Chapter 3

Managing Files, Directories, and Text

✓ **Objective 1.2: Given a scenario, manage files and directories**

In the original Linux years, to get anything done you had to work with the *Gnu/Linux shell*. The shell is a special interactive utility that allows users to run programs, manage files, handle processes, and so on. The shell provides a command-line interface, which furnishes a prompt at which you can enter text-based commands. These commands are actually programs. There are literally thousands of commands you can enter at the command line. However, you need to use only a few hundred commands on a regular basis in your daily job.

While it is highly likely that you have had multiple exposures to many of the commands in this chapter, you may not know all of them. In addition, there may be some shell commands you are using in an ineffective manner. Our purpose in this chapter is to improve your Linux command-line tool belt. We'll cover the basics of managing files and directories, reviewing text files, and finding information. The simple and oft-used `ls` command is covered as well as the interesting `diff` utility. Commands and concepts in this chapter will be built upon and used in later chapters.

Handling Files and Directories

Files on a Linux system are stored within a single directory structure, called a *virtual directory*. The virtual directory contains files from all the computer's storage devices and merges them into a single directory structure. This structure has a single base directory called the *root directory*, which is often simply called *root*.

Often one of the first skills learned at the command line is how to navigate the virtual directory structure as well as how to create directories and remove them. Viewing files, creating them, copying and moving them, and deleting them are also important skills. The following sections describe how to use commands at the command line to accomplish these various tasks.

Viewing and Creating Files

The most basic command for viewing a file's name and its various *metadata* is the list (`ls`) command. Metadata is information that describes and provides additional details about data.

To issue the list command, you type **ls** and any needed options or arguments. The basic syntax structure for the list command is:

```
ls [OPTION]... [FILE]...
```

In the list command's syntax structure, [*OPTION*] means there are various options (also called *switches*) you can add to display different file metadata. The brackets indicate that switches are optional. The [*FILE*] argument shows that you can add a directory or filename to the command's end to look at metadata for either specific files or files within other virtual directory structure locations. It too is optional, as denoted by the brackets.

Syntax structure is depicted for many command-line commands within the Linux system's manual pages, also called the *man pages*. To find a particular command's syntax structure, view its man page (e.g., `man ls`) and look in the Synopsis section.

When you issue the `ls` command with no additional arguments or options, it displays all the files' and subdirectories' names within the *present working directory*, as shown in Listing 3.1.

Listing 3.1: Using the `ls` and `pwd` commands

```
$ ls
Desktop    Downloads  Pictures       Public     Videos
Documents  Music      Project47.txt  Templates
$
$ pwd
/home/Christine
$
```

Your present working directory is your login process's current location within the virtual directory structure. You can determine this location's directory name by issuing the `pwd` command, which is also shown in Listing 3.1.

To display more than file and directory name metadata, you need to add various options to the list command. Table 3.1 shows a few commonly used options.

Table 3.1 has the best `ls` command options to memorize, because you will use them often. However, it is worthwhile to try all the various `ls` command options and option combinations. Take time to peruse the `ls` command's options in its man pages. You can, for example, try the `-lh` option combination, as shown in Listing 3.2, which makes the file size more human-readable. When you experiment with various command options, not only will you be better prepared for the Linux+ certification exam, you'll also find combinations that work well for your particular needs.

TABLE 3.1 The `ls` command's commonly used options

Short	Long	Description
`-a`	`--all`	Display all file and subdirectory names, including hidden files' names.
`-d`	`--directory`	Show a directory's own metadata instead of its contents.
`-F`	`--classify`	Classify each file's type using an indicator code (`*`,`/`,`=`,`>`,`@`, or `|`).
`-i`	`--inode`	Display all file and subdirectory names along with their associated index number.
`-l`	N/A	Display file and subdirectory metadata, which includes file type, file access permissions, hard link count, file owner, file's group, modification date and time, and filename.
`-R`	N/A	Show a directory's contents, and for any subdirectory within the original directory tree, consecutively show their contents as well (recursively).

Listing 3.2: Exploring the `ls -lh` command

```
$ pwd
/home/Christine/Answers
$
$ ls -l
total 32
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Everything
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Life
-rw-r--r--. 1 Christine Christine 29900 Aug 19 17:37 Project42.txt
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Universe
$
$ ls -lh
total 32K
drwxrwxr-x. 2 Christine Christine   6 Aug 19 17:34 Everything
drwxrwxr-x. 2 Christine Christine   6 Aug 19 17:34 Life
-rw-r--r--. 1 Christine Christine 30K Aug 19 17:37 Project42.txt
drwxrwxr-x. 2 Christine Christine   6 Aug 19 17:34 Universe
$
```

Be aware that some distributions include, by default, an *alias* for the `ls -l` command. It is `ll` (two lowercase *L* characters) and is demonstrated on a CentOS distribution in Listing 3.3. An alias at the Linux command line is simply a short command that represents

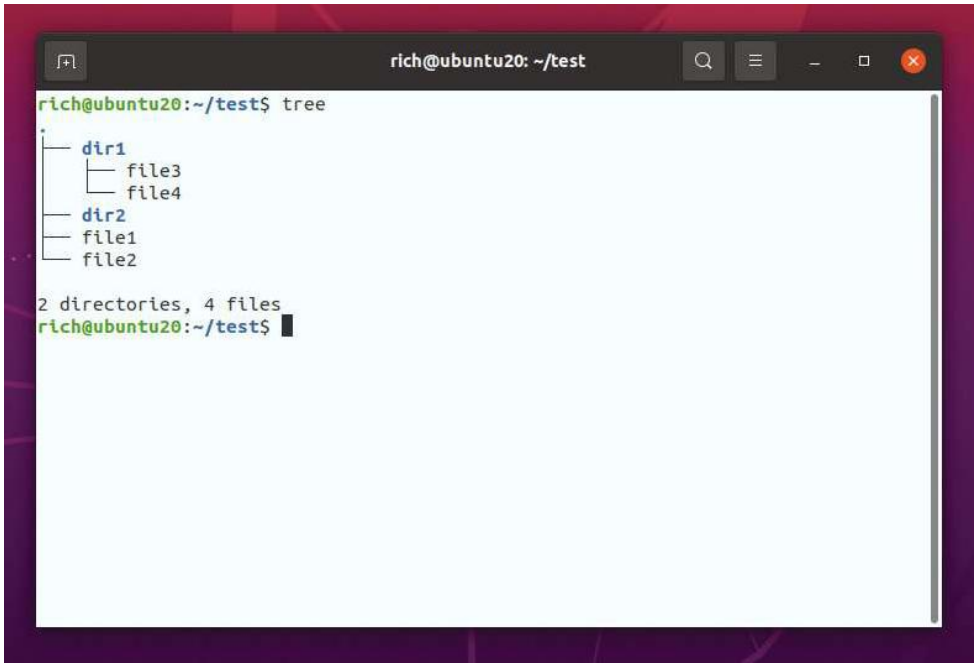
another, typically complicated, command. You can view all the current aliases your process has by typing **alias** at the command line.

Listing 3.3: Exploring the `ll` command

```
$ ls -l
total 32
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Everything
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Life
-rw-r--r--. 1 Christine Christine 29900 Aug 19 17:37 Project42.txt
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Universe
$
$ ll
total 32
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Everything
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Life
-rw-r--r--. 1 Christine Christine 29900 Aug 19 17:37 Project42.txt
drwxrwxr-x. 2 Christine Christine     6 Aug 19 17:34 Universe
$
```

If you're working with lots of files and directories, sometimes it helps to see a graphical overview of things. If you don't have access to a graphical desktop, you can still view things on the command line in a pseudo-graphical format using the `tree` command, as shown in Figure 3.1.

FIGURE 3.1 The `tree` command output



The output from the `tree` command creates a tiered structure, showing which files are associated with which directory, making it easier to sort things out.

The `touch` command will allow you to create empty files on the fly. This command's primary purpose in life is to update a file's timestamps—access and modification. However, for studying purposes, it is useful in that you can quickly create files with which to experiment, as shown in Listing 3.4.

Listing 3.4: Using the `touch` command

```
$ touch Project43.txt
$
$ ls
Everything  Life  Project42.txt  Project43.txt  Universe
$
$ touch Project44.txt Project45.txt Project46.txt
$
$ ls
Everything  Project42.txt  Project44.txt  Project46.txt
Life        Project43.txt  Project45.txt  Universe
$
```

Notice in Listing 3.4 that with the `touch` command you can create a single file or multiple files at a time. To create multiple files, just list the files' names after the command, separated by a space.

Directories are sometimes called *folders*. From a user perspective, a directory contains files, but in reality a directory is a special file used to locate other files. A file for which the directory is responsible has some of its metadata stored within the directory file. This metadata includes the file's name along with the file's associated index (inode) number. Therefore, a file can be located via its managing directory.

You can quickly create directories, but instead of using `touch`, use the `mkdir` command. The `-F` option on the `ls` command will help you in this endeavor. It displays any directories, including newly created ones, with a `/` indicator code following each directory's name. Listing 3.5 provides a few examples.

