

CHRISTOPHER NEGUS

UBUNTU[®] LINUX[®]

TOOLBOX

2nd
Edition

1000+
Commands
for Ubuntu and
Debian[®] Power Users

WILEY

Table of Contents

[Cover](#)

[Introduction](#)

[Ubuntu Takes Linux by Storm](#)

[Who Should Read This Book](#)

[What This Book Covers](#)

[How This Book Is Structured](#)

[What You Need to Use This Book](#)

[Conventions](#)

[Chapter 1: Starting with Ubuntu Linux](#)

[Ubuntu, Debian, and Linux](#)

[Finding Commands](#)

[Reference Information in Ubuntu](#)

[Summary](#)

[Chapter 2: Installing Ubuntu and Adding Software](#)

[Obtaining and Installing Ubuntu](#)

[Working with Debian Software Packages](#)

[Adding Software Collections with taskel](#)

[Managing Software with APT](#)

[Finding Packages with APT](#)

[Managing Software with dpkg](#)

[Managing Software with aptitude](#)

[Verifying Installed Packages with debsums](#)

[Building deb Packages](#)

[Summary](#)

[Chapter 3: Using the Shell](#)

[Terminal Windows and Shell Access](#)

[Using the Shell](#)

[Acquiring Super User Power](#)

[Using Environment Variables](#)

[Creating Simple Shell Scripts](#)

[Summary](#)

[Chapter 4: Working with Files](#)

[Understanding File Types](#)

[Setting File/Directory Permissions](#)

[Traversing the Filesystem](#)

[Copying Files](#)

[Changing File Attributes](#)

[Searching for Files](#)

[Finding Out More about Files](#)

[Summary](#)

[Chapter 5: Manipulating Text](#)

[Matching Text with Regular Expressions](#)

[Editing Text Files](#)

[Listing, Sorting, and Changing Text](#)

[Summary](#)

[Chapter 6: Playing with Multimedia](#)

[Working with Audio](#)

[Transforming Images](#)

[Playing with Video](#)

[Summary](#)

[Chapter 7: Administering Filesystems](#)

[Understanding Filesystem Basics](#)

[Creating and Managing Filesystems](#)

[Mounting and Unmounting Filesystems](#)

[Checking Filesystems](#)

[Creating Encrypted Filesystems](#)

[Checking RAID Disks](#)

[Finding Out about Filesystem Use](#)

[Summary](#)

[Chapter 8: Backups and Removable Media](#)

[Backing Up Data to Compressed Archives](#)

[Backing Up over Networks](#)

[Summary](#)

[Chapter 9: Checking and Managing Running Processes](#)

[Listing Active Processes](#)

[Finding and Controlling Processes](#)

[Summary](#)

[Chapter 10: Managing the System](#)

[Monitoring Resources](#)

[Mastering Time](#)

[Managing the Boot Process](#)

[Controlling Startup and Run Levels](#)

[Straight to the Kernel](#)

[Poking at the Hardware](#)

[Summary](#)

[Chapter 11: Managing Network Connections](#)

[Configuring Networks from the GUI](#)

[Managing Network Interface Cards](#)

[Managing Network Connections](#)

[Using Wireless Connections](#)

[Checking Name Resolution](#)

[Troubleshooting Network Problems](#)

[Summary](#)

[Chapter 12: Accessing Network Resources](#)

[Running Commands to Browse the Web](#)

[Transferring Files](#)

[Sharing Remote Directories](#)

[Chatting with Friends in IRC](#)

[Using Text-Based E-mail Clients](#)

[Summary](#)

[Chapter 13: Doing Remote System Administration](#)

[Doing Remote Login and Tunneling with SSH](#)

[Using Legacy Communications Tools](#)

[Using byobu and screen for Remote Shells](#)

[Using a Remote Windows Desktop](#)

[Using Remote Linux Desktop and Applications](#)

[Sharing Desktops Using VNC](#)

[Summary](#)

[Chapter 14: Locking Down Security](#)

[Working with Users and Groups](#)

[Checking on Users](#)

[Configuring the Built-In Firewall](#)

[Using Advanced Security Features](#)

[Summary](#)

[Chapter 15: Setting Up a Virtualization Host and Virtual Machines](#)

[Can Your Computer Support Virtualization?](#)

[Managing Virtual Machines with virt-manager](#)

[Managing Virtual Machines with Commands](#)

[Summary](#)

[Appendix A: Using vi or Vim Editors](#)

[Starting and Quitting the vi Editor](#)

[Moving Around in vi](#)

[Changing and Deleting Text in vi](#)

[Using Miscellaneous Commands](#)

[Modifying Commands with Numbers](#)

[Using ex Commands](#)

[Working in Visual Mode](#)

[Appendix B: Shell Special Characters and Variables](#)

[Using Special Shell Characters](#)

[Using Shell Variables](#)

[Appendix C: Getting Information from /proc](#)

[Viewing /proc Information](#)

[Changing /proc Information](#)

Introduction

The huge, enthusiastic Ubuntu community has swept up thousands and thousands of new Ubuntu Linux users. If you are one of them, you will probably soon find yourself wanting to dig beneath the surface of Ubuntu's applications and graphical tools. You'll want to become a power user.

Becoming a power user with any Linux system means being able to work from the command line. Few graphical interfaces will provide you with the options and flexibility you get with commands that address the same features.

Ubuntu Linux Toolbox provides you with more than 1000 specific command lines to help you dig deeply into Linux. Whether you are a systems administrator or desktop user, the book will show you the commands to create file systems, troubleshoot networks, lock down security, and dig out almost anything you care to know about your Linux system.

This book's focus for your Linux command-line journey is Ubuntu, the community-based Linux distribution sponsored by Canonical Ltd., and the Debian GNU/Linux system on which it is based. Tapping into the skills required to run those systems can help you to work with your own Linux systems as well as learn what you need to do as a Linux professional.

Ubuntu Takes Linux by Storm

Since its inaugural release in 2004, Ubuntu (www.ubuntu.com) has become the most popular and, arguably, best loved of the Linux distributions. From its name, which translates to humanity toward others, to its focus on support for many languages and special needs, Ubuntu has reflected its ideals of spreading free software beyond the standard Linux target markets of geeks and corporate servers.

The Ubuntu project does everything it can to help ease new users into using its Linux-based Ubuntu operating system. Ubuntu live CDs let a new user try out Ubuntu before installing it. If the user likes Ubuntu, a single click can start an Ubuntu install to hard disk. And because Ubuntu is based on Debian GNU/Linux, Ubuntu has made massive amounts of software from the Debian software repositories available free to Ubuntu users.

Although it's true that Ubuntu focuses on ease-of-use desktop systems, that doesn't mean Ubuntu has no commercial Linux value. In fact, Canonical offers paid enterprise-quality support through its Landscape initiative (www.ubuntu.com/management). Canonical also offers a range of free and paid support options for individuals and small

businesses (www.ubuntu.com/support). In other words, there are professional opportunities for those who learn to operate Ubuntu.

Who Should Read This Book

This book is for anyone who wants to access the power of a Linux system as a systems administrator or user. You may be a Linux enthusiast, a Linux professional, or possibly a computer professional who is increasingly finding the Windows systems in your data center supplanted by Linux boxes.

The bottom line is that you want to find quick and efficient ways of getting Ubuntu and other Debian-based systems working at peak performance. Those systems may be a few desktop systems at work, a file and print server at your school, or a home web server that you're doing just for fun.

In the best case, you should already have some experience with Linux. However, if you are a computer professional with skills managing other types of operating systems, such as Windows, you can easily adapt your knowledge to use the specific commands I cover in the book.

What This Book Covers

This is not a beginner's Linux book. Before you jump in, it would be best if you have a basic working knowledge of what Linux is, how the shell works, and what processes, filesystems, and network interfaces are. The book will then supplement that knowledge with information you need to do the following activities:

- **Get software**—Ubuntu offers the Ubuntu Software Center GUI tool for getting software. With tools such as `apt-get`, you'll learn the best ways to search for, download, install, update, and otherwise manage software from the command line.
- **Use the shell**—Find neat techniques and tips for using the shell.
- **Play with multimedia**—Play and stream multimedia content from your computer. You can also modify audio and image files, and then convert the content of those files to different formats. For video, you can play a variety of video formats, including commercial movies.
- **Work with files**—Use, manipulate, convert, and secure a wide range of file types in Linux.
- **Administer file systems**—Access, format, partition, and monitor your file storage hardware (hard disks, CD/DVD drives, floppy disks, USB flash drives, and so on). Then create, format, and check the file systems that exist on those hardware devices. You can even create encrypted file systems to protect your data.
- **Back up and restore data**—Use simple commands to gather, archive, and compress your files into efficient backup archives. Then store those archives locally or on remote computers.
- **Work with processes**—List running processes in a variety of ways, such as by CPU use, processor use, or process ID. Then change running processes to have them run in the background or foreground. Send signals to processes to have them re-read configuration files, stop and resume processing, or stop completely (abort).
- **Manage the system**—Run commands to check system resources, such as memory usage, runlevels, boot loaders, and kernel modules.
- **Monitor networks**—Bring wired and wireless network connections up and down. Check routing, DNS, and host information. Keep an eye on network traffic.
- **Get network resources**—Connect to Linux and Windows remote file systems using FTP, NFS, and Samba facilities. Use shell-based commands to browse the web.
- **Do remote administration**—Access and administer other computers using remote login (`ssh`, `telnet`, and so on), and `screen`. Learn about remote administration

interfaces, such as Webmin, SWAT, and CUPS.

- **Lock down security**—Set up firewalls and system logging to secure your Linux systems.
- **Set up a virtualization host**—Configure your Ubuntu system as a host KVM system, then install and manage virtual machines on that host.
- **Get reference information**—Use the appendixes at the end of this book to get more information about the shell (such as metacharacters and shell variables) and the state of the system (from `/proc`).

Hopefully, if I have done it right, it will be easier to use this book than to Google for the command lines or GUI tools you need.

After you have mastered many of the features described in this book, you'll have gained the following advantages:

- **Hundreds of commands**—By compressing a lot of information into a small space, you will have access to hundreds of useful commands, in over 1000 command lines, in a handy form to carry with you.
- **Critical Linux information**—This book lists connections to the most critical information on the web for succeeding with Linux in general and Ubuntu in particular.
- **Transferable knowledge**—Most of the same commands and options you use in Ubuntu will work exactly the same way on other Debian-based Linux systems. Different Linux distributions, on the other hand, offer different graphical administration tools. And even within a particular distribution, graphical tools change more often than commands do.
- **Quick problem solving**—By the time others have started up a desktop and launched a graphical administration tool, you will have already run a half dozen commands and solved the problem.
- **Enduring value**—Many of the commands described in this book were used in early Unix systems. So you are gaining tools that reflect the experience of thousands of computer experts for more than 40 years.

Because the full documentation for commands used in Linux consists of thousands of man pages, info text, and help messages, you will surely want to reach beyond the pages of this book from time to time. Luckily, Ubuntu and other Linux systems include helpful information installed on the system itself. Chapter 1 contains descriptions of how to access that information that is probably already installed on your Ubuntu system.

How This Book Is Structured

This book is neither a pure reference book (with alphabetically listed components) nor a guide (with step-by-step procedures for doing tasks). Instead, the book is organized by topics and aimed at including as many useful commands and options as I could fit.

Chapter 1 starts by giving you a basic understanding of what Ubuntu is and how it relates to other Linux systems, such as various Debian-based distributions. Then it describes some of the vast resources available to support your experience with this book (such as man pages, info material, and help text). Chapter 2 provides a quick overview of installation and then describes useful commands such as `apt-get` for getting and managing your Ubuntu software.

Commands that a regular user may find useful in Linux are described in Chapters 3, 4, 5, and 6. Chapter 3 describes tools for using the shell, Chapter 4 covers commands for working with files, and Chapter 5 describes how to manipulate text. Chapter 6 tells how to work with music, image, and video files.

Starting with Chapter 7, you get into topics relating to system administration. Creating and checking filesystems are covered in Chapter 7, while commands for doing data backups are described in Chapter 8. Chapter 9 describes how to manipulate running processes, and Chapter 10 describes administrative tools for managing basic components, such as hardware modules, CPU use, and memory use.

Chapter 11 is devoted to managing network resources by describing how to set up and work with wired and wireless network interfaces. Chapter 12 covers text-based commands for web browsing, file transfer, file sharing, chats, and e-mail. Tools for doing remote system administration are included in Chapter 13.

Chapter 14 tells you how to lock down security using features such as firewalls and logging. Chapter 15 describes how you can configure Ubuntu as a KVM virtualization host, then use GUI and command-line tools to install virtual machines and manage them from your Ubuntu KVM host.

After that, three appendixes provide reference information for text editing, shell features (metacharacters and variables), and system settings (from the `/proc` file system).

What You Need to Use This Book

Although I hope you enjoy the beauty of my prose, this is not meant to be a book you curl

up with in front of a nice fire with a glass of wine. I expect you will be sitting in front of a computer screen trying to connect to a network, fix a file system, or add a user. The wine is optional.

In other words, the book is meant to be a companion as you work on an Ubuntu or Debian operating system. If you don't already have an Ubuntu or Debian system installed, refer to Chapter 2 for information on getting and installing those systems.

All the commands in this book have been tested against Ubuntu on x86 or x86_64 architecture. However, because many of these commands have been around for a long time (some dating back over 30 years to the original UNIX days), most will work exactly as described here on Debian systems, regardless of CPU architecture.

Many of the commands described in this book will work on other Linux and Unix systems as well. Because this book focuses on Ubuntu, descriptions will differ from other Linux systems most prominently in the areas of packaging, installation, the boot process, and GUI administration tools.

To give this book the longest useful life, I have focused on the latest Ubuntu Long Term Support (LTS) release: Ubuntu 12.04. Support for that release is offered until 2019.

Conventions

To help you get the most from the text and keep track of what's happening, I've used a number of conventions throughout the book. In particular, I've created styles for showing commands that allowed me to fit as many command lines as possible in the book.

With command examples, computer output (shell prompts and messages) is shown in regular monofont text, computer input (the stuff you type) is shown in bold monofont text, and a short description (if included) appears in italics. Here is an example:

```
$ ls *jpg           List all JPEG files in the current directory
hat.jpg
dog.jpg
...
```

To save space, output is sometimes truncated (or skipped altogether). Three dots (...) are used to indicate that additional output was cut. If a command is particularly long, backslashes will appear at the end of each line to indicate that input is continuing to the next line. Here is an example:

```
# oggenc NewSong.wav -o NewSong.ogg \
-a Bernstein -G Classical \
-d 06/15/1972 -t "Simple Song" \
```

```
-l "Bernsteins Mass" \
-c info="From Kennedy Center"
```

In the example just shown, you can literally type the backslashes to have all that information included in the single command. Or, you can simply put all the information on a single line, excluding the backslashes.

Although a regular user can run many commands in Ubuntu, to run some commands, the user must have root privilege. Because Ubuntu is installed without a root password, you are expected to use the `sudo` command from an Ubuntu user session to run administrative commands. Here's an example:

```
chris@host1:/tmp$ sudo useradd -m joe
```

For clarity, and to save space, I typically show a regular user prompt as simply a dollar sign (\$):

```
$           Indicates a regular user prompt
```

On occasion, you will also see a pound sign prompt (#), indicating that you probably need to run the command with root privilege. So, if you see a # prompt you can either type the `sudo` command in front of the command line or gain root privilege using one of the ways described in Chapter 3.

Notes and warnings appear as follows:

Note Warnings, notes, and tips are offset and placed in italic like this.

As for styles in the text:

- I italicize new terms and important words with italics when we introduce them.
- I show keyboard combinations like this: Ctrl+a. If the command requires you to type an uppercase letter, the combination will show this: Ctrl+Shift+a.
- I show file names, URLs, and code within the text like so:

```
persistence.properties.
```

One final technique I use is to highlight text that describes what an upcoming command is meant to do. For example, I may say something like, “use the following command to **display the contents of a file**.” Highlighting descriptions in this way is done to provide quick visual cues to the readers, so you can easily scan the page for that command you just knew had to be there.

Chapter 1

Starting with Ubuntu Linux

IN THIS CHAPTER

- Introducing Ubuntu Linux
- Finding Ubuntu resources
- Learning quick and powerful commands
- Referencing useful utilities
- Working as Linux gurus do

Whether you make extensive use of Ubuntu Linux at work every day, or just putter around with it once in a while, a book that presents efficient and comprehensive ways to maintain, monitor, secure, and enhance Ubuntu can be an invaluable resource.

Ubuntu Linux Toolbox, Second Edition is that resource.

Ubuntu Linux Toolbox, Second Edition is aimed primarily at power users and systems administrators. To give you what you need, I will show you how to quickly find and install software for Ubuntu, as well as how to update, maintain, and monitor the health and security of your system. In short, I will show you the most efficient ways of using Ubuntu by working with some of the powerful tools that are at your fingertips.

The goal of this book is to pack as much useful information as possible into a small package that you can carry around with you. To that end, I describe:

- **Commands**—Tons of command line examples demonstrate clever and useful ways to navigate the often daunting command line.
- **GUI tools**—Quick tips for using graphical interface tools to administer and configure your Ubuntu system.
- **Software repositories**—Methods for downloading and installing the software, which is custom-made for your Ubuntu system.
- **Online resources**—Where to find useful and helpful information about Ubuntu, such as mailing lists that you can subscribe to, IRC channels, and other online resources.
- **Local documentation**—Tools for working with the man pages, the standard Linux and UNIX reference volumes, as well as specific documentation for the software you install.

Because this book is for people already familiar with Linux, there won't be a lot of

screenshots of icons and menus. What you get instead is the quickest path to using your Ubuntu system to its fullest extent. Primarily, that means unlocking the mysteries of the command line to do things you can only dream about doing from the desktop.

What you learn in this book will help you become more adept at working with your Ubuntu or Debian system, as well as Linux in general. If this sounds useful to you, please read on.

Ubuntu, Debian, and Linux

Ubuntu is an operating system based on Debian GNU/Linux (www.debian.org). Debian has been around since the early 1990s, and because of its maturity, is regarded as a leading Linux distribution in terms of stability and security. Debian is also known for its strict adherence to free software (www.debian.org/intro/free). It is on this foundation that Ubuntu has been formed.

Debian has given rise to not only Ubuntu, but many other Linux distributions (www.debian.org/misc/children-distros). Some are derived directly from Debian, while others are Ubuntu derivatives:

- **Xubuntu**—An Xfce-based desktop system based on Ubuntu
- **Kubuntu**—A KDE-based desktop system based on Ubuntu
- **Edubuntu**—An Ubuntu derivative focused on schools
- **Linux Mint**—An easy-to-use desktop system with both Ubuntu and Debian roots
- **Knoppix**—A KDE desktop-oriented live CD based on Debian
- **Kanotix**—A Debian-based live CD
- **Damn Small Linux**—A tiny (50MB) live CD based on Knoppix
- **Mepis**—A desktop live CD based on Ubuntu and Debian

Xubuntu, Kubuntu, and Edubuntu are the same Debian-based Ubuntu distribution under the hood. The only difference in these is the default desktop they run, or the collection of applications bundled with them. For example, Kubuntu features the KDE desktop and Adept package manager, which are not installed on Ubuntu by default. Edubuntu is geared toward educational applications, many of which are not installed by default on the other Ubuntu distros.

Because Debian and Ubuntu are open source systems with most parts built on the GNU General Public License (www.gnu.org/copyleft/gpl.html), anyone is free to take the GPL-based source code, or any part of the GPL'd system, and modify, strip down, build upon, extend, embed, reverse-engineer, and freely distribute those changes or modifications. Generally, the only requirement is that you abide by the terms of the GPL, which basically states that any changes you make to existing GPL works must be made

available for others to utilize in the same way (see www.debian.org/social_contract for other licenses Debian recognizes).

In the end, you have not only a superior system with a free, online, worldwide support base, but a product that is constantly evolving and that is driven by people with a passion for what they do. Many other Linux distributions offer these same advantages; however, Ubuntu has certainly pulled out in front in terms of popularity among desktop and first-time Linux users.

Understanding Ubuntu Releases

Every six months or so, a new release of Ubuntu comes out. You can choose which release you want to use from the Ubuntu Releases page (<http://releases.ubuntu.com>).

Short release cycles allow Ubuntu to always offer the latest open source software available. The downside of short release cycles, however, is that many businesses prefer stability over the latest bells and whistles. Traditionally, that is why business applications are more often run on Red Hat Enterprise Linux, which has a much longer major release cycle.

To deal with that issue, Canonical began offering releases of Ubuntu that were specified as Long Term Support (LTS) releases. For LTS releases, Canonical makes extra efforts at stability and offers longer support cycles. See the Ubuntu Wiki LTS page (<https://wiki.ubuntu.com/LTS>) for a description of the support cycles offered with long term releases.

To have the longest possible useful life, this book is focused on the Ubuntu 12.04 LTS (Precise Pangolin) release. To use the same version of software used in this book, go to the Precise Pangolin download page (<http://releases.ubuntu.com/precise/>) and choose server install media that's appropriate for your computer (x86 or 64-bit).

If you don't have the exact same version of Ubuntu, don't worry. Most of the commands in this book are ones you can rely on to not change much over time. They will form a foundation for your command line use that will help you to get up-to-speed quicker on new features as they are released.

Note Ever wonder where Ubuntu (Edgy Eft) and Debian (Woody) get those odd naming conventions from? Find out at <https://wiki.ubuntu.com/DevelopmentCodeNames> or www.debian.org/doc/manuals/project-history/ch-releases.en.html.

Ubuntu Compared to Other Linux Distributions

If you log into the command line of both an Ubuntu system and a Red Hat Enterprise Linux or Fedora system, very little will look different. There are common directories and utilities between the two, and functionality is fundamentally the same. So what makes Ubuntu different from other Linux distributions? Consider the following:

- **Ubuntu Unity desktop**—Taking its own direction, Ubuntu features its own Unity desktop, rather than use GNOME, KDE, or other common Linux desktop interfaces. Though based on GNOME, Unity seeks to simplify the user interface to make it more useful on smaller screens, such as those used on netbook computers.

A major plan for Unity is to change the underlying display management system from the X Window system (used in most Linux and UNIX desktop systems) to the OpenGL-based Wayland project. With Wayland, Ubuntu seeks to improve the user experience with smoother graphics and effects. Popular X-based applications would then be run in compatibility mode.

- **Mobile and entertainment devices**—While Ubuntu has made some inroads into the enterprise computing arena, a more natural transition for Ubuntu from the desktop has been to specialty devices. The Canonical Group (www.canonical.com), which runs the Ubuntu project, announced an Ubuntu phone (www.ubuntu.com/devices/phone). There is also an Ubuntu TV project (www.ubuntu.com/devices/tv).
- **Simplified installation**—The complexity of booting and installing Ubuntu has been narrowed down to a handful of mouse clicks, making many of the install decisions automatic based on assumptions as to what the average user may need and want. A simpler installation process made it possible for people to leverage the stability of Debian packages without needing to make sophisticated decisions about disk partitioning and package selection.
- **Software management**—Another major difference among Linux distributions is in software management tools. The aim of the utilities and packaging systems is the same for Debian as for other Linux distributions; however, the operation and implementations are significantly different. Ubuntu and most other Debian-based systems use the APT (Advanced Package Tool) family of utilities for managing software. You use APT to install, remove, query, and update Debian (deb) packages. Red Hat uses an RPM packaging system to handle the same tasks with its RPM packages.
- **Cloud computing**—Canonical is making several different plays into the cloud. Instead of running Ubuntu on a local computer, you could create an Ubuntu instance in an Amazon cloud using CloudInit (<https://help.ubuntu.com/community/CloudInit>). As for setting up your own cloud infrastructure, like other Linux distributions, Canonical is supporting the OpenStack project (www.openstack.org).

