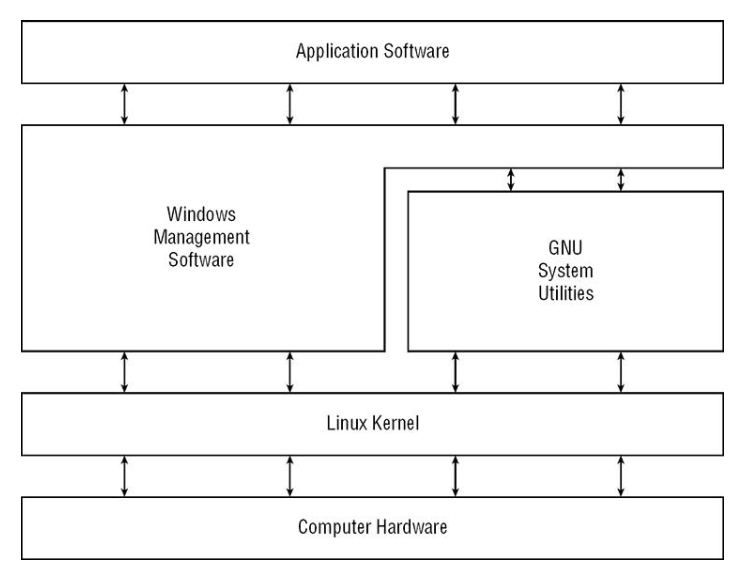
**Part I: The Linux Command Line**

* [Chapter 1](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-1-7.xhtml#ch01): Starting with Linux Shells
* [Chapter 2](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-2-11.xhtml#ch02): Getting to the Shell
* [Chapter 3](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-3-19.xhtml#ch03): Basic Bash Shell Commands
* [Chapter 4](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-4-29.xhtml#ch04): More Bash Shell Commands
* [Chapter 5](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-5-34.xhtml#ch05): Understanding the Shell
* [Chapter 6](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-6-39.xhtml#ch06): Using Linux Environment Variables
* [Chapter 7](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-7-48.xhtml#ch07): Understanding Linux File Permissions
* [Chapter 8](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-8-56.xhtml#ch08): Managing Filesystems
* [Chapter 9](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-9-61.xhtml#ch09): Installing Software
* [Chapter 10](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/chapter-10-68.xhtml#ch10): Working with Editors

For starters, four main parts make up a Linux system:

* The Linux kernel : The core of the Linux system is the *kernel*. The kernel controls all the hardware and software on the computer system, allocating hardware when necessary and executing software when required
* The GNU utilities
* A graphical desktop environment
* Application software

Each of these four parts has a specific job in the Linux system. Each one of the parts by itself isn't very useful. [Figure 1-1](https://cdn2.percipio.com/1735547352.e88499138ad3633b566b49e6285e8b5160708d9f/eod/books/156858/OEBPS/section-8.xhtml#c01-fig-0001) shows a basic diagram of how the parts fit together to create the overall Linux system.



Linus Torvalds. Linus is the person responsible for creating the first Linux kernel software while he was a student at the University of Helsinki.

**Linux Kernel**

The kernel is primarily responsible for four main functions:

* System memory management
* Software program management
* Hardware management
* Filesystem management

These can be a separate topics to explain in detail.

**System Memory Management**

One of the primary functions of the operating system kernel is memory management.

it can also create and manage virtual memory, or memory that does not actually exist.

It does this by using space on the hard disk, called the *swap space*.

**Software Program Management**

The Linux operating system calls a running program a *process*.

The kernel creates the first process, called the *init process*, to start all other processes on the system.