Listing 3.5: Exploring the `mkdir` command

```
$ ls -F
Everything/  Project42.txt  Project44.txt  Project46.txt
Life/        Project43.txt  Project45.txt  Universe/
$
$ mkdir Galaxy
$
$ ls -F
Everything/  Life/          Project43.txt  Project45.txt  Universe/
```

```
Galaxy/      Project42.txt  Project44.txt  Project46.txt
$
$ pwd
/home/Christine/Answers
$
$ mkdir /home/Christine/Answers/Galaxy/Saturn
$
$ ls -F Galaxy
Saturn/
$
```

To create a subdirectory in your present working directory, you simply enter the `mkdir` command followed by the subdirectory's name, as shown in Listing 3.5. If you want to build a directory in a different location than your present working directory, you can use an absolute directory reference, as was done for creating the `Saturn` directory in Listing 3.5.

If you are creating directories and moving into them from your present working directory, it is easy to become lost in the directory structure. Quickly move back to your previous present working directory using the `cd -` command or back to your home directory using just the `cd` command with no options.

Be aware when building directories that a few problems can occur. Specifically this can happen when attempting to create a directory tree, such as the example shown in Listing 3.6.

Listing 3.6: Avoiding problems with the `mkdir` command

```
$ ls -F
Everything/  Life/          Project43.txt  Project45.txt  Universe/
Galaxy/      Project42.txt  Project44.txt  Project46.txt
$
$ mkdir Projects/42/
mkdir: cannot create directory 'Projects/42/': No such file or directory
$
$ mkdir -p Projects/42/
$
$ ls -F
Everything/  Life/          Project43.txt  Project45.txt  Projects/
Galaxy/      Project42.txt  Project44.txt  Project46.txt  Universe/
$
$ ls -F Projects
42/
$
```

Notice that an error occurs when you attempt to use the `mkdir` command to build the directory `Projects` and its `42` subdirectory. A subdirectory (`42`) cannot be created without its parent directory (`Projects`) preexisting. The `mkdir` command's `-p` option allows you to overwrite this behavior, as shown in Listing 3.6, and successfully create directory trees.

It is tedious to enter the `ls -F` command after each time you issue the `mkdir` command to ensure that the directory was built. Instead, use the `-v` option on the `mkdir` command to receive verification that the directory was successfully constructed.

Copying and Moving Files

Copying, moving, and renaming files and directories are essential skills. There are several nuances between the commands to complete these tasks that are important for you to know.

To copy a file or directory locally, use the `cp` command. To issue this command, you use `cp` along with any needed options or arguments. The basic syntax structure for the command is:

```
cp [OPTION]... SOURCE DEST
```

The command options, as shown in the structure, are not required. However, the source (*`SOURCE`*) and destination (*`DEST`*) are required, as shown in a basic `cp` command example within Listing 3.7.

Listing 3.7: Using the `cp` command

```
$ pwd
/home/Christine/SpaceOpera/Emphasis
$
$ ls
melodrama.txt
$
$ cp melodrama.txt space-warfare.txt
$
$ ls
melodrama.txt  space-warfare.txt
$
$ cp melodrama.txt
cp: missing destination file operand after 'melodrama.txt'
Try 'cp --help' for more information.
$
```

In Listing 3.7, the first time the `cp` command is used, both the source file and its destination are specified. Thus no problems occur. However, the second time the `cp` command is used the destination file's name is missing. This causes the source file to not be copied and generates an error message.

There are several useful cp command options. Many will help protect you from making a grievous mistake, such as accidentally overwriting a file or its permissions. Table 3.2 shows a few commonly used options.

TABLE 3.2 The cp command's commonly used options

Short	Long	Description
-a	--archive	Perform a recursive copy and keep all the files' original attributes, such as permissions, ownership, and timestamps.
-f	--force	Overwrite any preexisting destination files with the same name as *DEST*.
-i	--interactive	Ask before overwriting any preexisting destination files with the same name as *DEST*.
-n	--no-clobber	Do not overwrite any preexisting destination files with the same name as *DEST*.
-R,-r	--recursive	Copy a directory's contents, and for any subdirectory within the original directory tree, consecutively copy its contents as well (recursive).
-u	--update	Only overwrite preexisting destination files with the same name as *DEST* if the source file is newer.
-v	--verbose	Provide detailed command action information as command executes.

To copy a directory, you need to add the -R (or -r) option to the cp command. This option enacts a recursive copy. A recursive copy will not only create a new directory (DEST), but it also copies any files the source directory manages, source directory subdirectories, and their files as well. Listing 3.8 shows an example of how to do a recursive copy as well as how *not* to do one.

Listing 3.8: Performing a recursive copy with the cp command

```
$ pwd
/home/Christine/SpaceOpera
$
$ ls -F
Emphasis/
$
```

```
$ cp Emphasis Story-Line
cp: omitting directory 'Emphasis'
$
$ ls -F
Emphasis/
$
$ cp -R Emphasis Story-Line
$
$ ls -F
Emphasis/  Story-Line/
$
$ ls -R Emphasis
Emphasis:
chivalric-romance.txt       melodrama.txt
interplanetary-battles.txt  space-warfare.txt
$
$ ls -R Story-Line/
Story-Line/:
chivalric-romance.txt       melodrama.txt
interplanetary-battles.txt  space-warfare.txt
$
```

Notice that the first time the `cp` command is used in Listing 3.8, the `-R` option is not used, and thus the source directory is not copied. The error message generated, `cp: omitting directory`, can be a little confusing, but essentially it is telling you that the copy will not take place. When the `cp -R` command is used to copy the source directory in Listing 3.8, it is successful. The recursive copy option is one of the few command options that can be uppercase, `-R`, or lowercase, `-r`.

To move or rename a file or directory locally, you use a single command: `mv`. The command's basic syntax is nearly the same as the `cp` command:

```
mv [OPTION]... SOURCE DEST
```

The commonly used `mv` command options are similar to `cp` command options. However, you'll notice in Table 3.3 that there are fewer typical `mv` command options than common `cp` options. As always, be sure to view the `mv` utility's man pages, using the `man mv` command, to review all the options for certification studying purposes and explore uncommon options, which may be useful to you.

The move command is simple to use. A few examples of renaming a file as well as employing the `-i` option to avoid renaming a file to a preexisting file are shown in Listing 3.9.

TABLE 3.3 The mv command's commonly used options

Short	Long	Description
-f	--force	Overwrite any preexisting destination files with the same name as *DEST*.
-i	--interac-tive	Ask before overwriting any preexisting destination files with the same name as *DEST*.
-n	--no-clobber	Do not overwrite any preexisting destination files with the same name as *DEST*.
-u	--update	Only overwrite preexisting destination files with the same name as *DEST* if the source file is newer.
-v	--verbose	Provide detailed command action information as the command executes.

Listing 3.9: Using the mv command

```
$ ls
chivalric-romance.txt       melodrama.txt
interplanetary-battles.txt  space-warfare.txt
$
$ mv space-warfare.txt risk-taking.txt
$
$ ls
chivalric-romance.txt       melodrama.txt
interplanetary-battles.txt  risk-taking.txt
$
$ mv -i risk-taking.txt melodrama.txt
mv: overwrite 'melodrama.txt'? n
$
```

When renaming an entire directory, there are no additional required command options. Just issue the mv command as you would for renaming a file, as shown in Listing 3.10.

Listing 3.10: Renaming a directory using the mv command

```
$ pwd
/home/Christine/SpaceOpera
$
$ ls -F
Emphasis/  Story-Line/
```

```
$
$ mv -i Story-Line Story-Topics
$
$ ls -F
Emphasis/  Story-Topics/
$
```

You can move a file and rename it all in one simple `mv` command, as shown in Listing 3.11. The *`SOURCE`* uses the file's current directory reference and current name. The *`DEST`* uses the file's new location as well as its new name.

Listing 3.11: Moving and renaming a file using the `mv` command

```
$ pwd
/home/Christine/SpaceOpera
$
$ ls
Emphasis  Story-Topics
$
$ ls Emphasis/
chivalric-romance.txt       melodrama.txt
interplanetary-battles.txt  risk-taking.txt
$
$ ls Story-Topics/
chivalric-romance.txt       melodrama.txt
interplanetary-battles.txt  space-warfare.txt
$
$ mv Emphasis/risk-taking.txt Story-Topics/risks.txt
$
$ ls Emphasis/
chivalric-romance.txt  interplanetary-battles.txt  melodrama.txt
$
$ ls Story-Topics/
chivalric-romance.txt       melodrama.txt  space-warfare.txt
interplanetary-battles.txt  risks.txt
$
```

In Listing 3.11, the file `risk-taking.txt` is located in the `Emphasis` directory. Employing a single `mv` command, it is moved to the `Story-Topics` directory and renamed to `risks.txt` at the same time.

For lightning-fast copies of big files or when you are copying large groups of files, the remote sync utility is rather useful. This tool is often used to create backups, can securely copy files over a network, and is accessed via the `rsync` command.