- **Administration with `sudo`**—One unique characteristic of an Ubuntu system is the intentional practice of locking the root user account, and instead implementing the use of `sudo` (www.gratisoft.us/sudo/intro.html), which allows you to run a command with root permissions, for system administration tasks (see Chapter 3 for details on the `sudo` command).

The root login on a Linux system has privileges that allow unrestrained access to nearly every component of the system. It would be trivial to remove an entire filesystem as the root user, so Ubuntu tries to limit use of this account to only times when it is prudent. Most Linux distributions require the user to log in or `su` to root to perform administration tasks; however, a user on an Ubuntu system does this through `sudo` using his or her own login password, and not a separate one for the root user. Each `sudo` command run is also logged, making it easier to track who makes changes (rather than just knowing that someone with the root password did it).

Ubuntu has unique features that have their advantages and disadvantages, but they are far from limiting. Ubuntu has the tools in place to allow you to customize, modify, experiment, and hack to your heart's content if that is what you want to do. Otherwise, the idea is to be an easily maintainable, secure system with a clear and concise application set that is neither limiting nor overwhelming. This makes Ubuntu a very fluid system so you can jump right in and become familiar with it very quickly.

Finding Ubuntu Resources

The Ubuntu community has a vast pool of knowledge you can draw from in the form of online resources. The following is a list of links to some of the most popular and useful venues:

- <http://ubuntuforums.org>—In this searchable web forum and moderated social network is a diverse and talented community of Ubuntu users and support staff. Here, people share their success and setbacks with one another as well as offering assistance and guidance. Chances are good that if you're having difficulty with something in Ubuntu, someone has already run into the same problem and found a solution.
- www.ubuntu.com/support—This site offers paid support from Canonical Ltd., the company behind Ubuntu. If you don't want to spend time searching through the forums, or waiting for responses, Canonical Ltd. is one avenue for telephone, e-mail, and web support, costing around \$20 a month. Ubuntu training aimed at companies and corporate users is also available.
- <https://help.ubuntu.com>—This site contains the official, up-to-date, online documentation for each Ubuntu release. As newer Ubuntu releases come out, you

can come here to find out what's new.

- <http://screencasts.ubuntu.com>—View recorded desktop sessions on how to do different things with Ubuntu, from setting up a printer, to setting up Samba file sharing, to installing updates to keep your Ubuntu system in top shape. Ubuntu users are encouraged to join the Ubuntu Screencasts Launchpad Team (<https://launchpad.net/~ubuntu-screencasts>) to contribute.
- <https://lists.ubuntu.com/mailman/listinfo/ubuntu-users>—Join the Ubuntu-users mailing list and interact with Ubuntu users over e-mail to discuss and solve problems that come up with everything from implementing mysql databases to setting up problematic network devices. An archive of past threads can be viewed at <https://lists.ubuntu.com/archives/ubuntu-users>.
- <https://wiki.ubuntu.com/IRCResourcePage>—If you are interested in live IRC chat support, you can visit the Ubuntu IRC resource page to find guidelines, clients, and chat servers, which are an available source of support, free at any time. It is advisable to visit the Ubuntu Code of Conduct page (www.ubuntu.com/project/about-ubuntu/conduct) if you have not taken part in IRC chat before.

If you plan on buying hardware to use with your Ubuntu or other Linux system, these sites may be helpful in determining where to spend your money:

- www.linux-usb.org—This website aims to maintain a working knowledge of USB devices known to be Linux-friendly. There is a search utility where you can plug in the name or model of a manufacturer and get an instant status report on the usability of that device with Linux.
- www.linuxfoundation.org/collaborate/workgroups/openprinting—The CUPS (<http://cups.org>) printing system is the standard printing system used on most Linux systems these days. If your printer model is not listed when you attempt to add a new printer to your Ubuntu system, you may need to search this site for an updated PPD file to add to your CUPS system.
- www.sane-project.org—Scanner Access Now Easy (SANE) is a site devoted to the topic of document scanning on Linux. If you are looking for a scanner or multifunction printer, check here to see how well the vendors stack up in terms of Linux support.
- <http://tldp.org>—The Linux Documentation Project is a culmination of Guides, How-To articles, and FAQs covering everything from how to make coffee with Linux to setting up QoS and Traffic Control.

Certainly, this is not a complete list, but these are good places to look first. You can also try searching for Linux-related support on a hardware vendor's website prior to making your purchase. If they intend their hardware to work with Linux, they may have

drivers or instructions available. And don't forget the wealth of information you can find by searching for Linux on your favorite search engine.

Finally, look for a local Linux User's Group (LUG) in your area. A LUG is a local community of people keenly interested in Linux and its implementations. You will find people with a wide range of experience, from system administrators to casual Linux users, to distro maintainers, to CEOs of companies. LUGs generally meet on a regular basis for group discussions and hold presentations to demonstrate ways they've found to implement Linux and other related technology.

Some LUGs sponsor local events like install fests (http://en.wikipedia.org/wiki/Install_fest) or other Linux advocacy-type events. Chances are good that if you ask a question at a LUG meeting, someone (but more likely several) will have an answer. A search engine should help you locate a LUG in your area if you decide to pursue this. Most LUGs have websites or mailing lists that can be easily found online.

Ubuntu Software

Most Ubuntu software can be found on the Ubuntu package website (<http://packages.ubuntu.com>). The standard tools—Synaptic, APT, and Update Manager—are the most common ways of installing software on your Ubuntu system. (Chapter 2 provides details on finding and installing software.)

As a mature Linux operating system, over time Ubuntu has had many, many open source software packages ported to work on it. There are even software packages offered with Ubuntu that include non-free components (binary-only or software encumbered by software patents or other restrictions). The bottom line is that you should look in the official Ubuntu repositories first for the software you want before venturing out to third-party software repositories. That said, some day you may want to experiment and look for software that is not available in the Ubuntu packages. Most packages will have an MD5sum or GPG key you can use to verify that downloaded software hasn't been tampered with (www.debian-administration.org/articles/375). You can also run into compatibility issues with non-standard software, making upgrades a difficult task. The key to experimenting with non-standard software is to test it out in ways that do not alter your system. The following list includes some websites you can check out to see what other software is out there.

Warning You should be careful about how you go about mixing the software on your Ubuntu system with software from non-Ubuntu sources. Carefully check the authenticity of anything you download.

- <http://freecode.com/>—Boasts the web’s largest collection of UNIX and cross-platform software, themes, eye-candy, and Palm-OS software. It also sports a discussion board for each software entry to facilitate discussions and feedback. These guys have been around for a very long time, although formerly, the site was known as freshmeat.net.
- <http://sourceforge.net>—When open source developers get together to start a new project, many have their project hosted at SourceForge. SourceForge offers web space as well as tools for managing projects, resources, communications, and code. If you are looking for software, certainly try some searching here.

Focusing on Linux Commands

These days, many important tasks in Linux can be done from both graphical interfaces and from commands. However, the command line has always been, and still remains, the interface of choice for Linux power users.

Graphical user interfaces (GUIs) are meant to be intuitive. With some computer experience, you can probably figure out, for example, how to add a user, change the time and date, and configure a printer from a GUI. For situations such as these, I’ll mention which graphical tool you could use for the job. For the following situations, however, you will probably need to rely on the command line:

- **Almost any time something goes wrong**—Ask a question at an online forum to solve some Linux problem you are having, and the help you get will almost always come in the form of commands to run. Also, command line tools typically offer much more feedback if there is a problem configuring a device or accessing files and directories.
- **Remote systems administration**—If you are administering a remote server, you may not have graphical tools available. Although remote GUI access (using X applications or VNC) and web-based administration tools may be available, they usually run more slowly than what you can do from the command line.
- **Features not supported by GUI**—GUI administration tools tend to present the most basic ways of performing a task. More complex operations often require options that are only available from the command line.
- **Scripted tasks**—GUIs are nice if you want to add one user. But what if you want to add a hundred users or gather complex sets of performance data from your system. By running commands together in what are referred to as shell scripts, you can create complex and recursive tasks that you can then repeat later by simply running the script again.
- **GUI is broken or not installed**—If no graphical interface is available, or if the

installed GUI isn't working properly, you may be forced to work from the command line. Broken GUIs can happen for lots of reasons, such as when you use a third-party, binary-only driver from NVIDIA, and a kernel upgrade makes the driver incompatible.

The bottom line is that to unlock the full power of your Linux system, you must be able to use shell commands. Thousands of commands are available for Linux to monitor and manage every aspect of your Linux system.

But whether you are a Linux guru or novice, one challenge looms large. How do you remember the most critical commands and options you need, when a command shell might only show you this:

\$

Ubuntu Linux Toolbox, Second Edition is not just another command reference or rehash of man pages. Instead, this book presents commands in Ubuntu Linux by the way you use them. In other words, instead of listing commands alphabetically, I group commands for working with filesystems, connecting to networks, and managing processes in their own sections, so you can access commands by what you want to do, not only by how they're named.

Likewise, I won't just give you a listing of every option available for every command. Instead, I'll show you working examples of the most important and useful options to use with each command. Then, I show you quick ways to find more options, if you need them, from man pages, the info facility, and help options.

Finding Commands

Some of the commands in this book may not be installed by default on your Ubuntu distro, but will certainly be available through APT or other sources. If the bash shell cannot find a command that you've typed in Ubuntu, there are several reasons why that might happen:

- The command may not exist at all.
- You may have mistyped the command name (“fat-fingered” it).
- The command is not in any of the directories the shell has been instructed to look in (`PATH` variable).
- The command, or Ubuntu package containing the command, is simply not installed.

The command shell will respond in different ways, depending on the reason the command isn't found. If the command is not installed, is not in your `PATH`, and isn't available in any known software package, you see a simple `command not found`

message:

```
$ sillycommand  
sillycommand: command not found
```

If the command is one that is in a known package that is simply not installed, the shell will tell you what to type to install that package. For example, here's what happens when you try to run the kate editor when it isn't installed:

```
$ kate  
The program 'kate' is currently not installed. Install it by  
typing:  
sudo apt-get install kate
```

If the command you type is not found, but the name is close to one or more available commands, the shell tries to guess what you mean and suggest packages to install to get the command. Here is what you see if you type `cate` instead of `kate`:

```
$ cate  
No command 'cate' found, did you mean:  
Command 'kate' from package 'kate' (universe)  
Command 'cat' from package 'coreutils' (main)  
Command 'date' from package 'coreutils' (main)  
Command 'late' from package 'late' (universe)  
Command 'cfte' from package 'fte' (universe)  
Command 'cake' from package 'cakephp-scripts' (universe)  
Command 'yate' from package 'yate' (universe)  
Command 'catg' from package 'nauty' (multiverse)  
cate: command not found
```

If you find a command listed in the output of `apt-cachesearch` or suspect that the command you want is not installed, you can install it from the Internet by running the command:

```
$ sudo apt-get install packagename
```

where `packagename` is the name of the package you want to install.

The following list shows some shell commands you can run on any Linux distribution to check whether the command you typed is on the system.

Note You may see an ellipsis (...) used in code output to note where non-essential information has been omitted for the sake of brevity.

- Show the current **PATH**:

```
$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
```

- Find the first occurrence of the **mount** command in the **PATH**:

```
$ which mount
```



```
/bin/mount
```

- Search the `/usr` filesystem for a filename or directory named `umount`:

```
$ find /usr -name umount
/usr/lib/klibc/bin/umount
```

- Show where the first binary and man page are for the `mount` command:

```
$ whereis mount
mount: /bin/mount /usr/share/man/man8/mount.8.gz
```

- Use the `locate` command to search its list of (configurable) directories for `mount`:

```
$ locate mount
...
/usr/bin/fdmountd
```

- Search the man page descriptions for instances of a keyword—in this example, `umount`:

```
$ apropos umount
...
umount (8)          - unmount file systems
```

- View section 8 of the man page for `umount` (type `q` to quit):

```
$ man 8 umount
Reformatting umount(8), please wait...
```

The following list shows similar commands specific to Ubuntu and Debian systems:

- Search the cached list of packages that may contain a command or description of `umount`:

```
$ apt-cache search umount
gnome-mount - wrapper for (un)mounting and ejecting storage devices
--
```

- Search the list of installed packages for the filename `umount`, revealing the package it is in:

```
$ dpkg-query -W -f='${Package} ${Version} ${Architecture}\n'
...
initscripts: /etc/init.d/umountnfs.sh
--
```

- List all the files contained in the `initscripts` package:

```
$ dpkg-query -W -f='${Package} ${Version} ${Architecture}\n'
...
/bin/mountpoint
--
```

- Refresh the list of cached packages:

```
$ sudo apt-get update
Password:
Get:1 http://security.ubuntu.com feisty-security Release.gpg [191B]
...
--
```

Reference Information in Ubuntu

Original Linux and UNIX documentation was all done on manual pages, generally referred to as **man pages**. A slightly more sophisticated documentation effort came a bit later with the GNU `info` facility. Within each command itself, help messages are almost always available.

This reference information is component oriented. There are separate man pages for nearly every command installed on the system. Man pages also document devices, file formats, system, developer info, and many other components of a Linux system. Documentation more closely aligned to whole software packages is typically stored in a subdirectory of the `/usr/share/doc` directory.

Ubuntu compresses much of this documentation so it needs to be uncompressed before it can be read. You can use the `gzip` program to do this, but instruct `gzip` to only print the contents of the file and not decompress the files to disk. Here's the command to unzip the change log for the `mount` command and list its content to the screen:

```
$ gzip -dc /usr/share/doc/mount/changelog.Debian.gz | lessutil-  
linux \  
(2.20.1-5.1ubuntu2) quantal; urgency=low  
...
```

The man pages, info facility, and `/usr/share/doc` directories are all available on most Linux systems.

Using help Messages

Nearly all commands on a Linux system print some form of brief usage information if asked to. Frequently, the way to ask for this usage info is by way of the `-h` or `--help` argument to the command, and nothing more. The following command shows how to ask the `ls` command to print its usage information:

```
$ ls --help  
Usage: ls [OPTION]... [FILE]...  
List information about the FILES (the current directory by  
default).  
...
```

Because there is so much information printed by the `--help` flag, you can again use a pager to limit the output to one screen at a time:

```
$ ls --help | less  
...
```

Note If you have used UNIX systems, the `more` command is probably the first pager command that you used. Although the `more` command is available with Linux, Linux systems favor the newer `less` command, which is whimsically named and ironically more functional than the `more` command. It allows you to page backwards in the output as well as forwards, allows the use of the arrow keys to scroll, and understands vi editor keystrokes for navigating and searching through text.

The preceding examples show you how to output the `ls` command help to the screen. You can also format the help output with the use of the `card` command, which will print directly to the default printer, or can be saved to a Postscript file to be viewed later with something like the `evince` utility, or converted into a PDF file with the `ps2pdf` utility.

Using man Pages

You can use the `apropos` command to search the man page database for any keyword or group of characters. The output will show man page sections that contain the word you supply to `apropos`.

```
$ apropos crontab
/etc/anacrontab (5) [anacrontab] - configuration file for anacron
anacrontab (5)      - configuration file for anacron
crontab (1)         - maintain crontab files for individual users
(V3)
crontab (5)         - tables for driving cron
```

The `apropos` output here shows the section and man page where the word `crontab` was found. Sections of man pages group together man pages by topic. Man pages in section 1 are Executable programs or shell commands. Man pages in section 5 fall under the topic of file formats and conventions. The man page sections will be the same on all Linux systems, but may vary a bit on other UNIX-type systems. You should be able to view the man page for `man` to find out which sections are represented on the system you're on:

```
$ man man
Reformatting man(1), please wait...
...
```

The following list shows the section numbers of the manual followed by the types of pages they contain:

- **1**—Executable programs or shell commands
- **2**—System calls (functions provided by the kernel)

- **3**—Library calls (functions within program libraries)
- **4**—Special files (usually found in `/dev`)
- **5**—File formats and conventions such as `/etc/passwd`
- **6**—Games
- **7**—Miscellaneous (including macro packages and conventions), such as `man(7)`, `groff(7)`
- **8**—System administration commands (usually only for root)
- **9**—Kernel routines (Non standard)

Given this information, you can see the `crontab` word you searched for has an entry in section 1 (Executable programs or shell commands) as well as section 5 (File formats and conventions). You can view the man pages from those sections by passing the section number as an argument to the `man` command.

```
$ man 5 crontab
Reformatting crontab(5), please wait...
CRONTAB(5)                                CRONTAB(5)
```

```
NAME
    crontab - tables for driving cron
```

```
DESCRIPTION
    A crontab file contains instructions to the cron(8) daemon of
    the general form: ``run this command at this time on this date''.
    ...
```

If you omit the section number, `man` will return the man page from the first section it finds. In the next example, `man` returns section 1 of the `crontab` man pages:

```
$ man crontab
Reformatting crontab(1), please wait...
CRONTAB(1)                                CRONTAB(1)
```

```
NAME
    crontab - maintain crontab files for individual users (V3)
    ...
```

In addition to section numbers, the `man` command takes several arguments to perform different tasks. Here are some examples:

- Shows all man page sections, in succession, for `crontab`:

```
$ man -a crontab
```

- Shows the section 5 man page for `crontab`:

```
$ man 5 crontab
```

- Uses the pager program `more` for paging through the `crontab` man page:

```
$ man -P more crontab
```

- Equivalent to the `whatis` command:

```
$ man -f crontab
```

- Equivalent to the `apropos` command:

```
$ man -k crontab
```

The `whatis` command is another man page searching utility. It is different from `apropos` in that it only prints man page descriptions that match the keyword you type in. Running the `apropos` command for the `route` command returns three different man pages where a reference to the word `route` was found:

```
$ apropos route  
NETLINK_ROUTE (7)      - Linux IPv4 routing socket  
route (8)              - show / manipulate the IP routing table  
traceroute6 (8)        - traces path to a network host
```

In running `whatis` for the `route` command, only the section 8 man page for the `route` command is returned:

```
$ whatis route  
route (8)              - show / manipulate the IP routing table
```

Using info Documents

In some cases, developers have put more complete descriptions of commands, file formats, devices, or other Linux components in the `info` database, a sort of linked set of online manual pages. You can enter the `info` database by simply typing the `info` command or by opening a particular component (use the `q` key to quit the `info` utility).

```
$ info ls
```

The previous command shows information on the `ls` command. You can navigate around the `info` utility using the up, down, left, and right arrow keys, as well as the Page Up and Page Down keys. The following text shows more about navigating in `info`.

- **?**—Display the basic commands to use in `info` screens.
- **Shift+l**—Go back to the previous node you were viewing.
- **n, p, u**—Go to the node that is next, previous, or up, respectively.
- **Enter**—Go to the hyperlink that is under the cursor.
- **Shift+r**—Follow a cross reference.
- **q** or **Shift+q**—Quit and exit from `info`.

Software packages that have particularly extensive text available in the `info` database include `gimp`, `festival`, `libc`, `automake`, `zsh`, `sed`, `tar`, and `bash`. Files used by the `info` database are stored in the `/usr/share/info` directory.

Summary

In one short chapter, I've covered some of the differences and similarities of Ubuntu Linux as compared to other Linux distributions and other UNIX-like systems. You've found out about several online resources specifically for Ubuntu as well as those for Linux in general.

You learned where to find Ubuntu-specific software as well as other Linux software. You installed a few packages using the Debian Advanced Package Tool (APT) and worked with ways of searching for commands and man pages on the system. You also worked with the `stdin` and `stdout` I/O streams by redirecting command output (`stdout`) to temporary files as well as the input streams (`stdin`) of other commands.

While you certainly can read this book from cover-to-cover if you like, it was designed to be a reference to hundreds of features in Ubuntu and Debian Linux that are the most useful to power users and systems administrators. Because information is organized by topic, instead of alphabetically, you don't have to know the commands in advance to find what you need to get the job done.

Most of the features described in this book will work equally well in all Linux-based systems, and many will carry over to legacy UNIX systems as well.

Chapter 2

Installing Ubuntu and Adding Software

IN THIS CHAPTER

- Installing Ubuntu
- Working with software repositories
- Getting software with APT
- Managing software with Debian package tools
- Extracting files from other package formats

Time-tested tools for initially installing Ubuntu, and later adding and managing software, include the APT (Advanced Package Tool) and dpkg (Debian package) utilities. These are some of the standard packaging utilities that serve as a back end to the more familiar desktop GUI tools for managing software on Ubuntu and other Debian-based systems. These package utilities interact with `.deb` files from online repositories, or local `.deb` files you've downloaded and have sitting on your hard disk.

This chapter highlights critical issues you need to know during Ubuntu initial installation. It covers information about online Ubuntu software repositories. Detailed examples of APT, dpkg, tasksel, and related command line utilities including aptitude are given later in this chapter.

Obtaining and Installing Ubuntu

Ubuntu and its close cousins Kubuntu, Xubuntu, and Edubuntu are all designed with ease of use and familiarity in transition in mind. These distributions focus on keeping things simple and clean to help smooth out the learning curves when you are adapting to a new system.

The Ubuntu installer (Ubiquity) is a prelude to the simplicity of the Ubuntu system, breaking down the install process into about 10 clicks. With an Internet connection, you can download one of the ISO images for free from the Ubuntu download page (www.ubuntu.com/download). From the download page, you can choose if you want installation media for an Ubuntu desktop, server, or cloud system. Once you make your selection, you are offered the chance to download the latest version of Ubuntu or the

most recent version of Ubuntu available for Long Term Support (LTS). For example:

- **Ubuntu 12.10**—Released in October 2012. The server version of this release is supported for 18 months (so support expires in April 2014).
- **Ubuntu 12.04 (LTS)**—Released in April 2012. Because it is a Long Term Support release, it comes with five years of guaranteed support (so it is supported until April 2017).

There are 64-bit and 32-bit versions of Ubuntu available when you choose Ubuntu Desktop, Ubuntu Server, or Cloud infrastructure from the Ubuntu Download page. The 64-bit version will not work if you have a standard x86 (32-bit) computer. The 32-bit version should work on a 64-bit machine, but not as efficiently. So choose the version that is right for your computer.

As you gain experience with Ubuntu, you can look into other ways of getting an installed Ubuntu system. Some of these methods, which are available from the Download page, include:

- **Juju**—With Juju, a system administrator can capture a deployed Ubuntu system and store it in what is referred to as **charms**. Those charms can then be used to reproduce the Ubuntu system and deploy it in various cloud environments.
- **Cloud Guest**—To just try Ubuntu in the cloud, you can select Cloud Guest. This allows you to try Ubuntu with some preconfigured applications in the cloud for one hour.

If you are fairly new to Ubuntu, however, I recommend downloading the appropriate installation media and installing it on an extra computer you have available. To do that, just select the version that matches your computer and download it.

After your download is complete, you may want to visit the HowToMD5SUM page and get the md5sum file for the version of Ubuntu you downloaded (<https://help.ubuntu.com/community/HowToMD5SUM>). This can help verify the integrity of the ISO image. Most open source software will have such a digital signature available, and I recommend that you verify this prior to installation, or before burning the ISO image to CD-ROM or DVD.

