



Linux

Succinctly

by Jason Cannon

Linux Succinctly

By

Jason Cannon

Foreword by Daniel Jebaraj



Copyright © 2014 by Syncfusion Inc.

2501 Aerial Center Parkway

Suite 200

Morrisville, NC 27560

USA

All rights reserved.

I mportant licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: Rui Machado

Copy Editor: Benjamin S. Ball

Acquisitions Coordinator: Hillary Bowling, marketing coordinator, Syncfusion, Inc.

Proofreader: Graham High, content producer, Syncfusion, Inc.

Table of Contents

The Story behind the <i>Succinctly</i> Series of Books	7
About the Author	9
Chapter 1 Introduction	10
What is Linux?	10
Linux Distributions	10
Chapter 2 Linux Directory Structure	11
Common Top-Level Directories	11
/ The Root Directory	11
/bin Binaries	11
/etc System Configuration Files	11
/home Home Directories	12
/opt Optional or Third-Party Software	12
/tmp Temporary Space	12
/usr User-Related Data, Read-Only	12
/var Variable Data	12
Comprehensive Listing of Top-Level Directories	12
Application Directory Structures	14
Organizational Directory Structures	15
Chapter 3 Command Line Interface	16
Basic Commands	17
Command Line Help	19
Chapter 4 Directories	22
Creating and Removing Directories	23

Chapter 5 Viewing File and Directory Details.....	24
Escaping Spaces and Special Characters.....	30
Chapter 6 Permissions	32
Decoding Permissions	34
Changing Permissions	35
Numeric Based Permissions.....	37
Commonly Used Permissions.....	39
Working with Groups.....	40
Directory Permissions	40
Default Permissions and the File Creation Mask.....	41
Special Modes	43
umask Examples.....	45
Chapter 7 Viewing and Editing Files	47
Editing Files	49
The Vim Editor	49
Command Mode	49
Insert Mode	50
Line Mode	50
Repeating Commands	51
Additional Commands.....	51
Emacs	53
Graphical Editors	54
Chapter 8 Deleting, Moving, and Renaming Files and Directories	56
Chapter 9 Finding, Sorting, and Comparing Files and Directories.....	59
Sorting.....	61
Comparing	62

Chapter 10 I/O Redirection	64
Chapter 11 Additional Command Line Concepts	69
Aliases.....	71
Personal Initialization Files	72
Shell History	73
Tab Completion.....	74
Line Continuation	75
Chapter 12 Processes and Jobs.....	76
Jobs.....	79
Chapter 13 Switching Users.....	84
Sudo Super User Do.....	85
Using Sudo	85
Chapter 14 Installing Software	87
RPM-Based Distributions.....	87
Using the rpm Command.....	90
DEB-Based Distributions	91
Using the dpkg Command	93

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!



About the Author

Jason Cannon started his career as a Unix and Linux System Engineer in 1999. Since that time, he has utilized his Linux skills at companies such as Xerox, UPS, Hewlett-Packard, and Amazon. Additionally, he has acted as a technical consultant and independent contractor for small businesses as well as Fortune 500 companies.

Jason has professional experience with CentOS, RedHat Enterprise Linux, SUSE Linux Enterprise Server, and Ubuntu. He has used several Linux distributions on personal projects including Debian, Slackware, CrunchBang, and others. In addition to Linux, Jason has experience supporting proprietary Unix operating systems including AIX, HP-UX, and Solaris.

He enjoys teaching others how to use and exploit the power of the Linux operating system. Jason is the author of [Command Line Kung Fu: Bash Scripting Tricks, Linux Shell Programming Tips, and Bash One-liners](#) as well as [Linux for Beginners: An Introduction to the Linux Operating System and Command Line](#). He is also the founder of <http://LinuxTrainingAcademy.com>, where he blogs and teaches online video training courses.

Chapter 1 Introduction

What is Linux?

Linux - unix like OS - at its core - Linux kernel - intermediary between apps and hardware.

Linux is a Unix-like open source operating system. At the core of the operating system is the Linux kernel. It acts as the intermediary between the applications which run in the operating system and the underlying hardware.

Linux Kernal = <https://github.com/torvalds/linux>

Linux Distributions => Linux kernel(heart of every distro) + a collection of software, makes the OS

A Linux distribution is the Linux kernel and a collection of software that, together, creates an operating system. Even though the Linux kernel is at the heart of every distribution, the software that is installed by default can vary greatly as each distribution its own goals and areas of focus. However, what you will learn in this book is applicable to any distribution as the concepts are fundamental to the Linux operating system as a whole.

This book is
distribution
independent

Some distros (distributions) are maintained by a community of volunteers, while others are backed by companies that charge fees for subscriptions and support. Some distros are designed to run on laptops and desktops, while others are designed to run on servers. The following are just a few of the most popular Linux distributions available today:

Distros can be
Classification based on
-> community or
company run
-> Desktops or Servers

- Linux Mint
- Ubuntu
- Debian
- Fedora
- openSUSE
- Arch Linux
- CentOS
- Red Hat Enterprise Linux

Linux Mint - Community-Run: Aimed at ease of use, designed as a user-friendly desktop alternative. - Desktop Focused.
Ubuntu - Company-Run: Developed by Canonical, offering both desktop and server versions.
Debian - Community-Run: A foundational Linux distro, serving as the base for Ubuntu and Mint. - Both Desktop and Server
Fedora - Company-Run (Sponsored): Sponsored by Red Hat and acts as a testing ground for RHEL features. Both Desktop and Server
openSUSE - Hybrid (Community/Company): Supported by SUSE; includes a strong community. Both Desktop and Server
Arch Linux - Community-Run: Known for simplicity, customization, and being bleeding-edge. - Desktop Focused.
CentOS - (Community Hybrid): Previously community-driven, now closely tied to RHEL as CentOS Stream. - Primarily Server.
RHEL (Red Hat Enterprise Linux) - Company-Run: Developed by Red Hat for enterprise-grade reliability and support. - Primarily Server: Designed for enterprise workloads and critical systems.

Chapter 2 Linux Directory Structure

Linux directory structure - like a tree - hierarchy starts from root or trunk - "/" - all the directories branch off from here.

Directory(linux) - Folder(other OSs)

Directory are separated by forward slash - "/"

The Linux directory structure is like a tree. The base of the Linux file system hierarchy begins at the root, or trunk, and directories branch off from there. Each one of these directories, called folders on other operating systems, can and often do contain other directories. The directories on a Linux system are separated by a forward slash.

```
sri@envy:/
$ ls
bin          home         lost+found  root        srv
bin.usr-is-merged  init        media       run         sys
boot         lib          mnt         sbin        tmp
dev          lib.usr-is-merged  opt         sbin.usr-is-merged  usr
etc          lib64        proc        snap        var
```

Common Top-Level Directories

What follows is a list of some of the most important top-level directories. Of course, all of the directories on a Linux system have a purpose, but understanding what these particular directories are for is rather important as a user of a Linux system. These top-level directories will be the ones that you interact with most often.

/ The Root Directory

-> Every file and directory is under / directory

-> It's contents are put up there.

-> / is also called "Trunk", "Slash" - it's analogous to C:\ drive.

-> when new drive is attached to windows D:\ drive is created, but in linux it's added to /mnt or /media/external

Every file and directory on a Linux system resides under the / directory. This directory is referred to as the root directory or sometimes "slash," a shorthand way of saying forward slash.

Even additional physical or virtual storage devices that are attached to a Linux system live somewhere underneath the / directory. The C:\ drive on a Windows system is analogous to / on Linux. When another storage device is attached to a Windows system, it is assigned a new drive letter such as D:\. On a Linux system, storage devices are attached, or mounted, to a directory such as /mnt or /media/external.

/bin Binaries -> Holds essential binaries and executables - basic and fundamental command line utilities reside here.
-> Non essential binaries are in /usr/bin - for eg graphical apps, browsers, mail, trivial cmd utilities

The /bin directory houses essential user binaries and other executable programs. The most basic and fundamental command line utilities reside in /bin. For example, some of the commands in /bin are used to list, copy, move, and view files. Other non-essential binaries are located in /usr/bin. You will find graphical applications such as web browsers and mail readers there, as well as various other command line utilities.

/etc System Configuration Files -> Controls how applications or OS behaves, are in here.
-> eg. a config file in /etc that tells to boot OS in text or Graphical.

Configuration files that control how applications or the operating system behave are located in the /etc directory. For example, there is a configuration file in /etc that tells the operating system whether to boot into a text mode or a graphical mode.

```
sri@envy:/
$ ls /bin | grep ls
dpkg-gensymbols
false
ls
lsattr
lsb_release
lsblk
...
```

/home Home Directories

- > Each user gets a subdirectory for their stuff in /home
- > Houses user-specific configs

Each user on a Linux system has a subdirectory dedicated to his or her account in the **/home** directory. For example, my user account is "jason" and thus my home directory is **/home/jason**. Since all users have their own home directory, they have the option of keeping their data private, sharing it with other users on the system, or a combination of the two.

Typical home directory contents include files created by the user, text documents, vacation pictures, music, etc. Additionally, user-specific configurations are stored in the home directory. These configuration files can control the behavior of the user's graphical or text environment, for example.

/opt Optional or Third-Party Software

- > softwares that don't come with Linux OS.
- > chrome gets installed in here.

Optional or third-party software resides in the **/opt** directory. The **/opt** directory is for software that is not bundled with the operating system. For example, the Google Chrome web browser is not part of the standard Linux operating system and installs in **/opt/google/chrome**.

/tmp Temporary Space

- > Temp space, can be used by apps or users.
- > Cleared at boot time.

Temporary space is available in **/tmp**. This directory can be used by applications or by individual users on the system. The **contents of /tmp are typically cleared at boot time**, so do not store anything in **/tmp** that you can't live without or that you want to store long-term.

/usr User-Related Data, Read-Only

- > Content is used by actual users and not OS.
- > /usr/bin -> all non-essential binaries - eg. git, free, dpkg, apt
- /usr/share/doc -> all documentation of those binaries.

The **/usr** directory is where user-related programs and read-only data reside. **The contents of /usr are meant to be used by actual users of the system as opposed to the operating system itself.** A whole directory hierarchy exists in **/usr**. For example, the **/usr/bin** directory contains binary files and applications, while **/usr/share/doc** contains documentation related to those applications.

/var Variable Data -> for log files and alike

Variable data, the **most notable being log files**, is stored in the **/var** directory. Several log files exist in the **/var/log** directory or a subdirectory thereof.

```
sri@envyr/  
$ ls /var  
backups crash local log opt snap tmp  
cache lib lock mail run spool
```

Comprehensive Listing of Top-Level Directories

In addition to the directories previously covered, there are additional top-level directories you may encounter on a Linux system. **Many of these directories will be of little concern to you in your day-to-day use of the operating system; however, they are an essential part of a functioning Linux system.** This may be used as a quick reference to help you understand the

'time' is a program that measures many of the CPU resources, such as time and memory, that other programs use. The GNU version can format the output in arbitrary ways by using a printf-style format string to include various resource measurements. Some systems do not provide much information about program resource use; 'time' reports unavailable information as zero values.

```

sri@envy:/
$ ls /
bin          home          lost+found  root          srv
bin.usr-is-merged init        media      run          sys
boot        lib          mnt        sbin         tmp
dev         lib.usr-is-merged opt        sbin.usr-is-merged usr
etc         lib64        proc       snap         var

```

general purpose of each of these top-level directories. Some subdirectories are included in this list to help clearly define the purpose of the top-level directory.

/ The starting point of the Linux file system hierarchy, called the root directory.

/bin Binaries and other executable programs.

/boot Files required to boot the operating system.

/cdrom Where CD-ROMs are attached or mounted. [Not there](#)

/cgroup Control groups hierarchy. [Not there](#)

/dev Device files, typically controlled by the operating system and the system administrators.

/etc System configuration files.

/home User home directories.

/init - missing

/lib System libraries.

/lib64 System libraries, 64-bit.

/lost+found Used by the file system to store recovered files after a file system check has been performed.

/media Used to **mount removable media like USB drives.**

/mnt Used to **mount external file systems.**

/opt Optional or third-party software.

/proc **Process information virtual file system.**

/root The **home directory for the root (superuser) account.**

/run - missing

/sbin System administration binaries.

/selinux Virtual file system used to display information about SELinux. [Not there](#)

/srv Contains data which is served by the system.

/srv/www Web server files.

/srv/ftp FTP files.

/sys Virtual file system used to display and sometimes configure the devices and buses known to the Linux kernel.

/tmp Temporary space, typically cleared on reboot.

Additionally, I have,
bin.usr-is-merged
lib.usr-is-merged
sbin.usr-is-merged

/usr User-related programs, libraries, and documentation.

/usr/bin Binaries and other executable programs.

/usr/lib Libraries.

/usr/local Locally installed software that is not part of the base operating system.

/usr/sbin System administration binaries.

/var Variable data, most notably log files.

/var/log Log files.

If you encounter other top-level directories that have not been listed here, those were more than likely created by the system administrator.

Application Directory Structures

-> Applications can follow the OS directory structure pattern

Pattern 1: following OS under 1 directory

Application directory structures can be patterned after the operating system. Here is a sample directory structure of an application named apache installed in **/usr/local**.

/usr/local/apache/bin The application's binaries and other executable programs.

/usr/local/apache/etc Configuration files for the application.

/usr/local/apache/lib Application libraries.

/usr/local/apache/logs Application log files.

Keeping the application is /usr/

If apache were to be installed in **/opt** it would look like this:

/opt/apache/bin The application's binaries and other executable programs.

/opt/apache/etc Configuration files for the application.

/opt/apache/lib Application libraries.

/opt/apache/logs Application log files.

Keeping the application is /opt/

Pattern 2: following OS under multiple directories

Another common application directory structure pattern includes moving the configuration and variable data outside of **/opt**. Instead of placing all of the application components in **/opt/app-name**, **/etc/opt/app-name** is used for configuration files and **/var/opt/app-name** is used for logs. Continuing with the apache application example, here is a demonstration of this method:

/etc/opt/apache Configuration files for the application.
instead of /opt/apache/etc/

etc contains only configs; hence creating /opt/apache conveys, that we have the config for apache present in opt here.

/opt/apache/bin The application's binaries and other executable programs.

/opt/apache/lib Application libraries.

/var/opt/apache Application log files. *instead of /opt/apache/var/*

Pattern 3: sharing a common directory structure with other non OS applications.

Not only can applications be segregated into their own directories, they can share a common directory structure with other applications that are not part of the standard operating system. For example, apache can be directly installed into **/usr/local**. In this case, the binaries would reside in **/usr/local/bin**, while the configuration would reside in **/usr/local/etc**. Since apache may not be the only locally installed application, it could share that space with the other programs.

/usr/local/(etc/lib/bin/logs) - multiple application share these directories.

Organizational Directory Structures

Directory structures can be based on an organization such as a company, group, or team. For example, if you work for the Widget corporation, you may find a directory named **/opt/widget** or **/usr/local/widget** on the company's Linux servers. In some cases, this base directory is treated much like an application directory. It will contain common subdirectories like **/etc** and **/bin**. Here is an example:

Managing directories 1: common directories

/opt/widget The top-level directory for the Widget company.

/opt/widget/bin Binaries and programs installed or created by the Widget company.

/opt/widget/etc Configuration files for the programs installed or created by the Widget company.

Managing directories 2: sub project directories

Further subdivisions can be made within this organizational directory structure. For instance, each application may receive its own subdirectory as follows:

/opt/widget The top-level directory for the Widget company.

/opt/widget/apache The top-level directory for the Widget company's installation of apache.

/opt/widget/apache/bin The apache binaries.

/opt/widget/apache/etc The apache configuration files.

Here are variations on the same theme, but based on a team within the company.

/opt/sysadmin The system administrator team's top-level directory.

/opt/widget/sysadmin The system administrator team's top-level directory.

/usr/local/widget/sysadmin The system administrator team's top-level directory.