When you're copying files over a network to a remote host, the file transfer process typically needs protection via encryption methods. The `rsync` command can be tunneled through OpenSSH to provide data privacy. Also, the `scp` command can be employed to provide a secure file copy mechanism. Both of these methods are covered in Chapter 12, "Protecting Files."

To quickly copy a file locally, the `rsync` command syntax is similar to the `mv` command's syntax. It is as follows:

```
rsync [OPTION]... SOURCE DEST
```

Certain `rsync` options will assist you in making quick file copies. Certain switches are helpful for copying large files or creating backups locally, so it's a good idea to review the commonly used `rsync` options listed in Table 3.4.

TABLE 3.4 The `rsync` command's commonly used local copy options

Short	Long	Description
`-a`	`--archive`	Use archive mode.
`-D`	N/A	Retain device and special files.
`-g`	`--group`	Retain file's group.
`-h`	`--human-readable`	Display any numeric output in a human-readable format.
`-l`	`--links`	Copy symbolic links as symbolic links.
`-o`	`--owner`	Retain file's owner.
`-p`	`--perms`	Retain file's permissions.
N/A	`--progress`	Display progression of file copy process.
`-r`	`--recursive`	Copy a directory's contents, and for any subdirectory within the original directory tree, consecutively copy its contents as well (recursive).
N/A	`--stats`	Display detailed file transfer statistics.
`-t`	`--times`	Retain file's modification time.
`-v`	`--verbose`	Provide detailed command action information as command executes.

Archive mode, turned on by the `-a` (or `--archive`) switch, is an interesting feature. Using this one switch is equivalent to using the option combination of `-rlptgoD`, which is a popular `rsync` command option set for creating directory tree backups.

An example of using the `rsync` utility to copy a large file is shown in Listing 3.12. Notice that when the copy is complete, the utility outputs useful information, such as the data transfer rate. If you want additional file transfer statistics, add the `--stats` option to your command.

Listing 3.12: Moving and renaming a file using the `rsync` command

```
# ls -sh /media/USB/Parrot-full-3.7_amd64.iso
3.6G /media/USB/Parrot-full-3.7_amd64.iso
#
# rsync -v /media/USB/Parrot-full-3.7_amd64.iso /home/Christine/
Parrot-full-3.7_amd64.iso

sent 3,769,141,763 bytes  received 35 bytes  3,137,030.21 bytes/sec
total size is 3,768,221,696  speedup is 1.00
#
# ls -sh /home/Christine/Parrot-full-3.7_amd64.iso
3.6G /home/Christine/Parrot-full-3.7_amd64.iso
#
```

The remote sync utility will often display a speedup rating in its output. This rating is related to conducting synchronized backups. If you are using the `rsync` command to conduct periodic backups of a particular directory to another directory location, the speedup rating lets you know how many files did not need to be copied because they had not been modified and were already backed up. For example, if 600 of 600 files had to be copied to the backup directory location, the speedup is 1.00. If only 300 of 600 files had to be copied, the speedup is 2.00. Thus, whenever you are using the `rsync` command to copy a single file to a new location, the speedup will always be 1.00.

Removing Files

Tidying up an entire filesystem or simply your own directory space often starts with deleting unneeded files and directories. Understanding the commands and their switches to do so is paramount to avoid mistakes in removing these items.

The most flexible and heavily used deletion utility is the remove tool. It is employed via the `rm` command, and the basic syntax is:

```
rm [OPTION]... FILE
```

There are many useful options for the `rm` utility, so be sure to view its man pages to see them all. However, the most commonly used options are listed in Table 3.5.

TABLE 3.5 The `rm` command's commonly used options

Short	Long	Description
`-d`	`--dir`	Delete any empty directories.
`-f`	`--force`	Continue on with the deletion process, even if some files designated by the command for removal do not exist, and do not ask prior to deleting any existing files.
`-i`	`--interactive`	Ask before deleting any existing files.
`-I`	N/A	Ask before deleting more than three files, or when using the `-r` option.
`-R,-r`	`--recursive`	Delete a directory's contents, and for any subdirectory within the original directory tree, consecutively delete its contents and the subdirectory as well (recursive).
`-v`	`--verbose`	Provide detailed command action information as command executes.

To simply delete a single file, you can use the `rm` command designating the filename to remove and not use any switches. However, it is always a good idea to use the `-i` (or `--interactive`) option to ensure that you are not deleting the wrong file, as demonstrated in Listing 3.13.

Listing 3.13: Deleting a file using the `rm` command

```
$ ls Parrot-full-3.7_amd64.iso
Parrot-full-3.7_amd64.iso
$
$ rm -i Parrot-full-3.7_amd64.iso
rm: remove write-protected regular file 'Parrot-full-3.7_amd64.iso'? y
$
$ ls Parrot-full-3.7_amd64.iso
ls: cannot access Parrot-full-3.7_amd64.iso: No such file or directory
$
$ rm -i Parrot-full-3.7_amd64.iso
rm: cannot remove 'Parrot-full-3.7_amd64.iso': No such file or directory
$
$ rm -f Parrot-full-3.7_amd64.iso
$
```

Notice also in Listing 3.13 that when the file has been deleted, if you reissue the `rm -i` command, an error message is generated, but if you issue the `rm -f` command, it is silent concerning the missing file. The `-f` (or `--force`) switch is useful when you are deleting many files and desire for no error messages to be displayed.

Removing a directory tree or a directory full of files can be tricky. If you just issue the `rm -i` command, you will get an error message, as shown in Listing 3.14. Instead, you need to add the `-R` or `-r` option in order for the directory and the files it is managing to be deleted.

Listing 3.14: Deleting a directory containing files using the `rm` command

```
$ cd SpaceOpera/
$
$ ls -F
Emphasis/  Story-Topics/
$
$ rm -i Emphasis/
rm: cannot remove 'Emphasis/': Is a directory
$
$ rm -ir Emphasis
rm: descend into directory 'Emphasis'? y
rm: remove regular empty file 'Emphasis/melodrama.txt'? y
rm: remove regular empty file 'Emphasis/interplanetary-battles.txt'? y
rm: remove regular empty file 'Emphasis/chivalric-romance.txt'? y
rm: remove directory 'Emphasis'? y
$
$ ls -F
Story-Topics/
$
```

If you have lots of files to delete, want to ensure that you are deleting the correct files, and don't want to have to answer **y** for every file to delete, employ the `-I` option instead of the `-i` switch. It will ask before deleting more than three files as well as when you are deleting a directory full of files and are using one of the recursive switches, as shown in Listing 3.15.

Listing 3.15: Employing the `rm` command's `-I` option

```
$ ls -F
Story-Topics/
$
$ rm -Ir Story-Topics/
rm: remove 1 argument recursively? y
$
$ ls -F
$
```

Deleting an empty directory, a directory containing no files, is simple. Use the remove empty directories tool by issuing the `rmdir` command. You'll find that adding the `-v` (or `--verbose`) switch is helpful as well, as shown in Listing 3.16.

Listing 3.16: Using the `rmdir` command

```
$ mkdir -v EmptyDir
mkdir: created directory 'EmptyDir'
$
$ rmdir -v EmptyDir/
rmdir: removing directory, 'EmptyDir/'
$
```

If you want to remove a directory tree, which is free of files but contains empty subdirectories, you can also employ the `rmdir` utility. The `-p` (or `--parents`) switch is required along with providing the entire directory tree name as an argument. An example is shown in Listing 3.17.

Listing 3.17: Using the `rmdir` command to delete an empty directory tree

```
$ mkdir -vp EmptyDir/EmptySubDir
mkdir: created directory 'EmptyDir'
mkdir: created directory 'EmptyDir/EmptySubDir'
$
$ rmdir -vp EmptyDir/EmptySubDir
rmdir: removing directory, 'EmptyDir/EmptySubDir'
rmdir: removing directory, 'EmptyDir'
$
```

You may have a situation where you need to remove only empty directories from a directory tree. In this case, you will need to use the `rm` command and add the `-d` (or `--dir`) switch, as shown in Listing 3.18.

Listing 3.18: Using the `rm` command to delete empty directories in a tree

```
$ mkdir -v EmptyDir
mkdir: created directory 'EmptyDir'
$
$ mkdir -v NotEmptyDir
mkdir: created directory 'NotEmptyDir'
$
$ touch NotEmptyDir/File42.txt
$
$ rm -id EmptyDir NotEmptyDir
rm: remove directory 'EmptyDir'? y
rm: cannot remove 'NotEmptyDir': Directory not empty
$
```

An important skill is understanding the commands used to create and remove directories along with the various commands to view, create, copy, move, rename, and delete files. Also, having a firm grasp on the commonly used command options is vital knowledge. This expertise is a valuable tool in your Linux command-line tool belt.