Note If you desire more security for your downloads beyond the MD5 Checksums, look at SecureApt. For more information on how APT uses digital authentication and encryption for software packages, visit the SecureApt section on the Ubuntu help website (<https://help.ubuntu.com/community/SecureApt>).

Preparing to Install

If you are going to erase everything on your computer's hard disk and install Ubuntu, you don't have to make preparations for your install in advance. If you want to keep any data from your hard disk, back up that data before proceeding. To keep existing data on your hard disk and add Ubuntu, you may need to resize existing disk partitions and repartition your disk. See Chapter 7 for information on disk resizing and partitioning commands.

Choosing Installation Options

To test the commands in this book, I use the 64-bit server installation of Ubuntu 12.04 (LTS). You may use other installation media versions, as you like. There should be relatively few differences with the commands shown in this book, regardless of whether you install a server or desktop system or even use a different release.

Note If you do choose a server install, keep in mind that it provides a minimal system, with the choice of adding a few services. Although there is no desktop software installed, you'll have a chance to add desktop software later in this chapter with a variety of command line tools. If you prefer, you can simply start with a desktop install instead.

After booting from the Install CD, you are asked to choose the language to use. Then you are presented with a menu of options, as shown in the following list:

- **Install Ubuntu Server**—Begin the installation process immediately. (Choose this option.) MAAS stands for Metal as a Service. You could use this feature if you are setting up multiple servers and you want to deploy preconfigured systems. (Because you are installing a single server, don't choose this option.)
- **Check disc for defects**—Test the CD for problems, reading the CD to find problems now instead of in the middle of the install.
- **Test memory**—If you suspect there are problems with your RAM, Ubuntu allows you to run Memtest86 (www.memtest.org/) to stress test your RAM to look for errors.
- **Boot from first hard drive**—If you've accidentally booted with the CD-ROM in the drive, simply pick this menu item to boot from your first hard drive.
- **Rescue a broken system**—The installation CD can also be used as a rescue CD. Use this if your Ubuntu system is unbootable and you want to get at the contents of your hard disk to fix the problem. (This selection is not used for a new installation.)

You can find out more about installing Ubuntu from the Installation guide for the particular installation type and version you are using. For example, for the Ubuntu 12.04 LTS server install, go to: <https://help.ubuntu.com/12.04/serverguide/installation.html>.

Answering Installation Questions

Most of the screens you see during Ubuntu installation are quite intuitive. The following list offers a quick review of those screens, along with tips where you might need some help:

- **Install welcome**—Select your language.
- **Where are you?**—Select the country where you are located to help set your time zone. (Worldwide servers are sometimes set to GMT time.)
- **Keyboard layout**—You can press keys to have the installer detect the keyboard layout or select from a list of available keyboard layouts.
- **Enter the hostname**—Choose a single hostname to represent your system. For example, you would simply type `abc` if your full hostname was `abc.example.com`.
- **Set up users and passwords**—Add the full name for the user, a shorter username, and password. This user will have super-user privileges (the root account is not configured by default in Ubuntu). You are also asked if you want to encrypt your home directory. (This makes your data more secure from someone who steals your hard drive, but it makes it so that you must remember the password you used to encrypt the directory or make your home directory inaccessible to yourself as well.)
- **Configure your clock**—The installer attempts to contact a network time server to set your clock and suggests your time zone (based on your current location). You can choose the time zone it recommends or choose a different one. As mentioned, some server admins like to set time zones to GMT if you need to sync up servers that are used worldwide.
- **Prepare disk space**—Select Guided partitioning if you want Ubuntu to guess how to lay out the disk. Select Manual if you want to determine the partitions yourself. I recommend that you choose the guided setup with LVM. Also, if you don't use all the space available, you can use the LVM commands later in Chapter 7 to grow and shrink your LVM partitions.
- **Start the installation**—Once partitioning completes, the installation begins.

You may be prompted during the installation process to:

- Configure a proxy server, if it is needed to reach Ubuntu software repositories to get the software needed to complete the installation.
- Choose whether or not to get automatic software updates.
- Select which software to install. (I simply added OpenSSH so I could remotely log in to the server. I'll add others later.)
- Install the GRUB boot loader (install it to the Master Boot Record if you have no other operating systems installed).

While the installer is running, multiple terminal sessions are available via the Ctrl+Alt+F2 and Ctrl+Alt+F3 key combination if you just need a shell prompt. Also, as the install progresses, Ctrl+Alt+F4 will show any messages or errors encountered during this process. Use Ctrl+Alt+F1 to return to the installation screen.

Working with Debian Software Packages

If you prefer to use a GUI tool for installing software, the SynapticPackage Manager is available from the desktop or an ssh session using the `-Xssh` parameter to tunnel X11 (see Chapter 13). The `aptitude` utility provides a nice curses (text-based) front end to APT when run with no arguments. A front end for the `dpkg` utility, `dselect`, is also available on most Debian systems, but can be difficult to learn to operate.

Ubuntu uses the Debian package format (an `ar` archive, actually), which is a standard method for packaging software for Debian-based systems. By gathering software components in separate Debian packages (`.deb` extension) the software can not only carry a self-contained archive of the software, it can also hold lots of information about the contents of the package. This metadata can include software descriptions, dependencies, computer architecture, vendor, size, licensing, and other information.

When a basic Ubuntu system is installed, you can add, remove, and otherwise manage `.deb` files to suit how you use that system. Ubuntu, Kubuntu, Xubuntu, Edubuntu, and most other Debian-based systems will use `.deb` files to install the bulk of the software on the system. The `aptitude` tool should work very well for most day-to-day software needs; however, many other tools for managing these packages exist, and you may need to use some of them occasionally:

- **APT**—Use APT to download and install packages from online repositories. The APT commands (`apt-get`, `apt-cache`, and so on) can be used to install packages locally. However, it's normally used for working with online software.
- **dpkg**—Use `dpkg` to work with `.deb` files from CD-ROM or other disk storage. The `dpkg` command has options for configuring, installing, and obtaining information on system software.
- **aptitude**—Use `aptitude` at the command line for working with online repositories. The `aptitude` tool is recommended as the first choice because it will automatically take care of some of the tasks you must do manually when working with `dpkg` or APT.

This chapter includes sections on each of these utilities, outlining the most appropriate

circumstances for using each tool.

Note For more information on these package tools, consult the man pages for APT and dpkg.

Ubuntu (and the other *buntu offerings) are installed from a single CD-ROM or DVD. After installing, you can run the `apt-cache stats` command to report on the total number of packages available:

```
$ apt-cache stats
Total package names : 50267 (1,005 k)
  Total package structures: 85142 (4,768 k)
  Normal packages: 58468
...
```

As you can see, from a barebones stock Ubuntu install, over 50,000 pieces of software are available. The Debian community is very careful to include only software that is appropriate for redistribution.

Note The Debian Tutorial at www.debian.org/doc/manuals/debian-tutorial/ch-introduction.html points out:

“Although Debian believes in free software, there are cases where people want or need to put proprietary software on their machine. Whenever possible, Debian will support this; although proprietary software is not included in the main distribution, it is sometimes available on the FTP site in the non-free directory, and there are a growing number of packages whose sole job is to install proprietary software we are not allowed to distribute ourselves.”

The Canonical group holds Ubuntu to similar standards (www.ubuntu.com/community/ubuntustory/licensing), offering software in four categories: main, restricted, universe, and multiverse:

- **main**—Contains software that is freely distributable and supported by the Ubuntu team. Much of this software is installed when you install Ubuntu.
- **restricted**—Contains software that is common to many Linux systems, supported by the Ubuntu team, but may not be under a completely free license.
- **universe**—Contains a snapshot of nearly every piece of open source software available in the Linux world and available under licenses that may not be as free as the others. Software in this component is not guaranteed for security fixes or support.
- **multiverse**—Contains software that does not meet the free concept of software as it applies to the Ubuntu main component license policy. Software in this component is

not supported in any way and it's up to you to determine licensing validity.

Working with Software Packages

The following sections describe the basics of package management, explaining what goes on behind the scenes and how to install packages. Learning this is a necessary first step prior to moving on to other tools such as `aptitude`.

The `dpkg` command is very powerful for installing single `deb` packages, but will not sift through and install dependencies that are needed by different pieces of software, nor does it care about software repositories, such as the Ubuntu components mentioned previously. APT, on the other hand, will resolve and install dependencies and consult the configured repositories, but it is not used to install `.deb` files located on a hard drive or other local disk.

Several other Linux distributions also use packaging systems similar to APT. Red Hat-based/derived distributions (including Red Hat Enterprise Linux, CentOS, Fedora, and Mandriva) have tools such as `yum`, `rpm`, `urpmi`, and `smart` to manage software. Although these tools are quite different from the ones Ubuntu uses, the ideas are similar; a configuration file is set up to tell the packaging tool where online to find the latest software packages. The packaging tool then works in conjunction with an installer to get the software on the system.

This system of having an online package fetcher (so to speak) and a back-end packaging tool is a very powerful combination to resolve dependency issues, digitally authenticate software integrity, easily keep a system up-to-date, and allow distribution maintainers to distribute changes simply and on a large scale.

Handling Locale Error Messages

If you are working at the command line with Ubuntu, you may see a locale error message like one of these while trying to install packages:

```
perl: warning: Setting locale failed.  
perl: warning: Please check that your locale settings:  
locale: Cannot set LC_CTYPE to default locale: No such  
file/directory
```

This seems to be a problem related to the installed language settings, or something with internationalized encoding in general. One workaround you can use to keep things satisfied is to export the `LC_ALL` environment variable and set it the same as your `LANG` setting.

```
$ export LC_ALL="$LANG"
```

There are several other possible workarounds on the help sites, but this one will be the easiest to undo in case the cure causes more problems than the condition. It should also work regardless of what language you speak. Note that you will have to run this command every time you open a local or ssh shell. You can automate this task by placing the command in your `~/.bashrc` file.

Enabling More Repositories for apt

In previous releases of Ubuntu, the multiverse, and universe repositories were not enabled by default. These repositories now come enabled by default with Ubuntu, so doing updates and searching for software will turn up many more options. One concern you may have, however, is that support, licensing, and patches may not be available for the universe and multiverse repositories. This could be a problem if you are considering an installation where you need to adhere to certain policies and procedures.

To disable the universe or multiverse repositories, open the file `/etc/apt/sources.list` in a text editor and comment out the lines that have multiverse or universe components enabled. You may want to initial the comments to make note of what you commented out, as shown by the `#cn` in the following examples:

```
#cn deb http://us.archive.ubuntu.com/ubuntu/ precise universe  
#cn deb-src http://us.archive.ubuntu.com/ubuntu/ precise universe  
#cn deb http://us.archive.ubuntu.com/ubuntu/ precise-updates  
universe  
#cn deb-src http://us.archive.ubuntu.com/ubuntu/ precise-updates  
universe  
#cn deb http://us.archive.ubuntu.com/ubuntu/ precise multiverse
```

```
#cn deb-src http://us.archive.ubuntu.com/ubuntu/ precise multiverse
#cn deb http://us.archive.ubuntu.com/ubuntu/ precise-updates
multiverse
#cn deb-src http://us.archive.ubuntu.com/ubuntu/ precise-updates
multiverse
```

Another reason to edit the `/etc/apt/sources.list` file is to add extra repositories that may be offered by individuals or companies. To edit this file, you must have root permissions:

```
$ sudo vi /etc/apt/sources.list
```

Insert a line starting with `deb` (for pre-built packages) or `deb-src` (for source packages), then the URL for the repository, along with the distribution (such as `precise` above), and the component descriptions (`universe` in the examples). Typically, you'll describe components as `contrib` for contributed (that is, not from the Ubuntu project) and `free` or `non-free`. Normally, you should receive all this information from the site that offers the repository.

If you do add other third-party repositories, be sure to look into the authenticity of the entity offering the software before modifying your Linux system. Although it's not a big problem with Linux these days, it is easy to add broken or malicious software to your system if you do not exercise care and reasonable caution.

Only use software from well-known sources, and always have a means to verify software you download prior to installing.

Adding Software Collections with `tasksel`

If you start with a minimal system install, as was illustrated in the “Obtaining and Installing Ubuntu” section of this chapter, you may find it convenient to add groups of packages to get an entire feature installed. This saves you from hunting around to get all the packages that are needed for a particular feature.

Note If you choose a software collection with `tasksel`, a large number of packages will be downloaded and installed. For example, when I installed the Ubuntu desktop with `tasksel`, more than 900 packages were downloaded and installed. The process took more than 30 minutes with a decent Internet connection. So, make sure you mean it before you select a package collection to install.

To run `tasksel`, simply type:

```
$ sudo tasksel
```

A menu of available software collections appears. If you don't have a desktop interface installed on your system, you could choose from several different desktop interfaces. Use the arrow key to move to the collection you want, press the spacebar to select it (which adds an asterisk), press Tab to highlight the OK button, and press Enter to begin installing.

Here is an example of a basic Ubuntu desktop being selected:

```
[ ] Ubuntu LXDE Desktop
[*] Ubuntu desktop
[ ] Ubuntu desktop USB
```

After packages are downloaded and installed, you might be prompted to do additional configuration. In some cases, you might need to reboot your computer for the feature to take effect.

Managing Software with APT

Although features in `dpkg` and APT can work to complement each other, most of the time APT will suffice for any software you need to install, download, upgrade, check, or search for on any Debian-based system. When I refer to APT, I'm talking about a set of software management commands that include `apt-get`, `apt-cache`, `apt-key`, and others.

For a quick command line reference of APT capabilities, use the `-h` option on the command line with any APT command that interests you. Some APT commands require super-user privileges, so they must be preceded by the `sudo` command to function properly. The following gives some common uses of APT commands.

Note The `aptitude` utility is preferred over APT; however, in the interest of fundamentals, I will cover APT first.

- This command consults `/etc/apt/sources.list` and updates the database of available packages. Be sure to run this command whenever `sources.list` is changed:

```
$ sudo apt-get update
```

- This command does a case-insensitive search of the package database for the keyword given. The package names and descriptions are returned where that keyword is found:

```
$ apt-cache search keyword
```

- Download and install the given package name as found in the package database using the following command. Starting with APT version 0.6, this command will

automatically verify package authenticity for GPG keys it knows about (<http://wiki.debian.org/SecureApt>):

```
$ sudo apt-get install package
```

- Download the package only, placing it in `/var/cache/apt/archives`, using the following command:

```
$ sudo apt-get -d install package
```

- Display information about the software from the named package with this command:

```
$ apt-cache show package
```

- Check updates for all installed packages and then prompt to download and install them with this command:

```
$ sudo apt-get upgrade
```

- This command updates the entire system to a new release, even if it means removing packages. (This is not the preferred method for updating a system.)

```
$ sudo apt-get dist-upgrade
```

- Run this command anytime to delete partially downloaded packages, or packages no longer installed:

```
$ sudo apt-get autoclean
```

- Remove all cached packages from `/var/cache/apt/archives` to free up disk space using this command:

```
$ sudo apt-get clean
```

- This command removes the named package and all its configuration files. Remove the `--purge` keyword to keep config files:

```
$ sudo apt-get --purge remove package
```

- This command does a sanity check for broken packages. This tries to fix any “unmet dependency” messages:

```
$ sudo apt-get -f install
```

- Print version information of installed APT utilities with this command:

```
$ apt-config -V
```

- This command lists GPG keys that APT knows about:

```
$ sudo apt-key list
```

- Print statistics on all packages installed with this command:

```
$ apt-cache stats
```

- Use this command to print dependencies for a package (whether it’s installed or not):

```
$ apt-cache depends
```

- List all packages installed on the system with this command:

```
$ apt-cache pkgnames
```

Finding Packages with APT

To find packages in any available repository, you can **query for new software** using the `apt-cache` command. For example, to search for the `bzflag` tank game, you could type the following:

```
$ apt-cache search bzflag
bzflag - a 3D first person tank battle game
bzflag-client - BZFlag client
bzflag-data - BZFlag data file
bzflag-server - bzfs - BZFlag game server
```

You can also **ask APT to show info** about the `bzflag` package:

```
$ apt-cache show bzflag
Package: bzflag
Priority: optional
Section: universe/games
Installed-Size: 212
Maintainer: Ubuntu Developers <ubuntu-devel-
discuss@lists.ubuntu.com>
Original-Maintainer: Tim Riker <Tim@Rikers.org>
...
```

Just how much extra software will `bzflag` require to be updated? **Check for dependencies** with the following:

```
$ apt-cache depends bzflag
Depends: bzflag-client
Depends: bzflag-server
...
```

Installing Packages with APT

Using `sudo` with APT or another package tool, you can **install software available from any software repository** you have enabled for Ubuntu. This shows `bzflag` being installed using APT:

```
$ sudo apt-get install bzflag
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

The following extra packages will be installed:

bzflag-client bzflag-data bzflag-serverThe following NEW packages will be installed:

bzflag bzflag-client bzflag-data bzflag-server

0 upgraded, 4 newly installed, 0 to remove and 61 not upgraded.

Need to get 12.0 MB of archives.

After this operation, 19.4 MB of additional disk space will be used.

Do you want to continue [Y/n]? Y

...

Once the installation is done, provided you have a desktop installed, you can run the bzflag program by typing **bzflag** on the command line, or selecting it from the Games menu on the Ubuntu desktop.

Upgrading Packages with APT

Over time, packages change and new versions add neat new features and fix problems. You can use APT to upgrade your system to new versions following a two-step process.

First, **check for updates** to the packages your Ubuntu system knows about using the `update` option to `apt-get`:

```
$ sudo apt-get update
```

This command searches the repositories for new versions of packages available for downloading, and updates the list of packages and versions cached on your Ubuntu system.

Second, upgrade the packages on your Ubuntu system by using the `upgrade` option to `apt-get`:

```
$ sudo apt-get upgrade
```

You should always update the package list prior to upgrading packages, so it is a good idea to always run these commands together. Separate the commands with a semicolon to tell the shell to run them both, one after the other:

```
$ sudo apt-get update; sudo apt-get upgrade
```

```
Get:1 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main  
base-files amd64 6.5ubuntu6.4 [69.9 kB]
```

```
Get:2 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main  
dpkg
```

```
amd64 1.16.1.2ubuntu7.1 [1,830 kB]
```

...

The following packages will be upgraded:

accountsservice apparmor apt apt-transport-https apt-utils
aptitude base-files

bind9-host busybox-initramfs busybox-static coreutils dbus

```
dnsutils
dpkg gnupg gpgv
...
59 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
Need to get 43.9 MB of archives.
After this operation, 569 kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
```

Upgrading a Single Package with APT

Upgrading a single system package on Ubuntu is pretty straightforward using `apt-get install <package>`. You need to run this command with `sudo` to gain root permissions. The old version is automatically updated to the newest one available.

Note It may seem counterintuitive, but the `upgrade` option to `apt-get` upgrades all packages. The `install` option installs a new package or installs an upgrade to one or more specific packages.

First, **check the version of the currently installed application**. In this example, I assume that an earlier version of the `minicom` application, an application for communicating over serial lines, is installed on your system. Like most commands, the `minicom` program supports an option to display its current version number:

```
$ minicom --version
minicom version 2.2 (compiled Mar  7 2007)
...
```

Now use APT to install the latest version of the `minicom` package from the repositories. APT tells you that you are upgrading a package:

```
$ sudo apt-get install minicom
...
The following packages will be upgraded:
  minicom
```

Now ask `minicom` for its version again and you can see it has indeed been upgraded:

```
$ minicom --version
minicom version 2.5 (compiled May  2 2011)
...
```

Removing Packages with APT

You can **remove a package** from your Ubuntu system by giving `apt-get` the `remove` option. You will be prompted to confirm before actually removing the software:

```
$ sudo apt-get remove bzflag
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
...
The following packages will be REMOVED:
  bzflag
0 upgraded, 0 newly installed, 1 to remove and 2 not upgraded.
After this operation, 217 kB disk space will be freed.
Do you want to continue [Y/n]? n
```

Cleaning Up Packages with APT

After your initial installation of an Ubuntu release, Ubuntu keeps downloaded packages cached in `/var/cache/apt/` to speed up downloading if you ever need them again. Over time, this can use up a lot of disk space. You can remove this package cache, but you will need to download a removed package again at a later date if it is needed for dependencies. Clean up the cache by giving `apt-get` the `clean` option. I show how this works by first running the Linux `find` command on the `/var/cache/apt/` directory to show the packages currently cached:

```
$ find /var/cache/apt/ -name \*.deb
/var/cache/apt/archives/nautilus_1%3a3.4.2-0ubuntu6_amd64.deb
/var/cache/apt/archives/gwibber-service-identica_3.4-
0ubuntu2.1_all.deb
/var/cache/apt/archives/girl1.2-peas-1.0_1.2.0-1ubuntu1_amd64.deb
```

Now clean up all the packages cached in the APT cache directory, and then verify that they are removed by running the `find` command again:

```
$ sudo apt-get clean
$ find /var/cache/apt/ -name \*.deb
```

The directory is empty now.

Note If you've run the `-h` option with `apt-get`, you may have noticed that the Ubuntu version of APT has Super Cow Powers. You can find out more about these Powers by asking `apt-get` to moo.

Downloading Packages with APT

If you want to download a package from a software repository without installing it, you can use the `download` option to `apt-get`. Once downloaded, the package can be investigated with the `dpkg` command before you install it (as described in the next section). To download the `minicom` package to the current directory, type the following:

```
$ sudo apt-get download minicom
```

Managing Software with dpkg

The `dpkg` utility (and related commands) works at a layer lower than the APT utilities do. APT uses `dpkg` behind the scenes to manage software on your Ubuntu system. APT and `dpkg` work in much the same way that `yum` and `rpm` do on Red Hat–based Linux distributions. Usually, APT will have enough functionality to get you through just about anything, but there are times when `dpkg` will be needed, such as finding out which package is associated with a given file on your system. The following gives some common `dpkg` commands and operations.

Note `dpkg` uses the `-D` flag to signify debugging information to be printed while performing various operations. If you want more information than the default output, try `-D1` with some of the `dpkg` commands. A section in the `dpkg` man page lists output levels for use with the `-D` flag.

- This command lists files that are installed by the `.deb` package file (replace `whatever.deb` with the name of the package you want to query). You must give the full or relative path to the `.deb` file:

```
$ dpkg -c whatever.deb
```

- This lists information about the `.deb` package file:

```
$ dpkg -I whatever.deb
```

- This lists information about the package:

```
$ dpkg -p whatever.deb
```

- List the packages where the given filename is found using this command. The filename can include a path to a file or just the name of a file:

```
$ dpkg -S filename
```

- This command lists all installed packages. This will also take options for more specific info:

```
$ dpkg -l
```

- This command lists all the files that have been installed from a package (the package must have been previously installed):

```
$ dpkg -L package
```

- The `-s` option lists the status of the given package:

```
$ dpkg -s package
```

- This installs the given `.deb` package file:

```
$ sudo dpkg -i whatever.deb
```

- This removes the given package from the system, but leaves configuration files behind. (That way, if the package is reinstalled later, it can resume using your configuration files.)