Chapter 3 Command Line Interface

Shell - a program - accepts commands(text or graphical instructions from users) and instructs OS(kernel) to execute it.
Eg. connecting to linux system over network(eg. College CentOS server) - shell is started - acts as interface - here in this case, it is called command line interpreter

A shell is a program that accepts commands and instructs the operating system to execute those commands. When you connect to a Linux system over the network, for example, a shell is started and acts as your interface to the system. The shell in this particular case is a command line interpreter. The command-line interface is often referred to by its abbreviation, CLI.

When you connect to a Linux system directly via an attached keyboard and display, you will either be presented with a textual interface or a graphical interface, depending on how that system is configured. In the case of a textual interface, you will have a very similar experience as if you had connected to that system over the network. When you log in, a command line shell is started and you are presented with a prompt.

Some linux system provides a graphical interface; to get to command line, you need TERMINAL EMULATOR APPLICATION

If you connect to a system that is in graphical mode, you will be interacting with a graphical user interface (GUI). In order to access the command line while logged into a GUI, you will need to start a terminal emulator application. Common terminal emulators include xterm, GNOME Terminal, Konsole, rxvt, and Eterm. The one you choose depends on personal preference and availability, but they all provide the same basic functionality—access to the command line.

Some terminal apps include GNOME, Terminal, Konsole, Eterm...

The following demonstrates logging into an Ubuntu Linux server at the command line.

```
Ubuntu 14.04 LTS linuxsvr tty1      Remember college server?
linuxsvr login: jason
Password:
Welcome to Ubuntu 14.04 LTS

username      servername
jason@linuxsvr:~$ -> Command prompt
current directory: home
```

Bash, or Bourne-Again Shell, is a command-line interface (CLI) shell program that allows users to interact with a computer's operating system.

The line `jason@linuxsvr:~$` is the command prompt. The default prompt varies from distribution to distribution and shell to shell. There are a variety of shells with the most common and popular one being Bash. All users can customize their shell prompt to their liking. The information provided in this shell prompt includes the username, the server name, and the current directory.

~ - tilde - represents home directory - `/home/<username>`

The tilde represents the home directory of the current user which is `/home/jason` in this example. You can also specify a username after the tilde, in which case it will expand to the home directory of that user. For example, `~john` expands to `/home/john`. No matter where the user's home directory is, `~username` will be translated to that directory. In the case of an application user such as `www-data`, `~www-data` expands to `/var/www`.

We can also specify `~<username>`; will take us to that particular user folder. No matter where you are, you can go there instantly.

The following are examples of various shell prompts.

```
[jason@linuxsvr /tmp]$
linuxsvr:/home/jason>
jason@linuxsvr:~>
```

Customization and various prompts that are available


```
[12:32:19 linuxsvr ~]$  
%  
>  
$
```

Shell prompts are not limited to a single line. The following example shell prompts span multiple lines.

```
linuxsvr:[/home/jason]  
$  
  
(jason@linuxsvr)-(09:22am--12/15)-]-  
(~)  
  
[Mon 14/12/15 09:22 EST][pts/3][x86_64]  
<jason@linuxsvr:~>  
zsh 26 %  
  
linuxsvr | Mon Dec 15 09:22am  
~/
```

In the remainder of examples in this book, the shell prompt will be abbreviated to just the dollar sign, unless displaying the entire prompt provides additional clarity.

Basic Commands

-> Commands are case-sensitive, they are usually in lowercase
-> Navigation commands - pwd - present working directory; cd - change directory

In Linux, commands are case-sensitive and are typically lowercase. Note that items surrounded by square brackets are optional. Let's start with two commands that will allow you to navigate around the system at the command line. They are **pwd** and **cd**.

pwd The **pwd** command displays the **present working directory** to your screen. This command allows you to keep track of where you are in the file system hierarchy.

if cd is executed without specifying a directory - it takes us to /home/<username>

cd [directory] The **cd** command **changes the current directory to the supplied directory**. If **cd** is executed without specifying a directory, it changes the current directory to your **home directory**. This is how you navigate around the system.

The following is an example of how the **pwd** and **cd** commands can be used. Remember that case matters.

```
$ pwd  
/home/jason  
$ Pwd Case matters!
```

```
Pwd: command not found
```

```
$ cd /home
```

```
$ pwd
```

```
/home
```

```
$ cd /var/log
```

```
$ pwd
```

```
/var/log
```

```
$ cd
```

```
$ pwd
```

```
/home/jason
```

```
$
```

`ls` - to view contents of a directory along with file informations(permission, size, and type...)
`cat` - to view contents of a file

The most common way to see the contents of a directory is to use the `ls` command. To view the contents of a file, use the `cat` command.

ls The `ls` command lists directory contents. Additionally, `ls` can be used to display information about files and directories including permissions, size, and type.

cat [file] The `cat` command concatenates, or displays, files.

```
$ pwd
```

```
/home/jason
```

```
$ ls
```

```
Desktop Documents Downloads Music Pictures to-do.txt
```

```
$ cat to-do.txt
```

```
This file contains my to-do list.
```

```
* Mow the lawn.
```

```
* Take over the world.
```

```
$ cd Music
```

```
$ ls
```

```
JohnColtrane
```

To quit command line session: `exit`, `logout`, `ctrl+d`

To end your command line session, type `exit`, `logout`, or `Ctrl-d`.

exit, **logout**, or **Ctrl-d** Exits the shell or your current session.

```
$ exit
```

```
logout
```

```
Connection to linuxsvr closed.
```

Command Line Help

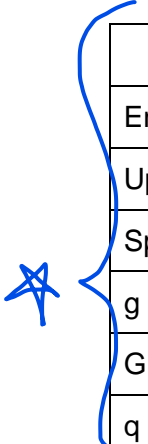
man - manual pages - built-in documentation - provides details of available options for a given command and even find a needed command.

The Linux operating system provides built-in documentation. To access these online manuals, also called man pages, use the `man` command. Man pages are great for looking up the available options for a given command or even finding a command that will help you accomplish the task at hand.

man [command] The `man` command displays the online manual for a given command.

Once you have executed the `man` command, you can navigate through the man page with the arrow keys, as well as the Page Up and Page Down keys. You can also use Enter to move down a line, the Spacebar to move down a page, `g` to move to the beginning, and capital `G` to move to the end of the man page. To quit, type `q`. To learn about even more commands available while viewing man pages, type `h` for help.

Table 1: Navigating Man Pages



Key	Action
Enter, Down Arrow	Move down one line.
Up Arrow	Move up one line.
Spacebar, Page Down	Move down one page.
g	Go to the start or top.
G	Go to the end or bottom.
q	Quit.

`h - help`

```
$ man
What manual page do you want?
$ man ls
LS(1)                User Commands                LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILES (the current directory by default).
    Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

...
```

If you want to do a global search across all the man pages the use k option
`man -k <keyword>`
will result the findings of this keyword across man pages.

```
Manual page ls(1) line 1 (press h for help or q to quit)
$
```

To search the man pages, supply a keyword to the `-k` option of the `man` command. If you are looking for a command that will reboot the system, you could search for "reboot." Once you have a list of man pages that contain that keyword, you can read the documentation for the most promising ones.

```
$ man -k reboot
grub-reboot (8) - set the default boot entry for GRUB for the next boot
only
halt (8)        - reboot or stop the system
poweroff (8)    - reboot or stop the system
reboot (2)      - reboot or enable/disable Ctrl-Alt-Del
reboot (8)      - reboot or stop the system
$ man reboot
NAME
    reboot, halt, poweroff - reboot or stop the system
...
$
```

sri@envy:/mnt/c/Users/sri
\$ man -k clear
proc_pid_clear_refs (5) - reset the PG_Referenced and ACCESSED/YOUNG bits
clear (1) - clear the terminal screen
clear_console (1) - clear the console
clearenv (3) - clear the environment
...

`-h` or `--help` argument, is available for many commands - provides options available and usage example

Some commands will print a help message when `-h` or `--help` is supplied as an argument. Even the `man` command follows this convention.

```
$ man -h
Usage: man [OPTION...] [SECTION] PAGE...

-C, --config-file=FILE    use this user configuration file
-d, --debug                emit debugging messages
-D, --default              reset all options to their default values
    --warnings[=WARNINGS] enable warnings from groff
...
$ man --help
Usage: man [OPTION...] [SECTION] PAGE...

-C, --config-file=FILE    use this user configuration file
-d, --debug                emit debugging messages
-D, --default              reset all options to their default values
    --warnings[=WARNINGS] enable warnings from groff
...
$ ls --help
Usage: ls [OPTION]... [FILE]...
```

List information about the FILES (the current directory by default).
Sort entries alphabetically if none of `-cftuvSUX` nor `--sort` is specified.

Mandatory arguments to long options are mandatory for short options too.

`-a, --all` do not ignore entries starting with `.`

...

\$

Exploring
commands:
got to `/bin` or
`/usr/bin` and
start checking
out
commands.

Given what you know about the Linux directory structure and the documentation that comes with the Linux operating system, you can start exploring commands on your own. For example, list the directory contents of `/bin` and `/usr/bin`. Pick out some commands that grab your attention and use the `man` command to find out what each one of them does.

```
$ cd /bin
$ ls
awk diff cal cat cp date du echo grep groups less more
$ man date
NAME
    date - print or set the system date and time
...
$ cd /usr/bin
$ ls
clear crontab cut dos2unix find kill mv pstree pwd sed strings touch ...
$ man clear
```

The output of the preceding `ls` commands was truncated. In reality, you will likely find dozens of commands in `/bin` and hundreds in `/usr/bin`.

clear command - for cleaning the screen;

Before we move on, I want to share one last basic command that you might find useful. It is the `clear` command. If you want to a fresh screen to work with, issue the `clear` command to clear the contents of your screen.

Commands seen so far:

```
> pwd
> cd
> ls
> cat
> exit
> logout
Shortcut - Ctrl-d
> man; man -k <keyword>; navigation: g, G, q, arrows, space, pg up/down
> <command> -h; <command> --help
> clear
```

Chapter 4 Directories

Paths

-> Absolute - starting from forward slash or root. eg. /home/sri/Desktop

-> Relative - just use the directory name; paths are relative to current working directory.

In addition to referencing directories by their full or absolute paths, you can reference directories by their relative paths. An absolute path starts with a forward slash. An example of a full path is /home/jason/Music. A relative path does not start with a forward slash. When using relative paths, the paths are relative to the current working directory. To change into the Music directory from /home/jason, you would simply type `cd Music`.

```
$ cd /home
$ pwd
/home
$ cd jason/Music
$ pwd
/home/jason/Music
$ cd JohnColtrane
$ pwd
/home/jason/Music/JohnColtrane
```

Note! Linux directories end with a trailing forward slash - you would have noticed this when using the TAB autocomplete option.

. (dot) - represents current directory

.. (double dot) - represents parent directory

Linux uses a `.` to represent the current directory and `..` to represent the parent directory. Also, directories end in a trailing forward slash, but this is often assumed. The following commands place you in the same directory.

```
$ pwd
/home/jason
$ cd ..
$ pwd
/home
$ cd /home
$ pwd
/home
$ cd /home/
$ pwd
/home
```

`cd -` => return to previous working directory.

To quickly return to your previous working directory, use the `cd -` command.

```
$ cd /var/log
$ pwd
/var/log
$ cd /etc/init.d
$ pwd
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ cd -
/mnt/c/Users/sri/Desktop
sri@envy:/mnt/c/Users/sri/Desktop
$ cd -
/mnt/c/Users/sri/Desktop/Linux
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ cd -
/mnt/c/Users/sri/Desktop
sri@envy:/mnt/c/Users/sri/Desktop
$ cd -
/mnt/c/Users/sri/Desktop/Linux
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$
```

create directory - mkdir
 remove directory - rmdir or rm
 rmdir - removes only empty directories
 rm -r - to delete directory and its contents
 mkdir -p directory - create directory with the intermediate ones
 eg. mkdir Book/Chapter01/Section01 - creates Chapter01/ subdirectory too.

```
/etc/init.d sri@envy:/mnt/c/Users/sri/Desktop
$ cd - $ mkdir Linux/Book/tmp
/var/log mkdir: cannot create directory 'Linux/Book/tmp': No such file or directory
$ pwd sri@envy:/mnt/c/Users/sri/Desktop
/var/log $ mkdir -p Linux/Book/tmp
$ sri@envy:/mnt/c/Users/sri/Desktop
$ ls Linux/ Linux/Book/
Linux: Linux_command_line_for_you_and_me_Release_0.1.pdf
Book Linux_Succinctly.pdf
$ Linux/Book:
tmp
sri@envy:/mnt/c/Users/sri/Desktop
$
```

```
sri@envy:/mnt/c/Users/sri/Desktop
$ rmdir Linux/Book/tmp
sri@envy:/mnt/c/Users/sri/Desktop
$ ls Linux/ Linux/Book/
Linux:
Book Linux_command_line_for_you_and_me_Release_0.1.pdf
Linux_Succinctly.pdf
```

```
Linux/Book:
sri@envy:/mnt/c/Users/sri/Desktop
$ mkdir -p Linux/Book/tmp
sri@envy:/mnt/c/Users/sri/Desktop
$ ls Linux/ Linux/Book/
Linux: Linux_command_line_for_you_and_me_Release_0.1.pdf
Book Linux_Succinctly.pdf
```

```
Linux/Book:
tmp
sri@envy:/mnt/c/Users/sri/Desktop
$ rmdir -p Linux/Book/tmp
rmdir: failed to remove directory 'Linux/Book/tmp': Directory not empty
$ ls Linux/
Linux_command_line_for_you_and_me_Release_0.1.pdf Linux_Succinctly.pdf
$ sri@envy:/mnt/c/Users/sri/Desktop
$ mkdir -p Linux/Book/tmp
sri@envy:/mnt/c/Users/sri/Desktop
$ touch Linux/Book/tmp/a
sri@envy:/mnt/c/Users/sri/Desktop
$ rmdir -p Linux/Book/tmp
rmdir: failed to remove 'Linux/Book/tmp': Directory not empty
sri@envy:/mnt/c/Users/sri/Desktop
$
```

Creating and Removing Directories

To create a directory, use the **mkdir** command. Directories can be deleted with the **rmdir** and **rm** commands.

mkdir [-p] directory Create a directory. When used with the **-p** (parents) option, intermediate directories are created.

rmdir [-p] directory Remove a directory. When used with the **-p** (parents) option, all the specified directories in the path are removed. The **rmdir** command only removes empty directories. To remove directories and their contents, use **rm**.

Options can be combined without a space after the dash and the order do not matter.

rm -rf directory The **rm** command removes files, directories, or both. To have **rm** recursively remove a directory and all of its contents, use the **-r** (recursive) and **-f** (force) options. Multiple options can be combined by using a dash followed by all the options without a space. Order does not matter. The commands **rm -r -f dir**, **rm -rf dir**, and **rm -fr dir** are all equivalent.

rm -f - suppresses all the warnings.