Linking Files and Directories

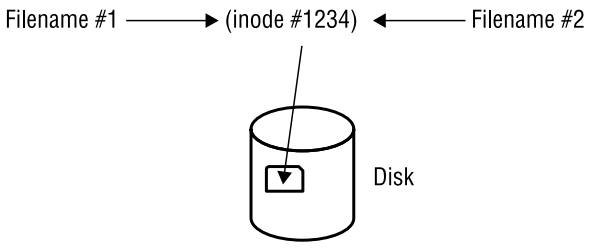
Understanding file and directory links is an essential part of your Linux journey. While many quickly pick up how to link files, they do not necessarily understand the underlying link structure. And that can be a problem. In this section, we'll explore linking files as well as their implications.

There are two types of links. One is a symbolic link, which is also called a *soft link*. The other is a hard link, and we'll take a look at it first.

Establishing a Hard Link

A hard link is a file or directory that has one index (inode) number but at least two different filenames. Having a single inode number means that it is a single data file on the filesystem. Having two or more names means the file can be accessed in multiple ways. Figure 3.2 shows this relationship. In this diagram, a hard link has been created. The hard link has two file names, one inode number, and therefore one filesystem location residing on a disk partition. Thus, the file has two names but is physically one file.

FIGURE 3.2 Hard link file relationship



A hard link allows you to have a pseudo-copy of a file without truly copying its data. This is often used in file backups where not enough filesystem space exists to back up the file's data. If someone deletes one of the file's names, you still have another filename that links to its data.

To create a hard link, use the `ln` command. For hard links, the original file must exist prior to issuing the `ln` command. The linked file must not exist. It is created when the command is issued. Listing 3.19 shows this command in action.

Listing 3.19: Using the `ln` command to create a hard link

```
$ touch OriginalFile.txt
$
$ ls
OriginalFile.txt
$
$ ln OriginalFile.txt HardLinkFile.txt
$
$ ls
HardLinkFile.txt  OriginalFile.txt
$
$ ls -i
2101459 HardLinkFile.txt  2101459 OriginalFile.txt
$
$ touch NewFile.txt
$
$ ls -og
total 0
-rw-rw-r--. 2 0 Aug 24 18:09 HardLinkFile.txt
-rw-rw-r--. 1 0 Aug 24 18:17 NewFile.txt
-rw-rw-r--. 2 0 Aug 24 18:09 OriginalFile.txt
$
```

In Listing 3.19, a new blank and empty file, `OriginalFile.txt`, is created via the `touch` command. It is then hard-linked to `HardLinkFile.txt` via the `ln` command. Notice that `OriginalFile.txt` was created prior to issuing the `ln` command and `HardLinkFile.txt` was created *by* issuing the `ln` command. The inode numbers for these files are checked using the `ls -i` command, and you can see that the numbers are the same for both files.

Also in Listing 3.19, after the hard link is created and the inode numbers are checked, a new empty file is created, called `NewFile.txt`. This was done to compare link counts. Using the `ls -og` command, the file's metadata is displayed, which includes file type, permissions, link counts, file size, creation dates, and filenames. This command is similar to `ls -l` but omits file owners and groups. You can quickly find the link counts in the command output. They are right next to the files' sizes, which are all `0` since the files are empty. Notice that both `OriginalFile.txt` and `HardLinkFile.txt` have a link count of `2`. This is because they are both hard-linked to one other file. `NewFile.txt` has a link count of `1` because it is *not* hard-linked to another file.

If you want to remove a linked file but not the original file, use the `unlink` command. Just type **`unlink`** at the command line and include the linked filename as an argument.

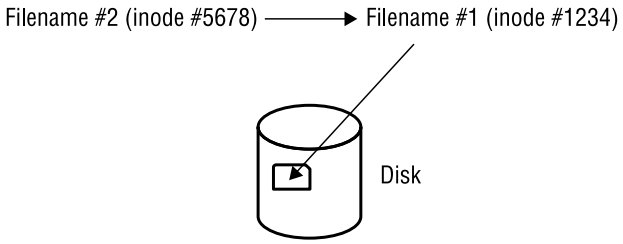
When creating and using hard links, there are a few important items to remember:

- The original file must exist before you issue the `ln` command.
- The second file listed in the `ln` command must *not* exist prior to issuing the command.
- An original file and its hard links share the same inode number.
- An original file and its hard links share the same data.
- An original file and any of its hard links can exist in different directories.
- An original file and its hard links must exist on the same filesystem.

Constructing a Soft Link

Typically, a soft link file provides a pointer to a file that may reside on another filesystem. The two files do not share inode numbers because they do not point to the same data. Figure 3.3 illustrates the soft link relationship.

FIGURE 3.3 Soft-link file relationship



To create a symbolic link, the `ln` command is used with the `-s` (or `--symbolic`) option. An example is shown in Listing 3.20.

Listing 3.20: Using the `ln` command to create a soft link

```
$ touch OriginalSFile.txt
$
$ ls
OriginalSFile.txt
$
$ ln -s OriginalSFile.txt SoftLinkFile.txt
$
$ ls -i
2101456 OriginalSFile.txt  2101468 SoftLinkFile.txt
$
$ ls -og
total 0
-rw-rw-r--. 1  0 Aug 24 19:04 OriginalSFile.txt
lrwxrwxrwx. 1 17 Aug 24 19:04 SoftLinkFile.txt -> OriginalSFile.txt
$
```

Similar to a hard link, the original file must exist prior to issuing the `ln -s` command. The soft-linked file must not exist. It is created when the command is issued. In Listing 3.20, you can see via the `ls -i` command that soft-linked files do not share the same inode number, unlike hard-linked files. Also, soft-linked files do not experience a link count increase. The `ls -og` command shows this, and it also displays the soft-linked file's pointer to the original file.

Sometimes you have a soft-linked file that points to another soft-linked file. If you want to quickly find the final file, use the `readlink -f` command and pass one of the soft-linked filenames as an argument to it. The `readlink` utility will display the final file's name and directory location.

When creating and using soft links, keep a few things in mind:

- The original file must exist before you issue the `ln -s` command.
- The second file listed in the `ln -s` command must not exist prior to issuing the command.
- An original file and its soft links do not share the same inode number.
- An original file and its soft links do not share the same data.
- An original file and any of its soft links can exist in different directories.
- An original file and its soft links can exist in different filesystems.

Stale links can be a serious security problem. A stale link, sometimes called a *dead link*, is when a soft link points to a file that was deleted or moved. The soft-linked file itself is not removed or updated. If a file with the original file's name and location is created, the soft link now points to that new file. If a malicious file is put in the original file's place, your server's security could be compromised. Use symbolic links with caution and employ the `unlink` command if you need to remove a linked file.

File and directory links are easy to create. However, it is important that you understand the underlying structure of these links in order to use them properly.

Reading Files

Linux systems contain many text files. They include configuration files, log files, data files, and so on. Understanding how to view these files is a basic but important skill. In the following sections, we'll explore several utilities you can use to read text files.

Reading Entire Text Files

The basic utility for viewing entire text files is the concatenate (`cat`) command. Though this tool's primary purpose in life is to join together text files and display them, it is often used just to display a single small text file. To view a small text file, use the `cat` command with the basic syntax that follows:

```
cat [OPTION]... [FILE]...
```

The `cat` command is simple to use. You just enter the command followed by any text file you want to read, as shown in Listing 3.21.

Listing 3.21: Using the `cat` command to display a file

```
$ cat numbers.txt
42
2A
52
0010 1010
*
$
```

The `cat` command simply spits out the entire text file to your screen. When you get your prompt back (shown as the `$` in Listing 3.21), you know that the line above the prompt is the file's last line.

There is a handy new clone of the `cat` command called `bat`. Its developer calls it "`cat` with wings" because of the `bat` utility's many additional features. You can read about its features at https://github.com/sharkdp/bat.

One `cat` command option that is useful is the `-n` (or `--number`) switch. Using this option will display line numbers along with the file text, as shown in Listing 3.22.

Listing 3.22: Employing the `cat -n` command to display a file

```
$ cat -n numbers.txt
     1  42
     2  2A
     3  52
     4  0010 1010
     5  *
$
```

Another useful command to view entire text files is the `pr` command. Its original use was to format text files for printing. However, nowadays it is far more useful for displaying a file when you need some special formatting. To view a small text file, use the `pr` command with the basic syntax that follows:

```
pr [OPTION]... [FILE]...
```

The special formatting options are what set this command apart from simply using the `cat` command. Table 3.6 shows some useful `pr` utility options for displaying a file.

TABLE 3.6 The `pr` command's useful file display options

Short	Long	Description
`-`*n*	`--columns=`n	Display the file(s) in column format, using *n* columns.
`-l` *n*	`--length=`*n*	Change the default 66-line page length to *n* lines long.
`-m`	`--merge`	When displaying multiple files, display them in parallel, with one file in each column, and truncate the files' lines.
`-s` *c*	`--separator=`c	Change the default column separator from tab to *c*.
`-t`	`--omit-header`	Do not display any file header or trailers.
`-w` *n*	`--width=`*n*	Change the default 72-character page width to *n* characters wide. The `-s` option overrides this setting.

To display one file, you need to use the page length option, `-l` (or `--length`), to shorten the page length. If you do not use this switch and have a very short file, the text will scroll off the screen. Also the `-t` (or `--omit-header`) option is useful if you only want to see what text is in the file. Listing 3.23 shows the `pr` command in action.