```
$ sudo dpkg -r package
```

- Removes package and configuration files of the given package.:

```
$ sudo dpkg -P package
```

- Extracts the files contained in the .deb package file to a destination directory. Note that this will reset permissions on the target directory or create the directory if it does not yet exist:

```
$ sudo dpkg -x whatever.deb /tmp/whatever
```

- Using the dpkg command, any user can query the package database. To use the command to install or remove software from your system, you must have root privileges.

Installing a Package with dpkg

The dpkg command focuses just on packages, while apt-get will take care of the messy details of finding out which repository hosts a package and downloading from there. For simplicity, the following example uses apt-get to download a package and then dpkg to install it, just to provide a flavor of the underlying dpkg command:

```
$ sudo apt-get download minicom  
Get:1 Downloading minicom 2.5-2 [297 kB]  
Fetched 297 kB in 0s (302 kB/s)
```

The command **downloads the package to the current directory**.

Now, **install the .deb file** using the command `dpkg -i` command (for install):

```
$ sudo dpkg -i /var/cache/apt/archives/minicom_2.5-2_amd64.deb  
Selecting previously deselected package minicom.  
(Reading database ... 185336 files and directories currently  
installed.)  
Unpacking minicom (from .../minicom_2.5-2_amd64.deb) ...  
Setting up minicom (2.5-2)  
Processing triggers for man-db ...
```

Removing a Package with dpkg

To **remove an installed package with dpkg**, use the `-r` option as follows:

```
$ sudo dpkg -r minicom  
(Reading database ... 185407 files and directories currently
```

```
installed.)
Removing minicom
Processing triggers for man-db ...
```

If you want to remove the package and its configuration files all at once, or remove the config files after removing the package, use the following:

```
$ sudo dpkg -P minicom
(Reading database ... 185335 files and directories currently
installed.)
Removing minicom ...
Purging configuration files for minicom ...
```

Extracting Files from a .deb File with dpkg

Debian and Ubuntu packages come bundled into single files, .deb files. Each .deb file contains one or more files that make up the package itself, such as a pre-built command, support files, documentation, and perhaps the source code.

So, a .deb file is basically an archive of files that you want to install to your computer, plus some header and control information that identifies the software (descriptions, Checksums, build information, and so on).

You can **extract a lot of this information on a package with the dpkg command**. This example extracts the files from the .deb file residing in the apt archives directory for the rsync package to a directory under /tmp:

```
$ mkdir /tmp/rsync_contents
$ sudo dpkg -x /var/cache/apt/archives/rsync_3.0.9-
1ubuntu1_amd64.deb \
/tmp/rsync_contents
$ ls /tmp/rsync_contents/
etc  lib  usr
```

You can replace the file rsync_3.0.9-1ubuntu1_amd64.deb in the preceding command with any .deb file you download.

Querying Information about .deb Packages

The following example shows how to **query installed packages for a package named rsync and display version information about that package**:

```
$ dpkg -p rsync
...
Version: 3.0.9-1ubuntu1
```

- Use the -I option to get information about a .deb file in any directory:

```
$ dpkg -I rsync_3.0.9-1ubuntu1_amd64.deb
```



```
new debian package, version 2.0.
```

```
...
```

- To **page through a list of all packages installed on your system**, type the following:

```
$ dpkg -l | less
```

```
...
```

```
ii  accountsservice 0.6.15-2ubuntu9.4 query and manipulate user acct info
```

```
...
```

- Or to just **see a listing of a specific package** use the `-l` option with a package name:

```
$ dpkg -l rsync
```

```
...
```

```
ii  rsync          3.0.9-1ubuntu1  fast, versatile, remote (and local) file-copyi
```

- **Check a file on your system to see what package the file belongs to, if any:**

```
$ dpkg -S /usr/bin/rsync
```

```
rsync: /usr/bin/rsync
```

Now that you know how to select the package(s) you want to query, let's get a little more information out of them. This example **lists standard details about an installed package**:

```
$ dpkg -s rsync
```

```
Package: rsync
```

```
Status: install ok installed
```

```
Priority: optional
```

```
Section: net
```

```
Installed-Size: 638
```

```
...
```

This **lists the content of a .deb file** in the local directory:

```
$ dpkg -c /var/cache/apt/archives/rsync_3.0.9-1ubuntu1_amd64.deb
```

```
drwxr-xr-x root/root          0 2011-11-08 14:05 ./
```

```
drwxr-xr-x root/root          0 2011-11-08 14:05 ./etc/
```

```
drwxr-xr-x root/root          0 2011-11-08 14:05 ./etc/default/
```

```
-rw-r--r-- root/root      1768 2011-11-08 14:05 ./etc/default/rsync
```

```
drwxr-xr-x root/root          0 2011-11-08 14:05 ./etc/init.d/
```

```
-rwxr-xr-x root/root     4395 2011-11-08 14:05 ./etc/init.d/rsync
```

```
...
```

This example **extracts the control scripts inside a .deb file on disk to a destination directory**. Use care when extracting as this command will reset the permissions on the target directory to 0755. (The 55 means that users other than you will have limited permissions on /tmp, and most applications assume they have wide open permissions to /tmp.) For this example, I'll create the directory /tmp/my_\$RANDOM (my_ and a random number) to work in:

```
$ cd /var/cache/apt/archives
```

```
$ sudo dpkg -e rsync_3.0.9-1ubuntu1_amd64.deb /tmp/my_$RANDOM
$ ls -lart /tmp/my_25445/
total 32
total 32
-rwxr-xr-x 1 root root 494 Nov 8 2011 prerm
-rwxr-xr-x 1 root root 113 Nov 8 2011 postrm
-rwxr-xr-x 1 root root 712 Nov 8 2011 postinst
-rw-r--r-- 1 root root 37 Nov 8 2011 conffiles
-rw-r--r-- 1 root root 1012 Nov 8 2011 control
-rw-r--r-- 1 root root 1743 Nov 8 2011 md5sums
```

To **extract all the non-control files** contained in the .deb file to a directory, use the `-x` option as follows (again, be aware that the directory permissions on the target directory will be reset to 0755):

```
$ sudo dpkg -x minicom_2.5-2_amd64.deb /tmp/dx_$RANDOM
$ ls -lart /tmp/dx_4921/
total 16
drwxr-xr-x 4 root root 4096 May 2 2011 usr
drwxr-xr-x 3 root root 4096 May 2 2011 etc
drwxr-xr-x 4 root root 4096 May 2 2011 .
drwxrwxrwt 15 root root 4096 Jan 27 22:03 ..
```

To **see the installed files a package is using on the system**, use the `-L` option:

```
$ dpkg -L minicom
/.
/usr
/usr/share
/usr/share/man
/usr/share/man/man1
/usr/share/man/man1/minicom.1.gz
/usr/share/man/man1/xminicom.1.gz
/usr/share/man/man1/ascii-xfr.1.gz
...
```

If the package is not removed completely, you may see some configuration files left over:

```
$ dpkg -L minicom
/etc
/etc/minicom
/etc/minicom/minicom.users
```

These examples cover some of the basic uses for `dpkg`, but by no means is this an exhaustive list. Other available options include those for reconfiguring packages (`dpkg-reconfigure`), telling `dpkg` what packages to ignore (`dpkg hold`), and setting selection states. Check the `dpkg` man page for more information.

Managing Software with aptitude

The dpkg and APT tools have been around for a long time and work well, but there are many nuances to both tools that can require a fair amount of understanding to use correctly. The aptitude tool tries to make things easier by automating some of the important package operations (such as running `apt-get update` before upgrading or installing) while allowing enough flexibility to be useful. For these reasons, I recommend the use of aptitude at the command line whenever possible.

The aptitude program aims to be both a curses application and command line tool. It is the command line usage I will focus on in this section. The following examples show a breakdown of ways to manage software packages with aptitude. Note that most of the options are similar to those of the `apt-get` command.

Note For more information on navigating the curses interface of aptitude, or other details, visit the Aptitude Survival Guide (<https://help.ubuntu.com/community/AptitudeSurvivalGuide>), or the Aptitude User's Manual (<http://people.debian.org/~dburrows/aptitude-doc/en/>), or run `man aptitude` at the command line.

- Running aptitude with no options starts the curses interface. Use Ctrl+t to access the menu and the q key to quit:

```
$ sudo aptitude
```

- This lists help for aptitude usage:

```
$ aptitude help
```

- This lists packages matching the given keyword:

```
$ aptitude search keyword
```

- This command updates the available package indexes from the APT sources:

```
$ sudo aptitude update
```

- This upgrades all packages in use to their latest versions:

```
$ sudo aptitude upgrade
```

- List information about the given package, installed or not, with the following command:

```
$ aptitude show package
```

- This downloads the given package, but does not install it:

```
$ sudo aptitude download package
```

- This removes all downloaded .deb files from the `/var/cache/apt/archives` directory:

```
$ sudo aptitude clean
```

- This removes all outdated `.deb` files from the `/var/cache/apt/archives` directory. This maintains a current cache without filling up the disk:

```
$ sudo aptitude autoclean
```

- This installs the given package to the system. Note that there are several options for selecting specific versions and using wildcards:

```
$ sudo aptitude install package
```

- This removes the given package from the system:

```
$ sudo aptitude remove package
```

- This upgrades all packages to their most recent versions, removing or installing packages as necessary. The `upgrade` option is preferable to `dist-upgrade`:

```
$ sudo aptitude dist-upgrade
```

Updating and Upgrading Packages with aptitude

By default, aptitude will always perform an `apt-get update` before installing or upgrading. You can, however, still issue a command to **perform only the update**:

```
$ sudo aptitude update
```

```
Ign http://security.ubuntu.com precise-security InRelease
Hit http://security.ubuntu.com precise-security Release.gpg
Ign http://us.archive.ubuntu.com precise InRelease
...
```

If you want to **upgrade all packages on the system**, you can send along the `upgrade` option with aptitude. This will install any new packages waiting in the repositories (in this example, there were no new packages on hand):

```
$ sudo aptitude upgrade
```

```
Resolving dependencies...
```

```
The following NEW packages will be installed:
```

```
linux-image-3.2.0-36-generic{a}
```

```
The following packages will be upgraded:
```

```
linux-image-server linux-server
```

```
2 packages upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
```

```
Need to get 38.5 MB of archives. After unpacking 149 MB will be
used.
```

```
Do you want to continue? [Y/n/?] Y
```

Querying Information about Packages with aptitude

You can **search with aptitude using keywords or full package names**, just as with the other package tools. Here is a search using the word **minic**, which returns the minicom package and and xfce window manager plug-in:

```
$ aptitude search minic
i   minicom          - friendly menu driven serial communication
program
$ aptitude show minicom
Package: minicom
State: not installed
Version: 2.5-2
Priority: optional
Section: universe/comm
...
```

Installing Packages with aptitude

You downloaded a package using `apt-get` before. Here you use `aptitude` to **download a package without installing it**:

```
$ sudo aptitude download minicom
...
Get:1 http://us.archive.ubuntu.com precise/main minicom 2.5-2
[297kB]
Fetched 276 kB in 0s (413 kB/s)
```

If you just want to **install the minicom package**, you can invoke `aptitude` like this:

```
$ sudo aptitude install minicom
...
Need to get 111 kB/408 kB of archives. After unpacking
  1,503 kB will be used.
Do you want to continue? [Y/n/?] n
```

If there is a **series of packages you want to install**, you can give `aptitude` a wildcard to select with. Here, I install any package containing the word `minic` (as with the `aptitude search` shown above). This also selects all of the dependencies for each package using what `aptitude` calls a **matcher**. Use the `~n` matcher, prefixed to your keyword, to install all packages containing the word `minic`:

```
$ sudo aptitude install "~nminic"
...
The following NEW packages will be automatically installed:
gcc-4.6-base:i386{a} libc6:i386{a} libcpan-mini-perl
  libencode-locale-perl{a}
  libfile-homedir-perl{a} libfile-listing-perl{a} libfile-which-
perl{a}
0 packages upgraded, 37 newly installed, 0 to remove and 0 not
```

```
upgraded.  
Need to get 6,038 kB/6,335 kB of archives. After unpacking  
18.1 MB will be used.  
...  
Accept this solution? [Y/n/q/?] n
```

Removing Packages with aptitude

Removing packages with aptitude is as easy as installing them. Just pass along the `remove` option:

```
$ sudo aptitude remove minicom  
...  
The following packages are unused and will be REMOVED:  
lrzsz  
The following packages will be REMOVED:  
minicom  
0 packages upgraded, 0 newly installed, 2 to remove and 0 not  
upgraded.  
Need to get 0B of archives. After unpacking 1401kB will be freed.  
Do you want to continue? [Y/n/?] Y
```

Cleaning Up Packages with aptitude

As you install things with aptitude, it will always download the `.deb` file and place it in the directory `/var/cache/apt/archives`. Over time, you will want to **purge these files from the cache** using the `clean` option, or at least the `autoclean` option to save disk space. If you look, you can see there are some files in the cache already:

```
$ ls /var/cache/apt/archives/  
lock  
lrzsz_0.12.21-5_amd64.deb  
minicom_2.5-2_amd64.deb  rsync_3.0.9-1ubuntu1_amd64.deb  
partial  
rsync_3.0.9-1ubuntu1_amd64.deb
```

Removing these with aptitude requires only that you use the `clean` or `autoclean` option:

```
$ sudo aptitude clean
```

Issuing the `ls` command again will show that the packages are indeed gone, so if you have a slow connection and it took you a week to download the last updates, you may want to think twice about this or use the `autoclean` option, which only removes outdated packages.

Useful Combinations of Options with aptitude

The `-v` option **adds verbosity to aptitude operations**. Used multiple times, you can get more than the usual information printed out as the operation executes. If you invoke `aptitude` with `-v`, it shows the MD5sum of the package. This is a digital fingerprint of sorts that can be used to test if the package has been tampered with or corrupted. **Using `-vv` will offer up even more information:**

```
$ aptitude show -vv minicom
Package: minicom
State: installed
...
Filename: pool/universe/m/minicom/minicom_2.5-2_amd64.deb
MD5sum: c2f4031b1fc6e688a783871cdca9890e
...
```

You can use the `-s` option with `aptitude` to tell it you want to **simulate what would happen without actually performing the operation**. This will work regardless of `aptitude` operation:

```
$ sudo aptitude -s install minicom
The following NEW packages will be installed:
...
Do you want to continue? [Y/n/?] y
Would download/install/remove packages.
```

Adding the `-v` option along with `-s` gives even more output:

```
$ sudo aptitude -vs install minicom
Reading package lists... Done
...
Do you want to continue? [Y/n/?] y
Inst lrzsz (0.12.21-5 Ubuntu:12.04/precise [amd64])
Inst minicom (2.5-2 Ubuntu:12.04/precise [amd64])
Conf lrzsz (0.12.21-5 Ubuntu:12.04/precise [amd64])
Conf minicom (2.5-2 Ubuntu:12.04/precise [amd64])
```

If you **don't want to be prompted** to answer the question "Do you want to continue?" you can answer ahead of time by adding the `-y` option to your command:

```
$ sudo aptitude -vs -y install "~ninc"
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
...
```

Be very careful with the `-y` option because there is no undo feature with `aptitude`.

Lastly, `aptitude -h` will return a reference of options that can be used if you need a refresher at any time. Interestingly enough, you can find out that the Ubuntu version of

aptitude is lacking something:

```
$ aptitude -h  
aptitude 0.6.6
```

```
...
```

This aptitude does not have Super Cow Powers.

No Super Cow Powers? I'm curious, so I ask aptitude to moo.

```
$ aptitude moo
```

There are no Easter Eggs in this program.

Easter eggs are items hidden in a program as a sort of surprise. Hmm. Maybe you can use the `-v` option for added verbosity:

```
$ aptitude -v moo
```

There really are no Easter Eggs in this program.

Maybe you need even more verbosity. Press aptitude for a little more information:

```
$ aptitude -vv moo
```

Didn't I already tell you that there are no Easter Eggs in this program?

Are you beginning to see a pattern here? Maybe adding more verbosity will lead somewhere.

Verifying Installed Packages with debsums

There are times when you will question the behavior of a binary or package installed on the system. It may not perform correctly, or may not even start at all. Problems with corrupt packages from unstable network connections or power outages do happen. In addition, malicious users may attempt to replace powerful commands with their own versions to cause further harm. It becomes useful to check the files on the filesystem against the information stored in the package.

The `debsums` program is a utility for Ubuntu and other Debian-based systems that **checks the MD5 sums of every installed package** against the `md5sum` files found in the `/var/lib/dpkg/info` directory.

You install this program with the following command:

```
$ sudo aptitude install debsums
```

Here are some of the most useful options for running `debsums`. See the man page for `debsums` to reveal all detailed information.


```

$ debsums -a      Check all files, include config files left out by
default
$ debsums -e      Check config files for packages only
$ debsums -c      List only changed files to stdout
$ debsums -l      List files that don't have md5sum info
$ debsums -s      List only errors; otherwise be silent
$ debsums package List the packages you want debsums to analyze

```

Note For many operations, you won't need to run this utility as root (using `sudo`). Some files may not have read access by regular users, so the use of `sudo` will be required if you get a message like this: `debsums: can't open at file /etc/at.deny (Permission denied)`.

If you run `debsums` with no options, it will check every file on the system that it knows about. The output can be redirected to a file if needed for later. The filename that `debsums` prints out will be accompanied by an `OK` status on the right side of the output if the MD5sum checks out for the file. Other messages may be printed out, such as `md5sums missing` for a certain file, or the word `REPLACED` if the MD5sum does not match.

You will need to be wary of false positives. If you want to use this tool as a baseline for assessments at a later date, you will want to get everything set up the way you want and re-generate MD5sums for stuff that is missing or incorrect. That way, you know you have the latest information.

This command will **check every file on the system against the stock md5sum files**. You can see there are some missing and replaced files. You would want to verify that the system does not already have problems with these files before you re-generate MD5sums for everything:

```

$ debsums
/usr/bin/acpi                                     OK
/usr/share/man/man1/acpi.1.gz                     OK
/usr/share/doc/acpi/README                         OK
/usr/share/doc/acpi/AUTHORS                       OK
...
/usr/share/app-install/icons/pybliographic.png    OK
debsums: no md5sums for bsduutils
debsums: no md5sums for bzip2
debsums: no md5sums for cdrecord
...
/usr/share/locale-langpack/en_AU/LC_MESSAGES/adduser.mo
REPLACED
/usr/share/locale-langpack/en_AU/LC_MESSAGES/alsa-utils.mo  OK
...

```

If you want to save this information to a file, you can **redirect both stdout and stderr**

streams into a file. Adding a final ampersand puts the command in the background, so you can continue working at the shell:

```
$ debsums &> /tmp/foo &
```

To **check the configuration files distributed with each package** for changes, run `debsums` with the `-a` option:

```
$ debsums -a  
/usr/bin/acpi OK  
/usr/share/man/man1/acpi.1.gz OK  
...
```

To check **only configuration files, and ignore everything else**, use the `-e` option. This is a good way to tell if you have inadvertently edited a config file you didn't want to. You can see some of the X configuration files have been changed:

```
$ debsums -e  
...  
/etc/X11/Xresources/x11-common OK  
/etc/X11/Xsession FAILED  
/etc/X11/rgb.txt OK  
/etc/init.d/x11-common OK  
/etc/X11/Xsession.d/50x11-common_determine-startup OK  
/etc/X11/Xsession.d/30x11-common_xresources OK  
/etc/X11/Xsession.d/20x11-common_process-args OK  
/etc/X11/Xsession.options FAILED  
...
```

As `debsums` spits out a lot of information, you may want to **see only changed files**. Issuing `debsums` with the `-c` options will do that:

```
$ debsums -c  
debsums: no md5sums for binutils  
debsums: no md5sums for libaudio  
...
```

With the preceding command, messages are printed for files that have no original md5sum info to check against. You can **check for files that have no md5sum info** by running `debsums` with the `-l` option:

```
$ debsums -l  
binutilslibaudio2...
```

If you want `debsums` to show only errors, use the `-s` option to **tell `debsums` to be silent except for errors**:

```
$ debsums -s  
debsums: no md5sums for binutils  
debsums: no md5sums for libaudio2...
```

To **check a specific package**, give `debsums` a package name as an argument:

```
$ debsums coreutils
/bin/cat                                OK
/bin/chgrp                             OK
/bin/chmod                             OK
...
```

This will check only the files listed in that package's `md5sum` file in the `/var/lib/dpkg/info` directory; so if the package does not come with an `md5sum` file, you will get an error:

```
$ debsums libaudio2
debsums: no md5sums for libaudio2
```

To generate the missing `md5sums` data for any package, use a combination of `dpkg`, the `md5sum` utility, and a little shell scripting. First, use `dpkg -L` to ask for a list of all the files `dpkg` knows about, in the package. The list `dpkg` returns will have other lines of data in it besides just the filenames, so you pipe that output to `grep` and filter out everything that does not start with a slash.

On the second line, you have the shell test whether the line of output from `dpkg` is a directory or a file (directories start with a slash also). If it is a file, `md5sum` is run on the line of output, which at this point should just be a filename. Last, all output at the third line is saved into a text file with the same naming convention as the `md5sum` files in the `/var/lib/dpkg/info` directory.

This example shows how to create this script for the `rsync` package. However, you can choose to use any package that does not have MD5sums available.

