chmod a=r filename

To give the user read and write permissions and everyone else read privileges, use

\$ chmod u=rw,go=r filename

### 6.18 Exercises

1. What are the rules for naming UNIX files and directories?

2. Which of the following would be valid names for ordinary UNIX files? Explain.

```plaintext
foo                     guess?   book.chap1       BOOK.chap2
2good2Btrue            {2bad}   >right>            <left>
name                   rank*    serial#            ^up^     
el_paso                w.lafayette New York /slash\
.hideNseek             .357     747                passwd
```

3. Which of the following would be valid directory names? Explain.

```plaintext
doo_wa                  dir1     Dir2           Directory.3
Game                    Set      Match         sticks
[Groucho]               'Chico'   Harpo.#        Karl?
                         .357     747                passwd
```

Exercises 4 through 11 refer to the hypothetical UNIX file system previously shown in Figure 6-4. (Hint: It may be helpful to sketch the directory structure as you go along.) These exercises should be done in order.

4. What are the absolute pathnames for root, bin, jill, and kangaroo?

5. Suppose that *Marupsial* is now your working directory. What are the relative pathnames of root, bin, jill, and kangaroo?
6. jack has two subdirectories, Continents and Oceans.
   a. What are the absolute pathnames of Continents and Oceans?
   b. From Oceans, what are the relative pathnames of root, etc, and bandi-coot?

7. Imagine that jack sets up additional subdirectories to hold geographical information. Continents contains Africa, Antarctica, Asia, Australia, Europe, NAmerica, and SAmerica. Each of these directories contains subdirectories for individual countries or regions. For example, NAmerica contains the subdirectories Canada, CentralAm, Mexico, and USA. Assuming every file and directory to be in its proper place, give the absolute pathnames of the following directories:
   a. Norway;
   b. India;
   c. Egypt;
   d. Argentina.

8. Suppose jack’s working directory is USA. Show how he could accomplish the following tasks, using a single command line and relative pathnames in each case:
   a. List the contents of the Marsupials directory belonging to jill.
   b. List the contents of Australia.
   c. Make a copy of jill’s file kangaroo, and place it under the name kangaroo in his Australia directory.

9. Repeat the previous problem using absolute pathnames.

10. The directory Canada has twelve subdirectories, one for each of the ten provinces and two territories. Suppose jack’s working directory is SAmerica. Show how he could accomplish the following tasks, using a single command line and relative pathnames in each case:
    a. List the contents of BC, the directory for British Columbia, Canada.
    b. Place a copy of the file for Vancouver, British Columbia, in the directory jill.

11. Repeat the previous problem using absolute pathnames.

12. Suppose you have a file named stuff in your working directory. Specify the command(s) you would use to do the following:
    a. Give everyone permission to read stuff; do not change any other privileges.
    b. Permit the owner and group members to read and write the file; remove all privileges from everyone else.
    c. Remove writing privileges from everyone but the owner.
    d. Give the owner and group members permission to execute stuff while giving the owner sole permission to read or write it.

13. Suppose you have a directory named MyStuff in your working directory. Specify the command(s) you would use to do the following:
Figure 6-4

```
  root (/)
    /     \
  bin    dev    etc    home    tmp    usr    var
      \   /     \     /     \     /     \     /     \    \\
  jack      jill
    /  \    /  \  \
Continents   Oceans   Bats   Marsupials
  /  \    /  \     \  \    \  \    \  \    \  \    \\
wombat  bandicoot  kangaroo  opossum  wombat
```

§ `ls`<RETURN>
bandicoot kangaroo opossum wombat

Suppose now that she wants to redirect this list into a file named `filelist`. She does this using the output redirection symbol (`>`):

§ `ls > filelist`<RETURN>

This time, nothing appears on the screen because the output was rerouted into the file. In UNIX terminology, the information was redirected from the standard output (the terminal screen) to the file. If `jill` lists her files now, she will see that there is a new one named `filelist` in the directory:

§ `ls`<RETURN>
bandicoot filelist kangaroo opossum wombat

Redirection is powerful and convenient, but it can be dangerous. If you redirect the output into a file that already exists, the original contents of the file will be lost.

**REDIRECTION INTO AN EXISTING FILE WILL OVERWRITE WHATEVER IS ALREADY IN THE FILE.**

If you want to add something to the end of an existing file while keeping the original contents, you can use the `append` operation. This requires two redirection symbols (`>>`):

§ `ls >> filelist`<RETURN>
7.13 Command Summary

Each command is typed in after the UNIX shell prompt, and each is terminated by a \(\text{\textbf{RETURN}}\). Note that \textit{file}, \textit{file1}, and \textit{file2} may be simple file names or pathnames.

**Making Calendars**

\texttt{ca1 m year} \quad show a calendar for month \textit{m} (1-12) of \textit{year} (1-9999)
\texttt{ca1 year} \quad show a calendar for \textit{year}
\texttt{ca1 year > file} \quad redirect calendar for \textit{year} into \textit{file}
\texttt{ca1 year >> file} \quad append calendar for \textit{year} to \textit{file}

**Listing and Viewing Files**

\texttt{ls} \quad list files in working directory
\texttt{cat file} \quad show contents of \textit{file} all at once
\texttt{more file} \quad show contents of \textit{file} one screen at a time; press spacebar to continue or \texttt{q} to quit
\texttt{pg file} \quad Like \texttt{more}. Press \(\text{\textbf{RETURN}}\) to see next screen, \texttt{q} to quit

**Printing Files**

\texttt{1pr file} \quad send \textit{file} to default line printer (BSD UNIX)
\texttt{lp file} \quad send \textit{file} to default line printer (AT&T UNIX)
\texttt{1pr -P\texttt{code} file} \quad send \textit{file} to printer designated by \texttt{code} (BSD)
\texttt{1p -d\texttt{code} file} \quad send \textit{file} to printer designated by \texttt{code} (AT&T)

**Copying, Renaming, and Removing Files**

\texttt{cp file1 file2} \quad copy \textit{file1} into \textit{file2}; retain both copies of the file
\texttt{mv file1 file2} \quad move (i.e., rename) \textit{file1} to \textit{file2}; retain only \textit{file2}
\texttt{rm file} \quad remove (i.e., delete) \textit{file}

7.14 Exercises

1. What are the rules for selecting UNIX file names?

2. Because of the need to make certain adjustments to the calendar, the month of September 1752 was a very unusual one. What was different about it?

3. The echo command takes a line that you type in and repeats it back on the screen. Thus if you type

\begin{verbatim}
  echo This is fun! \textbf{RETURN}
\end{verbatim}

The computer will respond with

\begin{verbatim}
  This is fun!
\end{verbatim}
Redirect this phrase into a file named fun.

4. Using the commands `who`, `who am i`, and `date`, append to the `fun` file (see Exercise 3 above) a list of the users currently logged onto the computer, your login, and the current date.

5. A hidden file has a name that begins with a period (.). Use the `cal` utility and the redirection operator (>) to create a file named `.hidden`, then use `ls` to list your files. Do you see the `.hidden` file? Now try the `ls -a` command. Does `.hidden` appear? What other hidden file entries do you see?

6. Many UNIX systems offer a utility named `file`, which classifies files according to their contents. The utility examines the file and tries to determine what kind of information it may contain. Some of the classifications used by `file` are:

```
  ascii text  c program text  commands
  data        directory      empty
  English text executable
```

Try out the `file` command on the files and directories in your system. Does `file` always classify files correctly?