Listing 3.23: Employing the `pr` command to display a file

```
$ pr -tl 15 numbers.txt
42
2A
52
0010 1010
*
$
```

Where the `pr` utility really shines is displaying two short text files at the same time. You can quickly view the files side by side. In this case, it is useful to employ the `-m` (or `--merge`) option, as shown in Listing 3.24.

Listing 3.24: Using the `pr` command to display two files

```
$ pr -mtl 15 numbers.txt random.txt
42                                      42
2A                                      Flat Land
52                                      Schrodinger's Cat
0010 1010                               0010 1010
*                                       0000 0010
$
```

If you want to display two files side by side and you do not care how sloppy the output is, you can use the `paste` command. Just like school paste, it will glue them together but not necessarily be pretty.

When a text file is larger than your output screen, if you use commands such as `cat` and `pr`, text may scroll off the screen. This can be annoying. Fortunately, there are several utilities that allow you to read portions of a text file, which are covered next.

Reading Text File Portions

If you just want to read a single file line or a small portion of a file, it makes no sense to use the `cat` command. This is especially true if you are dealing with a large text file.

The `grep` utility can help you find a file line (or lines) that contain certain text strings. While this utility, covered in more detail later, is primarily used to search for text patterns within a file, it is also very useful in searching for a single string. The basic syntax for the `grep` command is as follows:

```
grep [OPTIONS] PATTERN [FILE...]
```

When searching for a particular text string, you use the string for *`PATTERN`* in the command's syntax and the file you are searching as *`FILE`*. Listing 3.25 shows an example of using the *`grep`* command.

Listing 3.25: Using the `grep` command to find a file line

```
$ grep christine /etc/passwd
$
$ grep -i christine /etc/passwd
Christine:x:1001:1001::/home/Christine:/bin/bash
$
```

Be aware that the `grep` utility pays attention to case. If the string you enter does not match a string exactly (including case) within the file, the `grep` command will return nothing, as happened for the first command in Listing 3.25. If you employ the `-i` (or `--ignore-case`) switch, `grep` will search for any instance of the string disregarding case, as shown in Listing 3.25's second command.

Another handy tool for displaying portions of a text file is the head utility. The head command's syntax is as follows:

```
head [OPTION]... [FILE]...
```

By default, the head command displays the first 10 lines of a text file. An example is shown in Listing 3.26.

Listing 3.26: Employing the head command

```
$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
$
```

A good command option to try allows you to override the default behavior of only displaying a file's first 10 lines. The switch to use is either -n (or --lines=), followed by an argument. The argument determines the number of file lines to display, as shown in Listing 3.27.

Listing 3.27: Using the head command to display fewer lines

```
$ head -n 2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
$
$ head -2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
$
```

Notice in Listing 3.27 that the -n 2 switch and argument used with the head command display only the file's first two lines. However, the second command eliminates the n portion of the switch, and the command behaves just the same as the first command.

You can also eliminate the file's bottom lines by using a negative argument with the -n (or --lines=) switch. This is demonstrated in Listing 3.28.

Listing 3.28: Using the head command to not display bottom lines

```
$ head -n -40 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
$
$ head --40 /etc/passwd
head: unrecognized option '--40'
Try 'head --help' for more information.
$
```

Notice in Listing 3.28 that the `-n` switch's argument is negative this time (`-40`). This tells the `head` command to display all the file's lines except the last 40 lines. If you try to use a negative argument without using the `-n` switch, as you can do with a positive argument, you'll get an error message, as shown in Listing 3.28.

If you want to display the file's last lines instead of its first lines, employ the `tail` utility. Its general syntax is similar to the `head` command's syntax:

```
tail [OPTION]... [FILE]...
```

By default, the `tail` command will show a file's last 10 text lines. However, you can override that behavior by using the `-n` (or `--lines=`) switch with an argument. The argument tells `tail` how many lines from the file's bottom to display. If you add a plus sign (`+`) in front of the argument, the `tail` utility will start displaying the file's text lines starting at the designated line number to the file's end. There are three examples of using `tail` in these ways shown in Listing 3.29.

Listing 3.29: Employing the `tail` command

```
$ tail /etc/passwd
saslauth:x:992:76:Saslauthd user:/run/saslauthd:/sbin/nologin
pulse:x:171:171:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin
gdm:x:42:42::/var/lib/gdm:/sbin/nologin
setroubleshoot:x:991:985::/var/lib/setroubleshoot:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
sssd:x:990:984:User for sssd:/:/sbin/nologin
gnome-initial-setup:x:989:983::/run/gnome-initial-setup/:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
$
$ tail -n 2 /etc/passwd
tcpdump:x:72:72::/:/sbin/nologin
```

```
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
$
$ tail -n +42 /etc/passwd
gnome-initial-setup:x:989:983::/run/gnome-initial-setup/:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
$
```

One of the most useful `tail` utility features is its ability to watch log files. Log files typically have new messages appended to the file's bottom. Watching new messages as they are added is handy. Use the `-f` (or `--follow`) switch on the `tail` command and provide the log filename to watch as the command's argument. You will see a few recent log file entries immediately. As you keep watching, additional messages will display as they are being added to the log file.

Some log files have been replaced on various Linux distributions, and now the messages are kept in a journal file managed by `journald`. To watch messages being added to the journal file, use the `journalctl --follow` command.

To end your monitoring session using `tail`, you must use the Control+C key combination. An example of watching a log file using the `tail` utility is shown snipped in Listing 3.30.

Listing 3.30: Watching a log file with the `tail` command

```
$ sudo tail -f /var/log/auth.log
[sudo] password for Christine:
Aug 27 10:15:14 Ubuntu1804 sshd[15662]: Accepted password [...]
Aug 27 10:15:14 Ubuntu1804 sshd[15662]: pam_unix(sshd:sess[...]
Aug 27 10:15:14 Ubuntu1804 systemd-logind[588]: New sessio[...]
Aug 27 10:15:50 Ubuntu1804 sudo: Christine : TTY=pts/1 ; P[...]
Aug 27 10:15:50 Ubuntu1804 sudo: pam_unix(sudo:session): s[...]
Aug 27 10:16:21 Ubuntu1804 login[10703]: pam_unix(login:se[...]
Aug 27 10:16:21 Ubuntu1804 systemd-logind[588]: Removed se[...]
^C
$
```

If you are following along on your own system with the commands in your book, your Linux distribution may not have the `/var/log/auth.log` file. Try the `/var/log/secure` file instead.

Reading Text File Pages

One way to read through a large file's text is by using a *pager*. A pager utility allows you to view one text page at a time and move through the text at your own pace. The two commonly used pagers are the `more` and `less` utilities.

Though rather simple, the `more` utility is a nice little pager utility. You can move forward through a text file by pressing the spacebar (one page down) or the Enter key (one line down). However, you cannot move backward through a file. An example of using the `more` command is shown in Figure 3.4.

FIGURE 3.4 Using the `more` pager

```
#
# /etc/nsswitch.conf
#
# An example Name Service Switch config file. This file should be
# sorted with the most-used services at the beginning.
#
# The entry '[NOTFOUND=return]' means that the search for an
# entry should stop if the search in the previous entry turned
# up nothing. Note that if the search failed due to some other reason
# (like no NIS server responding) then the search continues with the
# next entry.
#
# Valid entries include:
#
#       nisplus                 Use NIS+ (NIS version 3)
#       nis                     Use NIS (NIS version 2), also called YP
#       dns                     Use DNS (Domain Name Service)
#       files                   Use the local files
#       db                      Use the local database (.db) files
#       compat                  Use NIS on compat mode
#       hesiod                  Use Hesiod for user lookups
#       [NOTFOUND=return]       Stop searching if not found so far
#

# To use db, put the "db" in front of "files" for entries you want to be
# looked up first in the databases
#
# Example:
#passwd:    db files nisplus nis
#shadow:    db files nisplus nis
#group:     db files nisplus nis

passwd:     files sss
shadow:     files sss
--More--(59%)
```

The output displayed in Figure 3.4 was reached by issuing the command `more /etc/nsswitch.conf` at the command line. Notice that the `more` pager utility displays at the screen's bottom how far along you are in the file. At any time you wish to exit from the `more` pager, you must press the q key. This is true even if you have reached the file's last line.

A more flexible pager is the `less` utility. While similar to the `more` utility in that you can move through a file a page (or line) at a time, this pager utility also allows you to move backward. Yet the `less` utility has far more capabilities than just that, which leads to the famous description of this pager, "`less` is more."

Figure 3.5 shows using the `less` utility on the `/etc/nsswitch.conf` text file. Notice that the display does not look that dissimilar from Figure 3.4, but don't let that fool you.

FIGURE 3.5 Using the `less` pager

The `less` page utility allows faster file traversal because it does not read the entire file prior to displaying the file's first page. You can also employ the up and down arrow keys to traverse the file as well as the spacebar to move forward a page and the Esc+V key combination to move backward a page. You can search for a particular word within the file by pressing the ? key, typing in the word you want to find, and pressing Enter to search backward. Replace the ? key with the / key and you can search forward. As with the `more` pager, you do need to use the q key to exit.