```
$ for file in `dpkg -L rsync | grep ^/`; do
test -f "$file" && md5sum "$file";
done > /tmp/rsync.md5sums
```

What you gain from this command is an `md5sum` database you can burn to CD-ROM and use to check your system. If the MD5sums are on CD-ROM, they cannot be deleted accidentally, or be subject to filesystem problems of a hard disk. If you want to check your MD5sums at a later time, you can use the `md5sum` command with the `-c` option and feed it the filename of the `md5sum` data:

```
$ md5sum -c /tmp/rsync.md5sums
...
/usr/bin/rsync: OK
/usr/share/doc/rsync/examples/rsyncd.conf: OK
/usr/share/doc/rsync/README.gz: OK
/usr/share/doc/rsync/TODO.gz: OK
...
```

To use the `rsync.md5sum` file with `debsums`, you need to make one modification that

will cause problems for `md5sum`, but is necessary for use with `debsums`: removing the leading slash in the filename. You can do this in a text editor, or with a little more shell scripting:

```
$ cat /tmp/rsync.md5sums
302916114c29191cd9c8cb51d67ee60a  /usr/bin/rsync
...
```

To remove the leading slash in front of `/usr/bin/rsync`, you could use a text editor or just use the Stream Editor (`sed`):

```
$ sed -e 's# /# #g' /tmp/rsync.md5sums > /tmp/rsync.debsums
$ cat /tmp/rsync.debsums
302916114c29191cd9c8cb51d67ee60a  usr/bin/rsync
```

With the leading slash removed, you can now copy `rsync.debsums` into the `/var/lib/dpkg/info` directory and `debsums` will be able to use it:

```
$ sudo mv /tmp/rsync.debsums /var/lib/dpkg/info/rsync.md5sums
$ debsums rsync
/usr/bin/rsync                                OK
/usr/share/doc/rsync/examples/rsyncd.conf     OK
/usr/share/doc/rsync/README.gz                OK
...
```

Building deb Packages

By rebuilding the `.deb` file that is used to build a Debian package, you can change it to better suit the way you use the software (for example, including an `md5sum` file). To begin, you need to extract a `.deb` file that you want to modify into a working directory. You then modify the file tree and control files to suit your needs.

For example, you could download and extract the `rsync` package and control files into the current directory by typing the following commands (your `$RANDOM` directory will be different of course):

```
$ aptitude download rsync
```

Then extract the package contents and the control files from the downloaded file. Note that the `$RANDOM` directory is found by typing `/tmp/rsync_` and pressing Tab:

```
$ sudo dpkg -x rsync_3.0.9-1ubuntu1_amd64 /tmp/rsync_$RANDOM
$ sudo dpkg -e rsync_3.0.9-1ubuntu1_amd64.deb /tmp/rsync_17197/
```

Now change to your package directory, where you extracted the `.deb` file to, and have a look around. You should see a directory structure that looks very similar to this:

```
$ cd /tmp/rsync_17197
$ ls -lart
```

```
total 36
-rw-r--r-- 1 root root 37 Nov 8 2011 conffiles
-rw-r--r-- 1 root root 1012 Nov 8 2011 control
drwxr-xr-x 4 root root 4096 Nov 8 2011 etc
drwxr-xr-x 3 root root 4096 Nov 8 2011 lib
-rw-r--r-- 1 root root 1743 Nov 8 2011 md5sums
-rwxr-xr-x 1 root root 712 Nov 8 2011 postinst
-rwxr-xr-x 1 root root 113 Nov 8 2011 postrm
-rwxr-xr-x 1 root root 494 Nov 8 2011 prerm
drwxr-xr-x 4 root root 4096 Nov 8 2011 usr
```

Now you have to configure the package directory to fit the formats that `dpkg` will want for building the `.deb` file. This involves creating a subdirectory named `rsync_3.0.9-1cn1.1/DEBIAN` and moving the install files into it. The control file itself is a specially formatted file that contains header and content fields and is parsed by the package tools to print out information about the package:

```
$ sudo mkdir -p rsync_3.0.9-1cn1.1/DEBIAN
$ sudo mv control conffiles prerm postrm postinst rsync_3.0.9-
1cn1.1/DEBIAN
```

You also need to move the `etc/` and `usr/` directories under the `rsync_3.0.9-1cn1.1` directory:

```
$ sudo mv usr etc rsync_3.0.9-1cn1.1
```

You should end up with everything filed away correctly, and all that is left are the `rsync_3.0.9-1cn1.1` directory, the `lib` directory, and the `md5sums` file in the current directory.

Now move the `md5sums` file you made earlier into your `DEBIAN` subdirectory and rename it **md5sums**. This will allow `debsums` to have some MD5sums to check:

```
$ sudo mv /var/lib/dpkg/info/rsync.md5sums rsync_3.0.9-
1cn1.1/DEBIAN/md5sums
```

Now edit the control file to modify some of the information. You certainly don't want to install your modified version of `rsync` with the same package info as the original. Open the control file in `vi` or another editor and change the `Version` line to reflect the one below. You will notice the word `Version` has a colon after it; this is the header field. The information field follows right after it. Be sure to maintain the space after the colon, and do not put any extra carriage returns or spaces in the file. It is very picky about formatting.

```
$ sudo vi rsync_3.0.9-1cn1.1/DEBIAN/control
...
Version: rsync_3.0.9-1cn1.1
...
```

A little farther down, you can add to the `Description` field. This will show up in the descriptions whenever someone views the package details. Notice the space right before the words `fast remote`. The space is part of the special formatting and is how `dpkg` tells the description text from the multiline header. Be sure to put a space in the first column if you wrap the description to the next line:

```
...
Description: Modified by CN 2013-02-01 to include md5sums.
             fast remote file copy program (like rcp)
...
```

Now build your new package using `dpkg -b` and the name of the control file subdirectory you created. You will get a warning about `Original-Maintainer` being a user-defined field. You can safely ignore the warning.

```
$ sudo dpkg -b rsync_2.6.9-1cn1.1
warning, `rsync_2.6.9-1cn1.1/DEBIAN/control' contains user-defined
field
`Original-Maintainer'
dpkg-deb: building package `rsync' in `rsync_3.0.9-1cn1.1.deb'.
dpkg-deb: ignoring 1 warnings about the control file(s)
```

You now have a new `.deb` file and can ask `dpkg` to display information about it. Just run `dpkg` with the `-I` option to see the new package info:

```
$ dpkg -I rsync_3.0.9-1cn1.1.deb
new debian package, version 2.0.
size 1004 bytes: control archive= 712 bytes.
    970 bytes,    21 lines      control
Package: rsync
Version: 3.0.9-1cn1.1
...
```

You could install the new `rsync` package at this point. This exercise is mainly a demonstration for building a custom package, not necessarily for hacking up the system needlessly. Nonetheless, the following code shows that this package will install and act like a regular Debian package. You want `debsums` to work also. Notice `dpkg` tells you about the downgrade:

```
$ sudo dpkg -i rsync_3.0.9-1cn1.1.deb
dpkg - warning: downgrading rsync from 3.0.9-1ubuntu1_amd64
to 3.0.9-1cn1.1.
(Reading database ... 88107 files and directories currently
installed.)
Preparing to replace rsync 3.0.9-1ubuntu1_amd64
  (using rsync_3.0.9-1cn1.1.deb) ...
Unpacking replacement rsync ...
Setting up rsync (3.0.9-1cn1.1) ...
```

The `debsums` utility now has some `md5sum` files to test with, and anywhere your new `rsync` package is installed, this will be the same:

```
$ debsums rsync
/usr/bin/rsync                                OK
/usr/share/doc/rsync/examples/rsyncd.conf     OK
/usr/share/doc/rsync/README.gz                OK
...
```

You can also ask `dpkg` to list your `rsync` package using the `-l` option to confirm that the new version is installed:

```
$ dpkg -l rsync
...
ii  rsync      3.0.9-1cn1.1  Modified by CN 2013-02-01 to include
md5sums.
```

Note You can find out more about building `.deb` files by visiting the Debian Binary Package Building HOWTO (<http://tldp.org/HOWTO/Debian-Binary-Package-Building-HOWTO>). The `dpkg-deb` man page is also a good source of info on deb package building.

Summary

Software for Ubuntu and other Debian-based distributions is packaged in the `deb` format. The Ubiquity installer is used to initially install Ubuntu. From the Boot menu, you can boot into a full Ubuntu environment and install from there, or run Ubuntu from a CD-ROM.

To install additional software, you can use the `aptitude` and `APT` utilities to get packages from online repositories. To install packages locally, as well as build custom Debian packages, you can use the `dpkg` utility. `APT`, `aptitude`, and `dpkg` all offer a means to query software. You can verify installed packages by using the `debsums` and `md5sum` utilities.

Chapter 3

Using the Shell

IN THIS CHAPTER

- Accessing the shell
- Using command history and completion
- Assigning aliases
- Gaining super user access
- Writing simple shell scripts

The use of a shell command interpreter (usually just called a shell) dates back to the early days of the first UNIX systems. Besides its obvious use of running commands, shells have many built-in features such as environment variables, aliases, and a variety of functions for programming.

Although the shell used most often with Linux systems is called the Bourne Again Shell (bash), other shells are available as well (such as sh, csh, ksh, and tcsh). In many cases, these shells, such as sh, are really symbolic links to other shell programs, such as bash. On Ubuntu Linux, sh is a symbolic link to `/bin/dash`. The sh shell is important as it is called in most shell scripts as the shell to run scripts. For interactive usage, bash forms the default shell.

This chapter offers information that will help you use Linux shells, in general, and the bash shell, in particular.

Terminal Windows and Shell Access

The most common way to access a shell from a Linux graphical interface is using a Terminal window. From a graphical interface, you can often access virtual terminals to get to a shell. With no graphical interface, with a text-based login you are typically dropped directly to a shell after login.

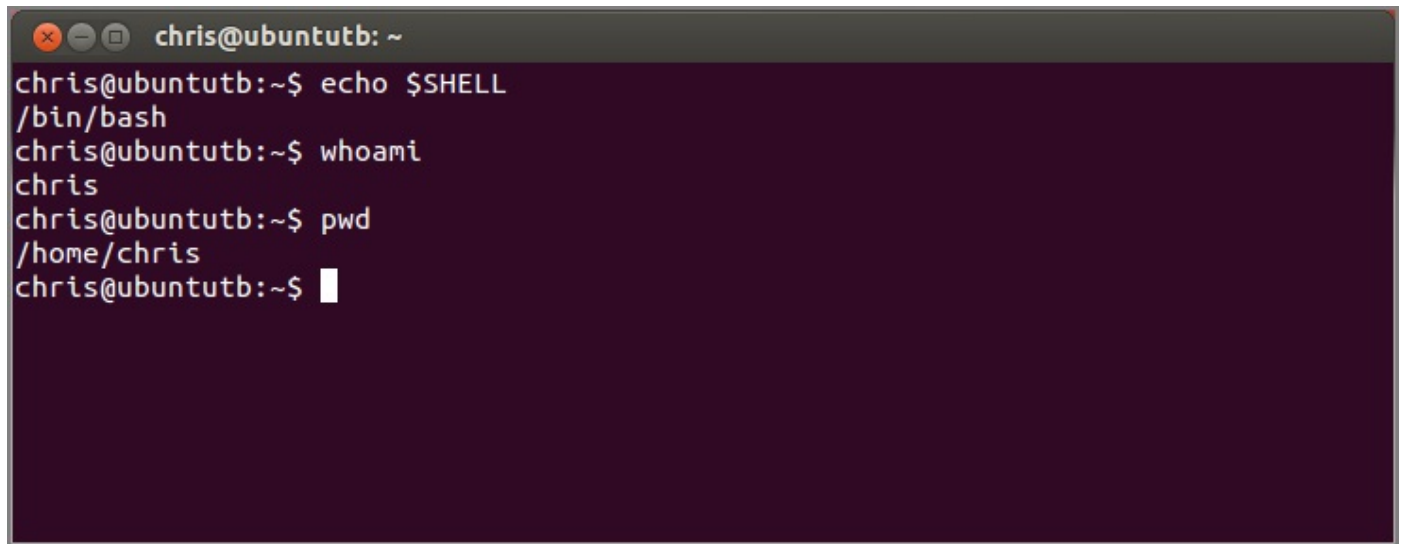
Using Terminal Windows

To open a Terminal window from Unity (the default Ubuntu desktop), select the Dashboard icon, and then begin typing terms in the search box. Look for a Terminal icon

to appear in the dashboard and select it. This opens a gnome-terminal window, displaying a bash shell prompt. [Figure 3-1](#) shows an example of a gnome-terminal window.

Commands shown in [Figure 3-1](#) illustrate that the **current shell** is the bash shell (`/bin/bash`), the **current user** is the desktop user who launched the window (chris), and the **current directory** is that user's home directory (`/home/chris`). The username (chris) and hostname (ubuntutb) appear in the title bar and the prompt.

[Figure 3-1](#): Type shell commands into a gnome-terminal window.



```
chris@ubuntutb:~$ echo $SHELL
/bin/bash
chris@ubuntutb:~$ whoami
chris
chris@ubuntutb:~$ pwd
/home/chris
chris@ubuntutb:~$
```

The gnome-terminal window not only lets you access a shell, it also has controls for managing your shells. Here are some examples:

- Select File ⇒ Open Tab to **open another shell on a different tab**.
- Select File ⇒ Open Terminal to **open a new Terminal window**.
- Select Terminal ⇒ Set Title to **set a new title in the title bar**.

You can also use control key sequences to work with a Terminal window. Open a **shell on a new tab** by pressing Shift+Ctrl+t, **open a new Terminal window** with Shift+Ctrl+n, **close a tab** with Shift+Ctrl+w, and **close a Terminal window** with Shift+Ctrl+q. Highlight text and copy it with Shift+Ctrl+c, and **then paste it in the same or different window** with Shift+Ctrl+v or by clicking the center button on your mouse.

Note In most applications, such as the LibreOffice word processor, Ctrl+c, not Shift+Ctrl+c, invokes the copy function, and Ctrl+v, not Shift+Ctrl+v, invokes the paste function. Because Ctrl+c means something special in a shell window (sending a signal to a program that normally causes it to die), the gnome-terminal window maps the expected graphical desktop functions using the Shift key as a modifier.

Other key sequences for controlling Terminal windows include pressing F11 to toggle the window in and out of **full screen mode**. Press Ctrl+Shift++ to **zoom in** (make text larger) or Ctrl+- (that's Ctrl and a minus sign) to **zoom out** (make text smaller). **Switch among tabs** using Ctrl+PageUp and Ctrl+PageDown (previous and next tab), or use Alt+1, Alt+2, Alt+3, and so on, to go to tab one, two, or three (and so on). Type Ctrl+d to exit the shell, which closes the current tab or the entire Terminal window (if it's the last tab).

The gnome-terminal window also supports profiles (select Edit ⇒ Profiles). Some profile settings are cosmetic (allowing bold text, cursor blinks, terminal bell, colors, images, and transparency). Other settings are functional. For example, by default, the terminal saves 512 scrollback lines. Some people like to be able to scroll back further and are willing to give up more memory to allow that.

If you launch gnome-terminal manually, you can add options. Here are some examples:

\$ <u>gnome-terminal -x alsamixer</u>	Start with alsamixer (ESC exits)
\$ <u>gnome-terminal --tab --tab --tab</u>	Start a terminal with 3 open tabs
\$ <u>gnome-terminal --geometry 80x20</u>	Start 80 characters by 20 lines
\$ <u>gnome-terminal --zoom=2</u>	Start terminal with larger font

Besides gnome-terminal, there are many other terminal windows you can use, such as `xterm` (basic terminal emulator that comes with the X Window System), `aterm` (terminal emulator modeled after the Afterstep XVT VT102 emulator), and `konsole` (terminal emulator delivered with the KDE desktop). The Enlightenment desktop project offers the `eterm` terminal (which includes features such as message logs on the screen background).

Using Virtual Consoles

When Ubuntu boots in multi-user mode (runlevel 2, 3, or 5), six virtual consoles (known as tty1 through tty6) are created with text-based logins. If an X Window System desktop is running, X is probably running in virtual console 7. If X isn't running, chances are

you're looking at virtual console 1.

From X, you can **switch to another virtual console** with Ctrl+Alt+F1, Ctrl+Alt+F2, and so on up to F6. From a text virtual console, you can switch using Alt+F1, Alt+F2, and so on. Press Alt+F7 to return to the X GUI. Each console allows you to log in using different user accounts. Switching to look at another console doesn't affect running processes in any of them. When you switch to virtual console 1 through 6, you see a login prompt similar to the following:

```
Ubuntu 12.04.2 LTS ubuntutb tty2
ubuntutb login:
```

Separate `getty` processes manage each virtual terminal. Type this command to see what `getty` processes look like before you log in to any virtual terminals:

```
$ ps aux | grep -v grep | grep getty
4366 tty4      Ss+    0:00 /sbin/getty -8 38400 tty4
4367 tty5      Ss+    0:00 /sbin/getty -8 38400 tty5
4372 tty2      Ss+    0:00 /sbin/getty -8 38400 tty2
4373 tty3      Ss+    0:00 /sbin/getty -8 38400 tty3
4374 tty1      Ss+    0:00 /sbin/getty -8 38400 tty1
4375 tty6      Ss+    0:00 /sbin/getty -8 38400 tty6
```

After I log in on the first console, `getty` handles my login, and then fires up a bash shell:

```
$ ps aux | grep -v grep | grep tty
4366 tty4      Ss+    0:00 /sbin/getty -8 38400 tty4
4367 tty5      Ss+    0:00 /sbin/getty -8 38400 tty5
4372 tty2      Ss    0:00 /bin/login --
4373 tty3      Ss+    0:00 /sbin/getty -8 38400 tty3
4374 tty1      Ss+    0:00 /sbin/getty -8 38400 tty1
4375 tty6      Ss+    0:00 /sbin/getty -8 38400 tty6
7214 tty2      S+     0:00 -bash
1153 tty7      Ss+    5:59 /usr/bin/X :0 -auth /var/run/lightdm/root/:0
      -nolisten tcp vt7 -novtswitch -background none
```

Virtual consoles are configured in the `/etc/init` directory. A script appears for each virtual console, such as `tty1.conf` for the `tty1` console, `tty2.conf` for the `tty2` console, and so on.

Note Other versions of Linux configure the consoles in one file, `/etc/inittab`. The `init` daemon uses `/etc/inittab` as its configuration file. Ubuntu Linux, on the other hand, uses a version of `init` from the `upstart` package, which uses the `/etc/init` directory to hold its configuration files.

Using the Shell

After you open a shell (whether from a text-based login or Terminal window), the shell environment is set up based on the user who started the shell. Bash shell settings for all users' shells are located in several files. You can make your own versions of these files to override the system settings. There are two types of files holding these settings: startup files and initialization files.

Bash runs startup files for any shell that is a login shell. These files define settings that apply across your entire login. Bash runs initialization files for shells run interactively that is, not running a shell script.

Bash uses the system-wide startup file `/etc/profile`, as well as several dot files in the user's home directory for personal settings (if they exist): `.bash_profile`, `.bash_login`, and `.profile`. It also includes scripts stored in the `/etc/profile.d` directory that end in `.sh`.

Bash includes the system-wide initialization file `/etc/bash.bashrc` and the `.bashrc` file in your home directory (for personal settings). Those files are included every time a new bash shell is opened.

Note Other versions of Linux store the system-wide files in `/etc/bashrc`.

When you exit from a login shell (for example, from a virtual console), any commands in your `~/.bash_logout` file are executed. Changing settings in these files permanently changes the user's shell settings but does not affect shells that are already running. (Other shells use different configuration files.)

There are a variety of ways in which you can list and change your shell environment. One of the biggest ways is to change which user you are—in particular, to become the super user (see the section “Acquiring Super User Power” later in this chapter).

Using Bash History

The Bourne Again Shell (bash) is the shell used by default by most modern Linux systems and quite a few other operating systems such as Mac OS X. Built into bash, as with other shells, is a history feature that lets you review, change, and reuse commands that you have run in the past. This can prove very helpful as many Linux commands are long and complicated.

When bash starts, it reads the `~/.bash_history` file and loads it into memory. This file is set by the value of `$HISTFILE`.

Note See the section “Using Environment Variables” later in this chapter for more on how to work with shell environment variables such as `$HISTFILE`.

During a bash session, commands are added to history in memory. When bash exits, history in memory is written back to the `.bash_history` file. The **number of commands held in history during a bash session** is set by `$HISTSIZE`, while the **number of commands actually stored in the history file** is set by `$HISTFILESIZE`:

```
$ echo $HISTFILE $HISTSIZE $HISTFILESIZE  
/home/chris/.bash_history 1000 2000
```

To **list the entire history**, type `history`. To **list a previous number of history commands**, follow `history` with a number. This lists the previous five commands in your history:

```
$ history 5  
975  mkdir extras  
976  mv *doc extras/  
977  ls -CF  
978  vi house.txt  
979  history 5
```

To **move among the commands** in your history, use the up arrow and down arrow keys. Once a command displays, you can use the keyboard to **edit the current command** like any other command: left arrow, right arrow, Delete, Backspace, and so on. Here are some other ways to recall and run commands from your bash history:

\$ <u>!!</u>	Run the previous command
\$ <u>!997</u>	Run command number 997 from history
ls -CF	
\$ <u>!997 *doc</u>	Append *doc to command 997 from history
ls -CF *doc	
\$ <u>!?CF?</u>	Run previous command line containing the CF
string	
ls -CF *doc	
\$ <u>!!ls</u>	Run the previous ls command
ls -CF *doc	
\$ <u>!ls:s/CF/l</u>	Run previous ls command, replacing CF with l
ls -l *doc	

Another way to **edit the command history** is using the `fc` command. With `fc`, you open the chosen command from the history using the nano editor. In nano, type your changes, and then type `Ctrl+o` and `Ctrl+x` to save your changes. The edited command runs when you exit the editor. Change to a different editor by setting the `FCEDIT` variable (for example, `export FCEDIT=gedit`) or on the `fc` command line. For example:

```
$ fc 978           Edit command number 978, then run it
$ fc              Edit the previous command, then run it
$ fc -e /usr/bin/vim 989 Use vim to edit command 989
```

Use Ctrl+r to **search for a string in history**. For example, typing Ctrl+r followed by the string `ss` resulted in the following:

```
# <Ctrl+r>
(reverse-i-search) 'ss': sudo /usr/bin/less /var/log/messages
```

Press Ctrl+r repeatedly to **search backward through your history list** for other occurrences of the `ss` string.

Note By default, bash command history editing uses emacs-style commands. If you prefer the vi editor, you can use vi-style editing of your history by using the `set` command to set your editor to vi. To do that, type the following: `set -o vi`. Add that line to your `.bashrc` file, as described later, to have vi set as your history editor every time you open a shell.

Using Command Line Completion

You can use the Tab key to complete different types of information on the command line. Here are some examples where you type a partial name, followed by the Tab key, to **have bash try to complete the information you want on your command line**:

```
$ ifc<Tab>           Command completion: Completes to ifconfig
command
$ cd /home/ch<Tab>   File completion: Completes to /home/chris
directory
$ cd ~jo<Tab>        User homedir completion: Completes to
/home/john
$ echo $PA<Tab>      Env variable completion: Completes to $PATH
$ ping @<Tab><Tab>    Host completion: Show hosts from /etc/hosts
@davinci.example.com @ritchie.example.com @thompson.example.com
@localhost           @zooney
```

Redirecting stdin and stdout

Typing a command in a shell makes it run interactively. The resulting process has two output streams: stdout for normal command output and stderr for error output. In the following example, when `/tmpp` isn't found, an error message goes to stderr but output from listing `/tmp` (which is found) goes to stdout:

```
$ ls /tmp /tmpp
ls: /tmpp: No such file or directory
```

```
/tmp/:  
gconfd-chris  keyring-b41WuB  keyring-ItEWbz  mapping-chris  orbit-  
chris
```

By default, all output is directed to the screen. Use the greater-than sign (>) to **direct output to a file**. More specifically, you can direct the standard output stream (using >) or standard error stream (using 2>) to a file. Here are examples:

```
$ ls /tmp /tmp > output.txt  
ls: /tmp: No such file or directory
```

```
$ ls /tmp /tmp 2> errors.txt  
/tmp/:  
gconfd-chris  keyring-b41WuB  keyring-ItEWbz  mapping-chris  orbit-  
chris  
$ ls /tmp /tmp 2> errors.txt > output.txt
```

```
$ ls /tmp /tmp > everything.txt 2>&1
```

In the first example, stdout is redirected to the file `output.txt`, while stderr is still directed to the screen. In the second example, stderr (stream 2) is directed to `errors.txt` whereas stdout goes to the screen. In the third example, the first two examples are combined. The last example directs both streams to the `everything.txt` file. **To append to a file** instead of overwriting it, use two greater-than signs:

```
$ ls /tmp >> output.txt
```

If you don't ever want to see an output stream, you can simply **direct the output stream to a special bit bucket file** (`/dev/null`). In this case, stderr is discarded and stdout is displayed:

```
$ ls /tmp 2> /dev/null
```

Note Another time that you may want to redirect stderr is when you run jobs with crontab. You could redirect stderr to a mail message that goes to the crontab's owner. That way, any error messages can be sent to the person running the job.

Just as you can direct standard output from a command, you can also **direct standard input to a command**. For example, the following command e-mails the `/etc/hosts` file to the user named `chris` on the local system:

```
$ mail chris < /etc/hosts
```

Using pipes, you can **redirect output from one process to another process** rather than just files. Here is an example where the output of the `ls` command is piped to the `sort -r` command to have the output sorted in reverse order:

```
$ ls /tmp | sort -r
```

In the next example, a **pipe and redirection are combined** (the stdout of the `ls` command is sorted and stderr is dumped to the bit bucket):

```
$ ls /tmp/ /tmmp 2> /dev/null | sort -r
```

Pipes can be used for tons of things:

```
$ dpkg-query -l | grep -i sql | wc -l
```

```
$ ps auwx | grep firefox
```

```
$ ps auwx | less
```

```
$ whereis -m bash | awk '{print $2}'
```

The first command line in the preceding examples lists all installed packages, grabs those packages that have `sql` in them (regardless of case), and does a count of how many lines are left (effectively counting packages with `sql` in the name). The second command line displays Firefox processes taken from the long process list (assuming the Firefox web browser is running), as well as any process whose command line references `firefox`, such as the command issued in this example. The third command line lets you page through the process list. The last line displays the word `bash`: followed by the path to the bash man page, and then displays only the path to the man page (the second element on the line).

Using backticks, you can **execute one section of a command line first and feed the output of that command to the rest of the command line**. Here are examples:

```
$ dpkg-query -S `which ps`
```

```
$ ls -l `which bash`
```

The first command line in the preceding example finds the full path of the `ps` command and finds the package that contains that `ps` command. The second command line finds the full path to the `bash` command and does a long list (`ls -l`) of that command.

A more advanced and powerful way to **take the output of one command and pass it as parameters to another** is with the `xargs` command. For example:

```
$ ls /bin/b* | xargs /usr/bin/dpkg-query -S
```

You can use the `-t` option to `xargs` to produce a verbose output of the command line before the command is executed. You can also have `xargs` pass each output string from `ls` as input to individual `dpkg-query` commands. You define `{}` as the placeholder for the string:

```
$ ls /bin/b* | xargs -t -I{} dpkg-query -S {}
```

```
dpkg-query -S /bin/bash
```

```
bash: /bin/bash
```

```
dpkg-query -S /bin/bunzip2
```

```
bzip2: /bin/bunzip2
```

```
dpkg-query -S /bin/bzcat
```



```
bzip2: /bin/bzcat
dpkg-query -S /bin/bzcmp
bzip2: /bin/bzcmp
dpkg-query -S /bin/bzdiff
bzip2: /bin/bzdiff
dpkg-query -S /bin/bzegrep
bzip2: /bin/bzegrep
dpkg-query -S /bin/bzexe
bzip2: /bin/bzexe
dpkg-query -S /bin/bzfgrep
bzip2: /bin/bzfgrep
dpkg-query -S /bin/bzgrep
bzip2: /bin/bzgrep
dpkg-query -S /bin/bzip2
bzip2: /bin/bzip2
dpkg-query -S /bin/bzip2recover
bzip2: /bin/bzip2recover
dpkg-query -S /bin/bzless
bzip2: /bin/bzless
dpkg-query -S /bin/bzmore
bzip2: /bin/bzmore
```

As you can see from the output, separate `dpkg-query -S` commands are run for each option passed by `ls`.

Using alias

You use the `alias` command to **set and list aliases**. Some aliases are already set in the system-wide or user-specific shell initialization files discussed previously. Here's how to **list the aliases that are currently set**:

```
$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

Notice that some aliases are set simply as a way of adding options to the default behavior of a command (such as `ls --color=auto`, so that the output of the `ls` command is always shown in color).

