Chapter 3

Exploring Linux Filesystems

Chapter Objectives

1. Navigate the Linux directory structure using relative and absolute pathnames.
2. Describe the various types of Linux files.
3. View filenames and file types.
4. Use shell wildcards to specify multiple filenames.
5. Display the contents of text files and binary files.
6. Search text files for regular expressions using `grep`.
7. Use the vi editor to manipulate text files.
8. Identify common alternatives to the vi editor.

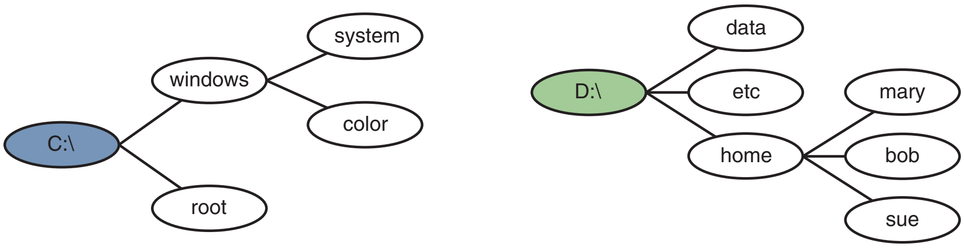
An understanding of the structure and commands surrounding the Linux filesystem is essential for effectively using Linux to manipulate data. In the first part of this chapter, you explore the Linux filesystem hierarchy by changing your position in the filesystem tree and listing filenames of various types. Next, you examine the shell wildcard metacharacters used to specify multiple filenames as well as view the contents of files using standard Linux commands. You then learn about the regular expression metacharacters used when searching for text within files and are introduced to the vi text editor and its alternatives.

The Linux Directory Structure

Fundamental to using the Linux operating system is an understanding of how Linux stores files on the filesystem. Typical Linux systems could have thousands of data and program files; thus, a structure that organizes those files is necessary to make it easier to find and manipulate data and run programs. Recall from the previous chapter that Linux uses a logical directory tree to organize files into **directories** (also known as folders). When a user stores files in a certain directory, the files are physically stored in the filesystem of a certain partition on a storage device (e.g., hard disk drive or SSD) inside the computer. Most people are familiar with the Windows operating system directory tree structure as shown in Figure 3-1; each filesystem on a storage device partition is referred to by a drive letter (such as C: or D:) and has a root directory (indicated by the \ character) containing subdirectories that together form a hierarchical tree.

It is important to describe directories in the directory tree properly; the **absolute pathname** to a file or directory is the full pathname of a certain file or directory starting from the root directory. In Figure 3-1, the absolute pathname for the color directory is C:\windows\color and the absolute

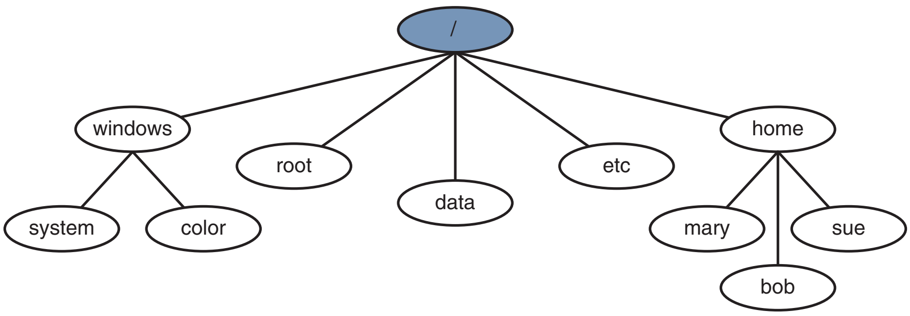
Figure 3-1 The Windows filesystem structure



pathname for the sue directory is D:\home\sue. In other words, you refer to C:\windows\color as the color directory below the windows directory below the root of C drive. Similarly, you refer to D:\home\sue as the sue directory below the home directory below the root of D drive.

Linux uses a similar directory structure, but with no drive letters. The structure contains a single root (referred to using the / character), with different filesystems on storage device partitions mounted (or attached) to different directories on this directory tree. The directory that each filesystem is mounted to is transparent to the user. An example of a sample Linux directory tree equivalent to the Windows sample directory tree shown in Figure 3-1 is shown in Figure 3-2. Note that the subdirectory named “root” in Figure 3-2 is different from the root (/) directory. You’ll learn more about the root subdirectory in the next section.

Figure 3-2 The Linux filesystem structure



In Figure 3-2, the absolute pathname for the color directory is /windows/color and the absolute pathname for the sue directory is /home/sue. In other words, you refer to the /windows/color directory as the color directory below the windows directory below the root of the system (the / character). Similarly, you refer to the /home/sue directory as the sue directory below the home directory below the root of the system.

Changing Directories

When you log into a Linux system, you are placed in your **home directory**, which is a place unique to your user account for storing personal files. Regular users usually have a home directory named after their user account under the /home directory, as in /home/sue. The root user, however, has a home directory called root under the root directory of the system (/root), as shown in Figure 3-2. Regardless of your user name, you can always refer to your own home directory using the **~ metacharacter**.

To confirm the system directory that you are currently in, simply observe the name at the end of the shell prompt or run the **`pwd` (print working directory) command** at a command-line prompt. If you are logged in as the root user, the following output is displayed on the terminal screen:

```
[root@server1 ~]# pwd
/root
[root@server1 ~]#_
```

However, if you are logged in as the user sue, you see the following output:

```
[sue@server1 ~]$ pwd
/home/sue
[sue@server1 ~]$_
```

To change directories, you can issue the **cd (change directory) command** with an argument specifying the destination directory. If you do not specify a destination directory, the cd command returns you to your home directory:

```
[root@server1 ~]# cd /home/mary
[root@server1 mary]# pwd
/home/mary
[root@server1 mary]# cd /etc
[root@server1 etc]# pwd
/etc
[root@server1 etc]# cd
[root@server1 ~]# pwd
/root
[root@server1 ~]#_
```

You can also use the ~ metacharacter to refer to another user's home directory by appending a user name at the end:

```
[root@server1 ~]# cd ~mary
[root@server1 mary]# pwd
/home/mary
[root@server1 mary]# cd ~
[root@server1 ~]# pwd
/root
[root@server1 ~]#_
```

In many of the examples discussed earlier, the argument specified after the cd command is an absolute pathname to a directory, meaning that the system has all the information it needs to find the destination directory because the pathname starts from the root (/) of the system. However, in most Linux commands, you can also use a relative pathname in place of an absolute pathname to reduce typing. A **relative pathname** is the pathname of a target file or directory relative to your current directory in the tree. To specify a directory below your current directory, refer to that directory by name (do not start the pathname with a / character). To refer to a directory one step closer to the root of the tree (also known as a **parent directory**), use two dots (..). An example of using relative pathnames to move around the directory tree is shown next:

```
[root@server1 ~]# cd /home/mary
[root@server1 mary]# pwd
/home/mary
[root@server1 mary]# cd ..
[root@server1 home]# pwd
/home
[root@server1 home]# cd mary
[root@server1 mary]# pwd
/home/mary
[root@server1 mary]#_
```

The preceding example used ".." to move up one parent directory and then used the word "mary" to specify the mary **subdirectory** relative to the current location in the tree; however, you can also move more than one level up or down the directory tree:

```
[root@server1 ~]# cd /home/mary
[root@server1 mary]# pwd
/home/mary
[root@server1 mary]# cd ../..
[root@server1 /]# pwd
/
[root@server1 /]# cd home/mary
[root@server1 mary]# pwd
/home/mary
[root@server1 mary]#_
```

Note 1

You can also use one dot (.) to refer to the current directory. Although this is not useful when using the `cd` command, you do use one dot later in this book.

Although absolute pathnames are straightforward to use as arguments to commands when specifying the location of a certain file or directory, relative pathnames can save you a great deal of typing and reduce the potential for error if your current directory is far away from the root directory. Suppose, for example, that the current directory is /home/sue/projects/acme/plans and you need to change to the /home/sue/projects/acme directory. Using an absolute pathname, you would type `cd /home/sue/projects/acme`; however, using a relative pathname, you only need to type `cd ..` to perform the same task because the /home/sue/projects/acme directory is one parent directory above the current location in the directory tree.

An alternate method for saving time when typing pathnames as arguments to commands is to use the **Tab-completion feature** of the BASH shell. To do this, type enough unique letters of a directory and press the Tab key to allow the BASH shell to find the intended file or directory being specified and fill in the appropriate information. If there is more than one possible match, the Tab-completion feature alerts you with a beep; pressing the Tab key again after this beep presents you with a list of possible files or directories.

Observe the directory structure in Figure 3-2. To use the Tab-completion feature to change the current directory to /home/sue, you type `cd /h` and then press the Tab key. This changes the previous characters on the terminal screen to display cd /home/ (the BASH shell was able to fill in the appropriate information because the /home directory is the only directory under the / directory that starts with the letter "h"). Then, you could add an s character to the command, so that the command line displays `cd /home/s`, and press the Tab key once again to allow the shell to fill in the remaining letters. This results in the command `cd /home/sue/` being displayed on the terminal screen (the sue directory is the only directory that begins with the s character under the /home directory). At this point, you can press Enter to execute the command and change the current directory to /home/sue.

Note 2

In addition to directories, the Tab-completion feature of the BASH shell can be used to specify the pathname to files and executable programs.

Viewing Files and Directories

The point of a directory structure is to organize files into an easy-to-use format. In order to locate the file you need to execute, view, or edit, you need to be able to display a list of the contents of a particular directory. You'll learn how to do that shortly, but first you need to learn about the various types of files and filenames, as well as the different commands used to select filenames for viewing.

File Types

Fundamental to viewing files and directories is a solid understanding of the various types of files present on most Linux systems. A Linux system can have several types of files; the most common include the following:

- Text files
- Binary data files
- Executable program files
- Directory files
- Linked files
- Special device files
- Named pipes and sockets

Most files on a Linux system that contain configuration information are **text files**. Another type of file is a program that exists on the filesystem before it is executed in memory to become a process. A program is typically associated with several supporting **binary data files** that store information such as common functions and graphics. In addition, directories themselves are actually files; they are special files that serve as placeholders to organize other files. When you create a directory, a file is placed on the filesystem to represent that directory.

Linked files are files that have an association with one another; they can represent the same data or they can point to another file (also known as a shortcut file). **Special device files** are less common than the other file types that have been mentioned, yet they are important for managing Linux because they represent different devices on the system, such as hard disk drives and SSDs. These device files are used in conjunction with commands that manipulate devices on the system; special device files are typically found only in the /dev directory and are discussed in later chapters of this book. As with special device files, **named pipe files** are less commonly used. Named pipes are files that pass information from one process in memory to another. One process writes to the file while another process reads from it to achieve this passing of information. Another variant of a named pipe file is a **socket file**, which allows a process on another computer to write to a file on your computer while another process on your computer reads from that file.

Filenames

Files are recognized by their **filenames**, which can include up to 255 characters, yet are rarely longer than 20 characters on most Linux systems. Filenames are typically composed of alphanumeric characters, the underscore (_) character, the dash (-) character, and the period (.) character.

Note 3

It is important to avoid using the shell metacharacters discussed in the previous chapter when naming files. Using a filename that contains a shell metacharacter as an argument to a Linux command might produce unexpected results.

Note 4

Filenames that start with a period (.) are referred to as hidden files. You need to use a special command to display them in a file list. This command is discussed later in this chapter.

Filenames used by the Windows operating system typically end with a period and three characters that identify the file type—for example, document.txt (a text file) and program.exe (an **executable program** file). While most files on a Linux filesystem do not follow this pattern, some do contain characters at the end of the filename that indicate the file type. These characters are commonly referred to as **filename extensions**. Table 3-1 lists common examples of filename extensions and their associated file types.

Table 3-1 Common filename extensions

Metacharacter	Description
.bin	Binary executable program files (similar to .exe files within Windows)
.c	C programming language source code files
.cc, .cpp	C++ programming language source code files
.html, .htm	HTML (Hypertext Markup Language) files
.ps	Files formatted for printing with postscript
.txt	Text files
.tar	Archived files (contain other files within)
.gz, .bz2, .xz, .Z	Compressed files
.tar.gz, .tgz, .tar.bz2, .tar.xz, .tar.Z	Compressed archived files
.conf, .cfg	Configuration files (contain text)
.so	Shared object (programming library) files
.o, ko	Compiled object files
.pl	PERL (Practical Extraction and Report Language) programs
.tcl	Tcl (Tool Command Language) programs
.jpg, .jpeg, .png, .tiff, .xpm, .gif	Binary files that contain graphical images
.sh	Shell scripts (contain text that is executed by the shell)

Listing Files

Linux hosts a variety of commands that can be used to display files and their types in various directories on filesystems. By far, the most common method for displaying files is to use the **ls command**. Following is an example of a file listing in the root user's home directory:

```
[root@server1 ~]# pwd
/root
[root@server1 ~]# ls
current myprogram project project12 project2 project4
Desktop myscript project1 project13 project3 project5
[root@server1 ~]#_
```

Note 5

The files listed previously and discussed throughout this chapter are for example purposes only. The hands-on projects use different files.

The ls command displays all the files in the current directory in columnar format; however, you can also pass an argument to the ls command indicating the directory to list if the current directory listing is not required. In the following example, the files are listed under the /home/bob directory without changing the current directory.

```
[root@server1 ~]# pwd
/root
[root@server1 ~]# ls /home/bob
```

```
assignment1 file1 letter letter2 project1
[root@server1 ~]#_
```

Note 6

When running the `ls` command, you will notice that files of different types are often represented as different colors; however, the specific colors used to represent files of certain types might vary depending on your terminal settings. As a result, do not assume color alone indicates the file type.

Note 7

Windows uses the `dir` command to list files and directories; to simplify the learning of Linux for Windows users, there is a `dir` command in Linux, which is either a copy of, or shortcut to, the `ls` command.

Recall from the previous chapter that you can use options to alter the behavior of commands. To view a list of files and their type, use the `–F` option to the `ls` command:

```
[root@server1 ~]# pwd
/root
[root@server1 ~]# ls -F
current@   myprogram*  project    project12  project2  project4
Desktop/   myscript*   project1   project13  project3  project5
[root@server1 ~]#_
```

The `ls –F` command appends a special character at the end of each filename displayed to indicate the type of file. In the preceding output, note that the filenames current, Desktop, myprogram, and myscript have special characters appended to their names. The @ symbol indicates a symbolically linked file (a shortcut to another file), the * symbol indicates an executable file, the / indicates a subdirectory, the = character indicates a socket, and the | character indicates a named pipe. Other file types do not have a special character appended to them and could be text files, binary data files, or special device files.

Note 8

It is a common convention to name directories starting with an uppercase letter, such as the D in the Desktop directory shown in the preceding output. This allows you to quickly determine which names refer to directories when running the `ls` command without any options that specify file type.

Although the `ls –F` command is a quick way of getting file type information in an easy-to-read format, at times you need to obtain more detailed information about each file. The `ls –l` command can be used to provide a long listing for each file in a certain directory.

```
[root@server1 ~]# pwd
/root
[root@server1 ~]# ls -l
total 548
lrwxrwxrwx    1 root    root          9 Apr  7 09:56 current -> project12
drwx------    3 root    root       4096 Mar 29 10:01 Desktop
-rwxr-xr-x    1 root    root     519964 Apr  7 09:59 myprogram
-rwxr-xr-x    1 root    root         20 Apr  7 09:58 myscript
-rw-r--r--    1 root    root         71 Apr  7 09:58 project
-rw-r--r--    1 root    root         71 Apr  7 09:59 project1
-rw-r--r--    1 root    root         71 Apr  7 09:59 project12
-rw-r--r--    1 root    root          0 Apr  7 09:56 project13
```

```
-rw-r--r--    1 root   root        71 Apr  7 09:59 project2
-rw-r--r--    1 root   root        90 Apr  7 10:01 project3
-rw-r--r--    1 root   root        99 Apr  7 10:01 project4
-rw-r--r--    1 root   root       108 Apr  7 10:01 project5
[root@server1 ~]#_
```

Each file listed in the preceding example has eight components of information listed in columns from left to right:

1. A file type character:
   - The d character represents a directory.
   - The l character represents a symbolically linked file (discussed in Chapter 4).
   - The b or c character represents a special device file (discussed in Chapter 5).
   - The n character represents a named pipe.
   - The s character represents a socket.
   - The – character represents all other file types (text files, binary data files).
2. A list of permissions on the file (also called the mode of the file and discussed in Chapter 4).
3. A hard link count (discussed in Chapter 4).
4. The owner of the file (discussed in Chapter 4).
5. The group owner of the file (discussed in Chapter 4).
6. The file size.
7. The most recent modification time of the file (or creation time if the file was not modified following creation).
8. The filename. Some files are shortcuts or pointers to other files and indicated with an arrow, as with the file called "current" in the preceding output; these are known as symbolic links and are discussed in Chapter 4.

For the file named "project" in the previous example, you can see that this file is a regular file because its long listing begins with a – character, the permissions on the file are rw-r--r--, the hard link count is 1, the owner of the file is the root user, the group owner of the file is the root group, the size of the file is 71 bytes, and the file was modified last on April 7 at 9:58 a.m.

Note 9

If SELinux is enabled on your system, you may also notice a period (.) immediately following the permissions on a file or directory that is managed by SELinux. SELinux will be discussed in Chapter 14.

Note 10

On most Linux systems, a shortcut to the `ls` command can be used to display the same columns of information as the `ls -l` command. Some users prefer to use this shortcut, commonly known as an alias, which is invoked when a user types `ll` at a command prompt. This is known as the **`ll` command**.

The `ls -F` and `ls -l` commands are valuable to a user who wants to display file types; however, neither of these commands can display all file types using special characters. To display the file type of any file, you can use the **`file` command**; you give the `file` command an argument specifying what file to analyze. You can also pass multiple files as arguments or use the * metacharacter to refer to all files in the current directory. An example of using the `file` command in the root user's home directory is:

```
[root@server1 ~]# pwd
/root
```

```
[root@server1 ~]# ls
current  myprogram  project   project12  project2  project4
Desktop  myscript   project1  project13  project3  project5
[root@server1 ~]# file Desktop
Desktop:   directory
[root@server1 ~]# file project Desktop
project:   ASCII text
Desktop:   directory
[root@server1 ~]# file *
Desktop:   directory
current:   symbolic link to project12
myprogram: ELF 64-bit LSB pie executable, x86_64, version 1 (SYSV),
dynamically linked, for GNU/Linux 3.2.0, stripped
myscript:  Bourne-Again shell script text executable
project:   ASCII text
project1:  ASCII text
project12: ASCII text
project13: empty
project2:  ASCII text
project3:  ASCII text
project4:  ASCII text
project5:  ASCII text
[root@server1 ~]#_
```

As shown in the preceding example, the `file` command can also identify the differences between types of executable files. The myscript file is a text file that contains executable commands (also known as a **shell script**), whereas the myprogram file is a 64-bit executable compiled program for the x86_64 CPU platform. The `file` command also identifies empty files such as project13 in the previous example.

You can also use the **`stat` command** to display additional details for a file, including the date and time a file was created (the birth time), as well as the last time the file was accessed, or its contents modified, or file information changed. Following is an example of using the `stat` command to view these details for the project file:

```
[root@server1 ~]# stat project
  File: project
  Size: 71            Blocks: 2          IO Block: 4096   regular file
Device: 8,3     Inode: 1179655     Links: 1
Access: (0644/-rw-r--r--) Uid: (0/root) Gid: (0/root)
Context: system_u:object_r:admin_home_t:s0
Access: 2023-09-03 12:15:40.462610154 -0400
Modify: 2023-09-02 22:00:11.840345812 -0400
Change: 2023-09-02 22:00:11.840345812 -0400
 Birth: 2023-09-01 16:55:46.462610154 -0400
[root@server1 ~]#_
```

Some filenames inside each user's home directory represent important configuration files or program directories. Because these files are rarely edited by the user and can clutter the listing of files, they are normally hidden from view when using the `ls` and `file` commands. Recall that filenames for hidden files start with a period character (.). To view them, pass the `–a` option to the `ls` command. Some hidden files that are commonly seen in the root user's home directory are shown next:

```
[root@server1 ~]# ls
current  myprogram  project   project12  project2  project4
```

```
Desktop   myscript   project1   project13   project3   project5
[root@server1 ~]# ls -a
.               .bash_profile  current    project    project2  .pki
..              .bashrc        Desktop    project1   project3  .tcshrc
.bash_history   .cache         myprogram  project12  project4
.bash_logout    .config        myscript   project13  project5
[root@server1 ~]#_
```

As discussed earlier, the (.) character refers to the current working directory and the (..) character refers to the parent directory relative to your current location in the directory tree. Each of these pointers is seen as a special (or fictitious) file when using the `ls -a` command, as each starts with a period.