By default, the Linux man page utility uses `less` as its pager. Learning the `less` utility's commands will allow you to search through various man pages with ease.

The `less` utility has amazing capabilities. It would be well worth your time to peruse the `less` pager's man pages and play around using its various file search and traversal commands on a large text file.

Finding Information

There are many ways to find various types of information on your Linux system. These methods are important to know so that you can make good administrative decisions and/or solve problems quickly. They will save you time as you perform your administrative tasks, as well as help you pass the certification exam. In the following sections, we'll explore several tools that assist in finding information.

Viewing File Information

It's not uncommon to look through a directory and see files that you're not familiar with, or perhaps even forgot why they're there. Linux has a couple of handy commands that can help you out with that.

The `file` command can provide basic information about the file type of a specified file, as shown in Listing 3.31.

Listing 3.31: Using the `file` command

```
$ file mytest
mytest: Bourne-Again shell script, ASCII text executable
$
```

The output from the `file` command shows that Linux recognizes the `mytest` file as a shell script file, in ASCII text format, and as an executable file.

If you'd like to see information about when a file was created, modified, or last accessed, use the `stat` command, as shown in Listing 3.32.

Listing 3.32: Using the `stat` command

```
$ stat mytest
  File: mytest
  Size: 1016  Blocks: 8 IO Block: 4096 regular file
Device: 805h/2053d   Inode: 1054186     Links: 1
Access: (0764/-rwxrw-r--)
Uid: ( 1000/    rich)
Gid: ( 1000/    rich)
Access: 2021-11-06 09:18:23.856584608 -0500
Modify: 2021-10-31 11:25:22.048406517 -0500
Change: 2021-10-31 11:25:22.048406517 -0500
 Birth: -
$
```

As seen in Listing 3.32, the output from the `stat` command provides basic information about the file, such as the file's name, size, inode number, and the physical device it's stored on. But it also provides some harder-to-find information, such as the last time the file was accessed and modified.

Exploring File Differences

A handy command to explore text file differences is the `diff` command. It allows you to make comparisons between two files, line by line. The basic syntax for the command is:

```
diff [OPTION]... FILES
```

With the `diff` utility you can perform a variety of comparisons. In addition, you can format the output to make the results easier for viewing. Table 3.7 shows a few commonly used options.

TABLE 3.7 The `diff` command's commonly used options

Short	Long	Description
`-e`	`--ed`	Create an ed script, which can be used to make the first file compared the same as the second file compared.
`-q`	`--brief`	If files are different, issue a simple message expressing this.
`-r`	`--recursive`	Compare any subdirectories within the original directory tree, and consecutively compare their contents and the subdirectories as well (recursive).
`-s`	`--report-identical-files`	If files are the same, issue a simple message expressing this.
`-W` *n*	`--width` *n*	Display a width maximum of *n* characters for output.
`-y`	`--side-by-side`	Display output in two columns.

To simply see if differences exist between two text files, you enter the `diff` command with the `-q` switch followed by the filenames. An example is shown in Listing 3.33. To help you see the exact file differences, the `pr` command is employed first to display both files side by side in a column format.

Listing 3.33: Quickly comparing files using the `diff -q` command

```
$ pr -mtw 35 numbers.txt random.txt
42                  42
2A                  Flat Land
52                  Schrodinger's Cat
0010 1010           0010 1010
*                   0000 0010
$
$ diff -q numbers.txt random.txt
Files numbers.txt and random.txt differ
$
```

For just a quick view of differences between files, modify the `diff` command's output to display the files in a column format. Use the `-y` (or `--side-by-side`) option along with

the `-W` (or `--width`) switch for an easier display to read, as shown in Listing 3.34. The pipe symbol (`|`) designates the second file's lines, which are different from those in the first file.

Listing 3.34: Using the `diff` command for quick file differences

```
$ diff -yW 35 numbers.txt random.txt
42                42
2A              | Flat Land
52              | Schrodinger's
0010 1010         0010 1010
*               | 0000 0010
$
```

The `diff` utility provides more than just differences; it also denotes what needs to be appended, changed, or deleted to make the first file identical to the second file. To see the exact differences between the files and any needed modifications, remove the `-q` switch. An example is shown in Listing 3.35.

Listing 3.35: Employing the `diff` command

```
$ diff numbers.txt random.txt
2,3c2,3
< 2A
< 52
---
> Flat Land
> Schrodinger's Cat
5c5
< *
---
> 0000 0010
$
```

The `diff` command's output can be a little confusing. In Listing 3.35, the first output line displays `2,3c2,3`. This output tells you that to make the first file, `numbers.txt`, just like the second file, `random.txt`, you will need to change the `numbers.txt` file's lines 2 through 3 to match the `random.txt` file's lines 2 through 3. The output's next six lines show each file's text content that does not match, separated by a dashed line. Next, the `5c5` designates that line *5* in `numbers.txt` needs to be changed to match line *5* in the `random.txt` file.

The `diff` command is rather powerful. Not only can it tell you the differences between text file lines, but it can also create a script for you to use. The script allows you to modify the first compared text file and turn it into a twin of the second text file. This function is demonstrated in the next chapter.

The letter c in the diff utility's output denotes that changes are needed. You may also see an a for any needed additions or d for any needed deletions.

Using Simple Pinpoint Commands

Commands that quickly locate (pinpoint) files are very useful. They allow you to determine if a particular utility is installed on your system, locate a needed configuration file, find helpful documentation, and so on. The beauty of the commands covered here is that they are simple to use.

The which command shows you the full path name of a shell command passed as an argument. Listing 3.36 shows examples of using this utility.

Listing 3.36: Using the which command

```
$ which diff
/usr/bin/diff
$
$ which shutdown
/usr/sbin/shutdown
$
$ which line
/usr/bin/which: no line in (/usr/local/bin:/usr/bin:/usr/local/sbin:
/usr/sbin:/home/Christine/.local/bin:/home/Christine/bin)
$
$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:
/home/Christine/.local/bin:/home/Christine/bin
$
```

In the first example in Listing 3.36, the which command is used to find the diff command's program location. The command displays the full path name of /usr/bin/diff. The shutdown utility is located in a sbin directory. However, the line program is not installed on this system, and the which utility displays all the directories it searched to find the program. It uses the PATH environment variable, whose contents are also displayed in Listing 3.36, to determine which directories to search.

Environment variables are configuration settings that modify your process's environment. When you type in a command (program) name, the PATH variable sets the directories Linux will search for the program binary. It is also used by other commands, such as the which utility. Note that directory names are separated by a colon (:) in the PATH list.

The which command is also handy for quickly determining if a command is using an alias. Listing 3.37 shows an example of this.

Listing 3.37: Using the `which` command to see a command alias

```
$ which ls
alias ls='ls --color=auto'
        /usr/bin/ls
$
$ unalias ls
$
$ which ls
/usr/bin/ls
$
```

When the `which` utility is used on the `ls` command in Listing 3.37, it shows that currently the `ls` command has an alias. Thus, when you type `ls`, it is as if you have typed in the `ls --color=auto` command. After employing the `unalias` command on `ls`, the `which` utility only shows the `ls` program's location.

Another simple pinpoint command is the `whereis` utility. This utility allows you to not only locate any command's program binaries but also locate source code files as well as any man pages. Examples of using the `whereis` utility are shown in Listing 3.38.

Listing 3.38: Employing the `whereis` command

```
$ whereis diff
diff: /usr/bin/diff /usr/share/man/man1/diff.1.gz
/usr/share/man/man1p/diff.1p.gz
$
$ whereis line
line:
$
```

The first command issued in Listing 3.38 searches for program binaries, source code files, and man pages for the `diff` utility. In this case, the `whereis` command finds a binary file as well as two man page files. However, when `whereis` is used to locate files for the fictitious `line` utility, nothing is found on the system.

A handy and simple utility to use in finding files is the `locate` program. This utility searches a database, `mlocate.db`, which is located in the `/var/lib/mlocate/` directory, to determine if a particular file exists on the local system. The basic syntax for the `locate` command is as follows:

```
locate [OPTION]... PATTERN...
```

Notice in the syntax that the `locate` utility uses a pattern list to find files. Thus, you can employ partial filenames and regular expressions and, with the command options, ignore case. Table 3.8 shows a few commonly used `locate` command options.

TABLE 3.8 The `locate` command's commonly used options

Short	Long	Description
`-A`	`--all`	Display filenames that match all the patterns, instead of displaying files that match only one pattern in the pattern list.
`-b`	`--basename`	Display only file names that match the pattern and do not include any directory names that match the pattern.
`-c`	`--count`	Display only the number of files whose name matches the pattern instead of displaying file names.
`-i`	`--ignore-case`	Ignore case in the pattern for matching filenames.
`-q`	`--quiet`	Do not display any error messages, such as `permission denied`, when processing.
`-r`	`--regexp` *R*	Use the regular expression, *R*, instead of the pattern list to match filenames.
`-w`	`--wholename`	Display filenames that match the pattern and include any directory names that match the pattern. This is default behavior.