Use the **rm** command with caution, especially **rm -rf**. The command line doesn't have a trash container where you can restore accidentally deleted files. When you delete something at the command line it is gone. The following demonstrates the use of **mkdir**, **rmdir**, and **rm**.

```
$ mkdir newdir
$ mkdir newdir/one/two
mkdir: cannot create directory 'newdir/one/two': No such file or directory
$ mkdir -p newdir/one/two
$ rmdir newdir
rmdir: directory "newdir": Directory not empty
$ rm -rf newdir
$ ls newdir
ls: newdir: No such file or directory
$ mkdir newerdir
$ rmdir newerdir
$ ls newerdir
ls: cannot access newerdir: No such file or directory
$
```

```
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ mkdir tmp
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ touch tmp/a tmp/b tmp/c
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ ls tmp/
a b c
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ rm tmp
rm: cannot remove 'tmp': Is a directory
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ rm -f tmp
rm: cannot remove 'tmp': Is a directory
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ rm -r tmp
sri@envy:/mnt/c/Users/sri/Desktop/Linux
$ ls tmp/
ls: cannot access 'tmp/': No such file or directory
sri@envy:/mnt/c/Users/sri/Desktop/Linux
```

cd ./ cd .. / cd -

Chapter 5 Viewing File and Directory Details

The `ls` command was briefly introduced in [Chapter 3](#). It not only lists files and directories, it can provide important details about those files and directories. One of the most common options to use with `ls` is `-l`, which displays a long listing format. The following is an example.

```
$ ls
Desktop  Documents  Downloads  Music  to-do.txt
$ ls -l
total 20
drwxrwxr-x 2 jason users 4096 May  3 08:33 Desktop
drwxrwxr-x 2 jason users 4096 May  3 08:35 Documents
drwxrwxr-x 2 jason users 4096 May  3 08:38 Downloads
drwxrwxr-x 3 jason users 4096 Jun 21 21:16 Music
-rw-r--r-- 1 jason users   73 Jun 22 19:34 to-do.txt
$
```

The information provided by `ls -l` starts with a series of characters that represent the permissions of the file or directory. Permissions will be covered in [Chapter 6](#). The number that follows the permissions string represents the number of links to the file or directory. Next, the owner is displayed followed by the group name. The file size is then displayed. The timestamp provided represents the modification time. The last item is the name of the file or directory itself.

```
$ ls -l to-do.txt
-rw-r--r-- 1 jason users 73 Jun 22 19:34 to-do.txt
```

File Name	Modification Time	Size in bytes	Group	User (owner)	Number of Links	Permissions
to-do.txt	Jun 22 19:34	73	users	jason	1	-rw-r--r--

By default, `ls` does not display files or directories that begin with a period. In Linux, such files are considered hidden. To display hidden files with `ls`, use the `-a` option to include all items. To display all items in a long listing format, use `-l` and `-a`. Remember that the options can be combined. These three commands are equivalent: `ls -l -a`, `ls -la`, and `ls -al`.

```
$ ls
Desktop Documents Downloads Music to-do.txt
$ ls -a
.  .. .bash_history .bash_logout .bashrc Desktop Documents Downloads
Music .profile .ssh to-do.txt
$ ls -a -l
total 48
drwxr-xr-x 7 jason users 4096 Jun 22 20:36 .
drwxr-xr-x 6 root  root  4096 May  4 10:26 ..
-rw----- 1 jason users 3738 Jun 22 19:37 .bash_history
-rw-r--r-- 1 jason users  220 Mar 30  2013 .bash_logout
-rw-r--r-- 1 jason users 3650 Jun 22 19:41 .bashrc
drwxrwxr-x 2 jason users 4096 May  3 08:33 Desktop
drwxrwxr-x 2 jason users 4096 May  3 08:35 Documents
drwxrwxr-x 2 jason users 4096 May  3 08:38 Downloads
drwxrwxr-x 3 jason users 4096 Jun 21 21:16 Music
-rw-r--r-- 1 jason users  675 Mar 30  2013 .profile
drwx----- 2 jason users 4096 May  3 12:44 .ssh
-rw-r--r-- 1 jason users   73 Jun 22 19:34 to-do.txt
$ ls -al
total 48
drwxr-xr-x 7 jason users 4096 Jun 22 20:36 .
drwxr-xr-x 6 root  root  4096 May  4 10:26 ..
-rw----- 1 jason users 3738 Jun 22 19:37 .bash_history
-rw-r--r-- 1 jason users  220 Mar 30  2013 .bash_logout
-rw-r--r-- 1 jason users 3650 Jun 22 19:41 .bashrc
drwxrwxr-x 2 jason users 4096 May  3 08:33 Desktop
drwxrwxr-x 2 jason users 4096 May  3 08:35 Documents
drwxrwxr-x 2 jason users 4096 May  3 08:38 Downloads
drwxrwxr-x 3 jason users 4096 Jun 21 21:16 Music
-rw-r--r-- 1 jason users  675 Mar 30  2013 .profile
drwx----- 2 jason users 4096 May  3 12:44 .ssh
-rw-r--r-- 1 jason users   73 Jun 22 19:34 to-do.txt
$ ls -la
total 48
drwxr-xr-x 7 jason users 4096 Jun 22 20:36 .
drwxr-xr-x 6 root  root  4096 May  4 10:26 ..
-rw----- 1 jason users 3738 Jun 22 19:37 .bash_history
```

```
-rw-r--r-- 1 jason users 220 Mar 30 2013 .bash_logout
-rw-r--r-- 1 jason users 3650 Jun 22 19:41 .bashrc
drwxrwxr-x 2 jason users 4096 May 3 08:33 Desktop
drwxrwxr-x 2 jason users 4096 May 3 08:35 Documents
drwxrwxr-x 2 jason users 4096 May 3 08:38 Downloads
drwxrwxr-x 3 jason users 4096 Jun 21 21:16 Music
-rw-r--r-- 1 jason users 675 Mar 30 2013 .profile
drwx----- 2 jason users 4096 May 3 12:44 .ssh
-rw-r--r-- 1 jason users 73 Jun 22 19:34 to-do.txt
```

To append a file type indicator to the name of the file or directory in the `ls` output, use the `-F` option.

```
$ ls
Desktop Documents Downloads link-to-to-do Music program to-do.txt
$ ls -F
Desktop/ Documents/ Downloads/ link-to-to-do@ Music/ program* to-
do.txt
$ ls -lF
total 24
drwxrwxr-x 2 jason users 4096 May 3 08:33 Desktop/
drwxrwxr-x 2 jason users 4096 May 3 08:35 Documents/
drwxrwxr-x 2 jason users 4096 May 3 08:38 Downloads/
lrwxrwxrwx 1 jason users 9 Jun 22 21:01 link-to-to-do -> to-do.txt
drwxrwxr-x 3 jason users 4096 Jun 21 21:16 Music/
-rwxr-xr-x 1 jason users 13 Jun 22 21:02 program*
-rw-r--r-- 1 jason users 73 Jun 22 19:34 to-do.txt
$
```

Table 2: File Type Indicators

Symbol	File Type
/	Directory
@	Symmlink. The file that follows the -> symbol is the target of the link.
*	Executable script or program
=	Socket
>	Door
	Named pipe

A symbolic link, sometimes called a symlink or just link, points to the location of an actual file or directory. The symlink is just a pointer, but you can operate on it as if it were the actual file or directory. Symbolic links are often used to create shortcuts to long names or long paths. Another common use for symlinks is to point to the current version of an application as in the following example.

```
$ cd /opt/nginx/
$ ls -F
1.6.0/ 1.7.1/ 1.7.2/ current@
$ ls -lF
total 12
drwxr-xr-x 2 root root 4096 Jun 22 21:12 1.6.0/
drwxr-xr-x 2 root root 4096 Jun 22 21:11 1.7.1/
drwxr-xr-x 2 root root 4096 Jun 22 21:11 1.7.2/
lrwxrwxrwx 1 root root    5 Jun 22 21:12 current -> 1.7.2/
$
```

To sort the output of the `ls` command by time, use the `-t` option. This displays the most recently modified items first. If you want to reverse the order, use `-r`. This can come in handy when you have a directory that contains many files. When you sort them by time in reverse the old files will scroll off the top of your screen, while the newest files will be displayed right above your prompt.

```
$ ls -t
program link-to-to-do to-do.txt Music Downloads Documents Desktop
$ ls -lt
total 24
-rwxr-xr-x 1 jason users 13 Jun 22 21:02 program
lrwxrwxrwx 1 jason users 9 Jun 22 21:01 link-to-to-do -> to-do.txt
-rw-r--r-- 1 jason users 73 Jun 22 19:34 to-do.txt
drwxrwxr-x 3 jason users 4096 Jun 21 21:16 Music
drwxrwxr-x 2 jason users 4096 May 3 08:38 Downloads
drwxrwxr-x 2 jason users 4096 May 3 08:35 Documents
drwxrwxr-x 2 jason users 4096 May 3 08:33 Desktop
$ ls -lrt
total 24
drwxrwxr-x 2 jason users 4096 May 3 08:33 Desktop
drwxrwxr-x 2 jason users 4096 May 3 08:35 Documents
drwxrwxr-x 2 jason users 4096 May 3 08:38 Downloads
drwxrwxr-x 3 jason users 4096 Jun 21 21:16 Music
-rw-r--r-- 1 jason users 73 Jun 22 19:34 to-do.txt
lrwxrwxrwx 1 jason users 9 Jun 22 21:01 link-to-to-do -> to-do.txt
-rwxr-xr-x 1 jason users 13 Jun 22 21:02 program
```

```
$
```

To perform a recursive listing, use the **-R** option.

```
$ ls -R
.:
Desktop Documents Downloads link-to-to-do Music program to-do.txt

./Desktop:

./Documents:
cat.jpg report.txt

./Downloads:

./Music:
JohnColtrane

./Music/JohnColtrane:
giant-steps.mp3
$
```

To accomplish the same goal, but in a more visually appealing way, use the **tree** command. To view only the directory structure, use **tree -d**. For colored output, use **tree -C**. The **tree** command is not always installed by default so you may have to rely on the **ls** command.

```
$ tree
.
|-- Desktop
|-- Documents
|   |-- cat.jpg
|   |-- report.txt
|-- Downloads
|-- link-to-to-do -> to-do.txt
|-- Music
|   |-- JohnColtrane
|       |-- giant-steps.mp3
|-- program
|-- to-do.txt

5 directories, 6 files
$ tree -d
```

```
.
|-- Desktop
|-- Documents
|-- Downloads
|-- Music
    |-- JohnColtrane

5 directories
$
```

When the **ls** command is run against a directory, the contents of the directory are displayed. To have **ls** operate on just the directory, use the **-d** option.

```
$ tree Music/
Music/
|-- JohnColtrane
    |-- giant-steps.mp3

1 directory, 1 file
$ ls Music/
JohnColtrane
$ ls -l Music/
total 4
drwxrwxr-x 2 jason users 4096 Jun 22 21:39 JohnColtrane
$ ls -d Music/
Music/
$ ls -ld Music/
drwxrwxr-x 3 jason users 4096 Jun 21 21:16 Music/
$
```

To colorize the output of the **ls** command, use the **--color** option. Much like the **-F** option, this option allows for the differentiation of file types.

```
$ ls --color
Desktop Documents Downloads link-to-to-do Music program to-do.txt
```

The following is a recap of the **ls** options covered in this chapter. Even though **ls** has many more options, these few will cover the most common use cases.

Table 3: Commonly Used Ls Options

Option	Description
-a	Display all files, including hidden files.
--color	Colorize output.
-d	List directories and not their contents.
-l	Use the long listing format.
-r	Reverse the order.
-R	List files recursively.
-t	Sort by time.

Escaping Spaces and Special Characters

Even though spaces are permitted in file and directory names, it can be easier to avoid them if possible. Instead of using spaces consider using hyphens or underscores. Another good option is to use CamelCase. For example, instead of naming a file **my to do list**, name it **my-to-do-list**, **my_to_do_list**, or even **MyToDoList**.

Even if you choose to avoid using spaces in file names, you may encounter file names created by others that do include spaces. The two ways of operating on files with spaces in their names is to use quotation marks or escaping. To operate on a file named **my to do list**, enclose it in quotation marks like so: **"my to do list"**. To escape the file name, precede the spaces with a backslash like so: **my\ to\ do\ list**. Escaping is like using quotation marks except that it is for single characters.

If you are unsure how to escape a file or directory name, let **ls** show you by using the **-b** option. Quoting and escaping not only applies to space, but to other special characters including **|**, **&**, **'**, **;**, **(**, **)**, **<**, **>**, **space**, and **tab**.

```
$ ls
my to do list
$ ls -l
total 4
-rw-r--r-- 1 jason users 73 Jun 22 22:16 my to do list
$ ls -l my to do list
ls: cannot access my: No such file or directory
ls: cannot access to: No such file or directory
ls: cannot access do: No such file or directory
ls: cannot access list: No such file or directory
$ ls -l "my to do list"
```

```
-rw-r--r-- 1 jason users 73 Jun 22 22:16 my to do list
$ ls -l my\ to\ do\ list
-rw-r--r-- 1 jason users 73 Jun 22 22:16 my to do list
$ ls -lb
total 4
-rw-r--r-- 1 jason users 73 Jun 22 22:16 my\ to\ do\ list
$
```

Chapter 6 Permissions

Looking back at the long listings provided by the `ls` command, we can now decipher the permissions for a given file or directory listing. Permissions are displayed at the beginning of long listings.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
```

The first character in the permissions string reveals the type. For example, `-` is a regular file, `d` is a directory, and `l` is a symbolic link. Those are the most common types you will encounter. However, there are other file types listed in the following table.

Table 4: File Types

File Type	Symbol
Regular file	-
Block special file	b
Character special file	c
Directory	d
Symbolic link	l
FIFO (named pipe)	p
Socket	s
Some other file type	?

The remaining characters in the permissions string represent the three main types of permissions: read, write, and execute. Each permission is represented by a single letter, also known as a symbol. Read is represented by `r`, write by `w`, and execute by `x`.

Table 5: Permissions

Permission	Symbol
Read	r
Write	w
Execute	x

For files as opposed to directories, read, write, and execute permissions have intuitive meanings. Read permissions allow you to view the contents of a file. Write permissions allow you to modify a file. Execute permissions allow you to run, or execute, a file as a program.

The meanings of the read, write, and execute permissions are not as intuitive when it comes to directories. Read permissions allow you to read the file names in a directory. Write permissions allow you to change the entries in a directory by renaming files, creating files, and deleting files. Execute permissions allow you to `cd` or change into the directory. Review the following differences between file and directory permissions.

Table 6: File and Directory Permissions

Permission	File Meaning	Directory Meaning
Read	Allows a file to be read.	Allows file names in the directory to be read.
Write	Allows a file to be modified.	Allows entries within the directory to be modified.
Execute	Allows the execution of a file.	Allows access to the contents and metadata of entries within the directory.

All files in Linux are owned by a user and a group. This allows for unique permissions to be applied across three sets of users: the user owner, the group owner, and others. When modifying permissions, these sets can be represented by a single letter: **u** for the user owner, **g** for the group owner, and **o** for others. In addition, the letter **a** can represent all three of these permissions groups. Note that these characters do not show up in an `ls` listing, but they can be used when changing permissions.

Table 7: User Categories

Category	Symbol
User	u
Group	g
Other	o
All	a

Going back to our original example, we can view the user and group owner of the file **sales.data**. The user owner is listed first, followed by the group owner. In this case, **jason** is the user owner, and **users** is the group owner.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
```

Every user is a member of at least one group, called their primary group. However, users can be members of many groups. Groups are used to organize users into logical sets. For example, a group named **sales** might be created and contain all the employees from the sales department. You could then set a file's group owner as the **sales** group, and allow members of the **sales** group read and write permissions to the file, or any other set of permissions for that matter.

To determine what groups you are a member of, run the **groups** command. If you supply another user's ID as an argument to the **groups** command, you will see the list of groups to which that user belongs. You can also run **id -Gn [user]** to get the same result.

```
$ groups
users sales
$ id -Gn
users sales
$ groups tracy
users projectx dba
$ groups john
users sales manager
```

Decoding Permissions

Now you have enough background information to start decoding permissions strings. The first character in the permissions string is the type. The next three characters represent the permissions available to the user, also known as the owner of the file. The next three characters represent the permissions available to the group. The last three characters represent the permissions available to all others.

In this case, order has meaning. Permission groups will always be displayed in this order: user, group, and others. Within these three permission groups, permission types will always be in this order: read, write, and execute. If a particular permission type is not granted, then a hyphen (-) will take its place.

Here is a colorized representation of the permission information displayed by **ls -l**. The file type is highlighted in red, the user permissions in green, the group permissions in blue, and the other permissions in purple.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
```

By examining the preceding example, you can determine that the file type is a regular file (-), the user owner is allowed read and write permissions (**rw-**), the group owner is granted read permissions (**r--**), and others are allowed read permissions as well (**r--**). The user owner is **jason**, and the group owner is **users**.

If there happens to be an additional character at the end of the permissions string, an alternative access control method has been applied. A trailing period (.) means that a SELinux (Security-Enhanced Linux) security context has been applied to the file or directory. A trailing plus sign (+) means that ACLs (Access Control Lists) are in use. SELinux and ACLs are beyond the scope of this book. However, you will be pleased to know that the use of these is rare. If you are having issues with permissions, look for an additional trailing character in the permissions string. If one is present be aware that further investigation may be necessary.