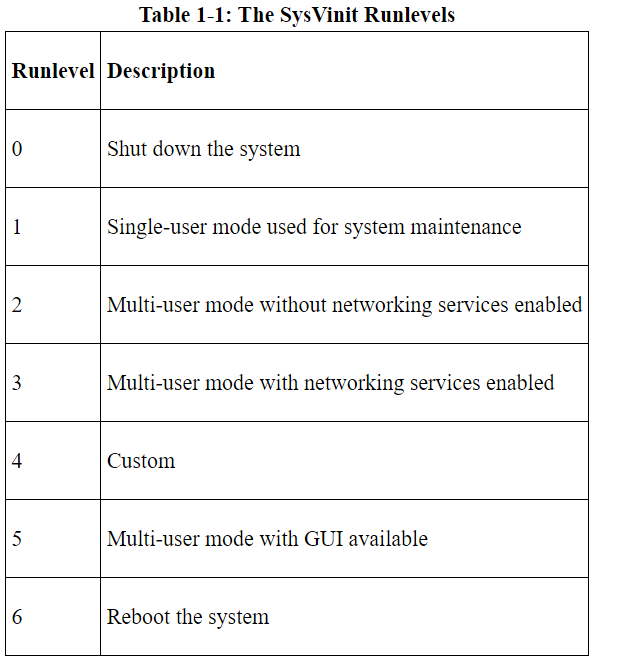
two most popular init process implementations are available in Linux,

* **Systemd:** The *systemd* initialization method, created in 2010, has become the most popular initialization and process management system used by Linux distributions.
* **SysVinit:** The SysVinit (SysV) initialization method, the original method used by Linux, was based on the Unix System V initialization method. Though it is not used by many Linux distributions these days, you still may find it around in older Linux distributions.

The /etc/inittab file defines the default runlevel for a system. The processes that start for specific runlevels are defined in subdirectories of the /etc/rc.d directory. You can view the current runlevel at any time using the runlevel command:

$ runlevel

N 5



The **systemd** initialization method became popular because it has the ability to start processes based on different events:

* When the system boots
* When a particular hardware device is connected
* When a service is started
* When a network connection is established
* When a timer has expired

The **systemctl** program allows you to start, stop, and list the unit files currently running on the system.

$ systemctl get-default

graphical.target

**Hardware Management**

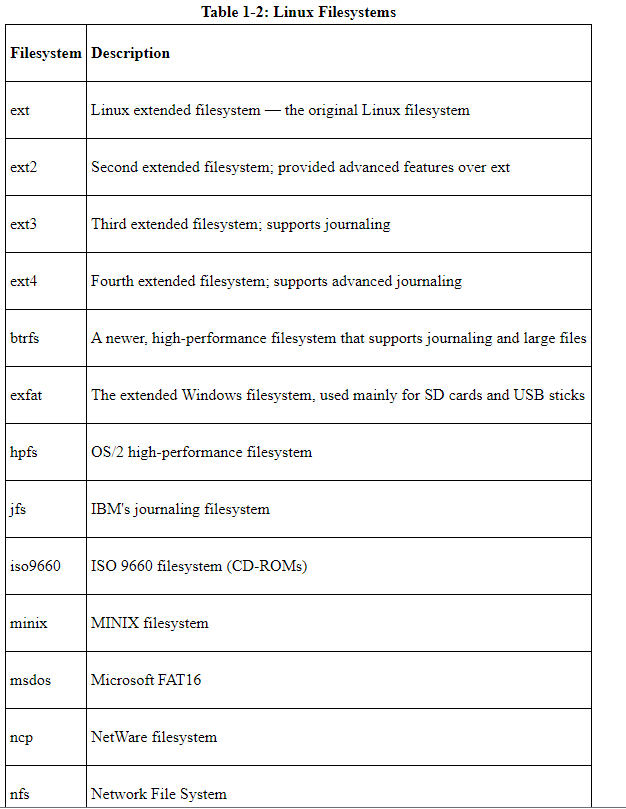
Another responsibility for the kernel is hardware management. Any device that the Linux system must communicate with needs driver code inserted inside the kernel code. The driver code allows the kernel to pass data back and forth to the device, acting as an intermediary between applications and the hardware. Two methods are used for inserting device driver code in the Linux kernel:

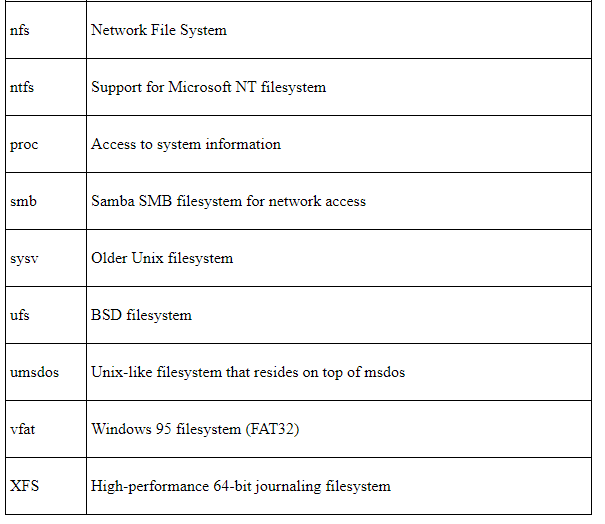
* Drivers compiled in the kernel
* Driver modules added to the kernel

**Filesystem Management**

Linux kernel can support different types of filesystems to read and write data to and from hard drives.

standard filesystems that a Linux system can use to read and write data.