To find a file with the `locate` command, just enter **`locate`** followed by the filename. If the file is on your system and you have permission to view it, the `locate` utility will display the file's directory path and name. An example of this is shown in Listing 3.39.

Listing 3.39: Using the `locate` command to find a file

```
$ locate Project42.txt
/home/Christine/Answers/Project42.txt
$
```

Using the `locate` command *PATTERN* can be a little tricky, due to default pattern *file globbing*. File globbing occurs when you use wildcards, such as an asterisk (`*`) or a question mark (`?`), added to a filename argument in a command, and the filename is expanded into multiple names. For example, `passw*d` could be expanded into the filename `password` or `passwrd`.

If you don't enter any wildcards into your pattern, the `locate` command, by default, adds wildcards to the pattern. So if you enter the pattern, **`passwd`**, it is automatically turned into **`*passwd*`**. Thus, if you just want to search for the base name `passwd`, with no file

globbing, you must add quotation marks (single or double) around the pattern and precede the pattern with the \ character. A few examples of this are shown in Listing 3.40.

Listing 3.40: Using the `locate` command with no file globbing

```
$ locate -b passwd
/etc/passwd
/etc/passwd-
/etc/pam.d/passwd
/etc/security/opasswd
/usr/bin/gpasswd
[...]
/usr/share/vim/vim74/syntax/passwd.vim
$
$ locate -b '\passwd'
/etc/passwd
/etc/pam.d/passwd
/usr/bin/passwd
/usr/share/bash-completion/completions/passwd
$
```

The first example in Listing 3.40 shows what would happen if you allow the default file globbing to occur. Many more files are displayed than those named `passwd`. So many files are displayed that the listing had to be snipped to fit. However, in the second example, file globbing is turned off with the use of quotation marks and the \ character. Using this pattern with the `locate` utility provides the desired results of displaying files named `passwd`.

If you do not have permission to view a directory's contents, the `locate` command cannot show files that match your *PATTERN*, which are located in that directory. Thus, you may have some files missing from your display.

Keep in mind that the `locate` command's *PATTERN* is really a pattern list. So, you can add additional patterns. Just be sure to separate them with a space, as shown in Listing 3.41.

Listing 3.41: Using the `locate` command with a pattern list

```
$ locate -b '\passwd' '\group'
/etc/group
/etc/passwd
/etc/iproute2/group
/etc/pam.d/passwd
```

```
/usr/bin/passwd
/usr/share/X11/xkb/symbols/group
/usr/share/bash-completion/completions/passwd
$
```

Another problem you can run into deals with newly created or downloaded files. The `locate` utility is really searching the `mlocate.db` database as opposed to searching the virtual directory structure. This database is typically updated only one time per day via a `cron` job. Therefore, if the file is newly created, `locate` won't find it.

The `mlocate.db` database is updated via the `updatedb` utility. You can run it manually using super user privileges if you need to find a newly created or downloaded file. Be aware that it may take a while to run.

Using Intricate Pinpoint Commands

While using simple commands to locate files is useful, they don't work in situations where you need to find files based on things such as metadata. Thankfully, there are more complex commands that can help.

The `find` command is flexible. It allows you to locate files based on data, such as who owns the file, when the file was last modified, permission set on the file, and so on. The basic command syntax is:

```
find  [PATH...] [OPTION] [EXPRESSION]
```

The *PATH* argument is a starting point directory, because you designate a starting point in a directory tree and `find` will search through that directory and all its subdirectories (recursively) for the file or files you seek. You can use a single period (.) to designate your present working directory as the starting point directory.

There are also options for the `find` command itself that handle such items as following or not following links and debugging. In addition, you can have a file deleted or a command executed if a particular file is located. See the *find* utility's man page for more information on these features.

The *EXPRESSION* command argument and its preceding *OPTION* control what type of metadata filters are applied to the search as well as any settings that may limit the search. Table 3.9 shows the more commonly used *OPTION* and *EXPRESSION* combinations.

The `find` utility has many features. Examples help clarify the use of this command. Listing 3.42 provides a few.

TABLE 3.9 The `find` command's commonly used options and expressions

Option	Expression	Description
`-cmin`	*n*	Display names of files whose status changed *n* minutes ago.
`-empty`	N/A	Display names of files that are empty and are a regular text file or a directory.
`-gid`	*n*	Display names of files whose group id is equal to *n*.
`-group`	*name*	Display names of files whose group is *name*.
`-inum`	*n*	Display names of files whose inode number is equal to *n*.
`-maxdepth`	*n*	When searching for files, traverse down into the starting point directory's tree only *n* levels.
`-mmin`	*n*	Display names of files whose data changed *n* minutes ago.
`-name`	*pattern*	Display names of files whose name matches *pattern*. Many regular expression arguments may be used in the *pattern* and need to be enclosed in quotation marks to avoid unpredictable results. Replace `-name` with `-iname` to ignore case.
`-nogroup`	N/A	Display names of files where no group name exists for the file's group ID.
`-nouser`	N/A	Display names of files where no username exists for the file's user ID.
`-perm`	*mode*	Display names of files whose permissions matches *mode*. Either octal or symbolic modes may be used.
`-size`	*n*	Display names of files whose size matches *n*. Suffixes can be used to make the size more human readable, such as G for gigabytes.
`-user`	*name*	Display names of files whose owner is *name*.

Listing 3.42: Employing the `find` command

```
$ find . -name "*.txt"
./Project47.txt
./Answers/Project42.txt
```

```
./Answers/Everything/numbers.txt
./Answers/Everything/random.txt
./Answers/Project43.txt
./Answers/Project44.txt
./Answers/Project45.txt
./Answers/Project46.txt
./SpaceOpera/OriginalSFile.txt
./SpaceOpera/SoftLinkFile.txt
$
$ find . -maxdepth 2 -name "*.txt"
./Project47.txt
./Answers/Project42.txt
./Answers/Project43.txt
./Answers/Project44.txt
./Answers/Project45.txt
./Answers/Project46.txt
./SpaceOpera/OriginalSFile.txt
./SpaceOpera/SoftLinkFile.txt
$
```

The first example in Listing 3.42 is looking for files in the present working directory's tree with a `txt` file extension. Notice that the `-name` option's *`pattern`* uses quotation marks to avoid unpredictable results. In the second example, a `-maxdepth` option is added so that the `find` utility searches only two directories: the current directory and one subdirectory level down.

The `find` command is handy for auditing your system on a regular basis as well as when you are concerned that your server has been hacked. The `-perm` option is useful for one of these audit types, and an example is shown in Listing 3.43.

Listing 3.43: Using the `find` command to audit a server

```
$ find /usr/bin -perm /4000
/usr/bin/newgrp
/usr/bin/chsh
/usr/bin/arping
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/traceroute6.iputils
/usr/bin/pkexec
/usr/bin/passwd
/usr/bin/sudo
$
```

In Listing 3.43, the `/usr/bin` directory is being audited for the potentially dangerous SUID permission by using the `find` utility and its `-perm` option. The expression used is `/4000`, which will ask the `find` utility to search for SUID settings (octal code 4) and, due to the forward slash (`/`) in front of the number, ignore the other file permissions (octal codes 000). The resulting filenames all legitimately use SUID, and thus, nothing suspicious is going on here.

On older Linux systems, to enact a search as shown in Listing 3.41, you would enter `+4000` to designate the permission. The plus sign (+) is now deprecated for this use and has been replaced by the forward slash (/) symbol for the `find` command.

Earlier in this chapter we briefly covered the `grep` command for the purpose of reading a portion of a text file. You can also use this clever utility to search for files on your system.

Suppose it has been a while since you last modified your `/etc/nsswitch.conf` configuration file. A problem arises that requires you to make a change to the `hosts:` setting within the file and you can't remember its exact name. Instead of digging around using the `ls` command, just employ the `grep` command as shown in Listing 3.44 and quickly find the file's name.

Listing 3.44: Using the `grep` command to find a file

```
$ sudo grep -d skip hosts: /etc/*
/etc/nsswitch.conf:hosts:          files [...]
$
```

In Listing 3.44, the `grep` command is used to search all the files within the `/etc/` directory for the `hosts:` setting. The `-d skip` option is used to skip any directory files in order to eliminate messages concerning them. The `grep` utility displays the configuration filename, followed by a colon (`:`) and the file's line where the setting is located. If you are not sure where in the `/etc/` directory tree the configuration file is placed, you can tack on the `-R` (or `-r`, or `--recursive`) option to recursively search through the specified directory tree. If you don't have permission to search through various files, the `grep` command will issue annoying error messages. You'll learn in the next chapter how to redirect those error messages.

Quickly finding files as well as various types of information on your Linux server can help you be a more effective and efficient system administrator. It is a worthwhile investment to try any of this section's commands or their options that are new to you.