You can also specify several options simultaneously for most commands on the command line and receive the combined functionality of all the options. For example, to view all hidden files and their file types, you could combine the `-a` and `-F` options:

```
[root@server1 ~]# ls -aF
.               .bash_profile  current@   project    project2  .pki/
..              .bashrc        Desktop/   project1   project3  .tcshrc
.bash_history   .cache/        myprogram  project12  project4
.bash_logout    .config/       myscript   project13  project5
[root@server1 ~]#_
```

To view files and subdirectories under a directory, you can add the recursive (`-R`) option to the to the `ls` command, or use the **`tree` command**. The following example uses these commands to display the files and subdirectories underneath the Desktop directory:

```
[root@server1 ~]# ls -R Desktop
Desktop/:
project-tracking  social  stuff

Desktop/project-tracking:
project1.xlsx  project2.xlsx  project3.xlsx

Desktop/social:
confirmations.docx  event-poster.pdf  events-calendar.cip

Desktop/stuff:
quotes.txt  todo.txt
[root@server1 ~]# tree Desktop
Desktop/
├── project-tracking
│   ├── project1.xlsx
│   ├── project2.xlsx
│   └── project3.xlsx
├── social
│   ├── confirmations.docx
│   ├── event-poster.pdf
│   └── events-calendar.cip
└── stuff
    ├── quotes.txt
    └── todo.txt

3 directories, 8 files
[root@server1 ~]#_
```

Note 11

To instead display only the subdirectories under the Desktop directory, you could add the `-d` option to the `ls` and `tree` commands shown in the previous example.

While the `ls` options discussed in this section (`-l`, `-F`, `-a`, `-R`, `-d`) are the most common you would use when navigating the Linux directory tree, there are many more available. Table 3-2 lists the most common of these options and their descriptions.

Table 3-2 Common options to the `ls` command

Option	Description
`-a` `--all`	Lists all filenames
`-A` `--almost-all`	Lists most filenames (excludes the . and .. special files)
`-C`	Lists filenames in column format
`--color=none`	Lists filenames without color
`-d` `--directory`	Lists directory names instead of their contents
`-f`	Lists all filenames without sorting
`-F` `--classify`	Lists filenames classified by file type
`--full-time`	Lists filenames in long format and displays the full modification time
`-l`	Lists filenames in long format
`-lhs` `-l --human-readable --size`	Lists filenames in long format with human-readable (easy-to-read) file sizes

`-lG`
`-l --no-group`
`-o`
	Lists filenames in long format but omits the group information

`-r` `--reverse`	Lists filenames reverse sorted
`-R` `--recursive`	Lists filenames in the specified directory and all subdirectories
`-s` `--size`	Lists filenames and their associated sizes in blocks (on most systems, each block is 1 KB)
`-S`	Lists filenames sorted by file size (largest first)
`-t`	Lists filenames sorted by modification time (newest first)
`-U`	Lists selected filenames without sorting
`-x`	Lists filenames in rows rather than in columns

Wildcard Metacharacters

In the previous section, you saw that the * metacharacter matches all the files in the current directory, much like a wildcard matches certain cards in a card game. As a result, the * metacharacter is called a **wildcard metacharacter**. Wildcard metacharacters can simplify commands that specify more than one filename on the command line, as you saw with the `file` command earlier. They match certain portions of filenames or the entire filename itself. Because they are interpreted by the shell, they can be used with most common Linux filesystem commands, including those that have already been mentioned (`ls`, `file`, `stat`, `tree`, and `cd`). Table 3-3 displays a list of wildcard metacharacters and their descriptions.

Table 3-3 Wildcard Metacharacters

Metacharacter	Description
*	Matches 0 or more characters in a filename
?	Matches 1 character in a filename
[aegh]	Matches 1 character in a filename—provided this character is either an a, e, g, or h
[a-e]	Matches 1 character in a filename—provided this character is either an a, b, c, d, or e
[!a-e]	Matches 1 character in a filename—provided this character is NOT an a, b, c, d, or e

Wildcards can be demonstrated using the `ls` command. Examples of using wildcard metacharacters to narrow the listing produced by the `ls` command are shown next.

```
[root@server1 ~]# ls
current    myprogram  project   project12  project2  project4
document1  myscript   project1  project13  project3  project5
[root@server1 ~]# ls project*
project  project1  project12  project13  project2  project3  project4  project5
[root@server1 ~]# ls project?
project1  project2  project3  project4  project5
[root@server1 ~]# ls project??
project12  project13
[root@server1 ~]# ls project[135]
project1  project3  project5
[root@server1 ~]# ls project[!135]
project2  project4
[root@server1 ~]#_
```

Note 12

Using wildcards to match multiple files or directories within a command is often called **file globbing**.

Displaying the Contents of Text Files

So far, this chapter has discussed commands that can be used to navigate the Linux directory structure and view filenames and file types; it is usual now to display the contents of these files. By far, the most common file type that Linux users display is text files. These files are usually shell scripts, source code files, user documents, or configuration files for Linux components or services. To view an entire text file on the terminal screen (also referred to as **concatenation**), you can use the **`cat` command**. The following is an example of using the `cat` command to display the contents of the fictitious file project4:

```
[root@server1 ~]# ls
current    myprogram  project   project12  project2  project4
document1  myscript   project1  project13  project3  project5
[root@server1 ~]# cat project4
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
[root@server1 ~]#_
```

You can also use the `cat` command to display the line number of each line in the file in addition to the contents by passing the `–n` option to the `cat` command. In the following example, the number of each line in the project4 file is displayed:

```
[root@server1 ~]# cat –n project4
     1  Hi there, I hope this day finds you well.
     2
     3  Unfortunately, we were not able to make it to your dining
     4  room this year while vacationing in Algonquin Park - I
     5  especially wished to see the model of the Highland Inn
     6  and the train station in the dining room.
     7
     8  I have been reading on the history of Algonquin Park but
     9  nowhere could I find a description of where the Highland
    10  Inn was originally located on Cache Lake.
    11
    12  If it is no trouble, could you kindly let me know such that
    13  I need not wait until next year when I visit your lodge?
    14
    15  Regards,
    16  Mackenzie Elizabeth
[root@server1 ~]#_
```

In some cases, you might want to display the contents of a certain text file in reverse order, which is useful when displaying files that have text appended to them continuously by system services. These files, also known as **log files**, contain the most recent entries at the bottom of the file. To display a file in reverse order, use the **`tac` command** (tac is cat spelled backwards), as shown next with the file project4:

```
[root@server1 ~]# tac project4
Mackenzie Elizabeth
Regards,
```

```
I need not wait until next year when I visit your lodge?
If it is no trouble, could you kindly let me know such that

Inn was originally located on Cache Lake.
nowhere could I find a description of where the Highland
I have been reading on the history of Algonquin Park but

and the train station in the dining room.
especially wished to see the model of the Highland Inn
room this year while vacationing in Algonquin Park - I
Unfortunately, we were not able to make it to your dining

Hi there, I hope this day finds you well.
[root@server1 ~]#_
```

If the file displayed is very large and you only want to view the first few lines of it, you can use the **head command**. The head command displays the first 10 lines (including blank lines) of a text file to the terminal screen but can also take a numeric option specifying a different number of lines to display. The following shows an example of using the head command to view the top of the project4 file:

```
[root@server1 ~]# head project4
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.
[root@server1 ~]# head -3 project4
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
[root@server1 ~]#_
```

Just as the head command displays the beginning of text files, the **tail command** can be used to display the end of text files. By default, the tail command displays the final 10 lines of a file, but it can also take a numeric option specifying the number of lines to display on the terminal screen, as shown in the following example with the project4 file:

```
[root@server1 ~]# tail project4

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
[root@server1 ~]# tail -2 project4
```

```
Regards,
Mackenzie Elizabeth
[root@server1 ~]#_
```

Note 13

The `-f` option to the `tail` command displays the final 10 lines of a file but keeps the file open so that you can see when additional lines are added to the end of the file. This is especially useful when viewing log files while troubleshooting a system problem. For example, you could run the `tail -f logfile` command in one terminal while performing troubleshooting actions in another terminal. After performing each troubleshooting action, the associated events will be displayed in the terminal running the `tail` command.

Although some text files are small enough to be displayed completely on the terminal screen, you might encounter text files that are too large to fit in a single screen. In this case, the `cat` command sends the entire file contents to the terminal screen; however, the screen only displays as much of the text as it has room for. To display a large text file in a page-by-page fashion, you need to use the `more` and `less` commands.

The **`more` command** gets its name from the `pg` command once used on UNIX systems. The `pg` command displayed a text file page-by-page on the terminal screen, starting at the beginning of the file; pressing the spacebar or Enter key displays the next page, and so on. The `more` command does more than `pg` did, because it displays the next complete page of a text file if you press the spacebar but displays only the next line of a text file if you press Enter. In that way, you can browse the contents of a text file page-by-page or line-by-line. The fictitious file project5 is an excerpt from Shakespeare's tragedy *Macbeth* and is too large to be displayed fully on the terminal screen using the `cat` command. Using the `more` command to view its contents results in the following output:

```
[root@server1 ~]# more project5
Go bid thy mistress, when my drink is ready,
She strike upon the bell. Get thee to bed.
Is this a dagger which I see before me,
The handle toward my hand? Come, let me clutch thee.
I have thee not, and yet I see thee still.
Art thou not, fatal vision, sensible
To feeling as to sight? or art thou but
A dagger of the mind, a false creation,
Proceeding from the heat-oppressed brain?
I see thee yet, in form as palpable
As this which now I draw.
Thou marshall'st me the way that I was going;
And such an instrument I was to use.
Mine eyes are made the fools o' the other senses,
Or else worth all the rest; I see thee still,
And on thy blade and dudgeon gouts of blood,
Which was not so before. There's no such thing:
It is the bloody business which informs
Thus to mine eyes. Now o'er the one halfworld
Nature seems dead, and wicked dreams abuse
The curtain'd sleep; witchcraft celebrates
Pale Hecate's offerings, and wither'd murder,
Alarum'd by his sentinel, the wolf,
--More--(71%)
```

As you can see in the preceding output, the more command displays the first page without returning you to the shell prompt. Instead, the more command displays a prompt at the bottom of the terminal screen that indicates how much of the file is displayed on the screen as a percentage of the total file size. In the preceding example, 71 percent of the project5 file is displayed. At this prompt, you can press the spacebar to advance one whole page, or you can press the Enter key to advance to the next line. In addition, the more command allows other user interactions at this prompt. Pressing the h character at the prompt displays a help screen, which is shown in the following output, and pressing the q character quits the more command completely without viewing the remainder of the file.

```
--More--(71%)
Most commands optionally preceded by integer argument k. Defaults in
brackets. Star (*) indicates argument becomes new default.
-----------------------------------------------------------------
<space>                 Display next k lines of text
z                       Display next k lines of text
<return>                Display next k lines of text [1]
d or ctrl-D             Scroll k lines [current scroll size, initially 11]
q or Q or <interrupt> Exit from more
s                       Skip forward k lines of text [1]
f                       Skip forward k screenfuls of text [1]
b or ctrl-B             Skip backward k screenfuls of text [1]
'                       Go to place where previous search started
=                       Display current line number
/<regular expression>  Search for kth occurrence of expression [1]
n                       Search for kth occurrence of last r.e [1]
!<cmd> or :!<cmd>       Execute <cmd> in a subshell
v                       Start up /usr/bin/vi at current line
ctrl-L                  Redraw screen
:n                      Go to kth next file [1]
:p                      Go to kth previous file [1]
:f                      Display current filename and line number
.                       Repeat previous command
-----------------------------------------------------------------
--More—(71%)
```

Just as the more command was named for allowing more user functionality, the **less command** is named for doing more than the more command (remember that "less is more," more or less). Like the more command, the less command can browse the contents of a text file page-by-page by pressing the spacebar and line-by-line by pressing the Enter key; however, you can also use the arrow keys on the keyboard to scroll up and down the contents of the file. The output of the less command, when used to view the project5 file, is as follows:

```
[root@server1 ~]# less project5
Go bid thy mistress, when my drink is ready,
She strike upon the bell. Get thee to bed.
Is this a dagger which I see before me,
The handle toward my hand? Come, let me clutch thee.
I have thee not, and yet I see thee still.
Art thou not, fatal vision, sensible
To feeling as to sight? or art thou but
A dagger of the mind, a false creation,
Proceeding from the heat-oppressed brain?
I see thee yet, in form as palpable
As this which now I draw.
```

```
Thou marshall'st me the way that I was going;
And such an instrument I was to use.
Mine eyes are made the fools o' the other senses,
Or else worth all the rest; I see thee still,
And on thy blade and dudgeon gouts of blood,
Which was not so before. There's no such thing:
It is the bloody business which informs
Thus to mine eyes. Now o'er the one halfworld
Nature seems dead, and wicked dreams abuse
The curtain'd sleep; witchcraft celebrates
Pale Hecate's offerings, and wither'd murder,
Alarum'd by his sentinel, the wolf,
Whose howl's his watch, thus with his stealthy pace.
project5
```

Like the `more` command, the `less` command displays a prompt at the bottom of the file using the : character or the filename of the file being viewed (project5 in our example), yet the `less` command contains more keyboard shortcuts for searching out text within files. At the prompt, you can press the `h` key to obtain a help screen or the `q` key to quit. The first help screen for the `less` command is shown next:

```
                   SUMMARY OF LESS COMMANDS

  Commands marked with * may be preceded by a number, N.
  Notes in parentheses indicate the behavior if N is given.
  A key preceded by a caret indicates the Ctrl key; thus ^K is ctrl-K.

h  H                 Display this help.
q  :q  Q  :Q  ZZ     Exit.
---------------------------------------------------------------------

                         MOVING

e  ^E  j  ^N  CR * Forward one line (or N lines).
y  ^Y  k  ^K  ^P * Backward one line (or N lines).
f  ^F  ^V  SPACE * Forward one window (or N lines).
b  ^B  ESC-v     * Backward one window (or N lines).
z                * Forward one window (and set window to N).
w                * Backward one window (and set window to N).
ESC-SPACE        * Forward one window, but don't stop at end-of-file.
d  ^D            * Forward one half-window (and set half-window to N).
u  ^U            * Backward one half-window (and set half-window to N).
ESC-) RightArrow * Right one half screen width (or N positions).

HELP -- Press RETURN for more, or q when done
```

The `more` and `less` commands can also be used in conjunction with the output of commands if that output is too large to fit on the terminal screen. To do this, use the | metacharacter after the command, followed by either the `more` or `less` command, as follows:

```
[root@server1 ~]# cd /etc
[root@server1 etc]# ls -l | more
total 3688
-rw-r--r--   1 root    root     15276 Mar 22 12:20 a2ps.cfg
```

```
-rw-r--r--    1 root    root       2562 Mar 22 12:20 a2ps-site.cfg
drwxr-xr-x    4 root    root       4096 Jun 11 08:45 acpi
-rw-r--r--    1 root    root         46 Jun 16 16:42 adjtime
drwxr-xr-x    2 root    root       4096 Jun 11 08:47 aep
-rw-r--r--    1 root    root        688 Feb 17 00:35 aep.conf
-rw-r--r--    1 root    root        703 Feb 17 00:35 aeplog.conf
drwxr-xr-x    4 root    root       4096 Jun 11 08:47 alchemist
-rw-r--r--    1 root    root       1419 Jan 26 10:14 aliases
-rw-r-----    1 root    smmsp     12288 Jun 17 13:17 aliases.db
drwxr-xr-x    2 root    root       4096 Jun 11 11:11 alternatives
drwxr-xr-x    3 amanda  disk       4096 Jun 11 10:16 amanda
-rw-r--r--    1 amanda  disk          0 Mar 22 12:28 amandates
-rw-------    1 root    root        688 Mar  4 22:34 amd.conf
-rw-r-----    1 root    root        105 Mar  4 22:34 amd.net
-rw-r--r--    1 root    root        317 Feb 15 14:33 anacrontab
-rw-r--r--    1 root    root        331 May  5 08:07 ant.conf
-rw-r--r--    1 root    root       6200 Jun 16 16:42 asound.state
drwxr-xr-x    3 root    root       4096 Jun 11 10:37 atalk
-rw-------    1 root    root          1 May  5 13:39 at.deny
-rw-r--r--    1 root    root        325 Apr 14 13:39 auto.master
-rw-r--r--    1 root    root        581 Apr 14 13:39 auto.misc
--More--
```

In the preceding example, the output of the `ls -l` command was redirected to the `more` command, which displays the first page of output on the terminal. You can then advance through the output page-by-page or line-by-line. This type of redirection is discussed in Chapter 7.

Note 14

You can also use the **`diff` command** to identify the content differences between two text files, which is often useful when comparing revisions of source code or configuration files on a Linux system. For example, the `diff file1 file2` command would list the lines that are different between file1 and file2.

Displaying the Contents of Binary Files

It is important to employ text file commands, such as `cat`, `tac`, `head`, `tail`, `more`, and `less`, only on files that contain text; otherwise, you might find yourself with random output on the terminal screen or even a dysfunctional terminal. To view the contents of binary files, you typically use the program that was used to create the file. However, some commands can be used to safely display the contents of most binary files. The **`strings` command** searches for text characters in a binary file and outputs them to the screen. In many cases, these text characters might indicate what the binary file is used for. For example, to find the text characters inside the /bin/echo binary executable program page-by-page, you could use the following command:

```
[root@server1 ~]# strings /bin/echo | more
/lib/ld-linux.so.2
PTRh|
<nt7<e
|[^_]
[^_]
[^_]
Try '%s --help' for more information.
Usage: %s [OPTION]... [STRING]...
```

```
Echo the STRING(s) to standard output.
  -n         do not output the trailing newline
  -e         enable interpretation of the backslash-escaped characters
             listed below
  -E         disable interpretation of those sequences in STRINGs
  --help     display this help and exit
  --version  output version information and exit
Without -E, the following sequences are recognized and interpolated:
  \NNN   the character whose ASCII code is NNN (octal)
  \\     backslash
--More--
```

Although this output might not be easy to read, it does contain portions of text that can point a user in the right direction to find out more about the /bin/echo command. Another command that is safe to use on binary files and text files is the **od command**, which displays the contents of the file in octal format (numeric base 8 format). An example of using the od command to display the first five lines of the file project4 is shown in the following example:

```
[root@server1 ~]# od project4 | head -5
0000000 064510 072040 062550 062562 020054 020111 067550 062560
0000020 072040 064550 020163 060544 020171 064546 062156 020163
0000040 067571 020165 062567 066154 006456 006412 052412 063156
0000060 071157 072564 060556 062564 074554 073440 020145 062567
0000100 062562 067040 072157 060440 066142 020145 067564 066440
[root@server1 ~]#_
```

Note 15

You can use the –x option to the od command to display a file in hexadecimal format (numeric base 16 format).

Searching for Text within Files

Recall that Linux was modeled after the UNIX operating system. The UNIX operating system is often referred to as the "grandfather" of all operating systems because it is over 40 years old and has formed the basis for most advances in computing technology. The major use of the UNIX operating system in the past 40 years involved simplifying business and scientific management through database applications. As a result, many commands (referred to as **text tools**) were developed for the UNIX operating system that could search for and manipulate text, such as database information, in many advantageous ways. A set of text wildcards was also developed to ease the searching of specific text information. These text wildcards are called **regular expressions (regexp)** and are recognized by text tools, as well as most modern programming languages, such as Python and C++.

Because Linux is a close relative of the UNIX operating system, these text tools and regular expressions are available to Linux as well. By combining text tools, a typical Linux system can search for and manipulate data in almost every way possible (as you will see later). As a result, regular expressions and the text tools that use them are frequently used today.

Regular Expressions

As mentioned earlier, regular expressions allow you to specify a certain pattern of text within a text document. They work similarly to wildcard metacharacters in that they are used to match characters, yet they have many differences:

- Wildcard metacharacters are interpreted by the shell, whereas regular expressions are interpreted by a text tool program.

- Wildcard metacharacters match characters in filenames (or directory names) on a Linux filesystem, whereas regular expressions match characters *within* text files on a Linux filesystem.
- Wildcard metacharacters typically have different definitions than regular expression metacharacters.
- More regular expression metacharacters are available than wildcard metacharacters.

In addition, regular expression metacharacters are divided into two categories: common (basic) regular expressions and extended regular expressions. Common regular expressions are available to most text tools; however, extended regular expressions are less common and available in only certain text tools. Table 3-4 shows definitions and examples of some common and extended regular expressions.