```
$ ls -l sales.data.selinux
-rw-r--r--. 1 jason users 1040 Jun 14 09:31 sales.data.selinux
$ ls -l sales.data.acl
-rw-r--r--+ 1 jason users 1040 Jun 14 09:31 sales.data.acl
```

Changing Permissions

Permissions are also known as modes. The command **chmod**, which is short for "change mode," is used to change permissions. The format of the **chmod** command is **chmod mode file**. There are two ways to specify the mode. The first way is called symbolic mode. The symbolic mode format is **chmod user_category operator permission**. Here is a table view of the **chmod** command using the symbolic mode format.

Table 8: Change Mode Command Symbols

Symbol	Description
chmod	The change mode command itself.
ugo	The user category. Use one or more of u for user, g for group, o for other, a for all.
+-=	One of + , - , or = . Use + to add permissions, - to subtract them, or = to explicitly set them.
rwX	The actual permissions. Use one or more of r for read, w for write, and x for execute.

You can add, subtract, or set permissions using user category and permission pairs. For example, if you want to add the write permission for the group owner, you would specify **chmod g+w file**.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
$ chmod g+w sales.data
$ ls -l sales.data
-rw-rw-r-- 1 jason users 10400 Jun 14 09:31 sales.data
```

After running **chmod g+w sales.data**, the permissions string changed from **-rw-r--r--** to **-rw-rw-r--**. Remember that the permissions are displayed in the order of user, group, and other. The group permission set now includes the **w** symbol, indicating that the write permission has been granted. Now **jason**, the owner of the file, and members of the **users** group can read and write to the **sales.data** file. The following example demonstrates how to subtract the write permission.

```
$ ls -l sales.data
-rw-rw-r-- 1 jason users 10400 Jun 14 09:31 sales.data
$ chmod g-w sales.data
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
```

Multiple permissions can be changed at once. For example, you can add write and execute permissions for the group owner by using **g+wx**.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
$ chmod g+wx sales.data
$ ls -l sales.data
-rw-rwxr-- 1 jason users 10400 Jun 14 09:31 sales.data
```

You can also modify multiple permissions groups at once. For example, **ug+wx** will add write and execute permissions for the user and group owners if they don't already have them. In this case, notice that the user owner already had write permission before the **chmod** command was executed. After running **chmod**, the user owner will still have write permissions, as well as the newly added execute permissions. Using **+** to add permissions will always add permissions, if applicable. It never takes them away.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
$ chmod ug+wx sales.data
$ ls -l sales.data
-rwxrwxr-- 1 jason users 10400 Jun 14 09:31 sales.data
```

If you want to set different permissions for different user categories, you can separate the specifications with a comma. You can mix and match to produce the permissions you desire. For example, **u=rwx,g+x** will set the read, write, and execute permissions for the file owner while adding the execute permission for the group. See how the permissions change for **sales.data** in the following example.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
$ chmod u=rwx,g+x sales.data
```

```
$ ls -l sales.data
-rwxr-xr-- 1 jason users 10400 Jun 14 09:31 sales.data
```

If you want to set the file to be readable, and only readable, by everyone, run **chmod a=r file**. When you use the equal sign (=), the current permissions are replaced by what is specified. In this case, **a=r** sets the read permission for user, group, and other. Any write or execute permissions will be removed.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
$ chmod a=r sales.data
$ ls -l sales.data
-r--r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
```

If you do not specify permissions following the equal sign, the permissions are removed. Here is an illustration of this behavior.

```
$ ls -l sales.data
-rw-r--r-- 1 jason users 10400 Jun 14 09:31 sales.data
$ chmod u=rwx,g=rx,o= sales.data
$ ls -l sales.data
-rwxr-x--- 1 jason users 10400 Jun 14 09:31 sales.data
```

Numeric Based Permissions

The second way to specify modes with the **chmod** command is called octal mode. Understanding symbolic mode will help you learn octal mode. Some Linux users never move beyond symbolic permissions. However, experienced Linux users find using octal mode quicker and easier in the long term because there are only a few commonly used permissions which can be readily memorized and recalled.

Octal mode permissions are based on the binary numeral system, also known as the base-2 numeral system. Each permission type is treated as a bit that is either set to off, represented by a zero (0), or on, represented by a one (1). In permissions, order has meaning. Permissions are always in read, write, and execute order. If **r**, **w**, and **x** are all set to off, the binary representation is **000**. If they are all set to on, the binary representation is **111**. To represent read and write permissions while omitting execute permissions, the binary number is **110**.

Table 9: Base-2 and Base-10 Representations of Permissions

	Read	Write	Execute
Binary and decimal value for off.	0	0	0

	Read	Write	Execute
Binary value for on.	1	1	1
Decimal value for on.	4	2	1

Supply the **chmod** command with the base-10, or decimal, value of the desired permissions. To convert the binary representation into decimal, remember that read equals **4**, write equals **2**, and execute equals **1**. The permissions number is determined by adding up the values for each permission type. For example, read and execute permissions are represented by 5 because **4** for read, plus **1** for execute, equals **5**. There are eight possible values from zero to seven, hence the name octal mode. The following table demonstrates all eight of the possible permissions.

Table 10: Octal Permissions

Permissions	String	Binary	Octal
No permissions	---	000	0
Execute only	--x	001	1
Write only	-w-	010	2
Write and Execute	-wx	011	3
Read only	r--	100	4
Read and Execute	r-x	101	5
Read and Write	rw-	110	6
Read, Write, and Execute	rwX	111	7

Remember that in permissions order has meaning. The user categories are always in this order: user, group, and other. Once the octal value is determined for each category, it must be specified in that order. For example, to get **-rwxr-xr-**permissions, run **chmod 754 file**. That means the owner of the file has read, write, and execute permissions; the members of the file's group have read and execute permissions; and others only have read permissions.

Table 11: Symbolic, Binary, and Decimal Representations of a Given Permission

	User	Group	Other
Symbolic	rwX	rw-	r--
Binary	111	110	100
Decimal	7	6	4

Commonly Used Permissions

The following table illustrates the most commonly used permissions. These five permission sets will cover most permission situations.

Table 12: Commonly Used Permissions

Meaning	Octal	Symbolic
Allows the file's owner full control over the file. No others on the system have access.	700	-rwx-----
Allows everyone on the system to execute the file but only the owner can edit it.	755	-rwxr-xr-x
Allows a group of people to modify the file and let others read it.	664	-rw-rw-r--
Allows a group of people to modify the file and not let others read it.	660	-rw-rw----
Allows everyone on the system to read the file but only the owner can edit it.	644	-rw-r--r--

Many times newcomers to the Linux operating system err on the side of permissive permissions. Instead of thinking through the required permissions they sometimes grant "everything to everybody" by using **777** or **666** permissions. Whenever you see a file or directory with **777** or **666** permissions, know that there is almost always a better permission set that can be used.

Granting unnecessary privileges to a file or directory not only has security implications, but it can also invite unwanted changes to those files or directories. If a file has **777** permissions, then anyone on the Linux system can edit that file. This can lead to a situation where someone accidentally saves changes to a file when all they really wanted to do was view the file's contents with an editor.

Also, a user on the system could purposefully use the weak permissions to escalate privileges, gain access to data they shouldn't see, or even destroy data. They could potentially insert malicious code into a script or program and wait for it to be executed by someone else on the system. Remember that anyone on the system can execute the file because all permissions, including the execute permission, have been granted.

If multiple people require write access to a file, make use of groups and limit the access of others. Consider it a best practice to avoid using **777** and **666** permission modes.

Working with Groups

Let's look at a situation where multiple people need access to the same file. For example, if the members of a sales team need to update a file named **sales.report**, the group owner of the file could be set to the Linux group named **sales** by using the **chgrp** command. Next, the permissions could be set to **664** (**rw-rw-r--**) or even **660** (**rw-rw---**) if you do not want others on the system to be able to read the file. Technically, **774** (**rw-rwxr--**) or **770** (**rw-rwx---**) permissions also work, but since **sales.report** is not an executable program, it makes more sense to use **664** (**rw-rw-r--**) or **660** (**rw-rw----**).

When a file is created, it is set to the current user's primary group. You can override this behavior by using the **newgrp** command, but remember by default a new file will inherit your default group. In the following example, Jason's primary group is **users**. The format of the **chgrp** command is **chgrp GROUP FILE**.

```
$ nano sales.report
$ ls -l sales.report
-rw-r--r-- 1 jason users 6 Jun 15 20:41 sales.report
$ chgrp sales sales.report
$ ls -l sales.report
-rw-r--r-- 1 jason sales 6 Jun 15 20:41 sales.report
$ chmod 664 sales.report
$ ls -l sales.report
-rw-rw-r-- 1 jason sales 6 Jun 15 20:41 sales.report
```

Sharing files from within individual user's home directories can be confusing. It's often easier to keep shared data in a shared location. If you have superuser privileges, you could create a **/usr/local/sales** directory for the sales team. If you don't have such permissions you can ask the system administrator to create that directory for you. The group owner of the shared directory should be set to **sales** and the permissions should be set to **775** (**rw-rwxr-x**) or **770** (**rw-rwx---**). Use **770** (**rw-rwx---**) if no one outside the sales team should have access to any files, directories, or programs located in **/usr/local/sales**.

```
$ ls -ld /usr/local/sales
drwxrwxr-x 2 root sales 4096 Jun 15 20:53 /usr/local/sales
$ mv sales.report /usr/local/sales/
$ ls -l /usr/local/sales
total 4
-rw-rw-r-- 1 jason sales 6 Jun 15 20:41 sales.report
```

Directory Permissions

A common problem encountered by Linux newcomers is incorrect directory permissions. Directory permissions usually only contain **0s**, **5s**, and **7s**. Common directory permissions

include **755**, **700**, **770**, and **750**. Incorrect directory permissions can prevent file access and file execution. If you determine that a file's permissions have been set correctly, look at the parent directory's permissions. Work your way toward the root of the file system by running `ls -ld .` in the current directory, moving up to the parent directory with `cd ..`, and repeating those two steps until you find the problem.

```
$ ls -ld directory/
drwxr-xr-x 2 jason users 4096 Sep 29 22:02 directory/
$ ls -l directory/
total 0
-rwxr--r-- 1 jason users    0 Sep 29 22:02 testprog
$ chmod 400 directory/
$ ls -ld directory/
dr----- 2 jason users 4096 Sep 29 22:02 directory/
$ ls -l directory/
ls: cannot access directory/testprog: Permission denied
total 0
-???????? ? ? ? ?           ? testprog
$ directory/testprog
-su: directory/testprog: Permission denied
$ chmod 500 directory/
$ ls -ld directory/
dr-x----- 2 jason users 4096 Sep 29 22:02 directory/
$ ls -l directory/
total 0
-rwxr--r-- 1 jason users 0    Sep 29 22:02 testprog
$ directory/testprog
This program ran successfully.
```

Default Permissions and the File Creation Mask

The file creation mask, also known as the **umask**, determines the default permissions of new files and directories. The **umask** is typically set by the system administrator; however, an individual user may override the setting by including a **umask** statement in his or her account's initialization files.

If no mask is applied, new directories receive **777** (**rw-rw-rw-**) permissions and new files receive **666** (**rw-rw-rw-**) permissions. When the **umask** is applied to these base permissions, it disables, or masks, certain permissions. For example, a **umask** of **000** will disable, or mask, zero bits. In this case, new directories receive **777** permissions and new files receive **666** permissions. At the other extreme, a **umask** of **777** disables all permissions bits. New files and directories receive **000** permissions in this instance.

umask [-S] [mode] Sets the file creation mask to a mode if specified. If a mode is omitted, the current mode will be displayed. Using the **-S** argument allows **umask** to display or set the mode with symbolic notation.

A quick way to estimate how a **umask** mode affects default permissions is to subtract the octal **umask** mode from **777** in the case of directories, or **666** in the case of files. The following is an example of a **022 umask**, which is typically the default **umask** used by Linux distributions or set by system administrators.

Table 13: Creation Permission Estimation

	File	Directory
Base Permission	666	777
Subtract the umask	-022	-022
Creation Permission	644	755

Using a **umask** of **002** is ideal for working with members of your group. When files or directories are created, the permissions allow members of the group to manipulate those files and directories.

Table 14: Creation Permission Estimation

	File	Directory
Base Permission	666	777
Subtract the umask	-002	-002
Creation Permission	664	775

Here is another possible **umask** to use for working with members of your group. By using **007**, no permissions are granted to users outside of the group.

Table 15: Creation Permission Estimation

	File	Directory
Base Permission	666	777
Subtract the Umask	-007	-007
Creation Permission	660 *	770

Again, using this octal subtraction method is a good estimation. You can see that the method breaks down with the **umask** mode of **007**. In reality, to get an accurate result each time, you need to convert the octal permissions into binary values. From there, you use a bitwise NOT operation on the **umask** mode and then perform a bitwise AND operation against that and the base permissions. Another way to think of this is that the **umask** disables the values specified. A

umask of **007** effectively means "disable all of the bits for the other users." A **umask** of **022** means "disable the write bits for the group and others."

The following table contains all the resulting permissions created by each one of the eight mask settings. Note that the most common and practical **umask** modes to use are **022**, **002**, **077**, and **007**.

Table 16: *umasks and Resulting Permissions*

File Permissions	Directory Permissions	Binary	Octal
rw-	rwX	000	0
rw-	rw-	001	1
r--	r-X	010	2
r--	r--	011	3
-w-	-wX	100	4
-w-	-w-	101	5
---	--X	110	6
---	---	111	7

Special Modes

When the **umask** command is queried for the current setting, it returns four characters instead of three. The following example shows the **umask** being clearly set to **002**, but **umask** returns **0022**.

```
$ umask 022
$ umask
0022
```

Until now, you have only been introduced to permissions for user, group, and other. However, there is a class of special modes called **setuid**, **setgid**, and sticky. These special modes are declared by prepending a character to the octal mode that you normally use with **umask** or **chmod**. The important point here is to know that **umask 022** is the same as **umask 0022**. Likewise, **chmod 644** is the same as **chmod 0644**.

The **setuid** permission allows the program to run as the owner of the file, not the user executing it. One example of where this permission is used is with the **passwd** command. The **passwd** command allows a user to change his or her own password, but it requires superuser privileges to modify the **/etc/passwd** and **/etc/shadow** files.

Prepend the number **4** when using octal mode to enable the **setuid** permission. For symbolic mode, use **u+s**.

```
$ ls -ld /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 Jul 26 2013 /usr/bin/passwd
$ chmod 4555 script
$ ls -l script
-r-sr-xr-x. 1 jason users 0 Jun 7 18:11 script
```

Similar to the **setuid** permission, the **setgid** permission allows a program to run with the group of the file, not the group of the user executing it. A Linux command that uses such a permission is the **locate** command.

When the **setgid** is used on a directory, it causes new entries in that directory to be created with the same group as the directory. When working with groups, using **setgid** on shared directories can prevent someone from accidentally creating a file in their default group instead of the intended group.

Prepend the number **2** when using octal mode to enable the **setgid** permission. For symbolic mode, use **g+s**.

```
$ ls -l /usr/bin/locate
-rwx--s--x 1 root slocate 35548 Oct 10 2012 /usr/bin/locate
$ chmod 2555 script
$ ls -l script
-r-xr-sr-x 1 jason users 0 Jun 7 18:28 script
$ mkdir salesdir
$ chgrp sales salesdir
$ chmod g+rx salesdir
$ ls -ld salesdir/
drwxrwxr-x 2 jason sales 4096 Jun 7 18:29 salesdir/
$ touch salesdir/file-before-setgid
$ ls -l salesdir/
total 0
-rw-r--r-- 1 jason users 0 Jun 7 18:29 file-before-setgid
$ chmod g+s salesdir
$ ls -ld salesdir
drwxrwsr-x 2 jason sales 4096 Jun 7 18:29 salesdir
$ touch salesdir/file-after-setgid
$ ls -l salesdir/
total 0
-rw-r--r-- 1 jason sales 0 Jun 7 18:30 file-after-setgid
-rw-r--r-- 1 jason users 0 Jun 7 18:29 file-before-setgid
$
```

The sticky bit prevents one user from deleting another user's files even if he or she would normally have permission to do so. The most common place you will see the sticky bit employed is on the `/tmp` and `/var/tmp` directories.