**The GNU Utilities**

Besides having a kernel to control hardware devices, a computer operating system needs utilities to perform standard functions, such as controlling files and programs.

**The Core GNU Utilities**

The GNU coreutils package consists of three parts:

* Utilities for handling files
* Utilities for manipulating text
* Utilities for managing processes

Each of these three main groups of utilities contains several utility programs that are invaluable to the Linux system administrator and programmer.

**The Shell**

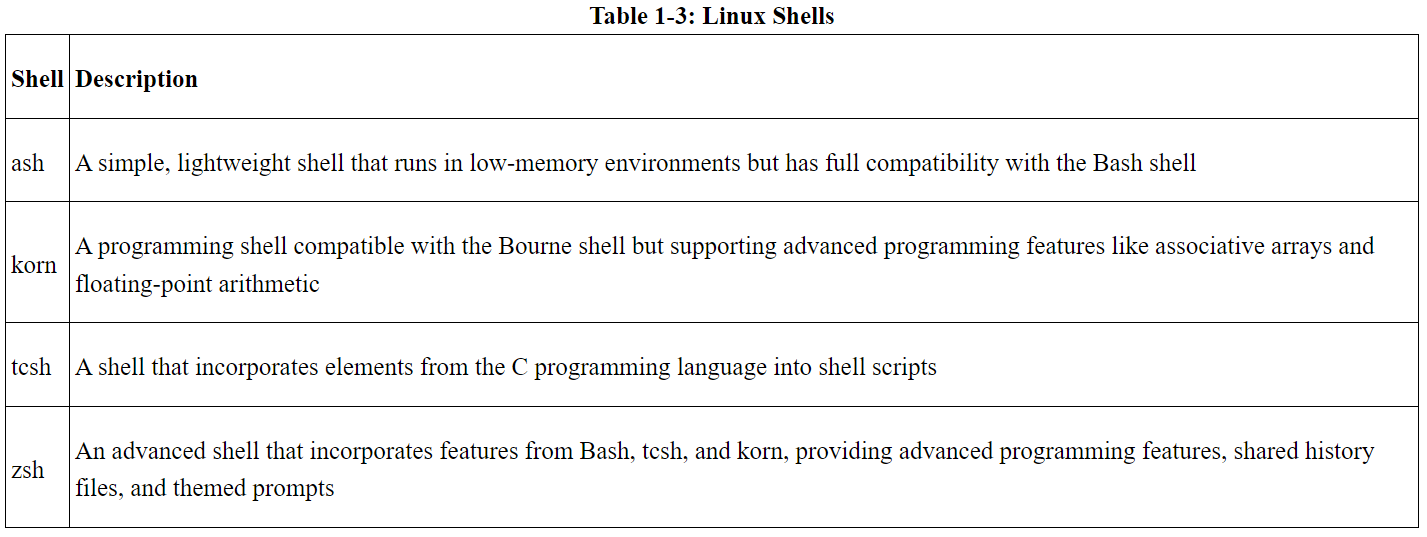
The GNU/Linux shell is a special interactive utility. It provides a way for users to start programs, manage files on the filesystem, and manage processes running on the Linux system. The core of the shell is the command prompt. The command prompt is the interactive part of the shell. It allows you to enter text commands, and then it interprets the commands and executes them in the kernel.

The shell contains a set of internal commands that you use to control tasks such as copying files, moving files, renaming files, displaying the programs currently running on the system, and stopping programs running on the system. Besides the internal commands, the shell allows you to enter the name of a program at the command prompt. The shell passes the program name off to the kernel to start it.