Table 3-4 Regular expressions

Regular Expression	Description	Example	Type
`*`	Matches 0 or more occurrences of the previous character	letter* matches lette, letter, letterr, letterrrr, letterrrrr, and so on	Common
`?`	Matches 0 or 1 occurrences of the previous character	letter? matches lette, letter	Extended
`+`	Matches 1 or more occurrences of the previous character	letter+ matches letter, letterr, letterrrr, letterrrrr, and so on	Extended
`. (period)`	Matches 1 character of any type	letter. matches lettera, letterb, letterc, letter1, letter2, letter3, and so on	Common
`[...]`	Matches one character from the range specified within the braces	letter[1238] matches letter1, letter2, letter3, and letter8; letter[a-c] matches lettera, letterb, and letterc	Common
`[^...]`	Matches one character NOT from the range specified within the braces	letter[^1238] matches letter4, letter5, letter6, lettera, letterb, and so on (any character except 1, 2, 3, or 8)	Common
`{ }`	Matches a specific number or range of the previous character	letter{3} matches letterrr, whereas letter {2,4} matches letterr, letterrr, and letterrrr	Extended
`^`	Matches the following characters if they are the first characters on the line	^letter matches letter if letter is the first set of characters in the line	Common
`$`	Matches the previous characters if they are the last characters on the line	letter$ matches letter if letter is the last set of characters in the line	Common
`(... | ...)`	Matches either of two sets of characters	(mother|father) matches the word "mother" or "father"	Extended

The grep Command

The most common way to search for information using regular expressions is the `grep` command. The **`grep` command** (the command name is short for global regular expression print) is used to display lines in a text file that match a certain common regular expression. To display lines of text that match extended regular expressions, you must use the **`egrep` command** (or the `-E` option to the `grep` command). In addition, the **`fgrep` command** (or the `-F` option to the `grep` command) does not interpret any regular expressions and consequently returns results much faster. Take, for example, the project4 file shown earlier:

```
[root@server1 ~]# cat project4
Hi there, I hope this day finds you well.
```

```
Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
[root@server1 ~]#_
```

The grep command requires two arguments at minimum, the first argument specifies which text to search for, and the remaining arguments specify the files to search. If a pattern of text is matched, the grep command displays the entire line on the terminal screen. For example, to list only those lines in the file project4 that contain the words "Algonquin Park," enter the following command:

```
[root@server1 ~]# grep "Algonquin Park" project4
room this year while vacationing in Algonquin Park - I
I have been reading on the history of Algonquin Park but
[root@server1 ~]#_
```

To return the lines that do not contain the text "Algonquin Park," you can use the -v option of the grep command to reverse the meaning of the previous command:

```
[root@server1 ~]# grep -v "Algonquin Park" project4
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
especially wished to see the model of the Highland Inn
and the train station in the dining room.

nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
[root@server1 ~]#_
```

Keep in mind that the text being searched is case sensitive; to perform a case-insensitive search, use the -i option to the grep command:

```
[root@server1 ~]# grep "algonquin park" project4
[root@server1 ~]#_
[root@server1 ~]# grep -i "algonquin park" project4
room this year while vacationing in Algonquin Park - I
I have been reading on the history of Algonquin Park but
[root@server1 ~]#_
```

Another important note to keep in mind regarding text tools such as `grep` is that they match only patterns of text; they are unable to discern words or phrases unless they are specified. For example, if you want to search for the lines that contain the word "we," you can use the following `grep` command:

```
[root@server1 ~]# grep "we" project4
Hi there, I hope this day finds you well.
Unfortunately, we were not able to make it to your dining
[root@server1 ~]#_
```

However, notice from the preceding output that the first line displayed does not contain the word "we"; the word "well" contains the text pattern "we" and is displayed as a result. To display only lines that contain the word "we," you can type the following to match the letters "we" surrounded by space characters:

```
[root@server1 ~]# grep " we " project4
Unfortunately, we were not able to make it to your dining
[root@server1 ~]#_
```

All of the previous `grep` examples did not use regular expression metacharacters to search for text in the project4 file. Some examples of using regular expressions (see Table 3-4) when searching this file are shown throughout the remainder of this section.

To view lines that contain the word "toe" or "the" or "tie," you can enter the following command:

```
[root@server1 ~]# grep " t.e " project4
especially wished to see the model of the Highland Inn
and the train station in the dining room.
I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
[root@server1 ~]#_
```

To view lines that start with the word "I," you can enter the following command:

```
[root@server1 ~]# grep "^I " project4
I have been reading on the history of Algonquin Park but
I need not wait until next year when I visit your lodge?
[root@server1 ~]#_
```

To view lines that contain the text "lodge" or "Lake," you need to use an extended regular expression and the `egrep` command, as follows:

```
[root@server1 ~]# egrep "(lodge|Lake)" project4
Inn was originally located on Cache Lake.
I need not wait until next year when I visit your lodge?
[root@server1 ~]#_
```

Editing Text Files

Recall that most system configuration is stored in text files, as are shell scripts and program source code. Consequently, most Linux distributions come with an assortment of text editor programs that you can use to modify the contents of text files. Text editors come in two varieties: editors that can be used on the command line, including vi (vim), nano, and Emacs, and editors that can be used in a desktop environment, including Emacs (graphical version) and gedit.

The vi Editor

The **vi editor** (pronounced "vee eye") is one of the oldest and most popular visual text editors available for UNIX operating systems. Its Linux equivalent (known as vim, which is short for "vi improved") is equally popular and widely considered the standard Linux text editor. Although the vi editor is not the easiest of the editors to use when editing text files, it has the advantage of portability. A Fedora Linux user who is proficient in using the vi editor will find editing files on all other UNIX and Linux systems easy because the interface and features of the vi editor are nearly identical across Linux and UNIX systems. In addition, the vi editor supports regular expressions and can perform over 1,000 different functions for the user.

To open an existing text file for editing, you can type `vi filename` (or `vim filename`), where *filename* specifies the file to be edited. To open a new file for editing, type `vi` or `vim` at the command line:

```
[root@server1 ~]# vi
```

The vi editor then runs interactively and replaces the command-line interface with the following output:

```
~
~
~
~                              VIM - Vi IMproved
~
~                               version 8.2.4621
~                           by Bram Moolenaar et al.
~                      Modified by <bugzilla@redhat.com>
~                 Vim is open source and freely distributable
~
~                           Sponsor Vim development!
~                type  :help sponsor<Enter>    for information
~
~                type  :q<Enter>               to exit
~                type  :help<Enter>  or  <F1>  for on-line help
~                type  :help version8<Enter>   for version info
~
~
~
```

The tilde (~) characters on the left indicate the end of the file; they are pushed further down the screen as you enter text. The vi editor is called a bimodal editor because it functions in one of two modes: **command mode** and **insert mode**. The vi editor opens command mode, in which you must use the keyboard to perform functions, such as deleting text, copying text, saving changes to a file, and exiting the vi editor. To insert text into the document, you must enter insert mode by typing one of the characters listed in Table 3-5. One such method to enter insert mode is to type the `i` key while in command mode; the vi editor then displays INSERT at the bottom of the screen and allows the user to enter a sentence such as the following:

```
This is a sample sentence.
~
~
~
~
~
~
~
~
~
-- INSERT --
```

When in insert mode, you can use the keyboard to type text as required, but when finished you must press the Esc key to return to command mode to perform other functions via keys on the keyboard. Table 3-6 provides a list of keys useful in command mode and their associated functions. After you are in command mode, to save the text in a file called samplefile in the current directory, you need to type the : (colon) character (by pressing the Shift and ; keys simultaneously) to reach a : prompt where you can enter a command to save the contents of the current document to a file, as shown in the following example and in Table 3-7.

```
This is a sample sentence.
~
~
~
~
~
~
~
~
~
:w samplefile
```

As shown in Table 3-7, you can quit the vi editor by typing the : character and entering q!, which then returns the user to the shell prompt:

```
This is a sample sentence.
~
~
~
~
~
~
~
~
~
:q!
[root@server1 ~]#_
```

Table 3-5 Common keyboard keys used to change to and from insert mode

Key	Description
i	Changes to insert mode and places the cursor before the current character for entering text
a	Changes to insert mode and places the cursor after the current character for entering text
o	Changes to insert mode and opens a new line below the current line for entering text
r	Changes to insert mode to replace the current character only (once this character has been replaced with another one you supply, the editor switches back to command mode)
I	Changes to insert mode and places the cursor at the beginning of the current line for entering text
A	Changes to insert mode and places the cursor at the end of the current line for entering text
O	Changes to insert mode and opens a new line above the current line for entering text
Esc	Changes back to command mode while in insert mode

Table 3-6 Key combinations commonly used in command mode

Key	Description
w, W	Moves the cursor forward one word to the beginning of the next word
e, E	Moves the cursor forward one word to the end of the next word
b, B	Moves the cursor backward one word
53G	Moves the cursor to line 53
G	Moves the cursor to the last line in the document
0, ^	Moves the cursor to the beginning of the line
$	Moves the cursor to the end of the line
X	Deletes the character the cursor is on
3x	Deletes three characters starting from the character the cursor is on
dw	Deletes one word starting from the character the cursor is on
d3w, 3dw	Deletes three words starting from the character the cursor is on
dd	Deletes one whole line starting from the line the cursor is on
d3d, 3dd	Deletes three whole lines starting from the line the cursor is on
d$	Deletes from the cursor character to the end of the current line
d^, d0	Deletes from the cursor character to the beginning of the current line
yw	Copies one word (starting from the character the cursor is on) into a temporary buffer in memory for later use
y3w, 3yw	Copies three words (starting from the character the cursor is on) into a temporary buffer in memory for later use
yy	Copies the current line into a temporary buffer in memory for later use
y3y, 3yy	Copies three lines (starting from the current line) into a temporary buffer in memory for later use
y$	Copies the current line from the cursor to the end of the line into a temporary buffer in memory for later use
y^, y0	Copies the current line from the cursor to the beginning of the line into a temporary buffer in memory for later use
p	Pastes the contents of the temporary memory buffer underneath the current line
P	Pastes the contents of the temporary memory buffer above the current line
J	Joins the line below the current line to the current line
Ctrl+g	Displays current line statistics
Ctrl+w followed by s	Splits the screen horizontally
Ctrl+g followed by v	Splits the screen vertically
Ctrl+ww	Move to the next screen
Ctrl+w followed by _	Minimize current screen
Ctrl+w followed by =	Restore a minimized screen
u	Undoes the last function (undo)
.	Repeats the last function (repeat)
/pattern	Searches for the first occurrence of pattern in the forward direction
?pattern	Searches for the first occurrence of pattern in the reverse direction
n	Repeats the previous search in the forward direction
N	Repeats the previous search in the reverse direction

Table 3-7 Key combinations commonly used at the command mode : prompt

Function	Description
:q	Quits from the vi editor if no changes were made
:q!	Quits from the vi editor and does not save any changes
:wq	Saves any changes to the file and quits from the vi editor
:w filename	Saves the current document to a file called filename
:!date	Executes the `date` command using a BASH shell
:r !date	Reads the output of the `date` command into the document under the current line
:r filename	Reads the contents of the text file called filename into the document under the current line
:set all	Displays all vi environment options
:set option	Sets a vi environment option
:s/the/THE/g	Searches for the regular expression "the" and replaces each occurrence globally throughout the current line with the word "THE"
:1,$ s/the/THE/g	Searches for the regular expression "the" and replaces each occurrence globally from line 1 to the end of the document with the word "THE"
:split proposal1	Creates a new screen (split horizontally) to edit the file "proposal1"
:vsplit proposal1	Creates a new screen (split vertically) to edit the file "proposal1"
:tabe notes	Creates a new tab called "notes"
:tabs	Displays all tabs
:tabn	Moves to the next tab
:tabp	Moves to the previous tab
:help p	Displays help for vi commands that start with p
:help holy-grail	Displays help for all vi commands

The vi editor also offers some advanced features to Linux users, as explained in Table 3-7. Examples of some of these features are discussed next, using the project4 file shown earlier in this chapter. To edit the project4 file, type `vi project4` and view the following screen:

```
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
~
```

```
~
~
~
~
~
~
"project4" 17L, 583C
```

Note that the name of the file as well as the number of lines and characters in total are displayed at the bottom of the screen (project4 has 17 lines and 583 characters in this example). To insert the current date and time at the bottom of the file, you can move the cursor to the final line in the file and type the following at the : prompt while in command mode:

```
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
~
~
~
~
~
~
~
~
:r !date
```

When you press Enter, the output of the `date` command is inserted below the current line:

```
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?
```

```
Regards,
Mackenzie Elizabeth
Sat Aug 7 18:33:10 EDT 2023
~
~
~
~
~
~
```

To change all occurrences of the word “Algonquin” to “ALGONQUIN,” you can type the following at the : prompt while in command mode:

```
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
Sat Aug 7 18:33:10 EDT 2023
~
~
~
~
~
~
:1,$ s/Algonquin/ALGONQUIN/g
```

The output changes to the following:

```
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in ALGONQUIN Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of ALGONQUIN Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?
```

```
Regards,
Mackenzie Elizabeth
Sat Aug 7 18:33:10 EDT 2023
~
~
~
~
~
~
~
```

Another attractive feature of the vi editor is its ability to customize the user environment through settings that can be altered at the : prompt while in command mode. Type `set all` at this prompt to observe the list of available settings and their current values:

```
:set all
--- Options ---
  aleph=224          fileencoding=      menuitems=25        swapsync=fsync
noarabic             fileformat=unix    modeline            switchbuf=
  arabicshape        filetype=          modelines=5         syntax=
noallowrevins      nofkmap              modifiable          tabstop=8
noaltkeymap          foldclose=         modified            tagbsearch
  ambiwidth=single   foldcolumn=0       more                taglength=0
noautoindent         foldenable         mouse=              tagrelative
noautoread           foldexpr=0         mousemodel=extend   tagstack
noautowrite          foldignore=#       mousetime=500       term=xterm
noautowriteall       foldlevel=0      nonumber            notermbidi
-- More --
```

Note in the preceding output that, although some settings have a configured value (e.g., fileformat=unix), most settings are set to either on or off; those that are turned off are prefixed with a "no." In the preceding example, line numbering is turned off (nonumber in the preceding output); however, you can turn it on by typing `set number` at the : prompt while in command mode. This results in the following output in vi:

```
      1 Hi there, I hope this day finds you well.
      2
      3 Unfortunately, we were not able to make it to your dining
      4 room this year while vacationing in ALGONQUIN Park - I
      5 especially wished to see the model of the Highland Inn
      6 and the train station in the dining room.
      7
      8 I have been reading on the history of ALGONQUIN Park but
      9 nowhere could I find a description of where the Highland
     10 Inn was originally located on Cache Lake.
     11
     12 If it is no trouble, could you kindly let me know such that
     13 I need not wait until next year when I visit your lodge?
     14
     15 Regards,
     16 Mackenzie Elizabeth
     17 Sat Aug 7 18:33:10 EDT 2023
     18
```

```
~
~
~
~
~
~
:set number
```

Conversely, to turn off line numbering, you could type `set nonumber` at the : prompt while in command mode.

Note 16

Most Linux distributions ship with the vim-minimal package, which provides a smaller version of the vi editor. You can install the vim-enhanced package to obtain full vi editor functionality. To do this on Fedora Linux, you can run the `dnf install vim-enhanced` command.

Other Common Text Editors

Although the vi editor is the most common text editor used on Linux and UNIX systems, you can instead choose a different text editor.

An alternative to the vi editor that offers an equal set of functionalities is the GNU **Emacs (Editor MACroS) editor**. Emacs is not installed by default on most Linux distributions. To install it on a Fedora system, you can run the command `dnf install emacs` at a command prompt to obtain Emacs from a free software repository on the Internet. Next, to open the project4 file in the Emacs editor in a command-line terminal, type `emacs project4` and the following is displayed on the terminal screen:

```
File  Edit  Options  Buffers  Tools  Conf Help
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth




-UU-:----F1  project4   Top L1    (Conf[Space])------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

The Emacs editor uses the Ctrl key in combination with certain letters to perform special functions, can be used with the LISP (LISt Processing) artificial intelligence programming language, and supports hundreds of keyboard functions, similar to the vi editor. Table 3-8 shows a list of some common keyboard functions used in the Emacs editor.

Table 3-8 Keyboard functions commonly used in the GNU Emacs editor

Key	Description
Ctrl+a	Moves the cursor to the beginning of the line
Ctrl+e	Moves the cursor to the end of the line
Ctrl+h	Displays Emacs documentation
Ctrl+d	Deletes the current character
Ctrl+k	Deletes all characters between the cursor and the end of the line
Esc+d	Deletes the current word
Ctrl+x + Ctrl+c	Exits the Emacs editor
Ctrl+x + Ctrl+s	Saves the current document
Ctrl+x + Ctrl+w	Saves the current document as a new filename
Ctrl+x + u	Undoes the last change

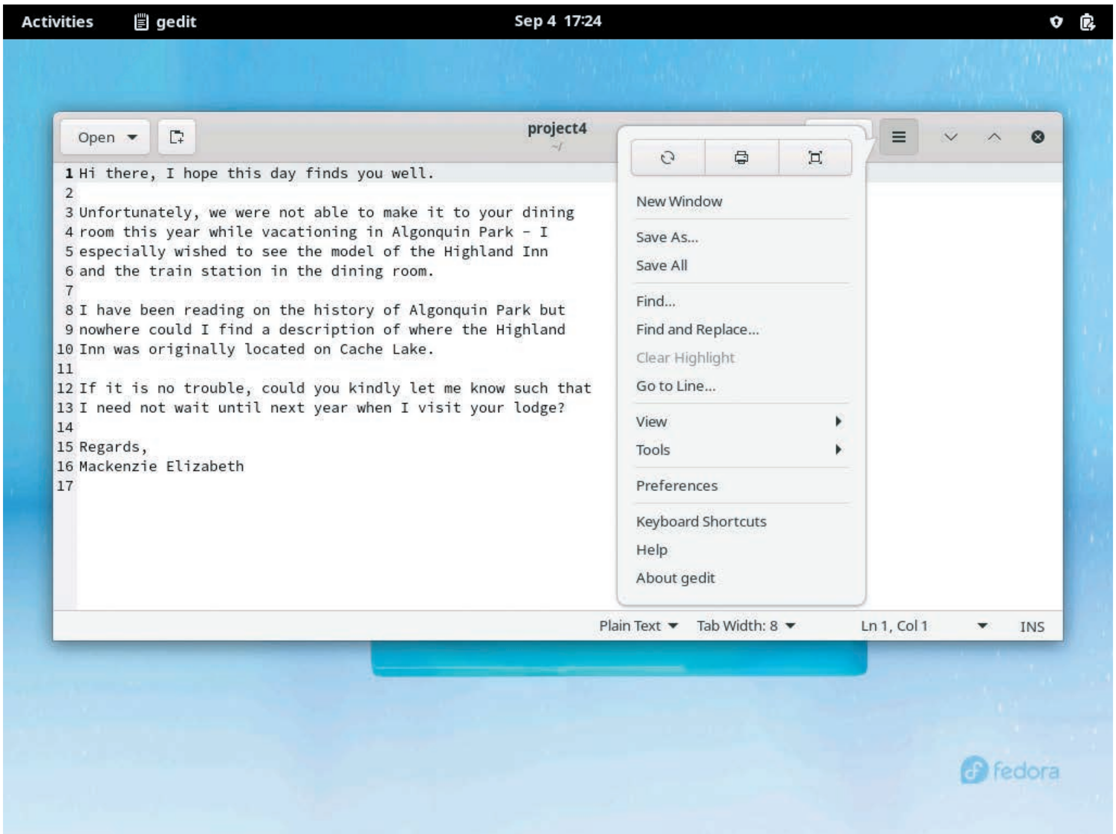
Another text editor that uses Ctrl key combinations for performing functions is the **nano editor** (based on the Pine UNIX editor). Unlike vi or Emacs, nano is a very basic and easy-to-use editor that many Linux administrators use to quickly modify configuration files if they don't need advanced functionality. As a result, nano is often installed by default on most modern Linux distributions. If you type `nano project4`, you will see the following displayed on the terminal screen:

```
GNU nano 6.0                    project4
Hi there, I hope this day finds you well.

Unfortunately, we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the train station in the dining room.

I have been reading on the history of Algonquin Park but
nowhere could I find a description of where the Highland
Inn was originally located on Cache Lake.