Prepend the number **1** when using octal mode to enable the sticky bit. For symbolic mode, use **+t**.

```
$ ls -ld /tmp
drwxrwxrwt 11 root root 20480 Jun  6 18:17 /tmp
$ ls -ld /var/tmp
drwxrwxrwt 4 root root 4096 Jun  7 16:46 /var/tmp
$ chmod 1777 tmp
$ ls -ld tmp
drwxrwxrwt 2 jason users 4096 Jun  7 16:50 tmp
```

Table 17: Special Modes

Permission	Octal
sticky bit	1
setgid	2
setuid	4

umask Examples

In the following example, new files and directories are created after setting the **umask**. Notice that the default file and directory permissions depend on the **umask** setting.

```
$ umask
0022
$ umask -S
u=rwx,g=rx,o=rx
$ mkdir directory
$ touch file
$ ls -l
total 4
drwxr-xr-x 2 jason users 4096 Jun  5 00:03 directory
-rw-r--r-- 1 jason users    0 Jun  5 00:03 file
$ rmdir directory
$ rm file
$ umask 007
$ umask
```

```
0007
$ umask -S
u=rwx,g=rwx,o=
$ mkdir directory
$ touch file
$ ls -l
total 4
drwxrwx--- 2 jason users 4096 Jun  5 00:04 directory
-rw-rw---- 1 jason users    0 Jun  5 00:04 file
```

Chapter 7 Viewing and Editing Files

In a preceding chapter, you learned that the **cat** command displays the entire contents of a file. If you would like to navigate the contents of a file, you can use a pager utility such as **more** or **less**. To display the top portion of a file, use the **head** command. The **tail** command allows you to display the end of a file.

cat file Concatenate (display) the entire contents of a file.

more file Browse through a text file. Press the Spacebar to advance to the next page. Press Enter to advance to the next line. Type **q** to quit viewing the file.

less file Like **more**, but allows backward movement and pattern searches.

head file Display the beginning portion of file.

tail file Display the ending portion of file.

```
$ cat goals.txt
1) Write a book.
2) Travel the world.
3) Learn a foreign language.
4) Learn to play an instrument.
5) Run a marathon.
6) Skydive.
7) Start a business.
8) Swim with dolphins.
9) Own a home.
10) Be an extra in a movie.
11) Win an Olympic medal.
12) Be a millionaire.
$ head goals.txt
1) Write a book.
2) Travel the world.
3) Learn a foreign language.
4) Learn to play an instrument.
5) Run a marathon.
6) Skydive.
7) Start a business.
8) Swim with dolphins.
9) Own a home.
10) Be an extra in a movie.
```

```
$ tail goals.txt
3) Learn a foreign language.
4) Learn to play an instrument.
5) Run a marathon.
6) Skydive.
7) Start a business.
8) Swim with dolphins.
9) Own a home.
10) Be an extra in a movie.
11) Win an Olympic medal.
12) Be a millionaire.
$ more goals.txt
1) Write a book.
2) Travel the world.
3) Learn a foreign language.
4) Learn to play an instrument.
5) Run a marathon.
...
```

The **head** and **tail** commands display 10 lines by default. To specify a specific number of lines to display, append **-N** to the command where *N* is the number of lines you want to display. For example, to display the first line in a file, run **head -1 file**.

```
$ head -2 goals.txt
1) Write a book.
2) Travel the world.
$ tail -1 goals.txt
12) Be a millionaire.
$
```

If you want to view files as they are being updated, use **tail -f file**. The **-f** flag makes the **tail** command follow the file as it grows. This is great for viewing log files. You can also use the **less** command. After running **less file**, type **F** to start following the file as it grows.

```
$ tail -f /opt/nginx/logs/access.log
10.10.10.10 - - [28/Jun/2014:18:38:48 +0000] "GET / HTTP/1.1" 200 612 "-"
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36"
11.11.11.11 - - [28/Jun/2014:18:39:16 +0000] "GET / HTTP/1.1" 200 612 "-"
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36"
...
```


Editing Files

An extremely simple, but ample text editor is nano. It doesn't have advanced editing features, but if you are looking to make simple changes to a file, this will surely work. To edit an existing file or create a new one, run **nano file-name**. When it loads, you will see the contents of the file and a list of available commands at the bottom of the screen. The caret symbol represents the Ctrl key. For example, to exit the editor type **Ctrl-x**, and to save the file type **Ctrl-o**. For help, type **Ctrl-g**.

In addition to using the navigation commands listed at the bottom of the screen, you can simply use the arrow keys, the Page Up and Page Down keys, and the Home and End keys. To add text, simply type it. Deleting text is as simple as using the Delete and Backspace keys. To delete an entire line, use **Ctrl-k**.

```
GNU nano 2.2.6           File: to-do.txt

This file contains my to-do list.
* Mow the lawn.
* Take over the world.

                                [ Read 3 lines ]
^G Get Help  ^O WriteOut ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

The Vim Editor

If you are looking for an editor that has advanced editing capabilities, and one that you can use at the command line, use vi or Emacs. On a Linux system, when you attempt to use vi, you will actually be using Vim, short for vi improved. The **vi** command is typically symlinked to Vim. The Vim editor is compatible with the commands found in the vi editor, which was originally created for the Unix operating system. Vim includes additional features not found in vi, including syntax highlighting, the ability to edit files over the network, multi-level undo and redo, and screen splitting. One advantage of learning Vim or vi is that you can apply the key mappings to other commands, such as man, more, less, and view.

Command Mode

One unique characteristic of the vi and Vim editors is the concept of modes. The three modes in Vim are command, insert, and line. Vim starts in command mode. To get back to command mode at any time, simply press the Escape key. When in command mode, the keys sent to the editor do not end up in the file, but are rather interpreted as commands. Command mode allows you to navigate about the file, perform searches, delete text, copy text, paste text, and more.

Table 18: Vim Navigation Keys

Key	Action
k	Move up one line.
j	Move down one line.
h	Move left one character.
l	Move right one character.
w	Move right one word.
b	Move left one word.
^	Move to the beginning of the line.
\$	Move to the end of the line.

The commands are case sensitive. For example, lowercase L moves the cursor right one character, but uppercase L moves the cursor to the bottom of the window. Even though the original vi editor did not allow you to use arrow keys, Vim does. Even though you can use the arrow keys, some argue that using the original vi key bindings can be faster since your hand does not have to leave the home row.

Insert Mode

Insert mode allows you to actually type text in a file. To enter insert mode, press i, I, a, or A. After you have entered the desired text, you can press the Escape key to return to command mode.

Table 19: Vim Insert Mode

Key	Action
i	Insert at the current position.
I	Insert at the beginning of the line.
a	Append after the cursor position.
A	Append at the end of the line.

Line Mode

Line mode, sometimes called command line mode, is accessed from command mode by typing a colon. Line mode allows you to save a file, exit the editor, replace text, and perform some forms of navigation. The following are some of the most common line mode commands you will want to familiarize yourself with.

Table 20: Vim Line Mode Commands

Key	Action
:w	Writes, or saves, the file.
:w!	Forces the file to be saved even if the write permission is not set.
:q	Quits the editor. This fails if there are unsaved changes to the file.
:q!	Quit without saving the file.
:wq!	Write and quit.
:x	Same as :wq!
:n	Position the cursor at line n.
:\$	Position the cursor on the last line of the file.
:set nu	Turn on line numbering.
:set nonu	Turn off line numbering.
:help [subcommand]	Access the built-in help documentation.

Repeating Commands

Most commands can be repeated by preceding them with a number. For example, to move the cursor down three lines, type **3j**. To insert the same piece of text 20 times, type **20i** followed by the desired text, and press Escape when you are finished. The insert operation will repeat 20 times. To insert a line of underscores, type **80i_** and press Escape.

Additional Commands

The following tables list some additional key combinations to use while in the command mode.

Table 21: Vim Command Mode—Deleting Text

Key	Action
x	Delete a character.
dw	Delete a word.
dd	Delete a line.

Key	Action
D	Delete from the current position to the end of the line.

Table 22: Vim Command Mode—Changing Text

Key	Action
r	Replace the current character.
cw	Change the current word.
cc	Change the current line.
c\$	Change the text from the current position to the end of the line.
C	Same as c\$.
~	Reverse the case of the character at the current position.

Table 23: Vim Command Mode—Copying and Pasting Text

Key	Action
yy	Yank, or copy, the current line.
y<position>	Yank the <position>. To yank a word, type yw .
p	Paste the most recent yanked or deleted text.
D	Delete from the current position to the end of the line.

Table 24: Vim Command Mode—Undo and Redo

Key	Action
u	Undo
Ctrl-r	Redo

Table 25: Vim Command Mode—Searching

Key	Action
/<pattern>	Start a forward search for <pattern>.
?<pattern>	Start a reverse search for <pattern>.

Emacs

Another powerful text editor that you can use at the command line is Emacs. Emacs relies heavily on compound keyboard shortcuts. In the Emacs documentation, you will see **C-*<character>***, which means press and hold Ctrl and then press *<character>*. For example, if you see **C-x**, that means hold down Ctrl and press x. You will also see sequences, like **C-x u**. That means hold down Ctrl and press x, release the Ctrl key, and then press u. **C-x C-c** means press and hold Ctrl, press x, and then press c while still holding Ctrl.

You will also encounter **M-*<character>***, which means hold down the meta key, which is the Alt key, and press *<character>*. A substitute to holding down Alt as the meta key is to press and release the Esc key instead. For example, you can press and hold Alt and press b to represent **M-b**, or you can press Esc followed by the b key. Some terminal emulators intercept the Alt key, so you may be forced to use Esc as the meta key in some situations.

Table 26: Emacs Basic Keyboard Shortcuts

Key	Action
C-h	Help.
C-x C-c	Exit.
C-x C-s	Save the file.
C-h t	Access the built-in tutorial.
C-h k <key>	Describe <key>.
C-u N <command>	Repeat <command> N times.

Table 27: Emacs Keyboard Shortcuts—Navigation

Key	Action
C-p	Move to the previous line.
C-n	Move to the next line.
C-b	Move backward one character.
C-f	Move forward one character.

Key	Action
M-f	Move forward one word.
M-b	Move backward one word.
C-a	Move to the beginning of the line.
C-e	Move to the end of the line.
M-<	Move to the beginning of the file.
M->	Move to the end of the file.

Table 28: Emacs Keyboard Shortcuts—Deleting Text

Key	Action
C-d	Delete a character.
M-d	Delete a word.

Table 29: Emacs Keyboard Shortcuts—Copying and Pasting Text

Key	Action
C-k	Kill, or cut, the rest of the current line.
C-y	Yank, or paste, from the previously killed text.
C-x u	Undo. Repeat for multiple-level undo.

Table 30: Emacs Keyboard Shortcuts—Searching

Key	Action
C-s	Start a forward search. Type the text you are looking for and press C-s to move to the next occurrence. Press Enter to stop searching.
C-r	Start a reverse search.

Graphical Editors

Nano, Vim, and Emacs are great for editing files at the command line. However, if you are using a GUI, you have many more options, but don't think that the effort you put into learning Vim or Emacs is worthless in a graphical environment. The graphical version of Vim is gVim. If you are using a GUI, Emacs detects this and starts in graphical mode.

If you are looking for a word processor, consider LibreOffice or AbiWord. LibreOffice is an office suite which not only includes a word processor, but ships with a spreadsheet program, a database application, and presentation software.

There are also specialty editors available for the Linux operating system. If you are looking for an IDE or a source code editor, consider jEdit, Geany, Kate, or Sublime Text. The editors listed in the following table are just a sampling of what is available.

Table 31: Graphical Editors

Editor	Description
gVim	Graphical version of Vim.
Emacs	Graphical version of Emacs.
gedit	A Notepad-like editor for the GNOME desktop environment.
KEdit	The default text editor for the KDE desktop environment.
AbiWord	Word processor.
LibreOffice	Office suite.
jEdit	Programmer's text editor.
Geany	A small and fast IDE.
Kate	A multi-document editor.
Sublime Text	An editor for source code.

Chapter 8 Deleting, Moving, and Renaming Files and Directories

Files and directories can be deleted with the `rm` command.

`rm file` Remove `file`.

`rm -r directory` To remove a directory with `rm`, the `-r` argument is required. The `-r` argument tells `rm` to remove files and directories recursively.

`rm -f file` Use the `-f` option to force removal without prompting for confirmation.

Search patterns in the form of wildcards can be used with commands like `rm` and `ls`. The most commonly used wildcards are the asterisk and the question mark. The asterisk matches anything, while the question mark matches a single character. Remember that files and directories that begin with a period are considered hidden and will not be matched by the asterisk. To include the hidden file in your search pattern, start your search with a period.

```
$ ls
Desktop Documents Downloads goals.txt Music Pictures to-do.txt
$ ls t*
to-do.txt
$ rm t*
$ ls t*
ls: cannot access t*: No such file or directory
$ ls g*txt
goals.txt
$ ls g????????
goals.txt
$ ls g?
ls: cannot access g?: No such file or directory
$ ls -d .*
. .. .bash_history .bash_logout .bashrc .hidden .profile
$ rm .hidden
$
```

The `cp` command is used to copy files and directories. To create a copy, run `cp source_file destination_file`. You can also copy one or more files to a directory by ending the `cp` command with a destination directory.

`cp source_file destination_file` Copy the `source_file` to the `destination_file`.

cp source_file1 [source_fileN ...] destination_directory Copy the **source_files** to the **destination_directory**.

cp -i source_file destination_file Use the **-i** option of **cp** to run in interactive mode. If the **destination_file** exists, **cp** will give you the opportunity to abort the operation or continue by overwriting the **destination_file**.

cp -r source_directory destination_directory The **-r** option of **cp** causes the **source_directory** to be recursively copied to the **destination_directory**. If the **destination_directory** exists, the source directory is copied into the **destination_directory**. Otherwise the **destination_directory** will be created with the contents of the **source_directory**.

```
$ ls
1file
$ cp 1file 2file
$ ls
1file 2file
$ mkdir 1dir
$ cp 1file 2file 1dir
$ ls 1dir/
1file 2file
$ cp -i 2file 1file
cp: overwrite `1file'? n
$ cp -r 1dir 2dir
$ ls 2dir/
1file 2file
$ cp 1dir 3dir
cp: omitting directory `1dir'
$ mkdir 3dir
$ cp -r 1dir 2dir 3dir
$ ls 3dir
1dir 2dir
$ tree 3dir
3dir
|-- 1dir
|   |-- 1file
|   |-- 2file
|-- 2dir
     |-- 1file
     |-- 2file

2 directories, 4 files
$
```

To move files and directories from one location to another, use the **mv** command. Additionally, the **mv** command is used to rename files and directories.

mv source destination Moves **source** to **destination**. If **destination** is a directory, **source** will be moved into **destination**. If **destination** is not a directory, then **source** will be renamed **destination**.

mv -i source destination Use the **-i** option of **mv** to run in interactive mode. If the **destination** exists, **mv** will give you the opportunity to abort the operation or continue by overwriting the **destination**.

In the following example, **1dir** is renamed to **1dir-renamed** using the **mv** command. Next, **1file** is renamed to **file1** and then moved into the **1dir-renamed** directory. If you do not specify the **-i** option to **mv**, it will overwrite an existing file without prompting you. This is demonstrated by moving **1file** to **2file**. Finally, the **-i** option is demonstrated with **2file** and **file1**.