You can also group shell commands into files to execute as a program. Those files are called *shell scripts*. Any command that you can execute from the command line can be placed in a shell script and run as a group of commands. This provides great flexibility in creating utilities for commonly run commands or processes that require several commands grouped together.

The default shell used in all Linux distributions is the Bash shell.

Other popular shells as below,



Most Linux distributions include more than one shell, although usually they pick one of them to be the default. If your Linux distribution includes multiple shells, feel free to experiment with different shells and see which one fits your needs.

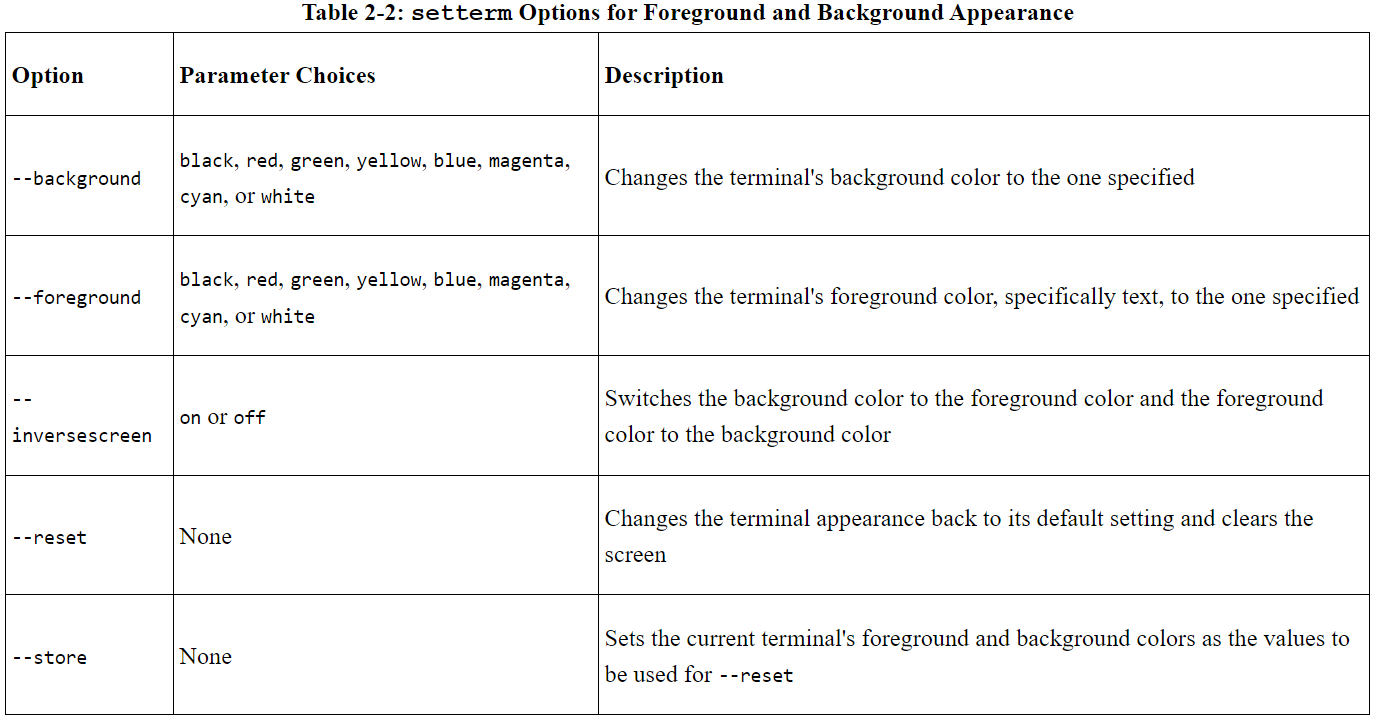
**Accessing CLI via a Linux Console Terminal**