Summary

Being able to effectively and swiftly use the right commands at the shell command line is important for your daily job. It allows you to solve problems, manage files, gather information, peruse text files, and so on.

This chapter's purpose was to improve your Linux command-line tool belt. Not only will this help you in your day-to-day work life, but it will also help you successfully pass the CompTIA Linux+ certification exam.

Exam Essentials

Explain basic commands for handling files and directories. Typical basic file and directory management activities include viewing and creating files, copying and moving files, and deleting files. For viewing and creating files and directories, use the `ls`, `touch`, and `mkdir` commands. When needing to duplicate, rename, or move files, employ one of the `mv`, `cp`, or `rsync` commands. For local large file copies, the `rsync` utility is typically the fastest. You can quickly delete an empty directory using the `rmdir` utility, but for directories full of files, you will need to use the `rm -r` command. Also, if you need to ensure that you are removing the correct files, be sure to use the `-i` option on the `rm` utility.

Describe both structures and commands involved in linking files. Linking files is easy to do with the `ln` command. However, it is important for you to describe the underlying link structure. Hard-linked files share the same inode number, whereas soft-linked files do not. Soft or symbolic links can be broken if the file they link to is removed. It is also useful to understand the `readlink` utility to help you explore files that have multiple links.

Summarize the various utilities that can be employed to read text files. To read entire text files, you can use the `cat`, `bat`, and `pr` utilities. Each utility has its own special features. If you need to read only the first or last lines of a text file, employ either the `head` or `tail` command. For a single text line out of a file, the `grep` utility is useful. For reviewing a file a page at a time, you can use either the `less` or the `more` pager utility.

Describe how to find information on your Linux system. To determine two text files' differences, the `diff` utility is helpful. With this utility, you can also employ redirection and modify the files to make them identical. When you need to quickly find files on your system and want to use simple tools, the `which`, `whereis`, and `locate` commands will serve you well. Keep in mind that the `locate` utility uses a database that is typically updated only one time per day, so you may need to manually update it via the `updatedb` command. When simple file location tools are not enough, there are more complex searching utilities, such as `find` and `grep`. The `grep` command can employ regular expressions to assist in your search.

Review Questions

1. You are looking at a directory that you have not viewed in a long time and need to determine which files are actually directories. Which command is the best one to use?
   - **A.** `mkdir -v`
   - **B.** `ls`
   - **C.** `ls -F`
   - **D.** `ls -i`
   - **E.** `ll`

2. You are using the `ls` command to look at a directory file's metadata but keep seeing metadata for the files within it instead. What command option will rectify this situation?
   - **A.** `-a`
   - **B.** `-d`
   - **C.** `-F`
   - **D.** `-l`
   - **E.** `-R`

3. You have just created an empty directory called `MyDir`. Which command did you most likely use?
   - **A.** `mkdir -v MyDir`
   - **B.** `touch MyDir`
   - **C.** `cp -R TheDir MyDir`
   - **D.** `mv -r TheDir MyDir`
   - **E.** `rmdir MyDir`

4. You have a file that is over 10 GB in size, and it needs to be backed up to a locally attached drive. What is the best utility to use in this situation?
   - **A.** `readlink -f`
   - **B.** `mv`
   - **C.** `cp`
   - **D.** `scp`
   - **E.** `rsync`

5. A long-time server administrator has left the company, and now you are in charge of her system. Her old user account directory tree, `/home/Zoe/`, has been backed up. Which command is the best one to use to quickly remove her files and still indicate that you are removing the correct directory, but without forcing you to confirm every file deletion?
   A. `cp -R /home/Zoe/ /dev/null/`
   B. `mv -R /home/zoe/ /dev/null/`
   C. `rm -Rf /home/Zoe/`
   D. `rm -ri /home/Zoe/`
   E. `rm -rI /home/Zoe`

6. There is a large directory structure that needs to be renamed. What `mv` command options should you consider employing? (Choose all that apply.)
   A. `-f`
   B. `-i`
   C. `-n`
   D. `-r`
   E. `-v`

7. You are trying to decide whether to use a hard link or a symbolic link for a data file. The file is 5 GB, has mission-critical data, and is accessed via the command line by three other people. What should you do?
   A. Create a hard link so that the file can reside on a different filesystem for data protection.
   B. Create three hard links and provide the links to the three other people for data protection.
   C. Create three symbolic links and protect the links from the three other people for data protection.
   D. Create a symbolic link so that the file can reside on a different filesystem.
   E. Create a symbolic link so that the links can share an inode number.

8. A short text-based control file is no longer working properly with the program that reads it. You suspect the file was accidentally corrupted by a control code update you performed recently, even though the file's control codes are all correct. Which command should you use next on the file in your problem investigation?
   A. `cat -v`
   B. `cat -z`
   C. `cat -n`
   D. `cat -s`
   E. `cat -E`

9. You have two short text files that have maximum record lengths of 15 characters. You want to review these files side by side. Which of the following commands would be the best to use?
   - **A.** `pr -m`
   - **B.** `pr -tl 20`
   - **C.** `cat`
   - **D.** `pr -mtl 20`
   - **E.** `pr -ml 20`

10. You have a lengthy file named `FileA.txt`. What will the `head -15 FileA.txt` command do?
   - **A.** Display all but the last 15 lines of the file.
   - **B.** Display all but the first 15 lines of the file.
   - **C.** Display the first 15 lines of the file.
   - **D.** Display the last 15 lines of the file.
   - **E.** Generate an error message.

11. You have issued the command `grep Hal` on a text file you generated using information from a failed login attempts file. It returns nothing, but you just performed a test case by purposely failing to log into the Hal account prior to generating the text file. Which of the following is the best choice as your next step?
   - **A.** Employ the `tail` command to peruse the text file.
   - **B.** Employ the `cat` command to view the text file.
   - **C.** Delete the text file and regenerate it using information from the failed login attempts file.
   - **D.** Issue the `grep -d skip Hal` command on the text file.
   - **E.** Issue the `grep -i Hal` command on the text file.

12. You are trying to peruse a rather large text file. A coworker suggests you use a pager. Which of the following best describes what your coworker is recommending?
   - **A.** Use a utility that allows you to view the first few lines of the file.
   - **B.** Use a utility that allows you to view one text page at time.
   - **C.** Use a utility that allows you to search through the file.
   - **D.** Use a utility that allows you to filter out text in the file.
   - **E.** Use a utility that allows you to view the last few lines of the file.

13. Which of the following does not describe the `less` utility?
   - **A.** It does not read the entire file prior to displaying the file's first page.
   - **B.** You can use the up and down arrow keys to move through the file.
   - **C.** You press the spacebar to move forward a page.
   - **D.** You can use the Esc+V key combination to move backward a page.
   - **E.** You can press the X key to exit from the utility.

14. Which `diff` option is the best option to allow you to quickly determine if two text files are different from one another?
    A. `-e`
    B. `-q`
    C. `-s`
    D. `-W`
    E. `-y`

15. You are working on a Linux server at the command line, and you try to issue a `diff` command and receive a response stating that the command was not found. What is the next best step to take in order to start the troubleshooting process?
    A. Hit your up arrow key and press Enter.
    B. Log out, log back in, and retry the command.
    C. Enter the `which diff` command.
    D. Enter the `whereis diff` command.
    E. Reboot the server and retry the command.

16. You are trying to find a file on your Linux server whose name is `conf`. Employing the `locate conf` command for your search shows many directories that contain the letters `conf`. What is the best description for why this is happening?
    A. The `locate` utility searches for only for directory names.
    B. You did not employ the `-d skip` switch.
    C. It is most likely because the `locate` database is corrupted.
    D. You did not employ the appropriate regular expression.
    E. It is due to file globbing on the pattern name.

17. You downloaded a large important file, `fortytwo.db`, from your company's local website to your Linux server but got interrupted by an emergency. Now you cannot remember where you stored the file. What is the best first step to fixing this problem?
    A. Issue the `sudo updatedb` command.
    B. Issue the `locate -b fortytwo.db` command.
    C. Issue the `locate -b 'fortytwo.db'` command.
    D. Download the file from the company's local website again.
    E. Issue the `locate fortytwo.db` command.

18. You want to search for a particular file, `main.conf`, using the `find` utility. This file most likely is located somewhere in the `/etc/` directory tree. Which of the following commands is the best one to use in this situation?

A. `find -r /etc -name main.conf`

B. `find / -name main.conf`

C. `find /etc -maxdepth -name main.conf`

D. `find /etc -name main.conf`

E. `find main.conf /etc`

19. Yesterday a coworker, Michael, was fired for nefarious behavior. His account and home directory were immediately deleted. You need to audit the server to see if he left any files out in the virtual directory system. Which of the following commands is the best one to use in this situation?

A. `find / -name Michael`

B. `find / -user Michael`

C. `find / -mmin 1440`

D. `find ~ -user Michael`

E. `find / -nouser`

20. You need to figure out what configuration file(s) hold a hostname directive. Which of the following commands is the best one to use?

A. `which`

B. `whereis`

C. `grep`

D. `locate`

E. `find`