```
$ ls -F
1dir/ 1file 2dir/ 2file 3dir/
$ mv 1dir 1dir-renamed
$ ls -F
1dir-renamed/ 1file 2dir/ 2file 3dir/
$ mv 1file file1
$ ls -F
1dir-renamed/ 2dir/ 2file 3dir/ file1
$ mv file1 1dir-renamed/
$ ls -F
1dir-renamed/ 2dir/ 2file 3dir/
$ ls -F 1dir-renamed/
1file 2file file1
$ cat 1dir-renamed/1file
The contents of 1file.
$ cat 1dir-renamed/2file
The contents of 2file.
$ mv 1dir-renamed/1file 1dir-renamed/2file
$ cat 1dir-renamed/2file
The contents of 1file.
$ ls -F 1dir-renamed/
2file file1
$ mv -i 1dir-renamed/2file 1dir-renamed/file1
mv: overwrite `1dir-renamed/file1'? n
$
```

Chapter 9 Finding, Sorting, and Comparing Files and Directories

To locate files or directories on a Linux system, you can use the **find** command. You can find files by owner, size, permissions, name, modification time, and more.

find [path...] [expression] Recursively find files and directories in **path** that match **expression**. When running **find** without arguments, **path** is assumed to be the current directory.

```
$ find
.
./.bash_history
./Pictures
./.bashrc
./Downloads
./.bash_logout
./viminfo
./Desktop
./Documents
./goals.txt
./to-do.txt
./profile
./Music
./Music/JohnColtrane
$
```

Table 32: Common find Commands

Command	Description
<code>find . -name pattern</code>	Display items whose names match pattern (case sensitive).
<code>find . -iname pattern</code>	Same as -name , but not case sensitive.
<code>find . -ls</code>	Perform an -ls operation on each of the items.
<code>find . -mtime number_of_days</code>	Display items that are number_of_days old.

Command	Description
<code>find . -size number</code>	Display items that are size number . The number can be followed by a character, which represents the unit of space: c for bytes, k for kilobytes, M for megabytes, and G for gigabytes.
<code>find . -newer file</code>	Display items that are newer than file .
<code>find . -exec command {} \;</code>	Run command against each of the found items. The braces ({}) act as a placeholder for the current file being processed.

The following are examples of using the **find** command. You can combine multiple find options, or expressions, to find exactly what you are looking for.

```
$ find /etc -name log*conf
/etc/logrotate.conf
$ find /opt -name Nginx
$ find /opt -iname Nginx
/opt/nginx
$ find /opt -iname Nginx -ls
655431      4 drwxr-xr-x   2 root   root           4096 Jul  1 03:34 /opt/nginx
$ find . -mtime +11 -mtime -14
./.bashrc
./.viminfo
$ find . -size +2M
./Music/JohnColtrane/giantsteps.mp3
$ find . -type d -newer to-do.txt
.
./Music/JohnColtrane
$ find . -name *mp3 -exec mpg123 {} \;
High Performance MPEG 1.0/2.0/2.5 Audio Player for Layers 1, 2 and 3
    version 1.12.1; written and copyright by Michael Hipp and others
    free software (LGPL/GPL) without any warranty but with best wishes

Directory: ./Music/JohnColtrane/
Playing MPEG stream 1 of 1: giantsteps.mp3 ...
Title:   Giant Steps
MPEG 1.0 layer III, 192 kbit/s, 44100 Hz stereo
[0:00] Decoding of giantsteps.mp3 finished.
$
```

The **find** command examines each file and directory in the provided path to determine if it matches the given expression. Sometimes this is a very quick operation if only a small number of items have to be examined. However, if you were to run **find / -name some_name**, **find** would examine every single file on the system and this could potentially be a slow process. There is another utility that you can use to find items on a Linux system and it's called **locate**.

locate pattern Display files and directories that match **pattern**.

The **locate** command queries an index, or database, which is updated daily by a process named **updatedb**. The advantage to this approach is that it's really fast since it doesn't have to examine files and directories in real time. The disadvantage is that it is not in real time. The **locate** command is great for finding files or directories that are older than a day, but it won't find items that have just been created. Also, **locate** and **updatedb** are sometimes not installed or enabled.

```
$ locate giant
/home/jason/Music/JohnColtrane/giantsteps.mp3
$ locate httpd.conf
/etc/apache2/httpd.conf
$
```

Sorting

Use the **sort** command to sort the contents of files.

Table 33: Commonly Used sort Options

Option	Description
sort file	Sort the text in file .
sort -k FIELD_NUM file	Sort by "key." Sort by the FIELD_NUM column.
sort -r file	Sort in reverse order.
sort -u file	Sort uniquely. No duplicates are displayed.

```
$ cat random-states
Tennessee      Nashville
Wyoming         Cheyenne
Indiana         Indianapolis
Indiana         Indianapolis
Arizona         Phoenix
Colorado        Denver
Indiana         Indianapolis
```

```

Georgia      Atlanta
$ sort random-states
Arizona      Phoenix
Colorado     Denver
Georgia      Atlanta
Indiana      Indianapolis
Indiana      Indianapolis
Indiana      Indianapolis
Tennessee    Nashville
Wyoming      Cheyenne
$ sort -u random-states
Arizona      Phoenix
Colorado     Denver
Georgia      Atlanta
Indiana      Indianapolis
Tennessee    Nashville
Wyoming      Cheyenne
$ sort -k2 -u random-states
Georgia      Atlanta
Wyoming      Cheyenne
Colorado     Denver
Indiana      Indianapolis
Tennessee    Nashville
Arizona      Phoenix
$

```

Comparing

You can use the **diff**, **sdiff**, and **vimdiff** commands to compare files and directories. The **diff** command displays just the differences, **sdiff** displays the two files side-by-side while highlighting the differences, and **vimdiff** uses the Vim editor to display the differences. Simply supply the command two items to compare.

```

$ cat random-states
Arizona      Phoenix
Colorado     Denver
Georgia      Atlanta
Indiana      Indianapolis
$ cat random-states.bak
Arizona      Phoenix
Colorado     Denver

```

```

Georgia          Savannah
Indiana          Indianapolis
$ diff random-states random-states.bak
3c3
< Georgia        Atlanta
---
> Georgia        Savannah
$ sdiff random-states random-states.bak
Arizona          Phoenix          Arizona          Phoenix
Colorado         Denver          Colorado         Denver
Georgia          Atlanta          | Georgia        Savannah
Indiana          Indianapolis      | Indiana        Indianapolis
$ vimdiff random-states random-states.bak
Arizona          Phoenix          | Arizona        Phoenix
Colorado         Denver          | Colorado       Denver
Georgia          Atlanta          | Georgia        Savannah
Indiana          Indianapolis      | Indiana        Indianapolis

random-states      1,1          All random-states.bak  1,1          All
"random-states.bak" 4L, 104C
$ tree
.
|-- dir1
|   |-- file1
|-- dir2
|   |-- file1
|   |-- file2

2 directories, 3 files
$ diff dir1 dir2
Only in dir2: file2
$

```

In the **diff** output, the text following the less than sign belongs to the first file while the text following the greater than sign belongs to the second file. Also **diff** provides information about the differences in a shorthand format. The first number represents line numbers from the first file. The next character will be a **c** for change, a **d** for deletion, or a **a** for an addition. The final number represents lines from the second file.

The **sdiff** command uses a pipe to represent changes on the same line. The less than sign is used to denote that particular line only exists in the first file, while the greater than sign means the line only exists in the second file.

Chapter 10 I/O Redirection

The Linux operating system features a concept called I/O streams. The three default I/O streams are standard input, standard output, and standard error. When a process is launched, it is connected to these three I/O streams, also called standard streams. By default, standard input comes from your keyboard while standard output and standard error are displayed on your screen. By convention, standard output is used for normal output while standard error is reserved for error messages.

Each stream is assigned a file descriptor. A file descriptor is referenced by a number and represents an open file. Standard input is assigned file descriptor **0**, standard output is assigned file descriptor **1**, and standard error is assigned file descriptor **2**. This effectively means that your keyboard and display are treated as files. As a matter of fact, your keyboard and display can be substituted for actual files. This layer of abstraction allows you to save output that would normally appear on your screen to a file. It also allows you send input to a command from a file. You can even use the output of one command as the input for another command.

Table 34: Standard Streams

File Descriptor	Abbreviation	Stream
0	stdin	standard input
1	stdout	standard output
2	stderr	standard error

Many Linux commands allow you to provide input by specifying a file as an argument or by accepting standard input. In the absence of a file, many commands expect standard input. Files, as well as standard input, are terminated with an end of file (EOF) marker. You can produce this EOF marker using your keyboard by typing **Ctrl-d**.

As an example of this behavior, let's look at the **sort** command. To have **sort** operate on a file, supply that file as an argument as in the following example.

```
$ cat test.txt
e
a
c
b
d
$ sort test.txt
a
b
c
```



```
d
e
$
```

To have **sort** operate on standard input, run the **sort** command without any arguments and start typing text. When you are finished, type **Ctrl-d** to send the EOF character. The standard input you provided will then be sorted.

```
$ sort
e
a
c
b
d
<Ctrl-d>
a
b
c
d
e
$
```

To send the standard output of one command as the standard input to another command, use a pipe symbol (**|**) between the commands. The following example demonstrates sending the output of **cat test.txt** as the input to the **sort** command.

```
$ cat test.txt | sort
a
b
c
d
e
$
```

To use the contents of a file as standard input, separate the command from the file with a less than sign (**<**).

```
$ sort < test.txt
a
b
c
d
```

```
e
$
```

To redirect the output of a command to a file, use the greater than sign (>) followed by a file name. If the file doesn't exist it will be created. If it does exist, it will be overwritten.

```
$ sort test.txt > sorted.txt
$ cat sorted.txt
a
b
c
d
e
$
```

If you want to append output to a file, use the double greater than sign (>>). If the file doesn't exist, it will be created, but if it does exist, the output from the command will be appended to the file.

```
$ sort test.txt >> sorted.txt
$ cat sorted.txt
a
b
c
d
e
a
b
c
d
e
$
```

You are not limited to just redirecting input or just redirecting output—you can do both at the same time. The following example demonstrates reading standard input from **test.txt** while redirecting standard output to **sorted.txt**.

```
$ sort < test.txt > sorted.txt
$ cat sorted.txt
a
b
c
```

```
d
e
$
```

Table 35: Redirecting I/O

Operator	Format	Action
>	cmd > file	Create or overwrite file with standard output from cmd .
>>	cmd >> file	Create or append to file with standard output from cmd .
<	cmd < file	Use the contents of file as standard input to cmd .

By default, input redirection operates on file descriptor 0 and output redirection operates on file descriptor 1. You can explicitly declare a file descriptor to use with redirection by immediately preceding the operator with the file descriptor number. Do not use a space between the file descriptor number and the redirection operator. If the file descriptor does not immediately precede the redirection operator, it will be interpreted as another item on the command line.

To capture error messages to a file while displaying standard output to your screen, use **2>file**. You can also redirect standard output to one file while redirecting standard output to another.

```
$ ls test.txt no-such-file
ls: cannot access no-such-file: No such file or directory
test.txt
$ ls test.txt no-such-file 2>errors
test.txt
$ cat errors
ls: cannot access no-such-file: No such file or directory
$ ls test.txt no-such-file 1>normal-output 2>errors
$ cat normal-output
test.txt
$ cat errors
ls: cannot access no-such-file: No such file or directory
$
```

Not only can you redirect output to a file, you can redirect it to another file descriptor using an ampersand followed by the file descriptor number. Using this method, you can combine standard output and standard error using **2>&1**. Do not use spaces.

The following command means, "send the standard output of the **ls** command to the **combined-output** file and append standard error to standard output." All output will be sent to the **combined-output** file because standard error is redirected to standard output and standard output is redirected to **combined-output**.

```
$ ls test.txt no-such-file > combined-output 2>&1
$ cat combined-output
ls: cannot access no-such-file: No such file or directory
test.txt
$
```

If you do not want to display the output of a command to your screen or save it to a file, you can redirect the output to the null device, **/dev/null**. This special file simply discards any input that is sent to it. The null device is sometimes referred to as the "black hole" or "bit bucket". The following example redirects the errors from **sort** to the null device.

```
$ ls test.txt no-such-file
ls: cannot access no-such-file: No such file or directory
test.txt
$ ls test.txt no-such-file 2>/dev/null
test.txt
$
```

Chapter 11 Additional Command Line Concepts

An environment variable is a name-value pair. Programs can use data from environment variables to determine how to behave in certain situations. For example, the default command used to display man pages can be altered by setting a value for the **PAGER** environment variable.

Environment variables are case-sensitive; however, by convention they are in all uppercase letters. To view the value of a known environment value, run **echo \$VAR_NAME** or **printenv VAR_NAME**. You can use the **env** or **printenv** commands without arguments to display all the environment variables that are currently set.

```
$ echo $HOME
/home/jason
$ printenv HOME
/home/jason
$ printenv
SHELL=/bin/bash
TERM=xterm
USER=jason
MAIL=/var/mail/jason
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
PWD=/home/jason
LANG=en_US.UTF-8
PS1=$
SHLV=1
HOME=/home/jason
LOGNAME=jason
OLDPWD=/home/jason
$ env
SHELL=/bin/bash
TERM=xterm
USER=jason
MAIL=/var/mail/jason
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
PWD=/home/jason
LANG=en_US.UTF-8
PS1=$
SHLV=1
HOME=/home/jason
```

```
LOGNAME=jason
OLDPWD=/home/jason
$
```

When a process is launched, it inherits the exported environment variables of its parent process. An environment variable that is set only affects the current running process, unless it is explicitly exported. In the following example, the **PAGER** environment variable is set to **less** for the current shell. If a subprocess is started without that variable being exported, such as another instance of the Bash shell, that environment variable is not inherited. When **PAGER** is exported and a new Bash shell is started, it is available to that process. To remove an environment variable, use the **unset** command.

```
$ echo $PAGER

$ PAGER=less
$ echo $PAGER
less
$ bash
$ echo $PAGER

$ exit
exit
$ export PAGER=less
$ bash
$ echo $PAGER
less
$ exit
exit
$ echo $PAGER
less
$ unset PAGER
$ echo $PAGER

$
```

Table 36: Common Environment Variables

Variable	Use
EDITOR	The program used to edit files.
HOME	The user's home directory.
LOGNAME	The user ID or login ID of the current user.

Variable	Use
MAIL	The location of the user's mailbox on the local system.
OLDPWD	The old, or previous, working directory.
PATH	The search path for commands.
PAGER	The program used for paging through a file.
PS1	The primary prompt string.
PWD	The present working directory.
USER	The user ID or login ID of the current user.

Aliases

You can use keyboard shortcuts, called aliases, at the command line. You can save yourself some time and typing by creating aliases for commands that you repeat often, that are long, that are hard to type, or that are difficult to remember. You can even use aliases to fix common typing mistakes. Some people even employ aliases to make Linux behave like another operating system they are familiar with.

alias [**alias_name**[=**value**]] Without any arguments, the alias command lists the current aliases that are in your environment. Use **alias alias_name=value** to create a new alias.

unalias alias_name Remove **alias_name**. Use **unalias -a** to delete all aliases.

```
$ alias ll='ls -l'
$ alias
alias ll='ls -l'
$ ls -l
total 32
drwxrwxr-x 2 jason jason 4096 Jun 21 22:01 Desktop
drwxrwxr-x 2 jason jason 4096 Jun 21 22:01 Documents
drwxrwxr-x 2 jason jason 4096 May 17 13:37 Downloads
-rw-rw-r-- 1 jason jason 274 Jun 28 14:52 goals.txt
drwxrwxr-x 3 jason jason 4096 Jun 21 22:05 Music
drwxrwxr-x 2 jason jason 4096 Jun 21 22:01 Pictures
-rw-rw-r-- 1 jason jason 73 Jun 29 02:30 to-do.txt
$ ll
total 32
drwxrwxr-x 2 jason jason 4096 Jun 21 22:01 Desktop
```

```

drwxrwxr-x 2 jason jason 4096 Jun 21 22:01 Documents
drwxrwxr-x 2 jason jason 4096 May 17 13:37 Downloads
-rw-rw-r-- 1 jason jason 274 Jun 28 14:52 goals.txt
drwxrwxr-x 3 jason jason 4096 Jun 21 22:05 Music
drwxrwxr-x 2 jason jason 4096 Jun 21 22:01 Pictures
-rw-rw-r-- 1 jason jason 73 Jun 29 02:30 to-do.txt
$ alias bu='/usr/local/bin/backup-database.sh'
$ bu
Starting database backup.
...
Database backup complete.
$ alias
alias bu='/usr/local/bin/backup-database.sh'
alias ll='ls -l'
$

```

Aliases only exist for your current session. So, if you were to create an alias, log out, and log in again, that alias would not be available. To make them persist between sessions, you have to add them to your personal initialization files.

Personal Initialization Files

To save customizations to your shell environment, place them in a personal initialization file. If you are using Bash, you can place your customizations in `~/.bashrc` or `~/.bash_profile`. The `~/.bash_profile` file is read and executed for login sessions. When bash is not started as a login shell, for example when you open a new tab in your terminal emulator application, `~/.bashrc` is read and executed. If you do not want or need this distinction, you can make `~/.bash_profile` source `~/.bashrc` and place all your customizations in `~/.bashrc`. Using this method will provide the same environment whether it's a login shell or not. Personal initialization files are often referred to as "dot files."