**Console Terminals**

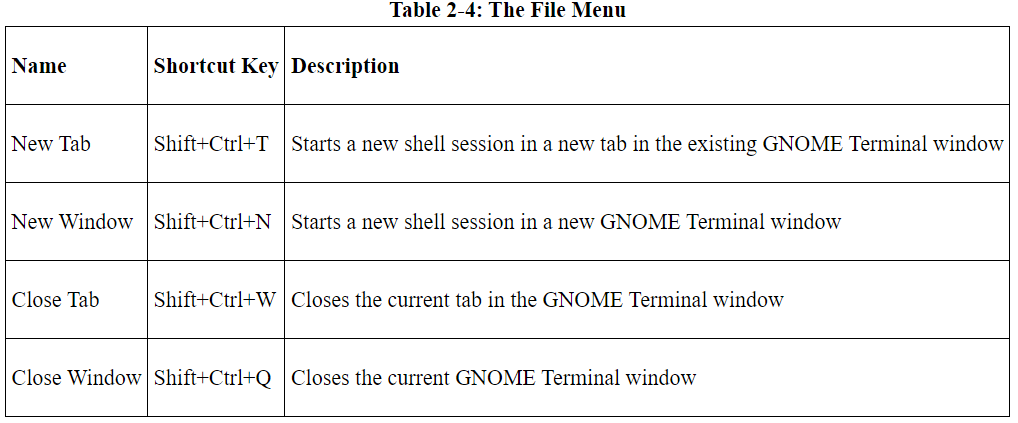
One way to get to a CLI is to access the Linux system via text mode. This provides nothing more than a simple shell CLI on the monitor, just like the days before graphical desktops. This mode is called the *Linux console.*

It may be easier on your eyes to set the background of the terminal to white and the text to black. After you have logged in, you can accomplish this modification in a couple of ways. One way is to type in the command **setterm --inversescreen on** and press the Enter key, as shown in Figure below,

Another way is to type two commands, one after the other. Type **setterm --background white** and press Enter, and then type **setterm --foreground black** and press Enter. Be careful because, when you change your terminal background first, it may be hard to see the commands you are typing.



[Table 2-4](https://cdn2.percipio.com/1735673567.1665d28656b9b7f00e57723f8db4f8a7912dd1e2/eod/books/156858/OEBPS/section-15.xhtml#c02-tbl-0004) shows the configuration options available within the GNOME Terminal File menu system. The File menu item contains items to create and manage your overall CLI terminal sessions.



Each tab session is considered to be an independent CLI session.

**Chapter 3: Basic Bash Shell Commands**

**In This Chapter**

* Interacting with the shell
* Using the Bash manual
* Traversing the filesystem
* Listing files and directories
* Managing files and directories
* Viewing file contents

The default shell used in many Linux distributions is the GNU Bash shell.

**Starting the Shell**

The GNU Bash shell is a program that provides interactive access to the Linux system.

In the /etc/passwd sample entry, the user christine has /bin/bash set as their default shell program. This means when christine logs into the Linux system, the GNU Bash shell program is automatically started.

if you log into the Linux system via a graphical desktop environment, you need to start a graphical terminal emulator to access the shell CLI prompt.

The default prompt symbol for the Bash shell is the dollar sign ($). This symbol indicates that the shell is waiting for you to enter text.

On this Ubuntu Linux system, the shell prompt looks like this:

machindra@Ubuntu2014:~$

On the CentOS Linux system, it looks like this:

[machindra @localhost ~]$

The prompt can provide additional helpful information. In the two preceding examples, the current user ID name, machindra, is shown in the prompt. Also, the name of the system is shown, Ubuntu2014 on the Ubuntu system and localhost on the CentOS machine.

**Bash Manual**

The man command provides access to the manual pages stored on the Linux system. Entering the man command followed by a specific command name provides that utility's manual entry.

The manual page divides information about a command into separate sections. Each section has a conventional naming standard, as shown below,

| **Table 3-1: The Linux Man Page Conventional Section Names** | |
| --- | --- |
| **Section** | **Description** |
| Name | Displays command name and a short description |
| Synopsis | Shows command syntax |
| Configuration | Provides configuration information |
| Description | Describes command generally |
| Options | Describes command option(s) |
| Exit Status | Defines command exit status indicator(s) |
| Return Value | Describes command return value(s) |
| Errors | Provides command error messages |
| Environment | Describes environment variable(s) used |
| Files | Defines files used by command |
| Versions | Describes command version information |
| Conforming To | Provides standards followed |
| Notes | Describes additional helpful command material |
| Bugs | Provides the location to report found bugs |
| Example | Shows command use examples |
| Authors | Provides information on command developers |
| Copyright | Defines command code copyright status |
| See Also | Refers to similar available commands |

Not every command's man page has all the section names described in [Table](https://cdn2.percipio.com/1735673567.1665d28656b9b7f00e57723f8db4f8a7912dd1e2/eod/books/156858/OEBPS/section-22.xhtml#c03-tbl-0001).