If it is no trouble, could you kindly let me know such that
I need not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth



                         [ Read 16 lines ]
^G Help ^O WriteOut  ^W Where Is ^K Cut   ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace  ^U Paste ^J Justify ^_ Go To Line
```

The bottom of the screen lists common Ctrl key combinations. The ^ symbol represents the Ctrl key. This means that, to exit nano, you can press Ctrl+x (^X = Ctrl+x).

If you are using a desktop environment, there is often a graphical text editor provided by your desktop environment, as well as others that can be optionally installed. The **gedit editor** is one of the most common Linux graphical text editors, and functionally analogous to the Windows WordPad and Notepad editors. If you type `gedit project4` within a desktop environment, you will be able to edit the project4 file content graphically, as well as access any gedit functionality from the drop-down menu shown in Figure 3-3.

Figure 3-3 The gedit text editor



Summary

- The Linux filesystem is arranged hierarchically using a series of directories to store files. The location of these directories and files can be described using a relative or absolute pathname. The Linux filesystem can contain many types of files, such as text files, binary data, executable programs, directories, linked files, and special device files.
- The `ls` command can be used to view filenames and offers a wide range of options to modify this view.
- Wildcard metacharacters are special keyboard characters. They can be used to simplify the selection of several files when using common Linux file commands.
- Text files are the most common file type whose contents can be viewed by several utilities, such as `head`, `tail`, `cat`, `tac`, `more`, and `less`.
- Regular expression metacharacters can be used to specify certain patterns of text when used with certain programming languages and text tool utilities, such as `grep`.
- Although many command-line and graphical text editors exist, vi (vim) is a powerful, bimodal text editor that is standard on most UNIX and Linux systems.

Key Terms

~ metacharacter
absolute pathname
binary data file
`cat` command
`cd (change directory)` command
command mode
concatenation
`diff` command
directory
`egrep` command
Emacs (Editor MACroS) editor
executable program
`fgrep` command
`file` command
file globbing
filename
filename extension
gedit editor
`grep` command
`head` command
home directory
insert mode
`less` command
linked file
`ll` command
log file
`ls` command
`more` command
named pipe file
nano editor
`od` command
parent directory
`pwd` (print working directory) command
regular expressions (regexp)
relative pathname
shell script
socket file
special device file
`stat` command
`strings` command
subdirectory
Tab-completion feature
`tac` command
`tail` command
text file
text tool
`tree` command
vi editor
wildcard metacharacter

Review Questions

1. A directory is a type of file.
   a. True
   b. False
2. Which command would a user type on the command line to find out the current directory in the directory tree?
   a. `pd`
   b. `cd`
   c. `where`
   d. `pwd`
3. Which of the following is an absolute pathname? (Choose all that apply.)
   a. C:\myfolder\resume
   b. resume
   c. /home/resume
   d. C:home/resume
4. A special device file is used to _____.
   a. enable proprietary custom-built devices to work with Linux
   b. represent hardware devices
   c. keep a list of device settings specific to each individual user
   d. do nothing in Linux
5. If a user's current directory is /home/mary/project1, which command could they use to move to the etc directory directly under the root?
   a. `cd ..`
   b. `cd etc`
   c. `cd /etc`
   d. `cd \etc`
6. After typing the `ls –a` command, you notice a file whose filename begins with a period (`.`). What does this mean?
   a. It is a binary file.
   b. It is a system file.
   c. It is a file in the current directory.
   d. It is a hidden file.
7. After typing the `ls –F` command, you notice a filename that ends with an * (asterisk) character. What does this mean?
   a. It is a hidden file.
   b. It is a linked file.
   c. It is a special device file.
   d. It is an executable file.
8. The vi editor can function in which two of the following modes? (Choose both that apply.)
   a. Command
   b. Input
   c. Interactive
   d. Insert
9. The `less` command offers less functionality than the `more` command.
   a. True
   b. False

10. Which command searches for and displays any text contents of a binary file?
    a. `text`
    b. `strings`
    c. `od`
    d. `less`
11. How can a user switch from insert mode to command mode when using the vi editor?
    a. Press the Ctrl+Alt+Del keys simultaneously.
    b. Press the Del key.
    c. Type a : character.
    d. Press the Esc key.
12. If "resume" is the name of a file in the home directory off the root of the filesystem and your present working directory is home, what is the relative name for the file named resume?
    a. /home/resume
    b. /resume
    c. resume
    d. \home\resume
13. What will the following wildcard expression return: `file[a-c]?`
    a. `filea-c`
    b. `filea, filec`
    c. `filea, fileb, filec`
    d. `fileabc`
14. What will typing `q!` at the : prompt in command mode do when using the vi editor?
    a. Quit as no changes were made.
    b. Quit after saving any changes.
    c. Nothing because the ! is a metacharacter.
    d. Quit without saving any changes.
15. A user types the command `head /poems/mary`. What will be displayed on the terminal screen?
    a. The first line of the file mary
    b. The first 10 lines of the file mary
    c. The header for the file mary
    d. The first 20 lines of the file mary
16. The `tac` command _____.
    a. displays the contents of a file in reverse order, last line first and first line last
    b. displays the contents of hidden files
    c. displays the contents of a file in reverse order, last word on the line first and first word on the line last
    d. is not a valid Linux command
17. How can you specify a text pattern that must be at the beginning of a line of text using a regular expression?
    a. Precede the string with a /.
    b. Follow the string with a \.
    c. Precede the string with a $.
    d. Precede the string with a ^.
18. Linux has only one root directory per directory tree.
    a. True
    b. False
19. Using a regular expression, how can you indicate a character that is *not* an a or b or c or d?
    a. `[^abcd]`
    b. `not [a-d]`
    c. `[!a-d]`
    d. `!a-d`
20. A user typed the command `pwd` and saw the output: /home/jim/sales/pending. How could that user navigate to the /home/jim directory?
    a. `cd ..`
    b. `cd /jim`
    c. `cd ../..`
    d. `cd ./.`

Hands-On Projects

These projects should be completed in the order given. The hands-on projects presented in this chapter should take a total of three hours to complete. The requirements for this lab include:

- A computer with Fedora Linux installed according to Hands-On Project 2-1.

Project 3-1

Estimated Time: 30 minutes
Objective: Navigate the Linux filesystem.
Description: In this hands-on project, you log in to the computer and navigate the file structure.

1. Boot your Fedora Linux virtual machine. After your Linux system has been loaded, switch to a command-line terminal (tty5) by pressing **Ctrl+Alt+F5** and log in to the terminal using the user name of **root** and the password of **LINUXrocks!**.

2. At the command prompt, type **pwd** and press **Enter** to view the current working directory. What is your current working directory?
3. At the command prompt, type **cd** and press **Enter**. At the command prompt, type **pwd** and press **Enter** to view the current working directory. Did your current working directory change? Why or why not?
4. At the command prompt, type **cd .** and press **Enter**. At the command prompt, type **pwd** and press **Enter** to view the current working directory. Did your current working directory change? Why or why not?
5. At the command prompt, type **cd ..** and press **Enter**. At the command prompt, type **pwd** and press **Enter** to view the current working directory. Did your current working directory change? Why or why not?
6. At the command prompt, type **cd root** and press **Enter**. At the command prompt, type **pwd** and press **Enter** to view the current working directory. Did your current working directory change? Where are you now? Did you specify a relative or absolute pathname to your home directory when you used the **cd root** command?
7. At the command prompt, type **cd etc** and press **Enter**. What error message did you receive and why?
8. At the command prompt, type **cd /etc** and press **Enter**. At the command prompt, type **pwd** and press **Enter** to view the current working directory. Did your current working directory change? Did you specify a relative or absolute pathname to the /etc directory when you used the `cd /etc` command?
9. At the command prompt, type **cd /** and press **Enter**. At the command prompt, type **pwd** and press **Enter** to view the current working directory. Did your current working directory change? Did you specify a relative or absolute pathname to the / directory when you used the `cd /` command?
10. At the command prompt, type **cd ~user1** and then press **Enter**. At the command prompt, type **pwd** and press **Enter** to view the current working directory. Did your current working directory change? Which command discussed earlier performs the same function as the `cd ~` command?
11. At the command prompt, type **cd Desktop** and press **Enter** (be certain to use a capital D). At the command prompt, type **pwd** and press **Enter** to view the current working directory. Did your current working directory change? Where are you now? What kind of pathname did you use here (absolute or relative)?
12. Currently, you are in a subdirectory of user1's home folder, three levels below the root. To go up three parent directories to the / directory, type **cd ../../..** and press **Enter** at the command prompt. Next, type **pwd** and press **Enter** to ensure that you are in the / directory.
13. At the command prompt, type **cd /etc/samba** and press **Enter** to change the current working directory using an absolute pathname. Next, type **pwd** and press **Enter** at the command prompt to ensure that you have changed to the /etc/samba directory. Now, type the command **cd ../sysconfig** at the command prompt and press **Enter**. Type **pwd** and press **Enter** to view your current location. Explain how the relative pathname seen in the `cd ../sysconfig` command specified your current working directory.
14. At the command prompt, type **cd ../../home/user1/Desktop** and press **Enter** to change your current working directory to the Desktop directory under user1's home directory. Verify that you are in the target directory by typing the **pwd** command at a command prompt and press **Enter**. Would it have been more advantageous to use an absolute pathname to change to this directory instead of the relative pathname that you used?
15. Type **exit** and press **Enter** to log out of your shell.

Project 3-2

Estimated Time: 10 minutes
Objective: Use the BASH Tab-completion feature.
Description: In this hands-on project, you navigate the Linux filesystem using the Tab-completion feature of the BASH shell.

1. Switch to a command-line terminal (tty5) by pressing **Ctrl+Alt+F5** and log in to the terminal using the user name of **root** and the password of **LINUXrocks!**.
2. At the command prompt, type **cd /** and press **Enter**.

3. Next, type **cd ro** at the command prompt and press **Tab**. What is displayed on the screen and why? How many subdirectories under the root begin with “ro”?
4. Press the **Ctrl**+**c** keys to cancel the command and return to an empty command prompt.
5. At the command prompt, type **cd b** and press **Tab**. Did the display change?
6. Press the **Tab** key again. How many subdirectories under the root begin with “b”?
7. Type the letter **i**. Notice that the command now reads “cd bi.” Press the **Tab** key again. Which directory did it expand to? Why? Press the **Ctrl**+**c** keys to cancel the command and return to an empty command prompt.
8. At the command prompt, type **cd m** and press **Tab**. Press **Tab** once again after hearing the beep. How many subdirectories under the root begin with “m”?
9. Type the letter **e**. Notice that the command now reads “cd me.” Press **Tab**.
10. Press **Enter** to execute the command at the command prompt. Next, type the **pwd** command and press **Enter** to verify that you are in the /media directory.
11. Type **exit** and press **Enter** to log out of your shell.

Project 3-3

Estimated Time: 20 minutes
Objective: View Linux filenames and types.
Description: In this hands-on project, you examine files and file types using the `ls` and `file` commands.

1. Switch to a command-line terminal (tty5) by pressing **Ctrl+Alt+F5** and log in to the terminal using the user name of **root** and the password of **LINUXrocks!**.
2. At the command prompt, type **cd /etc** and press **Enter**. Verify that you are in the /etc directory by typing **pwd** at the command prompt and press **Enter**.
3. At the command prompt, type **ls** and press **Enter**. What do you see listed in the four columns? Do any of the files have extensions? What is the most common extension you see and what does it indicate? Is the list you are viewing on the screen the entire contents of /etc?
4. At the command prompt, type **ls | more** and then press **Enter** (the | symbol is usually near the Enter key on the keyboard and is obtained by pressing the Shift and \ keys in combination). What does the display show? Notice the highlighted --More-- prompt at the bottom of the screen. Press **Enter**. Press **Enter** again. Press **Enter** once more. Notice that each time you press Enter, you advance one line further into the file. Now, press the **spacebar**. Press the **spacebar** again. Notice that with each press of the spacebar, you advance one full page into the displayed directory contents. Press the **h** key to get a help screen. Examine the command options.
5. Press the **q** key to quit the `more` command and return to an empty command prompt.
6. At the command prompt, type **ls | less** and then press **Enter**. What does the display show? Notice the : at the bottom of the screen. Press **Enter**. Press **Enter** again. Press **Enter** once more. Notice that each time you press Enter, you advance one line further into the file. Now press the **spacebar**. Press the **spacebar** again. Notice that with each press of the spacebar, you advance one full page into the displayed directory contents. Press the **h** key to get a help screen. Examine the command options, and then press **q** to return to the command output.
7. Press the ↑ (up arrow) key. Press ↑ again. Press ↑ once more. Notice that each time you press the ↑ key, you go up one line in the file display toward the beginning of the file. Now, press the ↓ (down arrow) key. Press ↓ again. Press ↓ once more. Notice that each time you press the ↓ key, you move forward into the file display.
8. Press the **q** key to quit the `less` command and return to a shell command prompt.
9. At the command prompt, type **cd** and press **Enter**. At the command prompt, type **pwd** and press **Enter**. What is your current working directory? At the command prompt, type **ls** and press **Enter**.

10. At the command prompt, type **ls /etc** and press **Enter**. How does this output compare with what you saw in Step 9? Has your current directory changed? Verify your answer by typing **pwd** at the command prompt and press **Enter**. Notice that you were able to list the contents of another directory by giving the absolute name of it as an argument to the `ls` command without leaving the directory in which you are currently located.
11. At the command prompt, type **ls /etc/skel** and press **Enter**. Did you see a listing of any files? At the command prompt, type **ls -a /etc/skel** and press **Enter**. What is special about these files? What do the first two entries in the list (. and ..) represent?
12. At the command prompt, type **ls -aF /etc/skel** and press **Enter**. Which file types are available in the /etc/skel directory?
13. At the command prompt, type **ls -F /bin/*** and press **Enter**. What file types are present in the /bin directory?
14. At the command prompt, type **ls -R /etc/ssh** and press **Enter**. Note the files and subdirectories listed. Type **tree /etc/ssh** and press **Enter** and note the same output in a more friendly format. Next, type **tree -d /etc/ssh** and press **Enter**. What does the `-d` option specify?
15. At the command prompt, type **ls /boot** and press **Enter**. Next, type **ls -l /boot** and press **Enter**. What additional information is available on the screen? What types of files are available in the /boot directory? At the command prompt, type **ll /boot** and press **Enter**. Is the output any different from that of the `ls -l /boot` command you just entered? Why or why not?
16. At the command prompt, type **file /boot/*** to see the types of files in the /boot directory. Is this information more specific than the information you gathered in Step 15?
17. At the command prompt, type **file /etc** and press **Enter**. What kind of file is etc?
18. At the command prompt, type **file /etc/issue** and press **Enter**. What type of file is /etc/issue?
19. At the command prompt, type **stat /etc/issue** and press **Enter**. Note the time this file was last accessed.
20. Type **exit** and press **Enter** to log out of your shell.

Project 3-4

Estimated Time: 20 minutes
Objective: View the contents of text files.
Description: In this hands-on project, you display file contents using the `cat`, `tac`, `head`, `tail`, `strings`, and `od` commands.

1. Switch to a command-line terminal (tty5) by pressing **Ctrl+Alt+F5,** and then log in to the terminal using the user name of **root** and the password of **LINUXrocks!**.
2. At the command prompt, type **cat /etc/hosts** and press **Enter** to view the contents of the file hosts, which reside in the directory /etc. Next, type **cat -n /etc/hosts** and press **Enter**. How many lines does the file have? At the command prompt, type **tac /etc/hosts** and press **Enter** to view the same file in reverse order.
3. To see the contents of the same file in octal format instead of ASCII text, type **od /etc/hosts** at the command prompt and press **Enter**.
4. At the command prompt, type **cat /etc/services** and press **Enter**.
5. At the command prompt, type **head /etc/services** and press **Enter**. How many lines are displayed, and why?
6. At the command prompt, type **head -5 /etc/services** and press **Enter**. How many lines are displayed and why? Next, type **head -3 /etc/services** and press **Enter**. How many lines are displayed and why?

7. At the command prompt, type `tail /etc/services` and press **Enter**. What is displayed on the screen? How many lines are displayed and why?
8. At the command prompt, type `tail -5 /etc/services` and press **Enter**. How many lines are displayed and why? Type the `cat -n /etc/services` command at a command prompt and press **Enter** to justify your answer.
9. At the command prompt, type `file /bin/nice` and press **Enter**. What type of file is it? Should you use a text tool command on this file?
10. At the command prompt, type `strings /bin/nice` and press **Enter**. Notice that you can see some text within this binary file. Next, type `strings /bin/nice | less` to view the same content page-by-page. When finished, press **q** to quit the `less` command.
11. Type `exit` and press **Enter** to log out of your shell.

Project 3-5

Estimated Time: 50 minutes
Objective: Use the vi editor.
Description: In this hands-on project, you create and edit text files using the vi editor.

1. Switch to a command-line terminal (tty5) by pressing **Ctrl+Alt+F5** and log in to the terminal using the user name of **root** and the password of **LINUXrocks!**.
2. At the command prompt, type `dnf install vim-enhanced` and press **Enter**. Press `y` when prompted to install the vim-enhanced package.
3. At the command prompt, type `pwd`, press **Enter**, and ensure that /root is displayed, showing that you are in the root user's home folder. At the command prompt, type `vi sample1` and press **Enter** to open the vi editor and create a new text file called sample1. Notice that this name appears at the bottom of the screen along with the indication that it is a new file.
4. At the command prompt, type `My letter` and press **Enter**. Why was nothing displayed on the screen? To switch from command mode to insert mode so you can type text, press `i`. Notice that the word Insert appears at the bottom of the screen. Next, type `My letter` and notice that this text is displayed on the screen. What types of tasks can be accomplished in insert mode?
5. Press **Esc**. Did the cursor move? What mode are you in now? Press ← two times until the cursor is under the last t in letter. Press **x**. What happened? Next, press **i** to enter insert mode, then type the letter `h`. Was the letter h inserted before or after the cursor?
6. Press **Esc** to switch back to command mode and then move your cursor to the end of the line. Next, press **o** to open a line underneath the current line and enter insert mode.
7. Type the following:

   ```
   It might look like I am doing nothing, but at the cellular level I
   can assure you that I am quite busy.
   ```
8. Type `dd` to delete the line in the file.
9. Press **i** to enter insert mode, and then type:

   ```
   Hi there, I hope this day finds you well.
   ```

   and press **Enter**. Press **Enter** again. Type:

   ```
   Unfortunately, we were not able to make it to your dining
   ```

   and press **Enter**. Type:

   ```
   room this year while vacationing in Algonquin Park - I
   ```

   and press **Enter**. Type:

   ```
   especially wished to see the model of the Highland Inn
   ```

and press **Enter**. Type:

`and the train station in the dining room.`

and press **Enter**. Press **Enter** again. Type:

`I have been reading on the history of Algonquin Park but`

and press **Enter**. Type:

`nowhere could I find a description of where the Highland`

and press **Enter**. Type:

`Inn was originally located on Cache Lake.`

and press **Enter**. Press **Enter** again. Type:

`If it is no trouble, could you kindly let me know such that`

and press **Enter**. Type:

`I need not wait until next year when I visit your lodge?`

and press **Enter**. Press **Enter** again. Type:

`Regards,`

and press **Enter**. Type:

`Mackenzie Elizabeth`

and press **Enter**. You should now have the sample letter used in this chapter on your screen. It should resemble the letter in Figure 3-3.

10. Press **Esc** to switch to command mode. Next press the **Shift** and **;** keys together to open the : prompt at the bottom of the screen. At this prompt, type `w` and press **Enter** to save the changes you have made to the file. What is displayed at the bottom of the file when you are finished?
11. Press the **Shift** and **;** keys together again to open the : prompt at the bottom of the screen, type `q`, and then press **Enter** to exit the vi editor.
12. At the command prompt, type `ls` and press **Enter** to view the contents of your current directory. Notice that there is now a file called sample1 listed.
13. Next, type `file sample1` and press **Enter**. What type of file is sample1? At the command prompt, type `cat sample1` and press **Enter**.
14. At the command prompt, type `vi sample1` and press **Enter** to open the letter again in the vi editor. What is displayed at the bottom of the screen? How does this compare with Step 10?
15. Use the cursor keys to navigate to the bottom of the document. Press the **Shift** and **;** keys together to open the : prompt at the bottom of the screen again, type `!date` and press **Enter**. The current system date and time appear at the bottom of the screen. As indicated, press **Enter** to return to the document.
16. Press the **Shift** and **;** keys together to open the : prompt at the bottom of the screen again, type `r !date` and press **Enter**. What happened and why?
17. Use the cursor keys to position your cursor on the line in the document that displays the current date and time, and type `yy` to copy it to the buffer in memory. Next, use the cursor keys to position your cursor on the first line in the document, and type `P` (capitalized) to paste the contents of the memory buffer above your current line. Does the original line remain at the bottom of the document?
18. Use the cursor keys to position your cursor on the line at the end of the document that displays the current date and time, and type `dd` to delete it.

19. Use the cursor keys to position your cursor on the "t" in the word "there" on the second line of the file that reads **Hi there, I hope this day finds you well.**, and type `dw` to delete the word. Next, press **i** to enter insert mode, type the word `Bob`, and then press **Esc** to switch back to command mode.
20. Press the **Shift** and **;** keys together to open the : prompt at the bottom of the screen, type `w sample2` and press **Enter**. What happened?
21. Press **i** to enter insert mode, and type the word `test`. Next, press **Esc** to switch to command mode. Press the **Shift** and **;** keys together to open the : prompt at the bottom of the screen, type `q`, and press **Enter** to quit the vi editor. Were you able to quit? Why not?
22. Press the **Shift** and **;** keys together to open the : prompt at the bottom of the screen, type `q!`, and press **Enter** to quit the vi editor and discard any changes since the last save.
23. At the command prompt, type `ls` and press **Enter** to view the contents of your current directory. Notice it now includes a file called sample2, which was created in Step 20. Type `diff sample1 sample2` and press **Enter** to view the difference in content between the two files you created.
24. At the command prompt, type `vi sample2` and press **Enter** to open the letter again in the vi editor.
25. Use the cursor keys to position your cursor on the line that reads **Hi Bob, I hope this day finds you well.**
26. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `s/Bob/Barb/g`, and press **Enter** to change all occurrences of "Bob" to "Barb" on the current line.
27. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `1,$ s/to/TO/g`, and press **Enter** to change all occurrences of the word "to" to "TO" for the entire file.
28. Press the **u** key. What happened?
29. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `wq`, and press **Enter** to save your document and quit the vi editor.
30. At the command prompt, type `vi sample3` and press **Enter** to open a new file called sample3 in the vi editor. Press `i` to enter insert mode. Next, type `P.S. How were the flies this year?` Press the **Esc** key when finished.
31. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `wq`, and press **Enter** to save your document and quit the vi editor.
32. At the command prompt, type `vi sample1`, press **Enter** to open the file sample1 again, and use the cursor keys to position your cursor on the line that reads "Mackenzie Elizabeth."
33. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `r sample3`, and press **Enter** to insert the contents of the file sample3 below your current line.
34. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `s/flies/flies and bears/g` and press **Enter**. What happened and why?
35. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `set number`, and press **Enter** to turn on line numbering.
36. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `set nonumber`, and press **Enter** to turn off line numbering.
37. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen, type `set all`, and press **Enter** to view all vi parameters. Press **Enter** to advance through the list, and press **q** when finished to return to the vi editor.
38. Press the **Shift** and **;** keys together to open the **:** prompt at the bottom of the screen again, type `wq`, and press **Enter** to save your document and quit the vi editor.
39. Type `exit` and press **Enter** to log out of your shell.

Project 3-6

Estimated Time: 20 minutes
Objective: Use wildcard metacharacters.
Description: In this hands-on project, you use the `ls` command alongside wildcard metacharacters in your shell to explore the contents of your home directory.

1. Switch to a command-line terminal (tty5) by pressing **Ctrl+Alt+F5** and log in to the terminal using the user name of **root** and the password of **LINUXrocks!**.
2. At the command prompt, type **`pwd`**, press **Enter**, and ensure /root is displayed showing that you are in the root user's home folder. At the command prompt, type **`ls`**. How many files with a name beginning with the word "sample" exist in **/root**?
3. At the command prompt, type **`ls *`** and press **Enter**. What is listed and why?
4. At the command prompt, type **`ls sample*`** and press **Enter**. What is listed?
5. At the command prompt, type **`ls sample?`** and press **Enter**. What is listed and why?
6. At the command prompt, type **`ls sample??`** and press **Enter**. What is listed and why?
7. At the command prompt, type **`ls sample[13]`** and press **Enter**. What is listed and why?
8. At the command prompt, type **`ls sample[!13]`** and press **Enter**. What is listed and why?
9. At the command prompt, type **`ls sample[1-3]`** and press **Enter**. What is listed and why?
10. At the command prompt, type **`ls sample[!1-3]`** and press **Enter**. What is listed and why?
11. Type **`exit`** and press **Enter** to log out of your shell.

Project 3-7

Estimated Time: 30 minutes
Objective: Use regular expressions.
Description: In this hands-on project, you use the `grep` and `egrep` commands alongside regular expression metacharacters to explore the contents of text files.

1. Switch to a command-line terminal (tty5) by pressing **Ctrl+Alt+F5** and log in to the terminal using the user name of **root** and the password of **LINUXrocks!**.
2. At the command prompt, type **`grep "Inn" sample1`** and press **Enter**. What is displayed?
3. At the command prompt, type **`grep -v "Inn" sample1`** and press **Enter**. What is displayed?
4. At the command prompt, type **`grep "inn" sample1`** and press **Enter**. What is displayed and why?
5. At the command prompt, type **`grep -i "inn" sample1`** and press **Enter**. What is displayed?
6. At the command prompt, type **`grep "I" sample1`** and press **Enter**. What is displayed?
7. At the command prompt, type **`grep " I " sample1`** and press **Enter**. What is displayed?
8. At the command prompt, type **`grep "t.e" sample1`** and press **Enter**. What is displayed?
9. At the command prompt, type **`grep "w...e" sample1`** and press **Enter**. What is displayed?
10. At the command prompt, type **`grep "^I" sample1`** and press **Enter**. What is displayed?
11. At the command prompt, type **`grep "^I " sample1`** and press **Enter**. What is displayed?
12. At the command prompt, type **`grep "(we|next)" sample1`** and press **Enter**. What is displayed? Why?
13. At the command prompt, type **`egrep "(we|next)" sample1`** and press **Enter**. What is displayed?
14. At the command prompt, type **`grep "Inn$" sample1`** and press **Enter**. What is displayed?
15. At the command prompt, type **`grep "?$" sample1`** and press **Enter**. What is displayed and why? Does the ? metacharacter have special meaning here? Why?
16. At the command prompt, type **`grep "^$" sample1`** and press **Enter**. Is anything displayed? (*Hint:* Be certain to look closely!) Can you explain the output?
17. Type **`exit`** and press **Enter** to log out of your shell.

Discovery Exercises

Discovery Exercise 3-1

Estimated Time: 30 minutes
Objective: Detail the commands used to navigate and view files.
Description: You are the systems administrator for a scientific research company that employs over 100 scientists who write and run Linux programs to analyze their work. All of these programs are stored in each scientist's home directory on the Linux system. One scientist has left the company, and you are instructed to retrieve any work from that scientist's home directory. When you enter the home directory for that user, you notice that there are very few files and only two directories (one named Projects and one named Lab). List the commands that you would use to navigate through this user's home directory and view filenames and file types. If there are any text files, what commands could you use to view their contents?

Discovery Exercise 3-2

Estimated Time: 20 minutes
Objective: Explain relative and absolute pathnames.
Description: When you type the `pwd` command, you notice that your current location on the Linux filesystem is the /usr/local directory. Answer the following questions, assuming that your current directory is /usr/local for each question.

a. Which command could you use to change to the /usr directory using an absolute pathname?
b. Which command could you use to change to the /usr directory using a relative pathname?
c. Which command could you use to change to the /usr/local/share/info directory using an absolute pathname?
d. Which command could you use to change to the /usr/local/share/info directory using a relative pathname?
e. Which command could you use to change to the /etc directory using an absolute pathname?
f. Which command could you use to change to the /etc directory using a relative pathname?

Discovery Exercise 3-3

Estimated Time: 30 minutes
Objective: Identify wildcard metacharacters.
Description: Using wildcard metacharacters and options to the `ls` command, view the following:

a. All the files that end with .cfg under the /etc directory
b. All hidden files in the /home/user1 directory
c. The directory names that exist under the /var directory
d. All the files that start with the letter "a" under the /bin directory
e. All the files that have exactly three letters in their filename in the /bin directory
f. All files that have exactly three letters in their filename and end with either the letter "t" or the letter "h" in the /bin directory

Discovery Exercise 3-4

Estimated Time: 40 minutes
Objective: Obtain command help.
Description: Explore the manual pages for the `ls`, `grep`, `cat`, `od`, `tac`, `head`, `tail`, `diff`, `pwd`, `cd`, `strings`, and `vi` commands. Experiment with what you learn using the file sample1 that you created earlier.

Discovery Exercise 3-5

Estimated Time: 20 minutes
Objective: Explain regular expressions.
Description: The famous quote from Shakespeare's Hamlet "To be or not to be" can be represented by the following regular expression:

```
(2b|[^b]{2})
```

If you used this expression when searching a text file using the `egrep` command (`egrep "(2b|[^b]{2})" filename`), what would be displayed? Try this command on a file that you have created. Why does it display what it does? That is the question.

Discovery Exercise 3-6

Estimated Time: 30 minutes
Objective: Use the vi editor.
Description: The vim-enhanced package you installed earlier in Hands-On Project 3-5 comes with a short 30-minute tutorial on its usage. Start this tutorial by typing **`vimtutor`** at a command prompt and then follow the directions.

Discovery Exercise 3-7

Estimated Time: 40 minutes
Objective: Use the vi editor.
Description: Enter the following text into a new document called question7 using the vi editor. Next, use the vi editor to fix the mistakes in the file using the information in Tables 3-5, 3-6, and 3-7 as well as the examples provided in this chapter.

```
Hi there,
Unfortunately we were not able to make it to your dining room
Unfortunately we were not able to make it to your dining room
this year while vacationing in Algonuin Park - I especially wished
to see the model of the highland inn and the train station in the
dining rooms.

I have been readng on the history of Algonuin Park but
no where could I find a description of where the Highland Inn was
originally located on Cache lake.

If it is not trouble, could you kindly let me that I need
not wait until next year when we visit Lodge?

I hope this day finds you well.
Regard
Elizabeth Mackenzie
```

Discovery Exercise 3-8

Estimated Time: 40 minutes
Objective: Use the Emacs editor.
Description: The knowledge gained from using the vi editor can be transferred easily to the Emacs editor. Perform Discovery Exercise 3-7 using the Emacs editor instead of the vi editor.

Discovery Exercise 3-9

Estimated Time: 20 minutes

Objective: Configure persistent vi environment settings.

Description: When you use the vi editor and change environment settings at the : prompt, such as `:set number` to enable line numbering, those changes are lost when you exit the vi editor. To continuously apply the same environment settings, you can choose to put any vi commands that can be entered at the : prompt in a special hidden file in your home directory called either .vimrc or .exrc. Each time the vi (vim) editor is opened, it looks for these files and automatically executes the commands within. Enter the vi editor and find two environment settings that you want to change in addition to line numbering. Then create a new file called .exrc in your home directory and enter the three lines changing these vi environment settings (do not start each line with a : character, just enter the `set` command—e.g., `set number`). When finished, open the vi editor to edit a new file and test to see whether the settings were applied automatically.

Chapter 4

Linux Filesystem Management

Chapter Objectives

1. Find files and directories on the filesystem.
2. Describe and create linked files.
3. Explain the function of the Filesystem Hierarchy Standard.
4. Use standard Linux commands to manage files and directories.
5. Modify file and directory ownership.
6. Define and change Linux file and directory permissions.
7. Identify the default permissions created on files and directories.
8. Apply special file and directory permissions.
9. Modify the default access control list (ACL).
10. View and set filesystem attributes.

In the previous chapter, you learned about navigating the Linux filesystem as well as viewing and editing files. This chapter focuses on the organization of files on the Linux filesystem as well as their linking and security. First, you explore standard Linux directories using the Filesystem Hierarchy Standard. Next, you explore common commands used to manage files and directories as well as learn methods that are used to find files and directories on the filesystem. Finally, you learn about file and directory linking, permissions, special permissions, and attributes.

The Filesystem Hierarchy Standard

The many thousands of files on a typical Linux system are organized into directories in the Linux directory tree. It's a complex system, made even more complex in the past by the fact that different Linux distributions were free to place files in different locations. This meant that you could waste a great deal of time searching for a configuration file on a Linux system with which you were unfamiliar. To simplify the task of finding specific files, the **Filesystem Hierarchy Standard (FHS)** was created.

FHS defines a standard set of directories for use by all Linux and UNIX systems as well as the file and subdirectory contents of each directory. Because the filename and location follow a standard convention, a Fedora Linux user will find the correct configuration file on an Arch Linux or macOS UNIX computer with little difficulty. The FHS also gives Linux software developers the ability to locate files on a Linux system regardless of the distribution, allowing them to create software that is not distribution-specific.

A comprehensive understanding of the standard types of directories found on Linux systems is valuable when locating and managing files and directories; some standard UNIX and Linux directories defined by FHS and their descriptions are listed in Table 4-1. These directories are discussed throughout this chapter and subsequent chapters.

Note 1

To read the complete Filesystem Hierarchy Standard definition, go to www.pathname.com/fhs.

Table 4-1 Linux directories defined by the Filesystem Hierarchy Standard

Directory	Description
/bin	Contains binary commands for use by all users (on most Linux systems, this directory is a shortcut to /usr/bin)
/boot	Contains the Linux kernel and files used by the boot loader
/dev	Contains device files
/etc	Contains system-specific configuration files
/home	Is the default location for user home directories
/lib /lib64	Contains shared program libraries (used by the commands in /bin and /sbin) as well as kernel modules (on most Linux systems, /lib is a shortcut to /usr/lib and /lib64 is a shortcut to /usr/lib64)
/media	A directory that contains subdirectories used for accessing (mounting) filesystems on removable media devices, such as DVDs and USB flash drives
/mnt	An empty directory used for temporarily accessing filesystems on removable media devices
/opt	Stores additional software programs
/proc	Contains process and kernel information
/root	Is the root user's home directory
/sbin	Contains system binary commands used for administration (on most Linux systems, this directory is a shortcut to /usr/sbin)
/sys	Contains configuration information for hardware devices on the system
/tmp	Holds temporary files created by programs
/usr	Contains most system commands and utilities—contains the following directories: /usr/bin—User binary commands /usr/games—Educational programs and games /usr/include—C program header files /usr/lib and /usr/lib64—Libraries /usr/local—Local programs /usr/sbin—System binary commands /usr/share—Files that are architecture independent /usr/share/X11—The X Window system (sometimes replaced by /etc/X11) /usr/src—Source code
/usr/local	Is the location for most additional programs
/var	Contains log files and spools

Managing Files and Directories

As mentioned earlier, using a Linux system involves navigating several directories and manipulating the files inside them. Thus, an efficient Linux user must understand how to create directories as needed, copy or move files from one directory to another, and delete files and directories. These tasks are commonly referred to as file management tasks.

Following is an example of a directory listing from the root user:

```
[root@server1 ~]# pwd
/root
[root@server1 ~]# ls -F
myprogram*  project   project12  project2  project4
myscript*   project1  project13  project3  project5
[root@server1 ~]#_
```

As shown in the preceding output, two executable files (myprogram and myscript), and several project-related files (project*) exist on this example system. Although this directory structure is not cluttered, typical home directories on a Linux system contain many more files. As a result, it is good practice to organize these files into subdirectories based on file purpose. Because several project files are in the root user's home directory in the preceding output, you could create a subdirectory called proj_files to contain the project-related files and decrease the size of the directory listing. To do this, you use the **mkdir (make directory) command**, which takes arguments specifying the absolute or relative pathnames of the directories to create. To create a proj_files directory under the current directory, you can use the `mkdir` command with a relative pathname:

```
[root@server1 ~]# mkdir proj_files
[root@server1 ~]# ls -F
myprogram* project  project12  project2  project4   proj_files/
myscript*  project1 project13  project3  project5
[root@server1 ~]#_
```

Now, you can move the project files into the proj_files subdirectory by using the **mv (move) command**. The `mv` command requires two arguments at minimum: the **source file/directory** and the **target file/directory**. For example, to move the /etc/sample1 file to the /root directory, you could use the command `mv /etc/sample1 /root`.

If you want to move several files, you include one source argument for each file you want to move and then include the target directory as the last argument. For example, to move the /etc/sample1 and /etc/sample2 files to the /root directory, you could use the command `mv /etc/sample1 /etc/sample2 /root`.

Note that both the source (or sources) and the destination can be absolute or relative pathnames and the source can contain wildcards if several files are to be moved. For example, to move all of the project files to the proj_files directory, you could type `mv` with the source argument `project*` (to match all files starting with the letters "project") and the target argument `proj_files` (relative pathname to the destination directory), as shown in the following output:

```
[root@server1 ~]# mv project* proj_files
[root@server1 ~]# ls -F
myprogram*  myscript*  proj_files/
[root@server1 ~]# ls -F proj_files
project  project12  project2  project4
project1 project13  project3  project5
[root@server1 ~]#_
```

In the preceding output, the current directory listing does not show the project files anymore, yet the listing of the proj_files subdirectory indicates that they were moved successfully.

Note 2

If the target is the name of a directory, the `mv` command moves those files to that directory. If the target is a filename of an existing file in a certain directory and there is one source file, the `mv` command overwrites the target with the source. If the target is a filename of a nonexistent file in a certain directory, the `mv` command creates a new file with that filename in the target directory and moves the source file to that file.

Another important use of the mv command is to rename files, which is simply moving a file to the same directory but with a different filename. To rename the myscript file from earlier examples to myscript2, you can use the following mv command:

```
[root@server1 ~]# ls -F
myprogram*  myscript*  proj_files/
[root@server1 ~]# mv myscript myscript2
[root@server1 ~]# ls -F
myprogram*  myscript2*  proj_files/
[root@server1 ~]#_
```

Similarly, the mv command can rename directories. If the source is the name of an existing directory, it is renamed to whatever directory name is specified as the target.

The mv command works similarly to a cut-and-paste operation in which the file is copied to a new directory and deleted from the source directory. In some cases, however, you might want to keep the file in the source directory and instead insert a copy of the file in the target directory. You can do this using the **cp (copy) command**. Much like the mv command, the cp command takes two arguments at minimum. The first argument specifies the source file/directory to be copied and the second argument specifies the target file/directory. If several files need to be copied to a destination directory, specify several source arguments, with the final argument on the command line serving as the target directory. Each argument can be an absolute or relative pathname and can contain wildcards or the special metacharacters "." (which specifies the current directory) and ".." (which specifies the parent directory). For example, to make a copy of the file /etc/hosts in the current directory (/root), you can specify the absolute pathname to the /etc/hosts file (/etc/hosts) and the relative pathname indicating the current directory (.):

```
[root@server1 ~]# cp /etc/hosts .
[root@server1 ~]# ls -F
hosts  myprogram*  myscript2*  proj_files/
[root@server1 ~]#_
```

You can also make copies of files in the same directory. For example, to make a copy of the hosts file called hosts2 in the current directory and view the results, you can run the following commands:

```
[root@server1 ~]# cp hosts hosts2
[root@server1 ~]# ls -F
hosts hosts2 myprogram* myscript2* proj_files/
[root@server1 ~]#_
```

Despite their similarities, the mv and cp commands work on directories differently. The mv command renames a directory, whereas the cp command creates a whole new copy of the directory and its contents. To copy a directory full of files in Linux, you must tell the cp command that the copy will be **recursive** (involve files and subdirectories too) by using the -r option. The following example demonstrates copying the proj_files directory and all of its contents to the /home/user1 directory without and with the -r option:

```
[root@server1 ~]# ls -F
hosts  myprogram*  myscript2*  proj_files/
[root@server1 ~]# ls -F /home/user1
Desktop/
[root@server1 ~]# cp proj_files /home/user1
cp: -r not specified; omitting directory 'proj_files'
[root@server1 ~]# ls -F /home/user1
Desktop/
[root@server1 ~]# cp -r proj_files /home/user1
[root@server1 ~]# ls -F /home/user1
```

```
Desktop/  proj_files/
[root@server1 ~]#_
```

If the target is a file that exists, both the mv and cp commands warn the user that the target file will be overwritten and will ask whether to continue. This is not a feature of the command as normally invoked, but it is a feature of the default configuration in Fedora Linux because the BASH shell in Fedora Linux contains aliases to the cp and mv commands.

Note 3

Aliases are special variables in memory that point to commands; they are fully discussed in Chapter 7.

When you type mv, you are actually running the mv command with the –i option without realizing it. If the target file already exists, both the mv command and the mv command with the –i option interactively prompt the user to choose whether to overwrite the existing file. Similarly, when you type the cp command, the cp –i command is actually run to prevent the accidental overwriting of files. To see the aliases in your current shell, type alias, as shown in the following output:

```
[root@server1 ~]# alias
alias cp='cp -i'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='(alias; declare -f) | /usr/bin/which --tty-only --read-
alias --read-functions --show-tilde --show-dot'
alias xzegrep='xzegrep --color=auto'
alias xzfgrep='xzfgrep --color=auto'
alias xzgrep='xzgrep --color=auto'
alias zegrep='zegrep --color=auto'
alias zfgrep='zfgrep --color=auto'
alias zgrep='zgrep --color=auto'
[root@server1 ~]#_
```

If you want to override this interactive option, which is known as **interactive mode**, use the –f (force) option to override the choice, as shown in the following example. In this example, the root user tries to rename the hosts file using the name "hosts2," a name already assigned to an existing file. The example shows the user attempting this task both without and with the –f option to the mv command:

```
[root@server1 ~]# ls -F
hosts  hosts2  myprogram*  myscript2*  proj_files/
[root@server1 ~]# mv hosts hosts2
mv: overwrite 'hosts2'? n
[root@server1 ~]# mv -f hosts hosts2
[root@server1 ~]# ls -F
hosts2  myprogram*  myscript2*  proj_files/
[root@server1 ~]#_
```

Creating directories, copying, and moving files are file management tasks that preserve or create data on the filesystem. To remove files or directories, you must use either the rm command or the rmdir command.

The **`rm` (remove) command** takes a list of arguments specifying the absolute or relative pathnames of files to remove. As with most commands, wildcards can be used to simplify the process of removing multiple files. After a file has been removed from the filesystem, it cannot be recovered. As a result, the `rm` command is aliased in Fedora Linux to the `rm` command with the `–i` option, which interactively prompts the user to choose whether to continue with the deletion. Like the `cp` and `mv` commands, the `rm` command accepts the `–f` option to override this choice and immediately delete the file. An example demonstrating the use of the `rm` and `rm –f` commands to remove the current and hosts2 files is shown here:

```
[root@server1 ~]# ls -F
hosts2   myprogram*   myscript2*   proj_files/
[root@server1 ~]# rm myprogram
rm: remove regular file 'myprogram'? y
[root@server1 ~]# rm -f hosts2
[root@server1 ~]# ls -F
myscript2*   proj_files/
[root@server1 ~]#_
```

To remove a directory, you can use the **`rmdir` (remove directory) command**; however, the `rmdir` command only removes a directory if it contains no files. To remove a directory and the files inside, you must use the `rm` command and specify that a directory full of files should be removed. As explained earlier in this chapter, you need to use the recursive option (`–r`) with the `cp` command to copy directories; to remove a directory full of files, you can also use a recursive option (`–r`) with the `rm` command. In the following example, the proj_files subdirectory and all of the files within it are removed without being prompted to confirm each file deletion by the `rm –rf proj_files` command:

```
[root@server1 ~]# ls -F
myscript2*   proj_files/
[root@server1 ~]# rmdir proj_files
rmdir: failed to remove 'proj_files': Directory not empty
[root@server1 ~]# rm -rf proj_files
[root@server1 ~]# ls -F
myscript2*
[root@server1 ~]#_
```

Note 4

In many commands, such as `rm` and `cp`, both the `–r` and the `–R` options have the same meaning (recursive).

Note 5

The recursive (`–r` or `-R`) option to the `rm` command is dangerous if you are not certain which files exist in the directory to be deleted recursively. As a result, this option to the `rm` command is commonly referred to as the résumé option; if you use it incorrectly in a production server environment, you might need to prepare your résumé, as there is no Linux command to restore deleted files.

Note 6

An alternative to the `rm` command is the **`unlink` command**. However, the `unlink` command can be used to remove files only (not directories).

The aforementioned file management commands are commonly used by Linux users, developers, and administrators alike. Table 4-2 shows a summary of these common file management commands.

Table 4-2 Common linux file management commands

Command	Description
mkdir	Creates directories
rmdir	Removes empty directories
mv	Moves/renames files and directories
cp	Copies files and directories full of files (with the -r or -R option)
alias	Displays BASH shell aliases
rm	Removes files and directories full of files (with the -r or -R option)
unlink	Removes files

Finding Files

Before using the file management commands mentioned in the preceding section, you must know the locations of the files involved. The fastest method to search for files in the Linux directory tree is to use the **locate command**. For example, to view all of the files under the root directory with the text "inittab" or with "inittab" as part of the filename, you can type `locate inittab` at a command prompt, which produces the following output:

```
[root@server1 ~]# locate inittab
/etc/inittab
/usr/share/augeas/lenses/dist/inittab.aug
/usr/share/vim/vim90/syntax/inittab.vim
[root@server1 ~]#_
```

The `locate` command looks in a premade database that contains a list of all the files on the system. This database is indexed much like a textbook for fast searching, yet it can become outdated as files are added and removed from the system, which happens on a regular basis. As a result, the database used for the `locate` command (/var/lib/plocate/plocate.db) is updated each day automatically and can be updated manually by running the `updatedb` command at a command prompt. You can also exclude specific directories, file extensions, and whole filesystems from being indexed by the `updatedb` command by adding them to the /etc/updatedb.conf file; this is called **pruning**. For example, to exclude the /etc directory from being indexed by the `updatedb` command, add the line `PRUNEPATHS=/etc` to the /etc/updatedb.conf file.

As the `locate` command searches all files on the filesystem, it often returns too much information to display on the screen. To make the output easier to read, you can use the `more` or `less` command to pause the output; if the `locate inittab` command produced too many results, you could run the command `locate inittab | less`. To prevent the problem entirely, you can do more specific searches.

A slower, yet more versatile, method for locating files on the filesystem is to use the **find command**. The `find` command does not use a premade index of files; instead, it searches the directory tree recursively, starting from a certain directory, for files that meet a certain criterion. The format of the `find` command is as follows:

```
find <start directory> -criterion <what to find>
```

For example, to find any files named "inittab" under the /etc directory, you can use the command `find /etc -name inittab` and receive the following output:

```
[root@server1 ~]# find /etc -name inittab
/etc/inittab
[root@server1 ~]#_
```

You can also use wildcard metacharacters with the find command; however, these wildcards must be protected from shell interpretation, as they must only be interpreted by the find command. To do this, ensure that any wildcard metacharacters are enclosed within quote characters. An example of using the find command with wildcard metacharacters to find all files that start with the letters "host" underneath the /etc directory is shown in the following output:

```
[root@server1 ~]# find /etc -name "host*"
/etc/hosts
/etc/host.conf
/etc/avahi/hosts
/etc/hostname
[root@server1 ~]#_
```

Although searching by name is the most common criterion used with the find command, many other criteria can be used with the find command as well. To find all files starting from the /var directory that have a size greater than 8192K (kilobytes), you can use the following command:

```
[root@server1 ~]# find /var -size +8192k
/var/lib/rpm/Packages
/var/lib/rpm/Basenames
/var/lib/PackageKit/system.package-list
/var/log/journal/034ec8ccdf4642f7a2493195e11d7df6/user-1000.journal
/var/log/journal/034ec8ccdf4642f7a2493195e11d7df6/user-42.journal
/var/log/journal/034ec8ccdf4642f7a2493195e11d7df6/system.journal
/var/cache/PackageKit/36/updates/gen/prestodelta.xml
/var/cache/PackageKit/36/updates/gen/primary_db.sqlite
/var/cache/PackageKit/36/updates/gen/filelists_db.sqlite
/var/cache/PackageKit/36/updates/gen/updateinfo.xml
/var/cache/PackageKit/36/updates/gen/other_db.sqlite
/var/cache/PackageKit/36/fedora-filenames.solvx
/var/cache/dnf/x86_64/36/fedora.solv
/var/cache/dnf/x86_64/36/updates-filenames.solvx
/var/tmp/kdecache-user1/plasma_theme_internal-system-colors.kcache
/var/tmp/kdecache-user1/plasma_theme_Heisenbug_v19.90.2.kcache
/var/tmp/kdecache-user1/icon-cache.kcache
[root@server1 ~]#_
```

As well, if you want to find all the directories only under the /boot directory, you can type the following command:

```
[root@server1 ~]# find /boot -type d
/boot
/boot/extlinux
/boot/lost+found
/boot/grub2
/boot/grub2/fonts
/boot/efi
/boot/efi/EFI
/boot/efi/EFI/BOOT
/boot/efi/EFI/fedora
/boot/efi/System
/boot/efi/System/Library
/boot/efi/System/Library/CoreServices
/boot/loader
/boot/loader/entries
[root@server1 ~]#_
```

Table 4-3 provides a list of some common criteria used with the `find` command.

Table 4-3 Common criteria used with the `find` command

Criteria	Description
`-amin -x`	Searches for files that were accessed less than *x* minutes ago
`-amin +x`	Searches for files that were accessed more than *x* minutes ago
`-atime -x`	Searches for files that were accessed less than *x* days ago
`-atime +x`	Searches for files that were accessed more than *x* days ago
`-empty`	Searches for empty files or directories
`-fstype x`	Searches for files if they are on a certain filesystem *x* (where x could be ext2, ext3, and so on)
`-group x`	Searches for files that are owned by a certain group or GID (*x*)
`-inum x`	Searches for files that have an inode number of *x*
`-mmin -x`	Searches for files that were modified less than *x* minutes ago
`-mmin +x`	Searches for files that were modified more than x minutes ago
`-mtime -x`	Searches for files that were modified less than *x* days ago
`-mtime +x`	Searches for files that were modified more than *x* days ago
`-name x`	Searches for a certain filename *x* (*x* can contain wildcards)
`-regex x`	Searches for certain filenames using regular expressions instead of wildcard metacharacters
`-size -x`	Searches for files with a size less than *x*
`-size x`	Searches for files with a size of *x*
`-size +x`	Searches for files with a size greater than *x*
`-type x`	Searches for files of type *x* where *x* is: b for block files c for character files d for directory files p for named pipes f for regular files l for symbolic links (shortcuts) s for sockets
`-user x`	Searches for files owned by a certain user or UID (*x*)

Although the `find` command can be used to search for files based on many criteria, it might take several minutes to complete the search if the number of directories and files being searched is large. To reduce the time needed to search, narrow the directories searched by specifying a subdirectory when possible. It takes less time to search the /usr/local/bin directory and its subdirectories, compared to searching the /usr directory and all of its subdirectories. As well, if the filename that you are searching for is an executable file, that file can likely be found in less time using the **`which` command**. The `which` command only searches directories that are listed in a special variable called the **PATH variable** in the current BASH shell. Before exploring the `which` command, you must understand the usage of PATH.

Executable files can be stored in directories scattered around the directory tree. Recall from FHS that most executable files are stored in directories named bin or sbin, yet there are over 20 bin and sbin directories scattered around the directory tree after a typical Fedora Linux installation. To ensure that users do not need to specify the full pathname to commands such as `ls` (which is the executable file /usr/bin/ls), a special variable called PATH is placed into memory each time a user logs in to the Linux system. Recall that you can see the contents of a certain variable in memory by using the $ metacharacter with the `echo` command:

```
[root@server1 ~]# echo $PATH
/root/.local/bin:/root/bin:/usr/local/sbin:
/usr/local/bin:/usr/sbin:/usr/bin
[root@server1 ~]#_
```

The PATH variable lists directories that are searched for executable files if a relative or absolute pathname was not specified when executing a command on the command line. Assuming the PATH variable in the preceding output, if a user types the `ls` command on the command line and presses Enter, the system recognizes that the command was not an absolute pathname (e.g., /usr/bin/ls) or relative pathname (e.g., ../../usr/bin/ls) and then proceeds to look for the ls executable file in the /root/.local/bin directory, then the /root/bin directory, then the /usr/local/sbin directory, then the /usr/local/bin directory, then the /usr/sbin directory, and finally the /usr/bin directory. If all the directories in the PATH variable are searched and no `ls` command is found, the shell gives an error message to the user stating that the command was not found. In the preceding output, the /usr/bin directory is in the PATH variable and, thus, the `ls` command is found and executed, but not until the previous directories in the PATH variable are searched first.

To search the directories in the PATH variable for the file called "grep," you could use the word "grep" as an argument for the `which` command and receive the following output:

```
[root@server1 ~]# which grep
alias grep='grep --color=auto'
        /usr/bin/grep
[root@server1 ~]#_
```

As shown in the previous output, the `which` command will also list any command aliases for a particular command. In this example, the `grep` command has the path /usr/bin/grep, but it also has an alias that ensures that each time the user runs the `grep` command, it runs it using the `--color=auto` option.

If the file being searched does not exist in the PATH variable directories, the `which` command lets you know in which directories it was not found, as shown in the following output:

```
[root@server1 ~]# which grepper
/usr/bin/which: no grepper in
(/root/.local/bin:/root/bin:/usr/local/sbin:/usr/local/bin:
/usr/sbin:/usr/bin)
[root@server1 ~]#_
```

There are two alternatives to the `which` command: the **`type` command** displays only the first result normally outputted by the `which` command, and the **`whereis` command** displays the location of the command as well as any associated man and info pages, as shown in the following output:

```
[root@server1 ~]# type grep
grep is aliased to 'grep --color=auto'
[root@server1 ~]# whereis grep
grep: /usr/bin/grep /usr/share/man/man1/grep.1.gz
/usr/share/man/man1p/grep.1p.gz /usr/share/info/grep.info.gz
[root@server1 ~]#_
```

Linking Files

Files can be linked to one another in two ways. In a **symbolic link**, one file is a pointer, or shortcut, to another file. In a **hard link**, two files share the same data.

To better understand how files are linked, you must understand how files are stored on a filesystem. On a structural level, a filesystem has three main sections:

- Superblock
- Inode table
- Data blocks

The **superblock** is the section that contains information about the filesystem in general, such as the number of inodes and data blocks, as well as how much data a data block stores, in kilobytes. The **inode table** consists of several **inodes** (information nodes); each inode describes one file or directory on the filesystem and contains a unique inode number for identification. What is more important, the inode stores information such as the file size, data block locations, last date modified, permissions, and ownership. When a file is deleted, only its inode (which serves as a pointer to the actual data) is deleted. The data that makes up the contents of the file as well as the filename are stored in **data blocks**, which are referenced by the inode. In filesystem-neutral terminology, blocks are known as allocation units because they are the unit by which disk space is allocated for storage.

Note 7

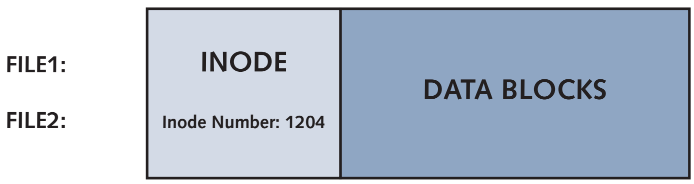
Each file and directory must have an inode. All files except for special device files also have data blocks associated with the inode. Special device files are discussed in Chapter 5.

Note 8

Recall that directories are simply files that are used to organize other files; they too have an inode and data blocks, but their data blocks contain a list of filenames that are located within the directory.

Hard-linked files share the same inode and inode number. As a result, they share the same inode number and data blocks, but the data blocks allow for multiple filenames. Thus, when one hard-linked file is modified, the other hard-linked files are updated as well. This relationship between hard-linked files is shown in Figure 4-1. You can hard-link a file an unlimited number of times; however, the hard-linked files must reside on the same filesystem.

Figure 4-1 The structure of hard-linked files



To create a hard link, you must use the **`ln` (link) command** and specify two arguments: the existing file to hard-link and the target file that will be created as a hard link to the existing file. Each argument can be the absolute or relative pathname to a file. Take, for example, the following contents of the root user's home directory:

```
[root@server1 ~]# ls -l
total 520
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
-rwxr-xr-x    1 root     root        519964 Apr  7 09:59 file1
-rwxr-xr-x    1 root     root          1244 Apr 27 18:17 file3
[root@server1 ~]#_
```

Suppose you want to make a hard link to file1 and call the new hard link file2. To accomplish this, you issue the command `ln file1 file2` at the command prompt; a file called file2 is created and hard-linked to file1. To view the hard-linked filenames after creation, you can use the `ls -l` command:

```
[root@server1 ~]# ln file1 file2
[root@server1 ~]# ls -l
total 1032
drwx------    3 root     root         4096 Apr  8 07:12 Desktop
-rwxr-xr-x    2 root     root       519964 Apr  7 09:59 file1
-rwxr-xr-x    2 root     root       519964 Apr  7 09:59 file2
-rwxr-xr-x    1 root     root         1244 Apr 27 18:17 file3
[root@server1 ~]#_
```

Notice from the preceding long listing that file1 and file2 share the same inode and data section, as they have the same size, permissions, ownership, modification date, and so on. Also note that the link count (the number after the permission set) for file1 has increased from the number one to the number two in the preceding output. A link count of one indicates that only one inode is shared by the file. A file that is hard-linked to another file shares two inodes and, thus, has a link count of two. Similarly, a file that is hard-linked to three other files shares four inodes and, thus, has a link count of four.

Although hard links share the same inode and data section, deleting a hard-linked file does not delete all the other hard-linked files; it simply removes one filename reference. Removing a hard link can be achieved by removing one of the files, which then lowers the link count.

To view the inode number of hard-linked files to verify that they are identical, you can use the `-i` option to the `ls` command in addition to any other options. The inode number is placed on the left of the directory listing on each line, as shown in the following output:

```
[root@server1 ~]# ls -li
total 1032
  37595 drwx------    3 root     root         4096 Apr  8 07:12 Desktop
   1204 -rwxr-xr-x    2 root     root       519964 Apr  7 09:59 file1
   1204 -rwxr-xr-x    2 root     root       519964 Apr  7 09:59 file2
  17440 -rwxr-xr-x    1 root     root         1244 Apr 27 18:17 file3
[root@server1 ~]#_
```

Note 9

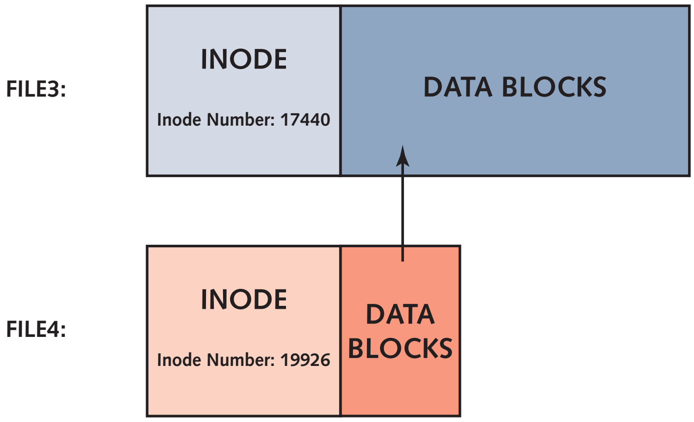
Directory files are not normally hard-linked on modern Linux systems, as the result would consist of two directories that contain the same contents. However, the root user has the ability to hard-link directories, using the `-F` or `-d` option to the `ln` command.

Symbolic links (shown in Figure 4-2) are different from hard links because they do not share the same inode and data blocks with their target file; one is merely a pointer to the other, thus both files have different sizes. The data blocks in a symbolically linked file contain only the pathname to the target file. When a user edits a symbolically linked file, the user is actually editing the target file. Thus, if the target file is deleted, the symbolic link serves no function, as it points to a nonexistent file.

Note 10

Symbolic links are sometimes referred to as "soft links" or "symlinks"

Figure 4-2 The structure of symbolically linked files



To create a symbolic link, you use the `–s` option to the `ln` command. For example, to create a symbolic link to file3 called file4 you can type `ln –s file3 file4` at the command prompt. As with hard links, the arguments specified can be absolute or relative pathnames. To view the symbolically linked filenames after creation, you can use the `ls –l` command, as shown in the following example:

```
[root@server1 ~]# ln -s file3 file4
[root@server1 ~]# ls -l
total 1032
drwx------     3 root     root          4096 Apr   8 07:12 Desktop
-rwxr-xr-x     2 root     root        519964 Apr   7 09:59 file1
-rwxr-xr-x     2 root     root        519964 Apr   7 09:59 file2
-rwxr-xr-x     1 root     root          1244 Apr 27 18:17 file3
lrwxrwxrwx     1 root     root             5 Apr 27 19:05 file4 -> file3
[root@server1 ~]#_
```

Notice from the preceding output that file4 does not share the same inode, because the permissions, size, and modification date are different from file3. In addition, symbolic links are easier to identify than hard links; the file type character (before the permissions) is l, which indicates a symbolic link, and the filename points to the target using an arrow. The `ls –F` command also indicates symbolic links by appending an @ symbol, as shown in the following output:

```
[root@server1 ~]# ls -F
Desktop/  file1*  file2*  file3*  file4@
[root@server1 ~]#_
```

Another difference between hard links and symbolic links is that symbolic links need not reside on the same filesystem as their target. Instead, they point to the target filename and do not require the same inode number, as shown in the following output:

```
[root@server1 ~]# ls -li
total 1032
37595 drwx------    3 root   root      4096 Apr   8 07:12 Desktop
 1204 -rwxr-xr-x    2 root   root    519964 Apr   7 09:59 file1
 1204 -rwxr-xr-x    2 root   root    519964 Apr   7 09:59 file2
17440 -rwxr-xr-x    1 root   root      1244 Apr 27 18:17 file3
19926 lrwxrwxrwx    1 root   root         5 Apr 27 19:05 file4 -> file3
[root@server1 ~]#_
```

Note 11

Unlike hard links, symbolic links are commonly made to directories to simplify navigating the filesystem tree. Also, symbolic links made to directories are typically used to maintain FHS compatibility with other UNIX and Linux systems. For example, on Fedora Linux, the /usr/tmp directory is symbolically linked to the /var/tmp directory for this reason.

File and Directory Permissions

Recall that all users must log in with a user name and password to gain access to a Linux system. After logging in, a user is identified by their user name and group memberships; all access to resources depends on whether the user name and group memberships have the required **permissions**. Thus, a firm understanding of ownership and permissions is necessary to operate a Linux system in a secure manner and to prevent unauthorized users from having access to sensitive files, directories, and commands.

File and Directory Ownership

When a user creates a file or directory, that user's name and **primary group** becomes the owner and group owner of the file, respectively. This affects the permission structure, as you see in the next section; however, it also determines who has the ability to modify file and directory permissions and ownership. Only two users on a Linux system can modify permissions on a file or directory or change its ownership: the owner of the file or directory and the root user.

To view your current user name, you can use the `whoami` command. To view your group memberships and primary group, you can use the `groups` command. An example of these two commands when logged in as the root user is shown in the following output:

```
[root@server1 ~]# whoami
root
[root@server1 ~]# groups
root bin daemon sys adm disk wheel
[root@server1 ~]#_
```

Notice from the preceding output that the root user is a member of seven groups, yet the root user's primary group is also called "root," as it is the first group mentioned in the output of the `groups` command.

Note 12

On Fedora Linux, the root user is only a member of one group by default (the "root" group).

If the root user creates a file, the owner is "root" and the group owner is also "root." To quickly create an empty file, you can use the **`touch` command**:

```
[root@server1 ~]# touch file1
[root@server1 ~]# ls -l
total 4
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
-rw-r--r--    1 root     root             0 Apr 29 15:40 file1
[root@server1 ~]#_
```

Note 13

Although the main purpose of the `touch` command is to update the modification date on an existing file to the current time, it will create a new empty file if the file specified as an argument does not exist.

Notice from the preceding output that the owner of file1 is "root" and the group owner is the "root" group. To change the ownership of a file or directory, you can use the **chown (change owner) command**, which takes two arguments at minimum: the new owner and the files or directories to change. Both arguments can be absolute or relative pathnames, and you can also change permissions recursively throughout the directory tree using the –R option to the chown command. To change the ownership of file1 to the user user1 and the ownership of the directory Desktop and all of its contents to user1 as well, you can enter the following commands:

```
[root@server1 ~]# chown user1 file1
[root@server1 ~]# chown -R user1 Desktop
[root@server1 ~]# ls -l
total 4
drwx------    3 user1    root      4096 Apr  8 07:12 Desktop
-rw-r--r--    1 user1    root         0 Apr 29 15:40 file1
[root@server1 ~]# ls -l Desktop
total 16
-rw-------    1 user1    root       163 Mar 29 09:58 Work
-rw-r--r--    1 user1    root      3578 Mar 29 09:58 Home
-rw-r--r--    1 user1    root      1791 Mar 29 09:58 Start Here
drwx------    2 user1    root      4096 Mar 29 09:58 Trash
[root@server1 ~]#_
```

Recall that the owner of a file or directory and the root user can change ownership of a particular file or directory. If a regular user changes the ownership of a file or directory that they own, that user cannot gain back the ownership. Instead, the new owner of that file or directory must change it to the original user. However, the root user always has the ability to regain the ownership:

```
[root@server1 ~]# chown root file1
[root@server1 ~]# chown -R root Desktop
[root@server1 ~]# ls -l
total 4
drwx------    3 root     root      4096 Apr  8 07:12 Desktop
-rw-r--r--    1 root     root         0 Apr 29 15:40 file1
[root@server1 ~]# ls -l Desktop
total 16
-rw-------    1 root     root       163 Mar 29 09:58 Work
-rw-r--r--    1 root     root      3578 Mar 29 09:58 Home
-rw-r--r--    1 root     root      1791 Mar 29 09:58 Start Here
drwx------    2 root     root      4096 Mar 29 09:58 Trash
[root@server1 ~]#_
```

Just as the chown (change owner) command can be used to change the owner of a file or directory, you can use the **chgrp (change group) command** to change the group owner of a file or directory. The chgrp command takes two arguments at minimum: the new group owner and the files or directories to change. As with the chown command, the chgrp command also accepts the –R option to change group ownership recursively throughout the directory tree. To change the group owner of file1 and the Desktop directory recursively throughout the directory tree, you can execute the following commands:

```
[root@server1 ~]# chgrp sys file1
[root@server1 ~]# chgrp -R sys Desktop
[root@server1 ~]# ls -l
total 4
drwx------    3 root     sys       4096 Apr  8 07:12 Desktop
-rw-r--r--    1 root     sys          0 Apr 29 15:40 file1
```

```
[root@server1 ~]# ls -l Desktop
total 16
-rw-------    1 root     sys       163 Mar 29 09:58 Work
-rw-r--r--    1 root     sys      3578 Mar 29 09:58 Home
-rw-r--r--    1 root     sys      1791 Mar 29 09:58 Start Here
drwx------    2 root     sys      4096 Mar 29 09:58 Trash
[root@server1 ~]#_
```

Note 14

Regular users can change the group of a file or directory only to a group to which they belong.

Normally, you change both the ownership and group ownership on a file when that file needs to be maintained by someone else. As a result, you can change both the owner and the group owner at the same time using the chown command. To change the owner to user1 and the group owner to root for file1 and the directory Desktop recursively, you can enter the following commands:

```
[root@server1 ~]# chown user1.root file1
[root@server1 ~]# chown -R user1.root Desktop
[root@server1 ~]# ls -l
total 4
drwx------    3 user1    root     4096 Apr  8 07:12 Desktop
-rw-r--r--    1 user1    root        0 Apr 29 15:40 file1
[root@server1 ~]# ls -l Desktop
total 16
-rw-------    1 user1    root      163 Mar 29 09:58 Work
-rw-r--r--    1 user1    root     3578 Mar 29 09:58 Home
-rw-r--r--    1 user1    root     1791 Mar 29 09:58 Start Here
drwx------    2 user1    root     4096 Mar 29 09:58 Trash
[root@server1 ~]#_
```

Note that there must be no spaces before and after the . character in the chown commands shown in the preceding output.

Note 15

You can also use the **:** character instead of the **.** character in the chown command to change both the owner and group ownership (e.g., chown –R user1:root Desktop).

To protect your system's security, you should ensure that most files residing in a user's home directory are owned by that user; some files in a user's home directory (especially the hidden files and directories) require this to function properly. To change the ownership back to the root user for file1 and the Desktop directory to avoid future problems, you can type the following:

```
[root@server1 ~]# chown root.root file1
[root@server1 ~]# chown -R root.root Desktop
[root@server1 ~]# ls -l
total 4
drwx------    3 root     root     4096 Apr  8 07:12 Desktop
-rw-r--r--    1 root     root        0 Apr 29 15:40 file1
[root@server1 root]# ls -l Desktop
total 16
-rw-------    1 root     root      163 Mar 29 09:58 Work
```

```
-rw-r--r--    1 root     root      3578 Mar 29 09:58 Home
-rw-r--r--    1 root     root      1791 Mar 29 09:58 Start Here
drwx------    2 root     root      4096 Mar 29 09:58 Trash
[root@server1 ~]#_
```

Note 16

You can override who is allowed to change ownership and permissions using a kernel setting. Many Linux distributions, including Fedora Linux, use this kernel setting by default to restrict regular (non-root) users from changing the ownership and group ownership of files and directories. This prevents these users from bypassing disk quota restrictions, which rely on the ownership of files and directories to function properly. Disk quotas are discussed in Chapter 5.

Managing File and Directory Permissions

Every file and directory file on a Linux filesystem contains information regarding permissions in its inode. The section of the inode that stores permissions is called the **mode** of the file and is divided into three sections based on the user(s) who receive the permissions to that file or directory:

- User (owner) permissions
- Group (group owner) permissions
- Other (everyone else on the Linux system) permissions

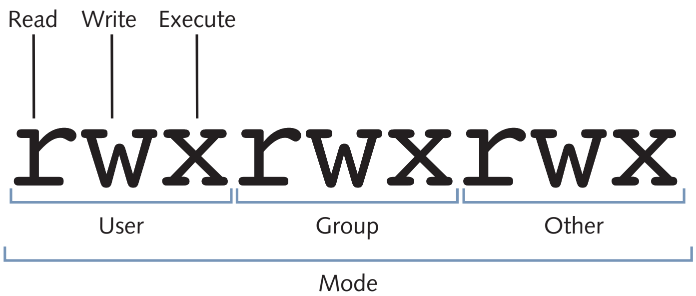
Furthermore, you can assign to each of these users the following regular permissions:

- Read
- Write
- Execute

Interpreting the Mode

Recall that the three sections of the mode and the permissions that you can assign to each section are viewed when you perform an `ls –l` command; a detailed depiction of this is shown in Figure 4-3. Note that the root user supersedes all file and directory permissions; in other words, the root user has all permissions to every file and directory regardless of what the mode of the file or directory indicates.

Figure 4-3 The structure of a mode



Consider the root user's home directory listing shown in the following example:

```
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
-r---w---x    1 bob      proj           282 Apr 29 22:06 file1
-------rwx    1 root     root           282 Apr 29 22:06 file2
```

```
-rwxrwxrwx    1 root     root          282 Apr 29 22:06 file3
----------    1 root     root          282 Apr 29 22:06 file4
-rw-r--r--    1 root     root          282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys           282 Apr 29 22:06 file6
[root@server1 ~]#_
```

Note from the preceding output that all permissions (as shown in Figure 4-3) need not be on a file or directory; if the permission is unavailable, a dash character (-) replaces its position in the mode. Be certain not to confuse the character to the left of the mode (which determines the file type) with the mode, as it is unrelated to the permissions on the file or directory. From the preceding output, the Desktop directory gives the **user** or **owner** of the directory (the root user) read, write, and execute permission, yet members of the **group** (the root group) do not receive any permissions to the directory. Note that **other** (everyone on the system) does not receive permissions to this directory either.

Permissions are not additive; the system assigns the first set of permissions that are matched in the mode order: user, group, other. Let us assume that the bob user is a member of the proj group. In this case, the file called file1 in the preceding output gives the user or owner of the file (the bob user) read permission, gives members of the group (the proj group) write permission, and gives other (everyone else on the system) execute permission only. Because permissions are not additive, the bob user will only receive read permission to file1 from the system.

Linux permissions should not be assigned to other only. Although file2 in our example does not give the user or group any permissions, all other users receive read, write, and execute permission via the other category. Thus, file2 should not contain sensitive data because many users have full access to it. For the same reason, it is bad form to assign all permissions to a file that contains sensitive data, as shown with file3 in the preceding example.

On the contrary, it is also possible to have a file that has no permissions assigned to it, as shown in the preceding example with respect to file4. In this case, the only user who has permissions to the file is the root user.

The permission structure that you choose for a file or directory might result in too few or too many permissions. You can follow some general guidelines to avoid these situations. The owner of a file or directory is typically the person who maintains it; members of the group are typically users in the same department or project and must have limited access to the file or directory. As a result, most files and directories that you find on a Linux filesystem have more permissions assigned to the user of the file/directory than to the group of the file/directory, and the other category has either the same permissions or less than the group of the file/directory, depending on how private that file or directory is. The file file5 in the previous output depicts this common permission structure. In addition, files in a user's home directory are typically owned by that user; however, you might occasionally find files that are not owned by that user. For these files, their permission definition changes, as shown in the previous example with respect to file1 and file6. The user (or owner) of file6 is user1, who has read and write permissions to the file. The group owner of file6 is the sys group; thus, any members of the sys group have read permission to the file. Finally, everyone on the system receives read permission to the file via the other category. Regardless of the mode, the root user receives all permissions to this file.

Interpreting Permissions

After you understand how to identify the permissions that are applied to user, group, and other on a certain file or directory, you can then interpret the function of those permissions. Permissions for files are interpreted differently than those for directories. Also, if a user has a certain permission on a directory, that user does not have the same permission for all files or subdirectories within that directory; file and directory permissions are treated separately by the Linux system. Table 4-4 shows a summary of the different permissions and their definitions.

Table 4-4 Linux permissions

Permission	Definition for Files	Definition for Directories
Read	Allows a user to open and read the contents of a file	Allows a user to list the contents of the directory (if the user has also been given execute permission)
Write	Allows a user to open, read, and edit the contents of a file	Allows a user to add or remove files to and from the directory (if the user has also been given execute permission)
Execute	Allows a user to execute the file in memory (if it is a program file or script)	Allows a user to enter the directory and work with directory contents

The implications of the permission definitions described in Table 4-4 are important to understand. If a user has the read permission to a text file, that user can use, among others, the `cat`, `more`, `head`, `tail`, `less`, `strings`, and `od` commands to view its contents. That same user can also open that file with a text editor such as vi; however, the user does not have the ability to save any changes to the document unless that user has the write permission to the file as well.

Recall from earlier that some text files contain instructions for the shell to execute and are called shell scripts. Shell scripts can be executed in much the same way that binary compiled programs are; the user who executes the shell script must then have execute permission to that file to execute it as a program.

Note 17

Avoid giving execute permission to files that are not programs or shell scripts. This ensures that these files will not be executed accidentally, causing the shell to interpret the contents.

Remember that directories are simply special files that have an inode and a data section, but what the data section contains is a list of that directory's contents. If you want to read that list (using the `ls` command for example), then you require the read permission to the directory. To modify that list by adding or removing files, you require the write permission to the directory. Thus, if you want to create a new file in a directory with a text editor such as vi, you must have the write permission to that directory. Similarly, when a source file is copied to a target directory with the `cp` command, a new file is created in the target directory. You must have the write permission to the target directory for the copy to be successful. Conversely, to delete a certain file, you must have the write permission to the directory that contains that file. A user who has the write permission to a directory can delete all files and subdirectories within it.

The execute permission on a directory is sometimes referred to as the search permission, and it works similarly to a light switch. When a light switch is turned on, you can navigate a room and use the objects within it. However, when a light switch is turned off, you cannot see the objects in the room, nor can you walk around and view them. A user who does not have the execute permission to a directory is prevented from listing the directory's contents, adding and removing files, and working with files and subdirectories inside that directory, regardless of what permissions the user has to them. In short, a quick way to deny a user from accessing a directory and all of its contents in Linux is to take away the execute permission on that directory. Because the execute permission on a directory is crucial for user access, it is commonly given to all users via the other category, unless the directory must be private.

Changing Permissions

To change the permissions for a certain file or directory, you can use the **`chmod` (change mode) command**. The `chmod` command takes two arguments at minimum; the first argument specifies the criteria used to change the permissions (see Table 4-5), and the remaining arguments indicate the filenames to change.

Table 4-5 Criteria used within the `chmod` command

Category	Operation	Permission
u (user)	+ (adds a permission)	r (read)
g (group)	- (removes a permission)	w (write)
o (other)	= (makes a permission equal to)	x (execute)
a (all categories)		

Take, for example, the directory list used earlier:

```
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root        4096 Apr  8 07:12 Desktop
-r---w---x    1 bob      proj         282 Apr 29 22:06 file1
-------rwx    1 root     root         282 Apr 29 22:06 file2
-rwxrwxrwx    1 root     root         282 Apr 29 22:06 file3
----------    1 root     root         282 Apr 29 22:06 file4
-rw-r--r--    1 root     root         282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys          282 Apr 29 22:06 file6
[root@server1 ~]#_
```

To change the mode of file1 to rw-r--r--, you must add the write permission to the user of the file, add the read permission and take away the write permission for the group of the file, and add the read permission and take away the execute permission for other.

From the information listed in Table 4-5, you can use the following command:

```
[root@server1 ~]# chmod u+w,g+r-w,o+r-x file1
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root        4096 Apr  8 07:12 Desktop
-rw-r--r--    1 bob      proj         282 Apr 29 22:06 file1
----r--rwx    1 root     root         282 Apr 29 22:06 file2
-rwxrwxrwx    1 root     root         282 Apr 29 22:06 file3
----------    1 root     root         282 Apr 29 22:06 file4
-rw-r--r--    1 root     root         282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys          282 Apr 29 22:06 file6
[root@server1 ~]#_
```

Note 18

You should ensure that there are no spaces between any criteria used in the `chmod` command because all criteria make up the first argument only.

You can also use the = criteria from Table 4-5 to specify the exact permissions to change. To change the mode on file2 in the preceding output to the same as file1 (rw-r--r--), you can use the following `chmod` command:

```
[root@server1 ~]# chmod u=rw,g=r,o=r file2
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root        4096 Apr  8 07:12 Desktop
-rw-r--r--    1 bob      proj         282 Apr 29 22:06 file1
```

```
-rw-r--r--    1 root     root          282 Apr 29 22:06 file2
-rwxrwxrwx    1 root     root          282 Apr 29 22:06 file3
----------    1 root     root          282 Apr 29 22:06 file4
-rw-r--r--    1 root     root          282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys           282 Apr 29 22:06 file6
[root@server1 ~]#_
```

If the permissions to change are identical for the user, group, and other categories, you can use the "a" character to refer to all categories, as shown in Table 4-5 and in the following example, when adding the execute permission to user, group, and other for file1:

```
[root@server1 ~]# chmod a+x file1
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root         4096 Apr  8 07:12 Desktop
-rwxr-xr-x    1 bob      proj          282 Apr 29 22:06 file1
-rw-r--r--    1 root     root          282 Apr 29 22:06 file2
-rwxrwxrwx    1 root     root          282 Apr 29 22:06 file3
----------    1 root     root          282 Apr 29 22:06 file4
-rw-r--r--    1 root     root          282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys           282 Apr 29 22:06 file6
[root@server1 ~]#_
```

However, if there is no character specifying the category of user to affect, all users are assumed, as shown in the following example when adding the execute permission to user, group, and other for file2:

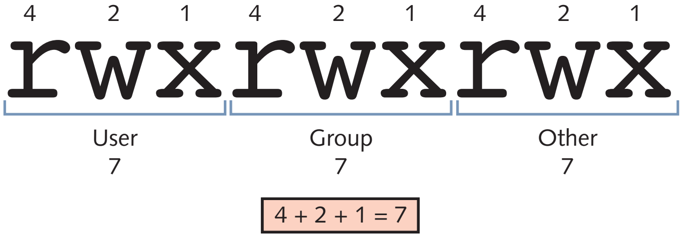
```
[root@server1 ~]# chmod +x file2
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root         4096 Apr  8 07:12 Desktop
-rwxr-xr-x    1 bob      proj          282 Apr 29 22:06 file1
-rwxr-xr-x    1 root     root          282 Apr 29 22:06 file2
-rwxrwxrwx    1 root     root          282 Apr 29 22:06 file3
----------    1 root     root          282 Apr 29 22:06 file4
-rw-r--r--    1 root     root          282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys           282 Apr 29 22:06 file6
[root@server1 ~]#_
```

All of the aforementioned `chmod` examples use the symbols listed in Table 4-5 as the criteria for changing the permissions on a file or directory. You might instead choose to use numeric criteria with the `chmod` command to change permissions. All permissions are stored in the inode of a file or directory as binary powers of two:

- read = 2^2 = 4
- write = 2^1 = 2
- execute = 2^0 = 1

Thus, the mode of a file or directory can be represented using the numbers 421421421 instead of rwxrwxrwx. Because permissions are grouped into the categories user, group, and other, you can then simplify this further by using only three numbers, one for each category that represents the sum of the permissions, as shown in Figure 4-4.

Figure 4-4 Numeric representation of the mode



Similarly, to represent the mode rw-r--r--, you can use the numbers 644 because user has read and write (4 + 2 = 6), group has read (4), and other has read (4). The mode rwxr-x--- can also be represented by 750 because user has read, write, and execute (4 + 2 + 1 = 7), group has read and execute (4 + 1 = 5), and other has nothing (0). Table 4-6 provides a list of the different permissions and their corresponding numbers.

Table 4-6 Numeric representations of the permissions in a mode

Mode (One Section Only)	Corresponding Number
rwx	4 + 2 + 1 = **7**
rw-	4 + 2 = **6**
r-x	4 + 1 = **5**
r--	4
-wx	2 + 1 = **3**
-w-	2
--x	1
---	0

To change the mode of the file1 file used earlier to r-xr-----, you can use the command `chmod 540 file1`, as shown in the following example:

```
[root@server1 ~]# chmod 540 file1
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
-r-xr-----    1 bob      proj           282 Apr 29 22:06 file1
-rwxr-xr-x    1 root     root           282 Apr 29 22:06 file2
-rwxrwxrwx    1 root     root           282 Apr 29 22:06 file3
----------    1 root     root           282 Apr 29 22:06 file4
-rw-r--r--    1 root     root           282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys            282 Apr 29 22:06 file6
[root@server1 ~]#_
```

Similarly, to change the mode of all files in the directory that start with the word "file" to 644 (which is common permissions for files), you can use the following command:

```
[root@server1 ~]# chmod 644 file*
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
-rw-r--r--    1 bob      proj           282 Apr 29 22:06 file1
```

```
-rw-r--r--    1 root     root          282 Apr 29 22:06 file2
-rw-r--r--    1 root     root          282 Apr 29 22:06 file3
-rw-r--r--    1 root     root          282 Apr 29 22:06 file4
-rw-r--r--    1 root     root          282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys           282 Apr 29 22:06 file6
[root@server1 ~]#_
```

Like the `chown` and `chgrp` commands, the `chmod` command can be used to change the permission on a directory and all of its contents recursively by using the `–R` option, as shown in the following example when changing the mode of the Desktop directory:

```
[root@server1 ~]# chmod -R 755 Desktop
[root@server1 ~]# ls -l
total 28
drwxr-xr-x    3 root     root         4096 Apr  8 07:12 Desktop
-rw-r--r--    1 bob      proj          282 Apr 29 22:06 file1
-rw-r--r--    1 root     root          282 Apr 29 22:06 file2
-rw-r--r--    1 root     root          282 Apr 29 22:06 file3
-rw-r--r--    1 root     root          282 Apr 29 22:06 file4
-rw-r--r--    1 root     root          282 Apr 29 22:06 file5
-rw-r--r--    1 user1    sys           282 Apr 29 22:06 file6
[root@server1 ~]# ls -l Desktop
total 16
-rwxr-xr-x    1 root     root      163 Mar 29 09:58 Work
-rwxr-xr-x    1 root     root     3578 Mar 29 09:58 Home
-rwxr-xr-x    1 root     root     1791 Mar 29 09:58 Start Here
drwxr-xr-x    2 root     root     4096 Mar 29 09:58 Trash
[root@server1 ~]#_
```

Default Permissions

Recall that permissions provide security for files and directories by allowing only certain users access, and that there are common guidelines for setting permissions on files and directories, so that permissions are not too strict or too permissive. Also important to maintaining security are the permissions that are given to new files and directories after they are created. New files are given rw-rw-rw- by the system when they are created (because execute should not be given unless necessary), and new directories are given rwxrwxrwx by the system when they are created (because execute needs to exist on a directory for other permissions to work). These default permissions are too permissive for most files, as they allow other full access to directories and nearly full access to files. Hence, a special variable on the system called the **umask** (user mask) takes away permissions on new files and directories immediately after they are created. The most common umask that you will find is 022, which specifies that nothing (0) is taken away from the user, write permission (2) is taken away from members of the group, and write permission (2) is taken away from other on new files and directories when they are first created and given permissions by the system.

Note 19

Keep in mind that the umask applies only to newly created files and directories; it is never used to modify the permissions of existing files and directories. You must use the `chmod` command to modify existing permissions.

An example of how a umask of 022 can be used to alter the permissions of a new file or directory after creation is shown in Figure 4-5.

Figure 4-5 Performing a umask 022 calculation

	New Files	New Directories
Permissions assigned by system	rw-rw-rw-	rwxrwxrwx
- umask	0 2 2	0 2 2
= resulting permissions	rw-r--r--	rwxr-xr-x

To verify the umask used, you can use the **`umask` command** and note the final three digits in the output. To ensure that the umask functions as shown in Figure 4-5, create a new file using the `touch` command and a new directory using the `mkdir` command, as shown in the following output:

```
[root@server1 ~]# ls -l
total 28
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
[root@server1 ~]# umask
0022
[root@server1 ~]# mkdir dir1
[root@server1 ~]# touch file1
[root@server1 ~]# ls -l
total 8
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
drwxr-xr-x    2 root     root          4096 May  3 21:39 dir1
-rw-r--r--    1 root     root             0 May  3 21:40 file1
[root@server1 ~]#_
```

Because the umask is a variable stored in memory, it can be changed. To change the current umask, you can specify the new umask as an argument to the `umask` command. Suppose, for example, you want to change the umask to 007; the resulting permissions on new files and directories is calculated in Figure 4-6.

Figure 4-6 Performing a umask 007 calculation

	New Files	New Directories
Permissions assigned by system	rw-rw-rw-	rwxrwxrwx
- umask	0 0 7	0 0 7
= resulting permissions	rw-rw----	rwxrwx---

To change the umask to 007 and view its effect, you can type the following commands on the command line:

```
[root@server1 ~]# ls -l
total 8
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
```

```
drwxr-xr-x    2 root     root          4096 May  3 21:39 dir1
-rw-r--r--    1 root     root             0 May  3 21:40 file1
[root@server1 ~]# umask 007
[root@server1 ~]# umask
0007
[root@server1 ~]# mkdir dir2
[root@server1 ~]# touch file2
[root@server1 ~]# ls -l
total 12
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
drwxr-xr-x    2 root     root          4096 May  3 21:39 dir1
drwxrwx---    2 root     root          4096 May  3 21:41 dir2
-rw-r--r--    1 root     root             0 May  3 21:40 file1
-rw-rw----    1 root     root             0 May  3 21:41 file2
[root@server1 ~]#_
```

Special Permissions

Read, write, and execute are the regular file permissions that you would use to assign security to files; however, you can optionally use three more special permissions on files and directories:

- SUID (Set User ID)
- SGID (Set Group ID)
- Sticky bit

Defining Special Permissions

The SUID has no special function when set on a directory; however, if the SUID is set on a file and that file is executed, the person who executed the file temporarily becomes the owner of the file while it is executing. Many commands on a typical Linux system have this special permission set; the **`passwd` command** (/usr/bin/passwd) that is used to change your password is one such file. Because this file is owned by the root user, when a regular user executes the `passwd` command to change their own password, that user temporarily becomes the root user while the `passwd` command is executing in memory. This ensures that any user can change their own password because a default kernel setting on Linux systems only allows the root user to change passwords. Furthermore, the SUID can only be applied to binary compiled programs. The Linux kernel will not interpret the SUID on an executable text file, such as a shell script, because text files are easy to edit and, thus, pose a security hazard to the system.

Contrary to the SUID, the SGID has a function when applied to both files and directories. Just as the SUID allows regular users to execute a binary compiled program and become the owner of the file for the duration of execution, the SGID allows regular users to execute a binary compiled program and become a member of the group that is attached to the file. Thus, if a file is owned by the group "sys" and also has the SGID permission, any user who executes that file will be a member of the group "sys" during execution. If a command or file requires the user executing it to have the same permissions applied to the sys group, setting the SGID on the file simplifies assigning rights to the file for user execution.

The SGID also has a special function when placed on a directory. When a user creates a file, recall that that user's name and primary group become the owner and group owner of the file, respectively. However, if a user creates a file in a directory that has the SGID permission set, that user's name becomes the owner of the file and the directory's group owner becomes the group owner of the file.

Finally, the sticky bit was used on files in the past to lock them in memory; however, today the sticky bit performs a useful function only on directories. As explained earlier in this chapter, the write permission applied to a directory allows you to add and remove any file to and from that directory. Thus, if you have the write permission to a certain directory but no permission to files within it, you could delete all of those files. Consider a company that requires a common directory that gives all employees the ability to add files; this directory must give everyone the write permission. Unfortunately, the write permission also gives all employees the ability to delete all files and directories within, including the ones that others have added to the directory. If the sticky bit is applied to this common directory in addition to the write permission, employees can add files to the directory but only delete those files that they have added and not others.

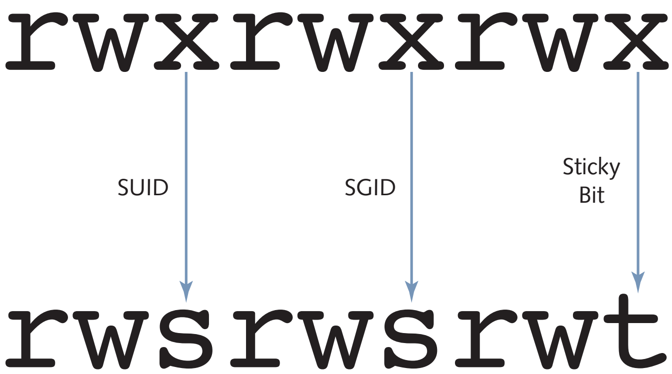
Note 20

Note that all special permissions also require the execute permission to work properly; the SUID and SGID work on executable files, and the SGID and sticky bit work on directories (which must have execute permission for access).

Setting Special Permissions

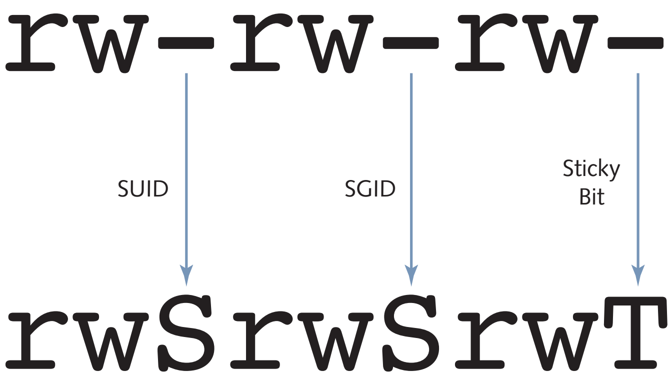
The mode of a file that is displayed using the `ls -l` command does not have a section for special permissions. However, because special permissions require execute, they mask the execute permission when displayed using the `ls -l` command, as shown in Figure 4-7.

Figure 4-7 Representing special permissions in the mode



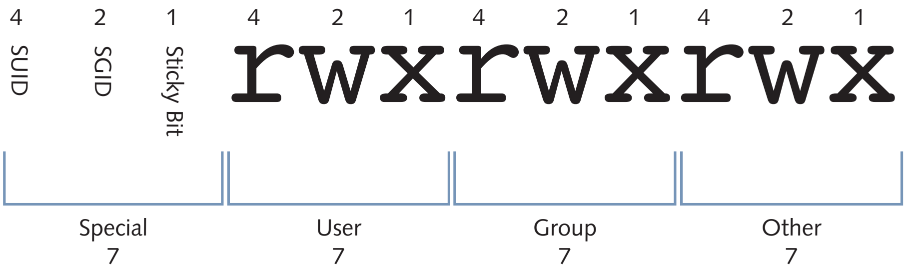
The system allows you to set special permissions even if the file or directory does not have execute permission. However, the special permissions will not perform their function. If the special permissions are set on a file or directory without execute permissions, then the ineffective special permissions are capitalized as shown in Figure 4-8.

Figure 4-8 Representing special permission in the absence of the execute permission



To set the special permissions, you can visualize them to the left of the mode, as shown in Figure 4-9.

Figure 4-9 Numeric representation of regular and special permissions



Thus, to set all of the special permissions on a certain file or directory, you can use the command `chmod 7777 name`, as indicated from Figure 4-9. However, the SUID and SGID bits are typically set on files. To change the permissions on the file1 file used earlier such that other can view and execute the file as the owner and a member of the group, you can use the command `chmod 6755 file1`, as shown in the following example:

```
[root@server1 ~]# ls -l
total 12
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
drwxr-xr-x    2 root     root          4096 May  3 21:39 dir1
drwx------    2 root     root          4096 May  3 21:41 dir2
-rw-r--r--    1 root     root             0 May  3 21:40 file1
-rw-------    1 root     root             0 May  3 21:41 file2
[root@server1 ~]# chmod 6755 file1
[root@server1 ~]# ls -l
total 12
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
drwxr-xr-x    2 root     root          4096 May  3 21:39 dir1
drwx------    2 root     root          4096 May  3 21:41 dir2
-rwsr-sr-x    1 root     root             0 May  3 21:40 file1
-rw-------    1 root     root             0 May  3 21:41 file2
[root@server1 ~]#_
```

Similarly, to set the sticky bit permission on the directory dir1 used earlier, you can use the command `chmod 1777 dir1`, which allows all users (including other) to add files to the dir1 directory. This is because you gave the write permission; however, users can only delete the files that they own in dir1 because you set the sticky bit. This is shown in the following example:

```
[root@server1 ~]# ls -l
total 12
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
drwxr-xr-x    2 root     root          4096 May  3 21:39 dir1
drwx------    2 root     root          4096 May  3 21:41 dir2
-rwsr-sr-x    1 root     root             0 May  3 21:40 file1
-rw-------    1 root     root             0 May  3 21:41 file2
[root@server1 ~]# chmod 1777 dir1
[root@server1 ~]# ls -l
total 12
drwx------    3 root     root          4096 Apr  8 07:12 Desktop
drwxrwxrwt    2 root     root          4096 May  3 21:39 dir1
drwx------    2 root     root          4096 May  3 21:41 dir2
-rwsr-sr-x    1 root     root             0 May  3 21:40 file1
-rw-------    1 root     root             0 May  3 21:41 file2
[root@server1 ~]#_
```

Also, remember that assigning special permissions without execute renders those permissions useless. For example, you may forget to give execute permission to user, group, or other, and the long listing covers the execute permission with a special permission. In that case, the special permission is capitalized, as shown in the following example when dir2 is not given execute underneath the position in the mode that indicates the sticky bit (t):

```
[root@server1 ~]# ls -l
total 12
drwx------    3 root     root         4096 Apr  8 07:12 Desktop
drwxrwxrwt    2 root     root         4096 May  3 21:39 dir1
drwx------    2 root     root         4096 May  3 21:41 dir2
-rwsr-sr-x    1 root     root            0 May  3 21:40 file1
-rw-------    1 root     root            0 May  3 21:41 file2
[root@server1 ~]# chmod 1770 dir2
[root@server1 ~]# ls -l
total 12
drwx------    3 root     root         4096 Apr  8 07:12 Desktop
drwxrwxrwt    2 root     root         4096 May  3 21:39 dir1
drwxrwx--T    2 root     root         4096 May  3 21:41 dir2
-rwsr-sr-x    1 root     root            0 May  3 21:40 file1
-rw-------    1 root     root            0 May  3 21:41 file2
[root@server1 ~]#_
```

Setting Custom Permissions in the Access Control List (ACL)

An **access control list (ACL)** is a list of users or groups that you can assign permissions to. As discussed earlier, the default ACL used in Linux consists of three entities: user, group, and other. However, there may be situations where you need to assign a specific set of permissions on a file or directory to an individual user or group.

Take, for example, the file doc1:

```
[root@server1 ~]# ls -l doc1
-rw-rw----    1 user1     acctg            0 May  2 22:01 doc1
[root@server1 ~]#_
```

The owner of the file (user1) has read and write permission, the group (acctg) has read and write permission, and everyone else has no access to the file.

Now imagine that you need to give read permission to the bob user without giving permissions to anyone else. The solution to this problem is to modify the ACL on the doc1 file and add a special entry for bob only. This can be accomplished by using the following **`setfacl` (set file ACL) command**:

```
[root@server1 ~]# setfacl -m u:bob:r-- doc1
[root@server1 ~]#_
```

The `–m` option in the preceding command modifies the ACL. You can use `g` instead of `u` to add a group to the ACL.

Now, when you perform a long listing of the file doc1, you will see a + symbol next to the mode to indicate that there are additional entries in the ACL for this file. To see these additional entries, use the **`getfacl` (get file ACL) command**:

```
[root@server1 ~]# ls -l doc1
-rw-rw----+   1 user1     acctg            0 May  2 22:01 doc1
 [root@server1 ~]# getfacl doc1
# file: doc1
# owner: user1
```

```
# group: acctg
user::rw-
user:bob:r--
group::rw-
mask::rw-
other::---
[root@server1 ~]#_
```

After running the `getfacl` command, you will notice an extra node in the output: the mask. The mask is compared to all additional user and group permissions in the ACL. If the mask is more restrictive, it takes precedence when it comes to permissions. For example, if the mask is set to `r--` and the user bob has `rw-`, then the user bob actually gets `r--` to the file. When you run the `setfacl` command, the mask is always made equal to the least restrictive permission assigned so that it does not affect additional ACL entries. The mask was created as a mechanism that could easily revoke permissions on a file that had several additional users and groups added to the ACL.

To remove all extra ACL assignments on the doc1 file, use the `–b` option to the `setfacl` command:

```
[root@server1 ~]# setfacl -b doc1
[root@server1 ~]# ls -l doc1
-rw-rw----   1 user1     acctg            0 May  2 22:01 doc1
[root@server1 ~]#_
```

Managing Filesystem Attributes

As with the Windows operating system, Linux has file attributes that can be set, if necessary. These attributes work outside Linux permissions and are filesystem-specific. This section examines attributes for the ext4 filesystem that you configured for your Fedora Linux system during Hands-On Project 2-1. Filesystem types will be discussed in more depth in Chapter 5.

To see the filesystem attributes that are currently assigned to a file, you can use the **`lsattr` (list attributes) command**, as shown here for the doc1 file:

```
[root@server1 ~]# lsattr doc1
--------------e---- doc1
[root@server1 ~]#_
```

By default, all files have the `e` attribute, which writes to the file in "extent" blocks (rather than immediately in a byte-by-byte fashion). If you would like to add or remove attributes, you can use the **`chattr` (change attributes) command**. The following example assigns the immutable attribute (`i`) to the doc1 file and displays the results:

```
[root@server1 ~]# chattr +i doc1
[root@server1 ~]# lsattr doc1
----i---------e---- doc1
[root@server1 ~]#_
```

The immutable attribute is the most commonly used filesystem attribute and prevents the file from being modified in any way. Because attributes are applied at a filesystem level, not even the root user can modify a file that has the immutable attribute set.

Note 21

Most filesystem attributes are rarely set, as they provide for low-level filesystem functionality. To view a full listing of filesystem attributes, visit the manual page for the `chattr` command.

Similarly, to remove an attribute, use the chattr command with the – option, as shown here with the doc1 file:

```
[root@server1 ~]# chattr -i doc1
[root@server1 ~]# lsattr doc1
--------------e---- doc1
[root@server1 ~]#_
```

Summary

- The Linux directory tree obeys the Filesystem Hierarchy Standard, which allows Linux users and developers to locate system files in standard directories.
- Many file management commands are designed to create, change the location of, or remove files and directories. The most common of these include cp, mv, rm, rmdir, and mkdir.
- You can find files on the filesystem using an indexed database (the locate command) or by searching the directories listed in the PATH variable (the which command). However, the most versatile command used to find files is the find command, which searches for files based on a wide range of criteria.
- Files can be linked two ways. In a symbolic link, one file serves as a pointer to another file. In a hard link, one file is a linked duplicate of another file.
- Each file and directory has an owner and a group owner. In the absence of system restrictions, the owner of the file or directory can change permissions and give ownership to others.
- File and directory permissions can be set for the owner (user), group owner members (group), as well as everyone else on the system (other).
- There are three regular file and directory permissions (read, write, execute) and three special file and directory permissions (SUID, SGID, sticky bit). The definitions of these permissions are different for files and directories.
- Permissions can be changed using the chmod command by specifying symbols or numbers.
- To ensure security, new files and directories receive default permissions from the system, less the value of the umask variable.
- The root user has all permissions to all files and directories on the Linux filesystem. Similarly, the root user can change the ownership of any file or directory on the Linux filesystem.
- The default ACL (user, group, other) on a file or directory can be modified to include additional users or groups.
- Filesystem attributes can be set on Linux files to provide low-level functionality such as immutability.

Key Terms

access control list (ACL)
chattr (change attributes) command
chgrp (change group) command
chmod (change mode) command
chown (change owner) command
cp (copy) command
data blocks
Filesystem Hierarchy Standard (FHS)
find command
getfacl (get file ACL) command
group
hard link
inode
inode table
interactive mode
ln (link) command
locate command
lsattr (list attributes) command
mkdir (make directory) command
mode
mv (move) command
other
owner
passwd command
PATH variable
permission
primary group
pruning
recursive
rm (remove) command