```

$ cat ~/.bash_profile
if [ -f ~/.bashrc ]; then
    source ~/.bashrc
fi
$

```

The **source** command reads and executes the commands from the given file in the current shell. You can also source files by using a period, so **source file1** and **. file1** are the same thing. The **if** statement in the preceding example simply checks to see if the `~/.bashrc` file exists before trying to source it.


```
$ echo "alias ll='ls -l'" >> ~/.bashrc
$ cat ~/.bashrc
# A line that begins with a pound sign is a comment.
# Place customizations in this file.
alias ll='ls -l'
$ . ~/.bashrc
$ alias ll
alias ll='ls -l'
$
```

Shell History

The commands that you execute at the command line are preserved in your shell history. Your history is retained in memory by Bash until your current session is ended. At that time, your history is saved to the `~/.bash_history` file. Different shells save history in different files, but they usually include the word history and are stored as a dot file in your home directory. Also, you can set the **HISTSIZE** environment variable to control the number of commands to save in your shell history. The default value is 500.

history When the history command is executed without arguments, it displays a list of commands in your shell history.

!N Repeat the command associated with line number **N**.

!! Repeat the previous command line.

!pattern Repeat the most recent command starting with **pattern**.

```
$ history
1 ls
2 diff random-states random-states.bak
3 history
$ !1
ls
Desktop Documents Downloads link-to-to-do Music program tmp to-
do.txt
$ echo $SHELL
/bin/bash
$ !!
echo $SHELL
/bin/bash
$ !d
diff random-states random-states.bak
```

```
3c3
< Georgia      Atlanta
---
> Georgia      Savannah
$
```

You can search through your shell history by typing **Ctrl-r**. This starts a reverse search indicated by **(reverse-i-search)`:** and allows you to type in a portion of a command in your history to retrieve. To keep traversing your history for other commands that match your search pattern, continue to press **Ctrl-r**. Once you find a command you want to execute, press Enter. If you want to change the command line before executing it, press Esc. To completely abandon your reverse search, type **Ctrl-c**.

```
$ diff random-states random-states.bak
3c3
< Georgia      Atlanta
---
> Georgia      Savannah
(reverse-i-search)`di': diff random-states random-states.bak
3c3
< Georgia      Atlanta
---
> Georgia      Savannah
$
```

Tab Completion

To invoke tab completion, simply start typing a command and press the Tab key. Tab completion attempts to complete partially typed commands when possible. If there are many possibilities, those options can be displayed by pressing Tab twice. You can continue to type and press the Tab key again at any time.

In addition to completing commands, you can use tab completion to complete file and directory names. This can be useful when a file or directory is used as an argument to a command like **ls**, **cat**, **rm**, and others.

```
$ # Typing jo[Tab][Tab] results in:
$ jo
jobs  join

$ # Typing job[Tab][Enter] results in:
$ jobs
```

```
[1]+  Running                  ./db-backup.sh &

$ ls r*
random-states  random-states.bak
$ # Typing cat[Space]r[Tab][Enter] results in:
$ cat random-states
Tennessee      Nashville
Wyoming         Cheyenne
Indiana         Indianapolis
Indiana         Indianapolis
Arizona         Phoenix
Colorado        Denver
Indiana         Indianapolis
Georgia         Atlanta
$
```

Line Continuation

If you want to create a command line that visually spans multiple lines but acts as a single command, use a backslash at the end of each line you want to continue. When the backslash appears at end of a command line, it acts as the line continuation character. If you use this at the command prompt, the continued lines will be prefixed with the greater than symbol. You may encounter this when reading documentation or examining shell scripts.

```
$ diff \
> random-states \
> random-states.bak
3c3
< Georgia      Atlanta
---
> Georgia      Savannah
$ diff random-states random-states.bak
3c3
< Georgia      Atlanta
---
> Georgia      Savannah
$
```

Chapter 12 Processes and Jobs

The **ps** command is used to list the currently running processes on a Linux system. If you run **ps** without any arguments, it displays the processes that are running as you and associated with your terminal. If you were to connect to a Linux server twice, you would see different output from the **ps** command. You might see the following for the first session, which is using **pts/0** (pseudo terminal 0).

```
$ ps
  PID TTY          TIME CMD
 1309 pts/0 00:00:00 bash
 1635 pts/0 00:00:00 ps
$
```

The following is the output from **ps** on the second connection, which is using **pts/1**.

```
$ ps
  PID TTY          TIME CMD
 1721 pts/1 00:00:00 bash
 1821 pts/1 00:00:00 ps
$
```

If you want to display all of your running processes, regardless of the associated terminal or lack thereof, use the command **ps -u username**.

```
$ ps -u jason
  PID TTY          TIME CMD
 1308 ?        00:00:00 sshd
 1309 pts/0 00:00:00 bash
 1720 ?        00:00:00 sshd
 1721 pts/1 00:00:00 bash
$
```

To see every process running on the system, use the command **ps -e**.

```
$ ps -e | head
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    6 ?        00:00:00 migration/0
```

```

 7 ?      00:00:00 watchdog/0
 8 ?      00:00:00 cpuset
 9 ?      00:00:00 khelper
10 ?      00:00:00 kdevtmpfs
11 ?      00:00:00 netns
$

```

By default, the information provided by **ps** is rather sparse. Typically when using **ps** you will supply additional arguments to display more detailed information. The following are some of the most common options to use with **ps**.

Table 37: Commonly Used PS Options

Option	Description
-e	Display all processes.
-f	Use a full format listing.
-u <username>	Display processes for username
-p <PID>	Display process information for process ID (PID).
-H	Display processes in a hierarchy (tree).
--forest	Display processes in a hierarchy using ASCII art.

The following table illustrates some useful ways to combine the preceding options.

Table 38: Commonly Used PS Commands

Command	Description
ps -e	Display all processes.
ps -ef	Display all processes using a full format listing.
ps -eH	Display all processes in a tree format.
ps -e -forest	Display all processes in a tree format with ASCII art.
ps -u <username>	Display processes running for username.
ps -fp <PID>	Display a full-format listing for process ID (PID).

The following demonstrates output from various **ps** commands.

```
$ ps
  PID TTY          TIME CMD
 1309 pts/0 00:00:00 bash
 2096 pts/0 00:00:00 ps

$ ps -f
UID          PID  PPID  C STIME TTY          TIME CMD
jason        1309   1308  0 15:15 pts/0        00:00:00 -bash
jason        2102   1309  0 15:45 pts/0        00:00:00 ps -f

$ ps -p 1309
  PID TTY          TIME CMD
 1309 pts/0 00:00:00 bash

$ ps -fp 1309
UID          PID  PPID  C STIME TTY          TIME CMD
jason        1309   1308  0 15:15 pts/0        00:00:00 -bash

$ ps -e | head
  PID TTY          TIME CMD
   1 ?            00:00:00 init
   2 ?            00:00:00 kthreadd
   3 ?            00:00:00 ksoftirqd/0
   6 ?            00:00:00 migration/0
   7 ?            00:00:00 watchdog/0
   8 ?            00:00:00 cpuset
   9 ?            00:00:00 khelper
  10 ?          00:00:00 kdevtmpfs
  11 ?          00:00:00 netns

$ ps -ef | head
UID          PID  PPID  C STIME TTY          TIME CMD
root          1     0  0 15:14 ?            00:00:00 /sbin/init
root          2     0  0 15:14 ?            00:00:00 [kthreadd]
root          3     2  0 15:14 ?            00:00:00 [ksoftirqd/0]
root          6     2  0 15:14 ?            00:00:00 [migration/0]
root          7     2  0 15:14 ?            00:00:00 [watchdog/0]
root          8     2  0 15:14 ?            00:00:00 [cpuset]
root          9     2  0 15:14 ?            00:00:00 [khelper]
root         10     2  0 15:14 ?            00:00:00 [kdevtmpfs]
root         11     2  0 15:14 ?            00:00:00 [netns]

$ ps -fu www-data
UID          PID  PPID  C STIME TTY          TIME CMD
www-data    1060  1057  0 15:15 ?            00:00:00 /usr/sbin/apache2 -k start
www-data    1061  1057  0 15:15 ?            00:00:00 /usr/sbin/apache2 -k start
www-data    1062  1057  0 15:15 ?            00:00:00 /usr/sbin/apache2 -k start
```

A command similar to running **ps** with the **-H** or **--forest** options is **pstree**.

```
$ pstree | head
init--accounts-daemon---{accounts-daemon}
    |-acpid
    |-apache2--apache2
    |   `--2*[apache2---26*[{apache2}]]
    |-at-spi-bus-laun--dbus-daemon
    |   `--3*[{at-spi-bus-laun}]
    |-at-spi2-registr---{at-spi2-registr}
    |-atd
    |-console-kit-dae---64*[{console-kit-dae}]
    |-cron
$
```

The **ps** command displays a point-in-time snapshot of the running processes. If you want an updating display of processes, use **top** or **htop**. The **top** and **htop** commands provide a system summary and process list. The commands are interactive, so while the program is running, you can sort processes by CPU usage, memory usage, or even kill a given process.

```
$ top
top - 16:05:29 up 50 min,  2 users,  load average: 0.00, 0.01, 0.05
Tasks:  88 total,   1 running,  87 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.7%us,  0.2%sy,  0.3%ni, 97.9%id,  0.8%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   503444k total,   346020k used,   157424k free,   45748k buffers
Swap:      0k total,      0k used,      0k free,   176524k cached

   PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
   974 root        20   0   285m  29m  9444  S   2.0   6.0    0:18.67 Xorg
  1440 lightdm    20   0   577m  19m  11m  S   2.0   3.9    0:07.14 unity-
greeter
     1 root        20   0   24596 2392 1268  S   0.0   0.5    0:00.34 init
...
```

The **top** command will be on any Linux system you encounter; however, you may have to install the **htop** command, as it is typically not a part of the base set of packages installed on most distributions.

Jobs

By default, when you execute a program at the command line, it runs in the foreground. While this program, or foreground process, is running, you aren't able to execute any other

commands. Once the program is finished, a new command prompt is displayed, ready to execute your next command. Many commands take anywhere from a fraction of a second to just a few seconds to run. However, you may want to execute a long running program and continue to perform other work in the meantime. To start a program in the background, end the command line with an ampersand. When you start a program in the background, the command prompt is immediately returned and allows you to continue other commands. These background programs and processes are often referred to as jobs.

Table 39: Job Control

Command	Description
command &	Start command in the background.
Ctrl-c	Kill the foreground process.
Ctrl-z	Suspend the foreground process.
bg [%num]	Background a suspended process.
fg [%num]	Foreground a backgrounded process.
kill [%num]	Kill a process by job number or PID .
jobs [%num]	List all jobs or %num job.

When a program is started in the background, two numbers are returned before the new prompt is displayed. Those two pieces of information are the job number, which is enclosed in brackets, and the process ID (PID). Job numbers can be referenced by preceding them with a percent sign. The following example demonstrates starting multiple processes in the background.

```

$ ./long-running-proc &
[1] 2793
$ ./long-running-proc &
[2] 2795
$ ./long-running-proc &
[3] 2807
$ ./long-running-proc &
[4] 2809
$ jobs
[1]   Done                  ./long-running-proc &
[2]   Running               ./long-running-proc &
[3]-  Running               ./long-running-proc &
[4]+  Running               ./long-running-proc &
$

```

In the output of the **jobs** command, you will notice a plus sign and a minus sign. The plus sign represents the current job, while the minus sign represents what is considered to be the

previous job. The current job is the last job that was started in the background or the most recent process that was stopped while it was running in the foreground. You can reference the current job by using double percent signs (%%) or a percent sign followed by a plus sign (%+). The previous job can be accessed by using a percent sign followed by a minus sign (%-). When working with the **fg** and **bg** commands, the current job is operated upon unless you explicitly specify a different job.

In the preceding output of the **jobs** command, job number 1 is reported as being done while the other jobs are in a running state. The shell reports job statuses right before a new prompt is displayed. The shell will not interrupt your current command line, even if it is empty, to report that a job has completed. To force a new prompt to be displayed, press the Enter key. If any of your jobs have completed, a status will be displayed before your new prompt is presented. The following is an example of that behavior.

```
$ <ENTER>
$ <ENTER>
[2]  Done                  ./long-running-proc &
$ jobs
[3]-  Running              ./long-running-proc &
[4]+  Running              ./long-running-proc &
$
```

In order to return a job to the foreground, use the **fg** command followed by a percent sign and job number. A shorthand way to perform the exact same task is to type a percent sign followed by the job number on the command line. So, **fg %2** and **%2** are equivalent.

Remember that the current job can be referenced by %% or %+. Also, the **fg** command operates on the current job unless another job is supplied. The following four commands are identical.

```
$ fg
$ fg %%
$ fg %+
$ %%
```

The following demonstrates bringing job number three into the foreground.

```
$ jobs
[3]-  Running              ./long-running-proc &
[4]+  Running              ./long-running-proc &
$ fg %3
./long-running-proc
```

To pause or suspend the foreground process, type **Ctrl-z**. A job that has been suspended can be resumed in the background or foreground. To resume a suspended job in the background, type the job specification followed by an ampersand, or use the **bg** command followed by the job

specification. If you want to background the process that was most recently suspended, you can omit the job specification as **bg** will operate on the current job. To resume the job in the foreground, use the **fg** command or just the job specification. The following demonstrates these methods.

```
$ jobs
[1]  Running      ./long-running-proc &
[2]  Running      ./long-running-proc &
[3]- Running      ./long-running-proc &
[4]+ Running      ./long-running-proc &
$ %2
./long-running-proc
^Z
[2]+  Stopped      ./long-running-proc
$ fg %3
./long-running-proc
^Z
[3]+  Stopped      ./long-running-proc
$ jobs
[1]  Running      ./long-running-proc &
[2]-  Stopped      ./long-running-proc
[3]+  Stopped      ./long-running-proc
[4]  Running      ./long-running-proc &
$ bg
[3]+ ./long-running-proc &
$ jobs
[1]  Running      ./long-running-proc &
[2]+  Stopped      ./long-running-proc
[3]  Running      ./long-running-proc &
[4]-  Running      ./long-running-proc &
$
```

To kill a job that is running in the foreground, type **Ctrl-c**. To kill a job that has been backgrounded, use the **kill** command. The kill command takes a job specification or a process ID as an argument. To list the PIDs in addition to the job numbers, use the **-l** option of the **jobs** command.

```
$ jobs
[1]  Running      ./long-running-proc &
[2]  Running      ./long-running-proc &
[3]- Running      ./long-running-proc &
[4]+ Running      ./long-running-proc &
$ fg %1
```

```

./long-running-proc
^C$ jobs
[2]    Running                  ./long-running-proc &
[3]-  Running                  ./long-running-proc &
[4]+  Running                  ./long-running-proc &
$ kill %3
[3]-  Terminated             ./long-running-proc
$ jobs -l
[2]-  2914 Running             ./long-running-proc &
[4]+  2918 Running             ./long-running-proc &
$ kill 2914
[2]-  Terminated             ./long-running-proc
$

```

The **kill** command simply sends a signal to a running process. The default signal, however, is termination. The termination signal is referred to as SIGTERM or just TERM for short. To display a list of signals and their corresponding numbers, use the **kill -l** command. To specify a signal to send to a process, follow the **kill** command with a dash and the signal name or number.