Many commands use a basic pattern:

*COMMAND-NAME [OPTION]... [ARGUMENT]...*

In the command's pattern structure,

* *COMMAND-NAME* is the name of the command used to run the desired program.
* [*OPTION*]s are additional items added to modify the command's behavior. There are typically many *OPTION*s (also called *switches*) you can add. The brackets ([]) indicate that *OPTION*s are not required, and the three dots (…) show that you can use more than one *OPTION* at a time.
* [*ARGUMENT*] is typically an item you pass to the command to let the program know you want it to operate on that item. You can see that it too is not required due to the brackets, and you can pass multiple *ARGUMENT*s to the program.

**Tip**

If you want to use more than one command option, often you can squish them together. For example, to use the options -a and -b, you type -ab.

Many commands were written by different individuals, so you'll find the way to use them varies as well. Thus, the command's synopsis section within its man page is a great place to find the proper syntax in order to get things done with the command.

**Tip**

If you can't remember a command's name, you can search the man pages using keywords. The syntax is man -k keyword. For example, to find commands dealing with the terminals, you type man -k terminal.

There are man page section areas. Each section area has an assigned number, starting at 1 and going to 9; they are listed in table below.

| **Table 3-2: The Linux Man Page Section Areas** | |
| --- | --- |
| **Section Number** | **Area Contents** |
| 1 | Executable programs or shell commands |
| 2 | System calls |
| 3 | Library calls |
| 4 | Special files |
| 5 | File formats and conventions |
| 6 | Games |
| 7 | Overviews, conventions, and miscellaneous |
| 8 | Super user and system administration commands |
| 9 | Kernel routines |

To get around the default section search order, type **man *section# topicname***. Thus, to see the hostname overview man pages in section 7, type **man 7 hostname**.

You can also step through an introduction to the various section content areas by typing **man 1 intro** to read about section 1, **man 2 intro** to read about section 2, **man 3 intro** to read about section 3, and so on.

**Navigating the Filesystem**

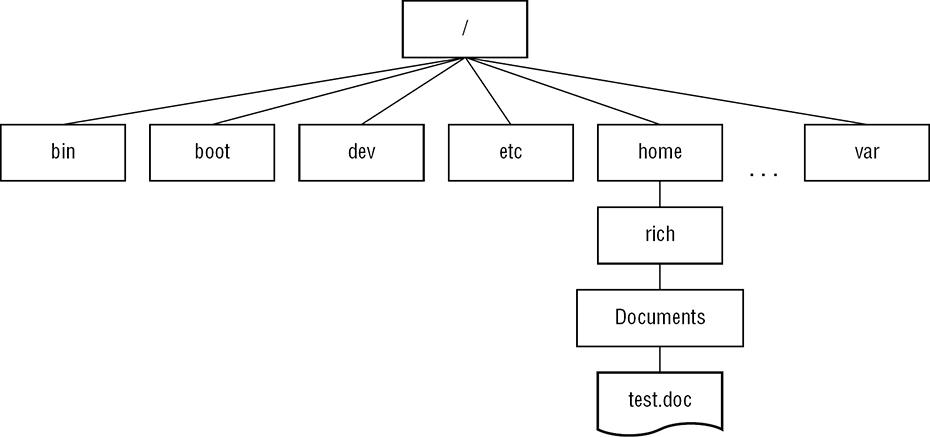
**Looking at the Linux Filesystem**

**Tip**

You'll notice that Linux uses a forward slash (/) instead of a backward slash (\) to denote directories in file paths. The backslash character in Linux denotes an escape character and causes all sorts of problems when you use it in a file path. This may take some getting used to if you're coming from a Windows environment.

/home/rich/Documents/test.doc

Figure 3-2: A Linux virtual directory file path



The first hard drive installed in a Linux system is called the *root drive*. The root drive contains the virtual directory core. Everything else builds from there.

On the root drive, Linux can use special directories as *mount points*. Mount points are directories in the virtual directory where you can assign additional storage devices. Linux causes files and directories to appear within these mount point directories, even though they are physically stored on a different drive.

Figure 3-3: The Linux file structure