```

$ kill -l | grep SIGTERM
11) SIGSEGV  12) SIGUSR2  13) SIGPIPE  14) SIGALRM  15) SIGTERM
$ kill 123
$ kill -SIGTERM 234
$ kill -TERM 345
$ kill -15 456

```

If a process does not terminate after it has been sent the TERM signal, use the KILL signal. The corresponding number for SIGKILL is 9.

```

$ ps | grep cannot-stop-me
2994 pts/1  00:00:00 cannot-stop-me
$ kill 2994
$ ps | grep cannot-stop-me
2994 pts/1  00:00:00 cannot-stop-me
$ kill -9 2994
$ ps | grep cannot-stop-me
$

```

Chapter 13 Switching Users

To switch users at the command line, use the **su** command. Without any arguments, **su** will switch to the superuser account, also known as root. Alternatively, you can execute **su root**. Switching users will not change your current working directory or environment variables, unless you specify a hyphen following **su**. By specifying a hyphen, you simulate logging into the system as that user, and thus are placed into that user's home directory with that user's environment. For example, **su - root**.

su [username] Change to username or become the superuser.

Common **su** options:

su - A hyphen is used to provide an environment similar to what the user would expect had the user logged in directly.

su -c command Specify a command to be executed. If the command is more than one word in length, it needs to be quoted.

```
jason@linuxsvr:~$ export TEST=1
jason@linuxsvr:~$ su oracle
Password:
oracle@linuxsvr:/home/jason$ echo $TEST
1
oracle@linuxsvr:/home/jason$ pwd
/home/jason
oracle@linuxsvr:/home/jason$ exit
exit
jason@linuxsvr:~$ su - oracle
Password:
oracle@linuxsvr:~$ echo $TEST

oracle@linuxsvr:~$ pwd
/home/oracle
oracle@linuxsvr:~$ exit
jason@linuxsvr:~$ su -c 'echo $ORACLE_HOME' oracle
Password:

jason@linuxsvr:~$ su -c 'echo $ORACLE_HOME' - oracle
Password:
/u01/app/oracle/product/current
jason@linuxsvr:~$
```

If you want to know what user you are working as, run the **whoami** command.

whoami Displays the effective username.

```
$ whoami
jason
$ su oracle
Password:
$ whoami
oracle
$
```

Sudo Super User Do

The **sudo** command allows you to run a command with the security privileges of another user. **sudo** will run the command as the superuser if no username is specified, hence the name "super user do." For example, **sudo ls** will run the **ls** command as the root user. **sudo** is commonly used to install, start, and stop applications that require superuser privileges.

sudo Execute a command as another user, typically the superuser.

One advantage of using **sudo** over the **su** command is that you do not need to know the password of the other user, usually the root user. This can eliminate issues that arise from using shared passwords and generic accounts. When you execute the **sudo** command, you are prompted for the current user's password. If the **sudo** configuration permits access, the command is executed. The **sudo** configuration is typically controlled by the system administrator and requires root access to change.

The **su** command is similar to **sudo**, but you should note these differences: **su** (switch user) asks for the new user's password, whereas **sudo** asks for the current user's password, or possibly no password at all. **Su** will change the current user of the shell, allowing multiple separate commands to be issued, whereas **sudo** runs a single command and is finished. For security reasons, **sudo** is generally preferable to **su**. The system administrator need not give the user the root password, and has full control over what commands work with **sudo**.

Using Sudo

Here are the common ways to use the **sudo** command.

sudo -l List available commands that can be executed with **sudo**.

sudo command Run command as the superuser.

sudo -u root command Same as **sudo** command.

sudo -u user command Run command as user.

sudo su Switch to the superuser account.

sudo su - Switch to the superuser account with an environment you would expect to see had you logged in as that user.

sudo su - username Switch to the username account with an environment you would expect to see had you logged in as that username.

```
$ sudo -l
User jason may run the following commands on this host:
(root) NOPASSWD: /etc/init.d/apache2
(fred) NOPASSWD: /opt/fredsApp/bin/start
(fred) NOPASSWD: /opt/fredsApp/bin/stop
(root) /bin/su - oracle
$ sudo /etc/init.d/apache2 start
* Starting web server apache2
$ sudo -u fred /opt/fredsApp/bin/start
Fred's app started as user fred.
$ sudo su - oracle
[sudo] password for jason:
oracle@linuxsvr:~$ whoami
oracle
oracle@linuxsvr:~$ exit
$ whoami
jason
$
```

The output of **sudo -l** displays what commands can be executed with **sudo** and under which account. In the previous example, **sudo** will not prompt for a password for the commands preceded with **NOPASSWD**. This type of configuration may be required to automate jobs via **cron** that require escalated privileges.

Chapter 14 Installing Software

The most common way to install software on a Linux system is through the use of packages. A package not only contains the files that are installed on the system, but also additional information called metadata. This metadata can include such information as the steps required to complete the installation in the form of pre-installation and post-installation scripts, the permission information for each of the files, the description of the package, the version, the package maintainer, and any additional packages that are required for it to function properly.

To install, upgrade, or remove packages, use a package manager. When you tell the package manager to install a given package, it not only installs that package, but also any other required packages, also called dependencies, based on the package's metadata. The package manager also maintains a database of package information. The package manager records what packages are installed, what versions are installed, and what files belong to what packages.

RPM-Based Distributions

RPM is a recursive acronym that stands for RPM Package Manager; however, it started its life as the RedHat Package Manager. RPM-based distributions include Red Hat Enterprise Linux (RHEL), CentOS, Fedora, Oracle Linux, and Scientific Linux. You can manipulate RPM packages directly with the `rpm` command or with another command line utility called `yum`.

`yum search search-pattern` Search for search-pattern.

`yum install [-y] package` Install package. Use the `-y` option to automatically answer yes to yum's questions.

`yum remove package` Remove or uninstall package.

`yum info [package]` Display information about package.

To search for available software, use **`yum search search-pattern`**.

```
$ yum search web browser
Loaded plugins: refresh-packagekit, security
===== N/S Matched: web, browser =====
icedtea-web.i686 : Additional Java components for OpenJDK - Java browser
                  : plug-in and Web Start implementation
elinks.i686      : A text-mode Web browser
firefox.i686    : Mozilla Firefox Web browser
lynx.i686       : A text-based Web browser
```

Full name and summary matches only, use "search all" for everything.

```
$ yum search firefox
Loaded plugins: refresh-packagekit, security
===== N/S Matched: firefox =====
firefox.i686 : Mozilla Firefox Web browser

Name and summary matches only, use "search all" for everything.
$
```

To install software, use the command **yum install package**. Installing software requires superuser privileges. Use **sudo** or switch to the root account with the **su** command before installing or removing software.

```
$ sudo yum install firefox
Loaded plugins: refresh-packagekit, security
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package firefox.i686 0:24.5.0-1.el6.centos will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch      Version                      Repository      Size
=====
Installing:
firefox                i686      24.5.0-1.el6.centos         updates         47 M

Transaction Summary
=====
Install      1 Package(s)

Total download size: 47 M
Installed size: 80 M
Is this ok [y/N]: y
Downloading Packages:
firefox-24.5.0-1.el6.centos.i686.rpm           | 47 MB    00:14
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : firefox-24.5.0-1.el6.centos.i686                1/1
```



```

Verifying   : firefox-24.5.0-1.el6.centos.i686                1/1

Installed:
  firefox.i686 0:24.5.0-1.el6.centos

Complete!
$

```

To uninstall a package, use the command **yum remove package**. Like installing software, removing software requires superuser privileges.

```

$ sudo yum remove firefox
Loaded plugins: refresh-packagekit, security
Setting up Remove Process
Resolving Dependencies
--> Running transaction check
---> Package firefox.i686 0:24.5.0-1.el6.centos will be erased
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package      Arch          Version                               Repository  Size
=====
Removing:
  firefox     i686          24.5.0-1.el6.centos                  @updates    80 M

Transaction Summary
=====
Remove      1 Package(s)

Installed size: 80 M
Is this ok [y/N]: y
Downloading Packages:
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Erasing   : firefox-24.5.0-1.el6.centos.i686                1/1
  Verifying : firefox-24.5.0-1.el6.centos.i686                1/1

Removed:

```

```
firefox.i686 0:24.5.0-1.el6.centos
```

```
Complete!
```

```
$
```

Using the rpm Command

You can also interact with the RPM package manager directly by using the **rpm** command.

rpm -qa List all the installed packages.

rpm -qf /path/to/file List the package that contains file.

rpm -ivh package.rpm Install a package from the file named package.rpm.

rpm -ql package List all files that belong to package.

```
$ rpm -qa | sort | head
acl-2.2.49-6.el6.i686
acpid-1.0.10-2.1.el6.i686
aic94xx-firmware-30-2.el6.noarch
alsa-lib-1.0.22-3.el6.i686
alsa-plugins-pulseaudio-1.0.21-3.el6.i686
alsa-utils-1.0.22-5.el6.i686
anaconda-13.21.215-1.el6.centos.i686
anaconda-yum-plugins-1.0-5.1.el6.noarch
apache-tomcat-apis-0.1-1.el6.noarch
apr-1.3.9-5.el6_2.i686
$ rpm -qf /usr/bin/sudo
sudo-1.8.6p3-12.el6.i686
$ sudo rpm -ivh SpiderOak-5.1.3-1.i386.rpm
Preparing... ##### [100%]
 1:SpiderOak ##### [100%]
$ rpm -ql sudo | head
/etc/pam.d/sudo
/etc/pam.d/sudo-i
/etc/sudo-ldap.conf
/etc/sudo.conf
/etc/sudoers
/etc/sudoers.d
/usr/bin/sudo
/usr/bin/sudoedit
```

```
/usr/bin/sudoreplay
/usr/libexec/sesh
$
```

DEB-Based Distributions

Linux distributions that are based on Debian use the DEB package format. Some of the more popular Debian-based distributions include Debian, Elementary OS, Linux Mint, and Ubuntu. The package manager for Debian-based distributions is called APT, the advanced packaging tool. APT is broken up into a few small commands. The two most commonly used APT commands are **apt-cache** and **apt-get**.

apt-cache search search-pattern Search for search-pattern.

apt-get install [-y] package Install package. Use the **-y** option to automatically answer yes to **apt-get**'s questions.

apt-get remove package Remove or uninstall package, leaving behind configuration files.

apt-get purge package Remove or uninstall package, deleting configuration files.

apt-cache show package Display information about package.

To search for software, use the command **apt-cache search search-pattern**.

```
$ apt-cache search web browser | head
abrowser - Safe and easy web browser from Mozilla - transitional package
abrowser-branding - Safe and easy web browser from Mozilla - transitional
package
akregator - RSS/Atom feed aggregator
firefox - Safe and easy web browser from Mozilla
firefox-branding - Safe and easy web browser from Mozilla - transitional
package
firefox-dbg - Safe and easy web browser from Mozilla - debug symbols
firefox-dev - Safe and easy web browser from Mozilla - development files
firefox-gnome-support - Safe and easy web browser from Mozilla - GNOME
support
firefox-gnome-support-dbg - Safe and easy web browser from Mozilla -
transitional package
gimp-help-de - Documentation for the GIMP (German)
$
```

To install software, use the command **apt-get install package**. Installing software requires superuser privileges. Use **sudo** or switch to the root account with the **su** command before installing or removing software.

```
$ sudo apt-get install firefox
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libdbusmenu-gtk4 xul-ext-ubufox
Suggested packages:
  ttf-lyx
The following NEW packages will be installed:
  firefox libdbusmenu-gtk4 xul-ext-ubufox
0 upgraded, 3 newly installed, 0 to remove and 193 not upgraded.
Need to get 36.1 MB of archives.
After this operation, 82.4 MB of additional disk space will be used.
Do you want to continue [Y/n]? y
Get:1 http://archive.ubuntu.com/ubuntu/ precise-updates/main libdbusmenu-
gtk4 amd64 0.6.2-0ubuntu0.2 [31.2 kB]
Get:2 http://archive.ubuntu.com/ubuntu/ precise-updates/main firefox amd64
29.0+build1-0ubuntu0.12.04.2 [36.0 MB]
Get:3 http://archive.ubuntu.com/ubuntu/ precise-updates/main xul-ext-ubufox
all 2.7-0ubuntu0.12.04.1 [56.8 kB]
Fetched 36.1 MB in 23s (1,535 kB/s)
Selecting previously unselected package libdbusmenu-gtk4.
(Reading database ... 102882 files and directories currently installed.)
Unpacking libdbusmenu-gtk4 (from .../libdbusmenu-gtk4_0.6.2-
0ubuntu0.2_amd64.deb) ...
Selecting previously unselected package firefox.
Unpacking firefox (from .../firefox_29.0+build1-0ubuntu0.12.04.2_amd64.deb)
...
Selecting previously unselected package xul-ext-ubufox.
Unpacking xul-ext-ubufox (from .../xul-ext-ubufox_2.7-
0ubuntu0.12.04.1_all.deb) ...
Processing triggers for desktop-file-utils ...
Processing triggers for bamfdaemon ...
Rebuilding /usr/share/applications/bamf.index...
Processing triggers for gnome-menus ...
Processing triggers for man-db ...
Setting up libdbusmenu-gtk4 (0.6.2-0ubuntu0.2) ...
Setting up firefox (29.0+build1-0ubuntu0.12.04.2) ...
```

```
update-alternatives: using /usr/bin/firefox to provide /usr/bin/gnome-www-  
browser (gnome-www-browser) in auto mode.  
update-alternatives: using /usr/bin/firefox to provide /usr/bin/x-www-  
browser (x-www-browser) in auto mode.  
Please restart all running instances of firefox, or you will experience  
problems.  
Setting up xul-ext-ubufox (2.7-0ubuntu0.12.04.1) ...  
Processing triggers for libc-bin ...  
ldconfig deferred processing now taking place  
$
```

To uninstall a package, use the command **apt-get remove package**. Like installing software, removing software requires superuser privileges.

```
$ sudo apt-get remove firefox  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages will be REMOVED:  
  firefox  
0 upgraded, 0 newly installed, 1 to remove and 193 not upgraded.  
After this operation, 81.8 MB disk space will be freed.  
Do you want to continue [Y/n]? y  
(Reading database ... 103024 files and directories currently installed.)  
Removing firefox ...  
Processing triggers for man-db ...  
Processing triggers for desktop-file-utils ...  
Processing triggers for bamfdaemon ...  
Rebuilding /usr/share/applications/bamf.index...  
Processing triggers for gnome-menus ...  
$
```

Using the dpkg Command

In addition to using the APT utilities, you can also interact directly with the package manager by using the **dpkg** command.

dpkg -l List all the installed packages.

dpkg -S /path/to/file List the package that contains file.

dpkg -i package.deb Install a package from the file named package.deb.

dpkg -L package List all files that belong to package.

```

$ dpkg -l | head
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-
pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                Version                Description
+++-----
=====
ii accountsservice      0.6.15-2ubuntu9.6      query and manipulate user
account information
ii acpid                 1:2.0.10-1ubuntu3      Advanced Configuration and
Power Interface event daemon
ii adduser               3.113ubuntu2            add and remove users and
groups
ii adium-theme-ubuntu    0.3.2-0ubuntu1          Adium message style for
Ubuntu
ii alsa-base             1.0.25+dfsg-0ubuntu1.1 ALSA driver configuration
files
$ dpkg -S /usr/bin/sudo
sudo: /usr/bin/sudo
$ sudo dpkg -i spideroak_5.1.3_i386.deb
Selecting previously unselected package spideroak.
(Reading database ... 153942 files and directories currently installed.)
Unpacking spideroak (from spideroak_5.1.3_i386.deb) ...
Setting up spideroak (1:5.1.3) ...
Processing triggers for man-db ...
Processing triggers for desktop-file-utils ...
Processing triggers for bamfdaemon ...
Rebuilding /usr/share/applications/bamf.index...
Processing triggers for gnome-menus ...
$ dpkg -L sudo | head
/.
/etc
/etc/sudoers.d
/etc/sudoers.d/README
/etc/pam.d
/etc/pam.d/sudo
/etc/sudoers
/etc/init.d
/etc/init.d/sudo
/usr
$

```