You can **define your own aliases for the current bash session** as follows:

```
$ alias lala='ls -la'
```

Add that line to your `~/.bashrc` file for the definition to occur for each new bash

session. **Remove an alias from the current bash session** using the `unalias` command, as follows:

```
$ unalias lala           Unalias the previously aliased lala command
$ unalias -a           Unalias all aliased commands
```

Watching Commands

If you need to keep an eye on a command whose output is changing, use the `watch` command. For example, to keep an eye on your load average, you can use the following:

```
$ watch `cat /proc/loadavg`
```

Every two seconds, `watch` runs the `cat` command again. Use `Ctrl+c` to quit. To change the refresh rate to 10 seconds, type the following:

```
$ watch -n 10 `ls -l`
```

To highlight the difference between screen updates, type:

```
$ watch -d `ls -l`
```

Press `Ctrl+c` to exit the `watch` command. Note that files need to change so that differences can be highlighted.

Watching Files

You can use the `watch` command to watch the size of a file. For example, to watch a large ISO file named `mydownload.iso` as it downloads, use the following command:

```
$ watch 'ls -l mydownload.iso'
```

To watch the contents of a plain text file grow over time, you can use the `tail` command. For example, you can watch as messages are added to the `/var/log/messages` file and see those messages displayed on your screen as follows:

```
$ sudo tail -f /var/log/messages
```

Pressing `Ctrl+c` will exit from the `tail` command.

Acquiring Super User Power

When you open a shell, you are able to run commands and access files and directories based on your user/group ID and the permissions set for those components. Many system features are restricted to the root user, also referred to as the super user.

There are three main ways to acquire super user power:

- Run one command with root user privileges with the `sudo` command.
- Log in as the root user.
- Temporarily become the root user with the `su` command.

Ubuntu Linux is set up for users to run the `sudo` command. In fact, a password is not even set for the root user by default. So, in most cases, to run an administrative command (such as `useradd` to add a new user), you would precede that command with the `sudo` command. For example:

```
$ sudo useradd -m joe           As root user, add a new user named
joe
```

Because Ubuntu restricts the system such that the root user cannot log in, Ubuntu is also not set up to use the `su` command, which is normally used on other Linux systems, to change to the root user.

If you find that you need to run a series of commands as root user, you could type the following command to open a shell as the root user:

```
$ sudo bash           Open a shell as root user
[sudo] password for chris:
*****
#
```

If you decide that you want to add a password for the root user (allowing you to log in as the root user or use the `su` command to temporarily become root), you can also do that using the `sudo` command:

```
$ sudo passwd root       Set the root user's password
```

Most Ubuntu desktop users, however, simply use `sudo` and never set a root password.

Delegating Power with `sudo`

The `sudo` command allows very granular delegation of power to users other than the root user. The `sudo` facility is a great tool for granting specific escalated privileges when you have multiple users and logging everything the users do with those privileges. Unless otherwise specified, `sudo` runs as root. Ubuntu Linux uses the `sudo` command to execute privileged commands, rather than the `su` command.

The `sudo` command is configured in `/etc/sudoers`.

Warning Never edit this file with your normal text editor. Instead, always use the `visudo` command. This opens the `sudoers` file in an editor, temporarily locks it from use by others, and does some error checking before you leave.

The file `/etc/sudoers` is restricted, so you need to use the `sudo` command to edit the file. For example:

```
$ sudo visudo
```

The `visudo` command launches an editor, by default the `nano` editor, discussed previously.

If you look at the `sudoers` file that shipped with your distribution, you'll see different empty sections delimited by comments and one active statement:

```
root    ALL=(ALL:ALL) ALL
```

This means that the user `root` is allowed on any hosts to run any command as any user.

Ubuntu Linux adds the following so that all users who are part of the `admin` or `sudo` group (as set in the `/etc/group` file) can acquire root privileges:

```
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) NOPASSWD: ALL
```

When a user in the `sudo` group requests root privilege, that user gains access to that privilege without entering a password. You can modify that `sudoers` line so a privileged user does have to provide a password (their own user password) when running `sudo` as follows:

```
%sudo  ALL=(ALL:ALL) ALL
```

When you installed Ubuntu Linux, the user account you created was automatically added to the `sudo` group. To allow additional users to acquire root privileges, you can simply add them to the `sudo` group in the `/etc/group` file. For example, to add `sudo` privilege for an existing user named `joe`, you could modify the line in `/etc/group` so it appears as follows:

```
sudo:x:27:chris,joe
```

You also have the choice of giving a user limited root privilege instead in the `/etc/sudoers` file. For example, you can add the following line, setting the first field to a user account on your system:

```
joe ALL= /usr/bin/less /var/log/messages
```

Note The preceding setting allows this user to run the `less` command with root privileges. This introduces a security issue, as the `less` command can allow this user to gain more information on the system by examining other system files.

Now `joe` (or whichever user you've added) can do the following:

```
$ sudo /usr/bin/less /var/log/messages
```

Password:

After joe types his own password, he can page through the `/var/log/messages` file. A timestamp is set at that time as well. For the next five minutes (by default), that user can type the preceding command and have it work without being prompted for the password.

Normally, though, you should add such users to the sudoers or admin group and not create individual entries in the `/etc/sudoers` file.

Every use of `sudo` gets logged in `/var/log/secure`:

```
Feb 24 21:58:57 localhost sudo: joe : TTY=pts/3 ; PWD=/home/joe ;  
USER=root; COMMAND=/usr/bin/less /var/log/messages
```

Next add this line to `/etc/sudoers`:

```
joe            server1=(chris)        /bin/ls /home/chris
```

Now joe can do the following:

```
$ sudo -u chris /bin/ls /home/chris
```

The `sudo` command just shown runs as `chris` and will work only on the host `server1`. In some organizations, the `/etc/sudoers` file is centrally managed and deployed to all the hosts, so it can be useful to specify `sudo` permissions on specific hosts.

The `sudo` command also allows the definition of aliases, or predefined groups of users, commands, and hosts. Check the `/etc/sudoers` file on your Linux system for examples of those features.

Using the su Command

If you did decide at some point to add a password to your root user account, with a shell open as a regular user, you can use the `su` (super user) command to **become the root user**. You can also use the `su` command to switch to a different, non-root user. The following sections describe how the `su` command works.

Simply using `su`, as in the following code, doesn't give you a login shell with root's environment:

```
$ su
```

Password: *****

```
# echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/ga
```

```
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/bin:
```

```
/usr/bin:/bin:/usr/X11R6/bin:/home/joe/bin
```

After running `su`, the user still has the user `joe`'s `PATH`. To **enable the root user's**

environment, use the `su` command with the dash option (`-`), as follows:

```
# exit
$ su -
Password: *****
# echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:
/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

In most cases, use `su -`, unless you have a very specific reason not to. If no user is specified, `su` defaults to the root user. However, `su` can also be used to **become other users**:

```
$ su - chris
```

The `su` command can also be used to **execute a single command as a particular user**:

```
$ su -c whoami
Password: *****
root
# su -c 'less /var/log/messages'
```

Although in the second example you are logged in as a regular user, when you run `whoami` with `su -c`, it shows that you are the root user. In the directly preceding example, the quotes are required around the `less` command line to identify `/var/log/messages` as an option to `less`. As shown above, `whoami` can be useful in **determining which user you're currently running a command as**:

```
$ whoami
chris
```

Using Environment Variables

Small chunks of information that are useful to your shell environment are stored in what are referred to as variables. By convention, shell variable names are all uppercase (although that convention is not enforced). If you use the bash shell, some variables are set for you from various bash start scripts covered previously.

How a variable is set determines whether it is a local variable (available only to the current shell) or an environment variable (which is inherited by other shells or applications run from that shell). To become an environment variable, the variable must be exported.

To **display, in alphabetical order, all of the current shell variables** that are already set for your shell, type the following:

```
$ set | less
BASH=/bin/bash
COLORS=/etc/DIR_COLORS.xterm
COLUMNS=118
DISPLAY=:0.0
HOME=/home/chris
HOSTNAME=ubuntutb
...
```

The output just shown contains only a few examples of the environment variables you will see. The `set` command lists local variables, environment variables, and functions as well. The `env` command just lists environment variables.

You can also **set, or reset, any variables** yourself. For example, to assign the value `123` to the variable `ABC` (then display the contents of `ABC`), type the following:

```
$ ABC=123
$ echo $ABC
123
```

The variable `ABC` exists only in the shell it was created in. If you launch a command from that shell (`ls`, `cat`, `firefox`, and so on), that new process will not see the variable. Start a new bash process and test this:

```
$ bash
$ echo $ABC

$
```

You can **make variables part of the environment and inheritable** by child processes by exporting them:

```
$ export ABC=123
$ bash
$ echo $ABC
123
```

Also, you can **concatenate a string to an existing variable**:

```
$ export PATH=$PATH:/home/chris/bin
```

Type the following to **list your bash's environment variables**:

```
$ env
```

When you go to create your own environment variables, avoid using names that are already commonly used by the system for environment variables. See Appendix B for a list of shell environment variables.

Creating Simple Shell Scripts

Shell scripts are good for automating repetitive shell tasks. Bash and other shells include the basic constructs found in various programming languages, such as loops, tests, case statements, and so on. The main difference is that there is only one type of variable: strings.

Editing and Running a Script

Shell scripts are simple text files that contain commands, variables, functions, aliases, or any other component you can run directly from a shell. The advantage of a shell script is that you can use it to gather a whole set of commands together so they can be easily run again (without retyping) or even run unattended (such as in a cron job).

You can create shell scripts using your favorite text editor (such as vi). To run, the shell script file must be executable. For example, if you created a shell script with a filename of `myscript.sh`, you could **make it executable** as follows:

```
$ chmod u+x myscript.sh
```

Also, the first line of your bash scripts should always be the following:

```
#!/bin/bash
```

The `#` in this case starts a comment. The `#!` syntax acts as a comment for shells that don't understand this special syntax. The `/bin/bash` part tells any running shell, be it bash or another shell, which program should be used to run the script. (Since historically not all systems came with bash, you will often see `/bin/sh` as the command to run a script.)

As with any command, in addition to being executable, the shell script you create must also either be in your PATH or be identified by its full or relative path when you run it. In other words, if you just try to run your script, you may get the following result:

```
$ myscript.sh  
bash: myscript.sh: command not found
```

In this example, the directory containing `myscript.sh` is not included in your PATH. To correct this problem, you can edit your PATH, copy the script to a directory in your PATH, or enter the full or relative PATH to your script. The four PATH examples just described, respectively, are shown here:

```
$ mkdir ~/bin ; cp myscript.sh ~/bin/ ; PATH=$PATH:~/bin  
$ cp myscript.sh /usr/local/bin  
$ ./myscript.sh  
$ /tmp/myscript.sh
```


Avoid putting a dot (.) into the PATH environment variable to indicate that commands can be run from the current directory. This is a technique that could result in commands with the same filename as important, well-known commands (such as `ls` or `cat`), which could be overridden if a command of the same name exists in the current directory. This can become a major security issue.

Adding Content to Your Script

Although a shell script can be a simple sequence of commands, shell scripts can also be used as you would any programming language. For example, a script can produce different results based on giving it different input. This section describes how to use compound commands, such as `if/then` statements, `case` statements, and `for/while` loops in your shell scripts.

The following example code assigns the string `abc` to the variable `MYSTRING`. It then tests the input to see if it equals `abc` and acts based on the outcome of the test. The test is between the brackets (`[]`):

```
MYSTRING=abc
if [ $MYSTRING = abc ] ; then
echo "The variable is abc"
fi
```

To **negate the test**, use `!=` instead of `=`, as shown in the following:

```
if [ $MYSTRING != abc ] ; then
echo "$MYSTRING is not abc";
fi
```

The following are examples of **testing for numbers**:

```
MYNUMBER=1
if [ $MYNUMBER -eq 1 ] ; then echo "MYNUMBER equals 1"; fi
if [ $MYNUMBER -lt 2 ] ; then echo "MYNUMBER <2"; fi
if [ $MYNUMBER -le 1 ] ; then echo "MYNUMBER <=1"; fi
if [ $MYNUMBER -gt 0 ] ; then echo "MYNUMBER >0"; fi
if [ $MYNUMBER -ge 1 ] ; then echo "MYNUMBER >=1"; fi
```

Let's look at some **tests on filenames**. In this example, you can check if a file exists (`-e`), if it's a regular file (`-f`), or if it is a directory (`-d`). These checks are done with `if/then` statements. If there is no match, then the `else` statement is used to produce the result.

```
filename="$HOME"
if [ -e $filename ] ; then echo "$filename exists"; fi
if [ -f "$filename" ] ; then
    echo "$filename is a regular file"
elif [ -d "$filename" ] ; then
```

```

    echo "$filename is a directory"
else
    echo "I have no idea what $filename is"
fi

```

[Table 3-1](#) shows examples of tests that you can perform on files, strings, and variables.

Table 3-1: Operators for Test Expressions

Operator	Test Being Performed
-a file	Check that the file exists (same as -e)
-b file	Check whether the file is a special block device
-c file	Check whether the file is a character special device (such as a serial device)
-d file	Check whether the file is a directory
-e file	Check whether the file exists (same as -a)
-f file	Check whether the file exists and is a regular file (for example, not a directory, socket, pipe, link, or device file)
-g file	Check whether the file has the set-group-id bit set
-h file	Check whether the file is a symbolic link (same as -L)
-k file	Check whether the file has the sticky bit set
-L file	Check whether the file is a symbolic link (same as -h)
-n string	Check whether the string length is greater than 0 bytes
-O file	Check whether you own the file
-p file	Check whether the file is a named pipe
-r file	Check whether the file is readable by you
-s file	Check whether the file exists and is larger than 0 bytes
-S file	Check whether the file exists and is a socket
-t fd	Check whether the file descriptor is connected to a terminal
-u file	Check whether the file has the set-user-id bit set
-w file	Check whether the file is writable by you
-x file	Check whether the file is executable by you
-z string	Check whether the length of the string is 0 (zero) bytes
expr1 -a expr2	Check whether both the first and the second expressions are true
expr1 -o expr2	Check whether either of the two expressions is true
file1 -nt file2	Check whether the first file is newer than the second file (using the modification timestamp)
file1 -ot file2	Check whether the first file is older than the second file (using the modification timestamp)
file1 -ef file2	Check whether the two files are associated by a link (a hard link or a symbolic link)

<code>var1 = var2</code>	Check whether the first variable is equal to the second variable
<code>var1 -eq var2</code>	Check whether the first variable is equal to the second variable
<code>var1 -ge var2</code>	Check whether the first variable is greater than or equal to the second variable
<code>var1 -gt var2</code>	Check whether the first variable is greater than the second variable
<code>var1 -le var2</code>	Check whether the first variable is less than or equal to the second variable
<code>var1 -lt var2</code>	Check whether the first variable is less than the second variable
<code>var1 != var2</code> <code>var1 -ne var2</code>	Check whether the first variable is not equal to the second variable

Another frequently used construct is the `case` command. Using the `case` statement, you can test for different cases and take an action based on the result. Similar to a `switch` statement in programming languages, `case` statements can take the place of several nested `if` statements.

```
case "$VAR" in
    string1)
        { action1 };;
    string2)
        { action2 };;
    *)
        { default action } ;;
esac
```

You can find examples of `case` usage in the system start-up scripts (`initscripts`) found in the `/etc/init.d/` directory. Each `init` script takes actions based on what parameter was passed to it (`start`, `stop`, and so on) and the selection is done via a large `case` construct.

Note The `init` scripts once stored in the `/etc/init.d` directory and related `/etc/rc?.d` directories are now stored in the `/etc/init` directory in Ubuntu.

The `bash` shell also offers **standard loop constructs**, illustrated by a few examples that follow. In the first example, all the values of the `NUMBER` variable (0 through 9) appear on the `for` line:

```
for NUMBER in 0 1 2 3 4 5 6 7 8 9
do
    echo The number is $NUMBER
done
```

In the following examples, the output from the `ls` command (a list of files) provides the variables that the `for` statement acts on:

```
for FILE in `ls`; do echo $FILE; done
```

Instead of feeding the whole list of values to a `for` statement, you can increment a value and **continue through a while loop until a condition is met**. In the following example, `VAR` begins as 0 and the `while` loop continues to increment until the value of `VAR` becomes 3:

```
"VAR=0"
while [ $VAR -lt 3 ]; do
    echo $VAR
    VAR=$((VAR+1))
done
```

Another way to get the same result as the `while` statement just shown is to use the `until` statement, as shown in the following example:

```
VAR=0
until [ $VAR -eq 3 ]; do echo $VAR; VAR=$((VAR+1)); done
```

If you are just starting with shell programming, refer to the Bash Guide for Beginners (<http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html>). Use that guide, along with reference material such as the bash man page, to step through many examples of good shell scripting techniques.

Summary

Despite improvements in graphical user interfaces, the shell is still the most common method for power users to work with Linux systems. The Bourne Again Shell (bash) is the most common shell used with Linux. It includes many helpful features for recalling commands (history), completing commands, assigning aliases, and redirecting output from and input to commands. You can make powerful commands of your own using simple shell scripting techniques.

Chapter 4

Working with Files

IN THIS CHAPTER

- Setting permissions
- Traversing the filesystem
- Creating/copying files
- Using hard/symbolic links
- Changing file attributes
- Searching for files
- Listing and verifying files

Everything in a Linux filesystem can be viewed as a file. This includes data files, directories, devices, named pipes, links, and other types of files. Associated with each file is a set of information that determines who can access the file and how they can access it. This chapter covers many commands for exploring and working with files.

Understanding File Types

Directories and regular files are by far the file types you will use most often. However, there are several other types of files you will encounter as you use Linux. From the command line, there are many ways you can create, find, and list different types of files.

Files that provide access to the hardware components on your computer are referred to as **device files**. There are character and block devices. There are **hard links** and **soft links** you can use to make the same file accessible from different locations. Less often used directly by regular users are **named pipes** and **sockets**, which provide access points for processes to communicate with each other.

Using Regular Files

Regular files consist of data files (documents, music, images, archives, and so on) and commands (binaries and scripts). You can determine the type of a file using the `file` command. In the following example, you change to the directory containing bash shell documentation (available if you have the `doc-base` package installed) and **use the file**

command to view some of the file types in that directory:

```
$ cd /usr/share/doc/
$ file doc-base/install-docs.html
doc-base/install-docs.html: XML document text
$ file doc-base/copyright
doc-base/copyright: ASCII English text
$ file doc-base/doc-base.html
doc-base/doc-base.html/: directory
$ file doc/doc-base/changelog.gz
doc-base/changelog.gz: gzip compressed data, from Unix, max
compression
$ file shared-mime-info/shared-mime-info-spec.pdf
shared-mime-info/shared-mime-info-spec.pdf: PDF document, version
1.4
```

The `file` command that was run shows document files in the Ubuntu documentation directories of different formats. It can look inside the files and determine that a file contains text that has been compressed, PDF or PostScript that can be sent to a printer, plain text, or HTML (web page) markup. There is even a subdirectory shown, which is unexpected because it has an odd name for a directory (`doc-base.html`).

You can create regular files with any application that can save its data. If you just want to **create some blank files to start** with, there are many ways to do that. Here are two examples:

```
$ touch /tmp/newfile.txt           Create a blank file
$ > /tmp/newfile2.txt             Create a blank file
```

Displaying a long list on a file is another way to determine its file type. For example:

```
$ ls -l /tmp/newfile2.txt           List a file to see its type
-rw-r--r--  1 chris chris 0 Sep 5 14:19 newfile2
```

A dash in the first character of the 10-character permission information (`-rw-r--r--`) indicates that the item is a regular file. (Permissions are explained in the “Setting File/Directory Permissions” section later in this chapter.) Commands are also regular files, but are saved as executables. Here are some examples:

```
$ ls -l /usr/bin/apt-key
-rwxr-xr-x 1 root root 8067 Dec 12 08:48 /usr/bin/apt-key
$ file /usr/bin/apt-key
/usr/bin/apt-key: POSIX shell script, UTF-8 Unicode text executable
$ file /bin/ls
/bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), ...
```

You can see that the `apt-key` command is executable by the `x` settings for owner, group, and others. By running `file` on `apt-key`, you can see that it is a shell script. Commands that are not scripts are binary executables, such as the `ls` command

indicated earlier.

Using Directories

A directory is a container for files and subdirectories. Directories are set up in a hierarchy from the root (/) down to multiple subdirectories, each separated by a slash (/). Directories are called folders when you access them from graphical file managers.

To create new directories for storing your data, you can use the `mkdir` command. Here are examples of using `mkdir` to **create directories in different ways**:

```
$ mkdir /tmp/new           Create "new" directory in /tmp
$ mkdir -p /tmp/a/b/c/new  Create parent directories as needed for
"new"
$ mkdir -m 700 /tmp/new2   Create new2 with drwx--- permissions
```

The first `mkdir` command simply adds the `new` directory to the existing `/tmp` directory. The second example creates directories as needed (subdirectories `a`, `b`, and `c`) to create the resulting `new` directory. The last command adds the `-m` option to set directory permissions as well.

You can **identify the file as a directory** because the first character in the 10-character permission string for a directory is a `d`:

```
$ file /tmp/new
/tmp/new: directory
$ ls -ld /tmp/new
drwxrwxr-x 2 chris chris 4096 Feb 10 15:40 /tmp/new
```

Another thing to notice about directories is that the execute bits (`x`) must be on if you want people to use the directory as their current directory.

Using Symbolic and Hard Links

Instead of copying files and directories to different parts of the filesystem, links can be set up to access that same file from multiple locations. Linux supports both **soft links**(usually called **symbolic links**) and **hard links**.

When you try to open a symbolic link that points to a file or change to one that points to a directory, the command you run acts on the file or directory that is the target of that link. The target has its own set of permissions and ownership that you cannot see from the symbolic link. The symbolic link can exist on a different disk partition than the target. In fact, the symbolic link can exist, even if the target doesn't.

A hard link, alternatively, can only be used on files (not directories) and is basically a way of giving multiple names to the same physical file. Every physical file has at least one hard link, which is commonly thought of as the file itself. Any additional names

(hard links) that point to that single physical file must be on the same partition as the original target file (in fact, one way to tell that files are hard links is that they all have the same inode number). Changing permissions, ownership, date/timestamps, or content of any hard link to a file results in all others being changed as well. However, deleting one link will not remove the file; it will continue to exist until the last link to the file is deleted.

Here are some examples of using the `ln` command to **create hard and symbolic links**:

```
$ touch myfile
$ ln myfile myfile-hardlink
$ ln -s myfile myfile-symlink
$ ls -li myfile*
4460742 -rw-rw-r-- 2 chris chris 0 Feb 10 18:01 myfile
4460742 -rw-rw-r-- 2 chris chris 0 Feb 10 18:01 myfile-hardlink
4460748 lrwxrwxrwx 1 chris chris 6 Feb 10 18:02 myfile-symlink ->
myfile
```

Note that after creating the hard and symbolic link files, I used the `ls -li` command to list the results. The `-li` option shows the inodes associated with each file. You can see that `myfile` and `myfile-hardlink` both have the inode number of 4460742 (signifying the exact same file on the hard disk). The `myfile-symlink` symbolic link has a different inode number. And although the hard link simply appears as a file (-), the symbolic link is identified as a link (l) with wide-open permissions. You won't know if you can access the file the symbolic link points to until you try it or list the link target.

Using Device Files

When applications need to communicate with your computer's hardware, they direct data to **device files**. By convention, device files are stored in the `/dev` directory. Devices are generally divided into block devices (such as storage media) and character devices (such as serial ports and terminal devices).

Note Device files are often called device drivers. In Linux and UNIX, the operating system treats almost everything as a file, hence the term device files.

Each device file is associated with a major number (indicating the type of device) and minor number (indicating the instance number of the device). For example, terminal (tty) devices are represented by major character device 4, while SCSI hard disks are represented by major block device number 8. Here are **examples of device files**:

```
$ ls -l /dev/tty0 /dev/sda1    List character and block special
devices
```



```
brw-rw---- 1 root disk 8, 1 Feb  3 17:24 /dev/sda1
crw--w---- 1 root tty  4, 0 Feb  3 17:24 /dev/tty0
```

A listing of device names and numbers allocated in Linux is available in Ubuntu in the online manual page for the `MAKEDEV` command. Most device files are created automatically for you at boot time or by `udev` when new hardware is connected live (such as when you plug in a USB flash drive). Most people never create device files manually. However, you can **create your own device file** using the `mknod` command. Here's an example:

```
$ sudo mknod /dev/ttyS40 c 4 68      Add device for 41st serial port
$ ls -l /dev/ttyS40                  List new device file
crw-rw---- 1 root dialout 4, 68 Feb  3 17:24 /dev/ttyS4
```

Using Named Pipes and Sockets

When you want to allow one process to send information to another process, you can simply pipe (`|`) the output from one to the input of the other. However, to provide a presence in the filesystem from which a process can communicate with other processes, you can create **named pipes** or **sockets**. Named pipes are typically used for interprocess communication on the local system, while sockets can be used for processes to communicate over a network.

Named pipes and sockets are often set up by applications in the `/tmp` directory. Here are some **examples of named pipes and sockets**:

```
$ ls -l /tmp/.TV-chris/tvtimefifo-local /tmp/.X11-unix/X0
prw----- 1 chris chris 0 Sep 26  2007 /tmp/.TV-chris/tvtimefifo-
local
srwxrwxrwx 1 root  root 0 Sep  4 01:30 /tmp/.X11-unix/X0
```

The first listing is a named pipe set up by the `tvtime` TV card player (note the `p` at the beginning indicating a named pipe). The second listing is a socket set up by the X GUI for interprocess communications.

To **create your own named pipe**, use the `mkfifo` command as follows:

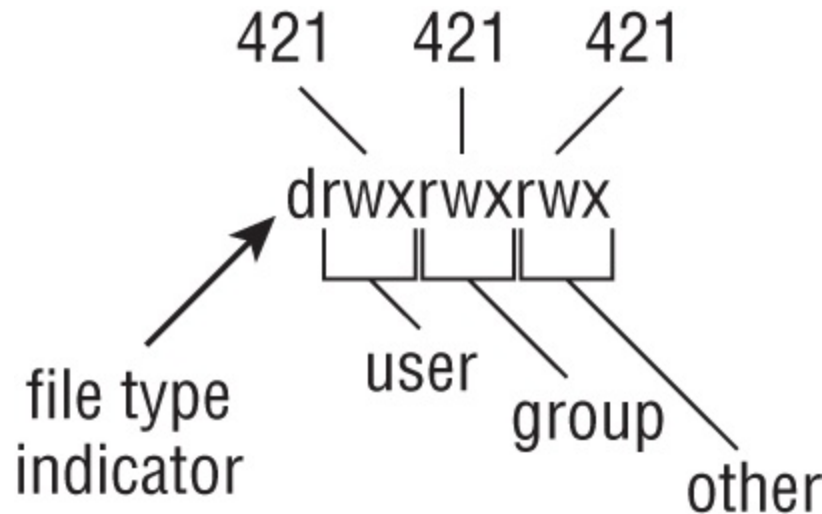
```
$ mkfifo mypipe
$ ls -l mypipe
prw-rw-r-- 1 chris chris 0 Sep 26 00:57 mypipe
```

Setting File/Directory Permissions

The ability to access files, run commands, and change to a directory can be restricted with permission settings for user, group, and other users. When you display a long list

(`ls -l`) of files and directories in Linux, the beginning 10 characters shown indicate what the item is (file, directory, block device, and so on) along with whether or not the item can be read, written, and/or executed. [Figure 4-1](#) illustrates the meaning of those 10 characters.

Figure 4-1: Read, write, and execute permissions are set for files and directories.



To follow along with examples in this section, create a directory called `/tmp/test` and a file called `/tmp/test/hello.txt`. Then do a long listing of those two items, as follows:

```
$ mkdir /tmp/test
$ echo "some text" > /tmp/test/hello.txt
$ ls -ld /tmp/test/ /tmp/test/hello.txt
drwxrwxr-x 2 chris sales 4096 Feb 10 19:24 /tmp/test/
-rw-rw-r-- 1 chris sales  16 Feb 10 19:24 /tmp/test/hello.txt
```

After creating the directory and file, the first character of the long listing shows `/tmp/test` as a directory (`d`) and `hello.txt` as a file (`-`). Other types of files available in Linux that would appear as the first character include character devices (`c`), block devices (`b`) or symbolic links (`l`), named pipes (`p`), and sockets (`s`).

The next nine characters represent the permissions set on the file and directory. The first `rwx` indicates that the owner (`chris`) has read, write, and execute permissions on the directory. Likewise, the group `sales` has the same permissions (`rwx`). Then all other users have only read and execute permissions (`r-x`); the dash indicates the missing write permission. For the `hello.txt` file, the user and members of the group have read and write permissions (`rw-`) and all others have read permission (`r--`).

When you set out to change permissions, each permission can be represented by an octal number (where read is 4, write is 2, and execute is 1) or a letter (`rwx`). Generally speaking, read permission lets you view the contents of the directory, write lets you change (add or modify) the contents of the directory, and execute lets you change to (in

other words, access) the directory.

If you don't like the permissions you see on files or directories you own, you can change those permissions using the `chmod` command.

Changing Permissions with `chmod`

The `chmod` command lets you **change the access permissions of files and directories**.

The following sections show several `chmod` command lines and how access to the directory or file changes.

`chmod 0700` directory

Original Permission: `any`

New Permission: `drwx-----`

Description: The directory's owner can read or write files in that directory as well as change to it. All other users (except root) have no access.

`chmod 0711` directory

Original Permission: `any`

New Permission: `drwx--x--x`

Description: Same as for the owner. All others can change to the directory, but not view or change files in the directory. This can be useful for server hardening, where you prevent someone from listing directory contents but allow access to a file in the directory if someone already knows it's there.

`chmod go+r` directory

Original Permission: `drwx-----`

New Permission: `drwxr--r--`

Description: Adding read permission to a directory may not give desired results. Without execute on, others can't view the contents of any files in that directory.

`chmod 0777` directory and `chmod a=rwx` directory

Original Permission: `any`

New Permission: `drwxrwxrwx`

Description: All permissions are wide open.

chmod 0000 directory and chmod a-rwx directory

Original Permission:any

New Permission:d-----

Description: All permissions are closed. Good to protect a directory from errant changes. However, backup programs that run as non-root may fail to back up the directory's contents.

chmod 666 file

Original Permission:any

New Permission:-rw-rw-rw-

Description: Open read-write permissions completely on a file.

chmod go-rw file

Original Permission:-rw-rw-rw-

New Permission:-rw-----

Description: Don't let anyone except the owner view, change, or delete the file.

chmod 644 file

Original Permission:any

New Permission:-rw-r--r--

Description: Only the owner can change or delete the file, but all can view it.

General Information on Changing Permissions

This section discusses what you should know generally on changing permissions using the command line.

The first 0 in the mode line can usually be dropped (so you can use 777 instead of 0777). That placeholder has special meaning. It is an octal digit that can be used on commands (executables) to indicate that the command can run as a set-UID program (4), run as a set-GID program (2), or become a sticky program (1). With set-UID and set-GID, the command runs with the assigned user or group permissions (instead of running with permission of the user or group that launched the command).

Warning SUID should not be used on shell scripts. Here is a warning from the Linux Security HOWTO: “SUID shell scripts are a serious security risk, and for this reason the kernel will not honor them. Regardless of how secure you think the shell script is, it can be exploited to give the cracker a root shell.”

Having the sticky bit on for a directory keeps users from removing or renaming files from that directory that they don't own (/tmp is an example). Given the right permission settings, however, users can change the contents of files they don't own in a sticky bit directory. The final permission character is `t` instead of `x` on a sticky directory. A command with sticky bit on used to cause the command to stay in memory, even while not being used. This is an old UNIX feature that is not supported in Linux.

The `-R` option is a handy feature of the `chmod` command. With `-R`, you **can recursively change permissions of all files and directories starting from a point in the filesystem**. Here are some examples:

```
$ sudo chmod -R 700 /tmp/test      Open permission to owner below
/tmp/test
$ sudo chmod -R 000 /tmp/test      Close all permissions below
/tmp/test
$ sudo chmod -R a+rwX /tmp/test    Open all permissions below
/tmp/test
```

Note that the `-R` option is inclusive of the directory you indicate. So the permissions in the preceding code, for example, would change for the `/tmp/test` directory itself, and not just for the files and directories below that directory.

Setting the umask

Permissions given to a file or directory are assigned originally at the time that item is created. How those permissions are set is based on the user's current umask value. Using the `umask` command, you can **set the permissions given to files and directories when you create them**.

```
$ umask 0066      Make directories drwx--x--x and files -rw-----
$ umask 0077      Make directories drwx----- and files -rw-----
$ umask 0022      Make directories drwxr-xr-x and files -rw-r--r--
$ umask 0777      Make directories d----- and files -----
```

Changing Ownership

When you create a file or directory, your user account is assigned to that file or

directory. So is your primary group. As root user, you can **change the ownership (user) and group assigned to a file to a different user and/or group** using the `chown` and `chgrp` commands. Here are some examples:

```
$ chown chris test/           Change owner to chris
$ chown chris:market test/      Change owner to chris and group to
market
$ chgrp market test/          Change group to market
$ chown -R chris test/         Change all files below test/ to owner
chris
```

The recursive option to `chown` (`-R`) just shown is useful if you need to **change the ownership of an entire directory structure**. As with `chmod`, using `chown` recursively changes permissions for the directory named, along with its contents. You might use `chown` recursively when a person leaves a company or stops using your web service. You can use `chown -R` to reassign their entire `/home` directory to a different user.

Related commands for changing group assignments and passwords include `newgrp` and `gpasswd`, as well as the `/etc/gshadow` file.

Traversing the Filesystem

Basic commands for changing directories (`cd`), checking the current directory (`pwd`), and listing directory contents (`ls`) are well-known to even casual shell users. So this section focuses on some less-common options to those commands, as well as other lesser-known features for moving around the filesystem. Here are some quick examples of `cd` for **moving around the filesystem**:

```
$ cd                          Change to your home directory
$ cd $HOME                     Change to your home directory
$ cd ~                         Change to your home directory
$ cd ~chris                   Change to chris' home directory
$ cd -                        Change to previous working directory
$ cd $OLDPWD                  Change to previous working directory
$ cd ~/public_html           Change to public_html in your home
directory
$ cd ..                       Change to parent of current directory
$ cd /usr/bin                 Change to usr/bin from root directory
$ cd usr/bin                 Change to usr/bin beneath current
directory
```

If you want to **find out what your current directory is**, use `pwd` (print working directory):

```
$ pwd
/home/chris
```

Creating symbolic links is a way to access a file from other parts of the filesystem (see the section “Using Symbolic and Hard Links” earlier in the chapter for more information on symbolic and hard links). However, symbolic links can cause some confusion about how parent directories are viewed. The following commands **create a symbolic link** to the `/tmp` directory from your home directory and show how to tell where you are related to a linked directory:

```
$ cd $HOME
$ ln -s /tmp tmp-link
$ ls -l tmp-link
lrwxrwxrwx 1 chris chris 13 Mar 24 12:41 tmp-link -> /tmp
$ cd tmp-link/
$ pwd
/home/chris/tmp-link
$ pwd -P
/tmp
$ pwd -L
/home/chris/tmp-link
$ cd -L ..
$ pwd
/home/chris
$ cd tmp-link
$ cd -P ..
$ pwd
/
```

Using the `-P` and `-L` options to `pwd` and `cd`, you can **work with symbolically linked directories in their permanent or link locations**, respectively. For example, `cd -L ..` takes you up one level to your home directory, whereas `cd -P ..` takes you up one level above the permanent directory (`/`). Likewise, `-P` and `-L` options to `pwd` show permanent and link locations.

Bash can remember a list of working directories. Such a list can be useful if you want to return to previously visited directories. That list is organized in the form of a stack. Use `pushd` and `popd` **to add and remove directories**:

```
$ pwd
/home/chris
$ pushd /usr/share/man/
/usr/share/man ~
$ pushd /var/log/
/var/log /usr/share/man ~
$ dirs
/var/log /usr/share/man ~
$ dirs -v
0 /var/log
1 /usr/share/man
```

```
2 ~
$ popd
/usr/share/man ~
$ pwd
/usr/share/man
$ popd
~
$ pwd
/home/chris
```

The `dirs`, `pushd`, and `popd` commands can also be used to manipulate the order of directories on the stack. For example, `pushd -0` pushes the last directory on the stack to the top of the stack (making it the current directory). The `pushd -2` command pushes the third directory from the bottom of the stack to the top.

Copying Files

Provided you have write permission to the target directory, copying files and directories can be done with some fairly simple commands. The standard `cp` command will **copy a file to a new name or the same name in a new directory**, with a new timestamp associated with the new file. Other options to `cp` let you retain date/timestamps, copy recursively, and prompt before overwriting. Here are some examples:

```
$ cd ; touch index.html
$ mkdir /tmp/html
$ cp -i index.html /tmp/html/
$ cp -il index.html /tmp/html/index2.html
$ mkdir /tmp/back
$ cp -a /tmp/html /tmp/back/
$ cp -R /tmp/html /tmp/back/
```

The preceding examples show ways of copying files. In the first `cp` example, if an `index.html` file exists in `/tmp/html`, you are prompted before overwriting it with the new file. In the next example, the `index.html` file is hard-linked to a file of the same name in the `/tmp/html` directory. In that case, because both hard links point to the same file, editing the file from either location will change the contents of the file in both locations. (The link can only be made if `/tmp/html` and your home directory are in the same filesystem.)

The `cp -a` command copies all files below the `/tmp/html` directory, retaining all ownership and permission settings. If, for example, `/tmp/back` represented a USB flash drive, that command would be a way to copy the contents of your web server to that drive. The `-R` option also recursively copies a directory structure, but assigns ownership to the current user and adds current date/timestamps.

The `dd` command is another way to **copy data**. This command is very powerful because on Linux systems, everything is a file, including hardware peripherals. Here is an example:

```
$ dd if=/dev/zero of=/tmp/mynullfile count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 4.5447e-05 s, 11.3 MB/s
```

`/dev/zero` is a special file that generates null characters. In the example just shown, the `dd` command takes `/dev/zero` as input file and outputs to `/tmp/mynullfile`. The `count` is the number of blocks. By default, a block is 512 bytes. The result is a 512-bytes-long file full of null characters. You could use `less` or `vi` to view the contents of the file. However, a better tool to view the file would be the `od` (Octal Dump) command:

```
$ od -vt x1 /tmp/mynullfile      View an octal dump of a file
```

Here's another example of the `dd` command:

```
$ dd if=/dev/zero of=/tmp/mynullfile count=10 bs=2
10+0 records in
10+0 records out
20 bytes (20 B) copied, 0.000173996 s, 115 kB/s
```

This time, set the block size to 2 bytes and copy 10 blocks (20 bytes). The following command **clones the first partition of the primary master IDE drive** to the second partition of the primary slave IDE drive (back up all data before trying anything like this):

```
$ sudo dd if=/dev/hda1 of=/dev/hdb9
```

Warning Be very careful with this command. You normally do not want to blindly overwrite parts of your hard drives.

The next example **makes a compressed backup** of the first partition of the primary master IDE drive. Typically the partition should be unmounted before a backup such as this.

```
$ sudo umount /dev/hda1
$ sudo dd if=/dev/hda1 | gzip > bootpart.gz
```

The following command copies an ISO image file from a CD or DVD to your USB flash drive (assuming the drive appears as `/dev/sdb1`):

```
$ sudo dd if=whatever.iso of=/dev/sdb9
```

Note that this command is making a binary copy of the bytes in the file, which may not be what you want to do.

This next example copies the Master Boot Record from the primary master IDE hard drive to a file named `mymbrfile`:

```
$ dd if=/dev/hda of=mymbrfile bs=512 count=1
```

If you want to make a copy of the ISO image that was burned to a CD or DVD, insert that medium into your CD/DVD drive and (assuming `/dev/cdrom` is associated with your computer's CD drive) type the following command:

```
$ dd if=/dev/cdrom of=whatever.iso
```

Note Ubuntu also creates `/dev/cdrw` and `/dev/dvd` devices files as well as `/dev/cdrom`.

Changing File Attributes

Files and directories in Linux filesystems all have read, write, and execute permissions associated with user, group, and others. However, there are also other attributes that can be attached to files and directories that are specific to certain filesystem types.

Files on `ext2` and `ext3` filesystems have special attributes that you may choose to use. You can **list these attributes** with the `lsattr` command. Most attributes are obscure and not turned on by default. Here's an example of using `lsattr` to see some files' attributes:

```
$ lsattr /etc/host*
-----e- /etc/host.conf
-----e- /etc/hosts
-----e- /etc/host.allow
-----e- /etc/host.deny
$ lsattr -aR /tmp/ | less      Recursively list all /tmp
attributes
```

The dashes represent 13 `ext2/ext3` attributes that can be set. None are on by default. Those attributes are the following: `a` (append only), `c` (compressed), `d` (no dump), `e` (extent format), `i` (immutable), `j` (data journaling), `s` (secure deletion), `t` (no tail-merging), `u` (undeletable), `A` (no atime updates), `D` (synchronous directory updates), `S` (synchronous updates), and `T` (top of directory hierarchy). You can **change these attributes** using the `chattr` command. Here are some examples:

```
$ sudo chattr +d whatever.iso
$ sudo chattr +A -R /home/chris/images/*
$ sudo chattr +d ubuntu.iso
$ lsattr whatever.iso /home/chris/images/* ubuntu.iso
----i----- whatever.iso
```

```
-----A----- /home/chris/images/einstein.jpg
-----A----- /home/chris/images/goth.jpg
-----d----- ubuntu.iso
```

As shown in the preceding example, with the `+i` option set, the `whatever.iso` file becomes immutable, meaning that it can't be deleted, renamed, or changed, or have a link created to it. Here, this prevents any arbitrary changes to the file. (Not even the root user can change the file until the `i` attribute is gone.) You can use this to help protect system files.

The `-R` option in the example recursively sets the `+A` option, so all files in the `images` directory and below can't have access times (`atime` record) modified. Setting `A` attributes can save some disk I/O on laptops or flash drives. If you use the `dump` command to back up your `ext2/ext3` filesystems, the `+d` option can prevent selected files from being backed up. In this case, you can choose to not have a large ISO image backed up.

To **remove an attribute** with `chattr`, use the minus sign (`-`). For example:

```
$ sudo chattr -i whatever.iso
```

Note Crackers who successfully break into a machine will often replace some system binaries (such as `ls` or `ps`) with corrupt versions and make them immutable. It's a good idea to occasionally check the attributes set for your executables (in `/bin`, `/usr/bin`, `/sbin`, and `/usr/sbin`, for example).

Searching for Files

Ubuntu keeps a database of all the files in the filesystem (with exceptions defined in `/etc/updatedb.conf`) using features of the `mlocate` package. The `locate` command allows you to search that database. (In Ubuntu, the `locate` command is a symbolic link to the secure version of the command, `mlocate`.)

The results of the `locate` command come back instantly because the database is searched and not the actual filesystem. Before `locate` was available, most Linux users ran the `find` command to find files in the filesystem. Both `locate` and `find` are covered here.

Finding Files with locate

Because the database contains the name of many different kinds of files in the

filesystem, and not just commands, you can use `locate` to **find commands, devices, man pages, data files, or anything else identified by a name** in the filesystem. Here is an example:

```
$ locate e1000
/lib/modules/2.6.20-16-generic/kernel/drivers/net/e1000
/lib/modules/2.6.20-16-generic/kernel/drivers/net/e1000/e1000.ko
/lib/modules/2.6.20-15-generic/kernel/drivers/net/e1000
/lib/modules/2.6.20-15-generic/kernel/drivers/net/e1000/e1000.ko
/usr/src/linux-headers-2.6.20-16-generic/include/config/e1000
/usr/src/linux-headers-2.6.20-16-
generic/include/config/e1000/napi.h
/usr/src/linux-headers-2.6.20-16-generic/include/config/e1000.h
/usr/src/linux-headers-2.6.20-15-generic/include/config/e1000
/usr/src/linux-headers-2.6.20-15-
generic/include/config/e1000/napi.h
/usr/src/linux-headers-2.6.20-15-generic/include/config/e1000.h
/usr/src/linux-headers-2.6.20-15/include/config/e1000.h
/usr/src/linux-headers-2.6.20-15/drivers/net/e1000
/usr/src/linux-headers-2.6.20-15/drivers/net/e1000/Makefile
/usr/src/linux-headers-2.6.20-16/include/config/e1000.h
/usr/src/linux-headers-2.6.20-16/drivers/net/e1000
/usr/src/linux-headers-2.6.20-16/drivers/net/e1000/Makefile
```

The preceding example found two versions of the `e1000.ko` and `e1000.ko` kernel modules. `locate` is case sensitive unless you use the `-i` option. Here's an example:

```
$ locate -i itco_wdt
/lib/modules/2.6.20-16-
generic/kernel/drivers/char/watchdog/iTCO_wdt.ko
/lib/modules/2.6.20-15-
generic/kernel/drivers/char/watchdog/iTCO_wdt.ko
```

The `mlocate` package includes a cron job that runs the `updatedb` command once per day to update the `locate` database of files.

To **update the locate database immediately**, you can run the `updatedb` command manually:

```
$ sudo updatedb
```

As mentioned earlier, the `/etc/updatedb.conf` file contains information that excludes certain directories and filesystem types from having their files gathered by the update database. Here is an example of information in the `updatedb.conf` file:

```
PRUNEPATHS="/tmp /var/spool /media /home/.ecryptfs"
PRUNEFS="NFS nfs nfs4 rpc_pipefs afs binfmt_misc proc smbfs
autofs iso9660 ncpfs coda devpts ftpfs devfs mfs shfs sysfs cifs
lustre_lite tmpfs usbfs udf fuse.glusterfs fuse.sshfs curlftpfs
ecryptfs fusesmb devtmpfs"
```

The `PRUNEPATHS` variable sets directories for which files and subdirectories are loaded into the `locate` database. These include places where temporary files are stored (such as `/var/spool` and `/tmp`). Filesystem types that are excluded by the `PRUNEFS` line include those of temporary filesystem types and remotely mounted filesystem types (such as `nfs` and `cifs`).

Locating Files with `find`

Before the days of `locate`, users would utilize the `find` command to find files. Although `locate` will come up with a file faster, `find` has many other powerful options for finding files based on attributes other than the name.

Note Searching the entire filesystem can take a long time to complete. Before searching the whole filesystem, consider searching a subset of the filesystem or excluding certain directories or remotely mounted filesystems.

This example searches the root filesystem (`/`) recursively for files named `e100`:

```
$ find / -name "e100*" -print
find: /usr/lib/audit: Permission denied
find: /usr/libexec/utempter: Permission denied
/sys/module/e100
/sys/bus/pci/drivers/e100
...
```

Running `find` as a normal user can result in long lists of `Permission denied` as `find` tries to enter a directory you do not have permissions to. You can **filter out the inaccessible directories**:

```
$ find / -name e100 -print 2>&1 | grep -v "Permission denied"
```

or **send all errors to the `/dev/null` bit bucket**:

```
$ find / -name e100 -print 2> /dev/null
```

Because searches with `find` are case sensitive and must match the name exactly (`e100` won't match `e100.ko`), you can use **regular expressions to make your searches more inclusive**. (To be case-insensitive, you could use `-iname` instead.) Here's an example:

```
$ find / -name 'e100*' -print
/lib/modules/2.6.20-16-generic/kernel/drivers/net/e1000
/lib/modules/2.6.20-16-generic/kernel/drivers/net/e1000/e1000.ko
/lib/modules/2.6.20-16-generic/kernel/drivers/net/e100.ko
/lib/modules/2.6.20-15-generic/kernel/drivers/net/e1000
/lib/modules/2.6.20-15-generic/kernel/drivers/net/e1000/e1000.ko
/lib/modules/2.6.20-15-generic/kernel/drivers/net/e100.ko
```

```
/usr/src/linux-headers-2.6.20-16-generic/include/config/e100.h
/usr/src/linux-headers-2.6.20-16-generic/include/config/e1000
/usr/src/linux-headers-2.6.20-16-generic/include/config/e1000.h
/usr/src/linux-headers-2.6.20-15-generic/include/config/e100.h
/usr/src/linux-headers-2.6.20-15-generic/include/config/e1000
/usr/src/linux-headers-2.6.20-15-generic/include/config/e1000.h
/usr/src/linux-headers-2.6.20-15/include/config/e100.h
/usr/src/linux-headers-2.6.20-15/include/config/e1000.h
/usr/src/linux-headers-2.6.20-15/drivers/net/e1000
/usr/src/linux-headers-2.6.20-16/include/config/e100.h
/usr/src/linux-headers-2.6.20-16/include/config/e1000.h
/usr/src/linux-headers-2.6.20-16/drivers/net/e1000
```

You can also **find files based on timestamps**. The following command line finds files in `/usr/bin/` that have been accessed in the past two minutes:

```
$ find /usr/bin/ -amin -2 -print
/usr/bin/
/usr/bin/find
```

The following command line finds files that have not been accessed in `/home/chris` for more than 60 days:

```
$ find /home/chris/ -atime +60
```

Use the `-type d` option to **find directories**. The following command line finds all directories under `/etc` and redirects stderr to the bit bucket (`/dev/null`):

```
$ find /etc -type d -print 2> /dev/null
```

This command line finds files in `/sbin` by **permissions** that match 755:

```
$ find /sbin/ -perm 755 -print
```

The `exec` option to `find` is very powerful because it lets you **act on the files found with the find command**. The following command finds all the files in `/var` owned by the user `chris` (must be a valid user) and executes the `ls -l` command on each one:

```
$ find /var -user chris -exec ls -l {} \;
```

An alternative to `find`'s `exec` option is `xargs`:

```
$ find /var -user chris -print | xargs ls -l
```

There are big differences on how the two commands just shown operate, leading to very different performance. The `find -exec` spawns the command `ls` for each result it finds. The `xargs` command works more efficiently by passing many results as input to a single `ls` command. (Note that for the `ls -l` command, you can use `-ls` on the `find` command line instead of using `xargs ls -l`.)

To **negate a search criterion**, place an exclamation point (!) before it. The next example finds all the files that are not owned by the group `root` and are regular files, and

then does an `ls -l` on each:

```
$ find / ! -group root -type f -print 2> /dev/null | xargs ls -l
```

The next example finds the files in `/sbin` that are regular files and are not writable by others, and then feeds them to an `ls -l` command:

```
$ find /sbin/ -type f ! -perm /o+w -print | xargs ls -l
-rwxr-xr-x 1 root root      3056 2007-03-07 15:44
/sbin/acpi_available
-rwxr-xr-x 1 root root    43204 2007-02-18 20:18 /sbin/alsactl
```

Finding files by size is a great way to determine what is filling up your hard disks. The following command line finds all files that are greater than 10MB (`+10M`), lists those files from largest to smallest (`ls -ls`), and directs that list to a file (`/tmp/bigfiles.txt`):

```
$ find / -xdev -size +10M -print | xargs ls -ls > /tmp/bigfiles.txt
```

In this example, the `-xdev` option prevents any mounted filesystems, besides the root filesystem, from being searched. This is a good way to keep the `find` command from searching the `/proc` directory and any remotely mounted filesystems, as well as other locally mounted filesystems.

Using Other Commands to Find Files

Other commands for finding files include the `whereis` and `which` commands. Here are some examples of those commands:

```
$ whereis man
man: /usr/bin/man /usr/X11R6/bin/man /usr/bin/X11/man
    /usr/local/man
    /usr/share/man /usr/share/man/man1/man.1.gz
    /usr/share/man/man7/man.7.gz
$ which ls
/bin/ls
```

The `whereis` command is useful because it **finds** not only commands, but also **man pages and configuration files associated with a command**. From the example of `whereis` for the word `man`, you can see the `man` executable, its configuration file, and the location of man pages for the `man` command. The `which` example shows where the `ls` executable is (`/bin/ls`). The `which` command is useful when you're looking for the actual location of an executable file in your `PATH`, as in this example:

```
$ dpkg-query -S `which ps`
procps: /bin/ps
```

Finding Out More about Files

Now that you know how to find files, you can get more information about those files. Using less common options to the `ls` command enables you to list information about a file that you won't see when you run `ls` without options. Commands such as `file` help you identify a file's type. With `md5sum` and `sha1sum`, you can verify the validity of a file.

Listing Files

Although you are probably quite familiar with the `ls` command, you may not be familiar with many of the useful options for `ls` that can help you find out a lot about the files on your system. Here are some examples of **using ls to display long lists (-l)** of files and directories:

```
$ ls -l           Files and directories in current directory
$ ls -la          Includes files/directories beginning with dot (.)
$ ls -lt          Orders files by time recently changed
$ ls -lu          Orders files by time recently accessed
$ ls -ls          Orders files by size
$ ls -li          Lists the inode associated with each file
$ ls -ln          List numeric user/group IDs, instead of names
$ ls -lh          List file sizes in human-readable form (K, M, etc.)
$ ls -lR          List files recursively, from current and
subdirectories
```

When you list files, there are also ways to **have different types of files appear differently** in the listing:

```
$ ls -F           Add a character to indicate file type
myfile-symlink@  config/  memo.txt  pipefile|  script.sh*
xpid.socket=
$ ls --color=always  Show file types as different colors
$ ls -C           Show files listing in columns
```

In the `-F` example, the output shows several different file types. The `myfile-symlink@` indicates a symbolic link to a directory, `config/` is a regular directory, `memo.txt` is a regular file (no extra characters), `pipefile|` is a named pipe (created with `mkfifo`), `script.sh*` is an executable file, and `xpid.socket=` is a socket. The next two examples display different file types in different colors and lists output in columns, respectively.

Verifying Files

When files such as software packages and CD or DVD images are shared over the Internet, often each has a SHA1SUM or MD5SUM file that is published with it. Those files contain checksums that can be used to make sure that the file you downloaded is exactly the one that the repository published.

The following are examples of the `md5sum` and `sha1sum` commands being used to produce checksums of files:

```
$ md5sum whatever.iso
d41d8cd98f00b204e9800998ecf8427e  whatever.iso
$ sha1sum whatever.iso
da39a3ee5e6b4b0d3255bfef95601890afd80709  whatever.iso
```

Which command you choose depends on whether the provider of the file you are checking distributed `md5sum` or `sha1sum` information. There are also other `sha*` commands that use a higher number of bits for encryption. For example, here is what some of the md5 hashes for the Ubuntu Quantal Quetzal distribution ISO images looked like:

```
7b7c56c74008da7d97bd49669c8a045d  ubuntu-12.10-desktop-amd64+mac.iso
7ad57cadae955bd04019389d4b9c1dcb  ubuntu-12.10-desktop-amd64.iso
b4191c1d1d6fdf358c154f8bf86b97dd  ubuntu-12.10-desktop-i386.iso
b8a4d9513ed53dc1be05576113b113e8  ubuntu-12.10-server-amd64+mac.iso
4bd3270bde86d7e4e017e3847a4af485  ubuntu-12.10-server-amd64.iso
...
```

Within each Ubuntu ISO image, there is also an `mdsum.txt` file in the top-level directory of that image. This file lists the MD5 checksums for all files on the CD or DVD image.

Any time you have stored the `md5sums` in a file, you can use that file to check the files it contains. That way, you can validate the `md5sums` for many files at once. For example, I mounted the ISO image `ubuntu-11.10-desktop-amd64.iso` and, from the top level of that image, I ran the `md5sum -c` command as follows:

```
$ md5sum -c md5sum.txt

./casper/initrd.lz: OK
./casper/filesystem.manifest-remove: OK
./casper/filesystem.manifest: OK
./casper/filesystem.squashfs: OK
./casper/vmlinuz: OK
./casper/filesystem.size: OK
./dists/oneiric/Release: OK
...
```

To verify only one of the files listed in the file, you could do something like the following:

```
$ cat md5sum.txt | grep initrd.lz | md5sum -c
./dists/feisty/Release.gpg: OK
```

If you had an `SHA1SUM` file instead of an `md5sum.txt` file to check against, you could use the `shasum` command in the same way. By combining the `find` command described earlier in this chapter with the `md5sum` command, you can verify any part of your filesystem. For example, here's how to **create an MD5 checksum for every file in the `/etc` directory** so they can be checked later to see if any have changed:

```
$ sudo find /etc -type f -exec md5sum {} \; > /tmp/md5.lst 2>
/dev/null
```

The result of the previous command line is a `/tmp/md5.lst` file that contains a 128-bit checksum for every file in the `/etc` directory. Later, you could type the following command to see if any of those files have changed:

```
$ cd /etc
$ md5sum -c /tmp/md5.list | grep -v 'OK'
./hosts.allow: FAILED
md5sum: WARNING: 1 of 1668 computed checksums did NOT match
```

As you can see from the output, only one file changed (`hosts.allow`). So the next step is to check the changed file and see if the changes to that file were intentional.

Summary

There are dozens of commands for exploring and working with files in Linux. Commands such as `chmod` can change the permissions associated with a file, whereas commands that include `lsattr` and `chattr` can be used to list and change file attributes that are associated with `ext2` and `ext3` filesystem types.

To move around the filesystem, people use the `cd` command most often. However, to move repeatedly among the same directories, you can use the `pushd` and `popd` commands to work with a stack of directories.

Copying files is done with the `cp` command. However, the `dd` command can be used to copy files (such as disk images) from a device (such as a CD-ROM drive). For creating directories, you can use the `mkdir` command.

Instead of keeping multiple copies of a file around on the filesystem, you can use symbolic links and hard links to have multiple filenames point to the same file or directory. Symbolic links can be anywhere in the filesystem, while hard links must exist on the same partition that the original file is on.

To search for files, Linux offers the `locate` and `find` commands. To verify the integrity of files you download from the Internet, you can use the `md5sum` and `shasum`

commands.

Chapter 5

Manipulating Text

IN THIS CHAPTER

- Matching text with regular expressions
- Editing text files with vi, JOE, or nano
- Using graphical text editors
- Listing text with cat, head, and tail
- Paging text with less and more
- Paginating text with pr
- Searching for text with grep
- Counting words, lines, and characters with wc
- Sorting output with sort
- Stream editing with sed, tr, cut, and awk
- Searching binaries for text with strings
- Finding differences in files with diff
- Converting text files with unix2dos/dos2unix

With only a shell available on the first UNIX systems (on which Linux was based), using those systems meant dealing primarily with commands and plain text files. Documents, program code, configuration files, e-mail, and almost anything you created or configured was represented by text files. To work with those files, early developers created many text manipulation tools.

Despite having graphical tools for working with text, most seasoned Linux users find command line tools to be more efficient and convenient. Text editors such as vi (vim), Emacs, JOE, nano, and Pico are available with most Linux distributions. Commands such as `grep`, `sed`, and `awk` can be used to find, and possibly change, pieces of information within text files.

This chapter shows you how to use many popular commands for working with text files in Ubuntu. It also explores some of the less common uses of text manipulation commands that you might find interesting.

Matching Text with Regular Expressions

Many of the tools for working with text enable you to use regular expressions, sometimes referred to as regex, to identify the text you are looking for based on some pattern. You can use these patterns to find text within a text editor or use them with search commands to scan multiple files for the strings of text you want.

A regex search pattern can include a specific string of text (as in a word such as Linux) or a location (such as the end of a line or the beginning of a word). It can also be specific (find just the word hello) or more inclusive (find any word beginning with h and ending with o).

Appendix B includes reference information for shell metacharacters that can be used in conjunction with regular expressions to do the exact kinds of matches you are looking for. This section shows examples of using regular expressions with several different tools you encounter throughout this chapter.

The list that follows shows some examples that use basic regular expressions to match text strings.

Many different types of regular expressions are used in examples throughout this chapter. Keep in mind that not every command that incorporates regex uses its features the same way.

Expression	Matches
<code>a*</code>	<code>a</code> , <code>ab</code> , <code>abc</code> , and <code>aecjeich</code>
<code>^a</code>	Any "a" appearing at the beginning of a line
<code>*a\$</code>	Any "a" appearing at the end of a line
<code>a.c</code>	Three-character strings that begin with a and end with c
<code>[bcf]at</code>	<code>bat</code> , <code>cat</code> , or <code>fat</code>
<code>[a-d]at</code>	<code>aat</code> , <code>bat</code> , <code>cat</code> , <code>dat</code> , but not <code>Aat</code> , <code>Bat</code> , and so on
<code>[A-D]at</code>	<code>Aat</code> , <code>Bat</code> , <code>Cat</code> , and <code>Dat</code> , but not <code>aat</code> , <code>bat</code> , and so on
<code>1[3-5]7</code>	<code>137</code> , <code>147</code> , and <code>157</code>
<code>\tHello</code>	A tab character preceding the word <code>Hello</code>
<code>\.[tT][xX][Tt]</code>	<code>.txt</code> , <code>.TXT</code> , <code>.TxT</code> , or other case combinations

Editing Text Files

There are many text editors in the Linux/UNIX world. The editor that is most common is `vi`, which can be found virtually on any UNIX or Linux system available today. That is why knowing how to at least make minor file edits in `vi` is a critical skill for any Linux administrator. One day, if you find yourself in a minimalist, foreign Linux environment trying to bring a server back online, `vi` is the tool that will almost always be there.

On Ubuntu, make sure you have the `vim-enhanced` package installed. Vim (Vi Improved) with the `vim-enhanced` package will provide the most up-to-date, feature-

rich, and user-friendly vi editor. For more details about using vi, refer to Appendix A.

Note Ubuntu installs vim by default.

Traditionally, the other popular UNIX text editor has been Emacs and its more graphical variant, XEmacs. Emacs is a powerful multi-function tool that can also act as a mail/news reader or shell and perform other functions. Emacs is also known for its very complex series of keyboard shortcuts that require three arms to execute properly.

In the mid-90s, Emacs was ahead of vi in terms of features. Now that Vim is widely available, both can provide all the text editing features you'll ever need. If you are not already familiar with either vi or Emacs, I recommend you start by learning vi.

Many other command line and GUI text editors are available for Linux. Text-based editors that you may find to be simpler than vi and Emacs include JED, JOE, and nano. Start any of those editors by typing its command name, optionally followed by the filename you want to edit. The following sections offer some quick descriptions of how to use each of those editors.

Using the JOE Editor

If you have used classic word processors such as WordStar that worked with text files, you might be comfortable with the JOE editor. To use JOE, install the joe package. To use the spell checker in JOE, make sure the Aspell package is installed. (Ubuntu installs Aspell by default.) To install JOE, run the following command:

```
$ sudo apt-get install joe
```

With JOE, instead of entering a command or text mode, you are always ready to type. To move around in the file, you can use control characters or the arrow keys. To **open a text file for editing**, just type **joe** and the filename or use some of the following options:

\$ <u>joe memo.txt</u>	Open memo.txt for editing
\$ <u>joe -wordwrap memo.txt</u>	Turn on wordwrap while editing
\$ <u>joe -lmargin 5 -tab 5 memo.txt</u>	Set left margin to 5 and tab to 5
\$ <u>joe +25 memo.txt</u>	Begin editing on line 25

To **add text**, just begin typing. You can use **keyboard shortcuts** for many functions. Use arrow keys to move the cursor left, right, up, or down. Use the Delete key to delete text under the cursor or the Backspace key to erase text to the left of the cursor. Press Enter to add a line break. Press Ctrl+k+h to see the help screen. The following list shows the most commonly used control keys for editing in JOE.

<u>Key Combo</u>	<u>Result</u>
<u>Cursor</u>	

Ctrl+b	Left
Ctrl+p	Up
Ctrl+f	Right
Ctrl+n	Down
Ctrl+z	Previous word
Ctrl+x	Next word
<u>Search</u>	
Ctrl+k+f	Find text
Ctrl+l	Find next
<u>Block</u>	
Ctrl+k+b	Begin
Ctrl+k+k	End
Ctrl+k+m	Move block
Ctrl+k+c	Copy block
Ctrl+k+w	Write block to file
Ctrl+k+y	Delete block
Ctrl+k+/ <u>Misc</u>	Filter
Ctrl+k+a	Center line
Ctrl+t	Options
Ctrl+r	Refresh
<u>File</u>	
Ctrl+k+e	Open new file to edit
Ctrl+k+r	Insert file at cursor
Ctrl+k+d	Save
<u>Goto</u>	
Ctrl+u	Previous screen
Ctrl+v	Next screen
Ctrl+a	Line beginning
Ctrl+e	End of line
Ctrl+k+u	Top of file
Ctrl+k+v	End of file
Ctrl+k+l	To line number
<u>Delete</u>	
Ctrl+d	Delete character
Ctrl+y	Delete line
Ctrl+w	Delete word right
Ctrl+o	Delete word left
Ctrl+j	Delete line to right
Ctrl+-	Undo
Ctrl+6	Redo
<u>Exit</u>	
Ctrl+k+x	Save and quit
Ctrl+c	Abort
Ctrl+k+z	Shell
Ctrl+[+n	Word
Ctrl+[+l	File

Using the Pico and nano Editors

Pico is a popular, very small text editor, distributed as part of the Pine e-mail client. Although Pico is free, it is not truly open source. Therefore, many Linux distributions, including Ubuntu, don't offer Pico. Instead, they offer an open source clone of Pico called nano (nano's another editor). This section describes the nano editor.

Note Ubuntu links the command `pico` to the program for the nano editor.

Nano (represented by the `nano` command) is a compact text editor that runs from the shell, but is screen-oriented (owing to the fact that it is based on the curses library). Nano is popular with those who formerly used the Pine e-mail client because nano's editing features are the same as those used by Pine's Pico editor.

On the rare occasion that you don't have the `vi` editor available on a Linux system (such as when installing a minimal Gentoo Linux), nano is almost always available. Ubuntu installs nano by default. You need the `spell` command, rather than `aspell`, to perform a spelling check within nano.

As with the JOE editor, instead of having command and typing modes, you can just begin typing. **To open a text file for editing**, just type `nano` and the filename, or use some of the following options:

\$ <code>nano memo.txt</code>	Open memo.txt for editing
\$ <code>nano -B memo.txt</code> ~.filename	When saving, back up previous to ~.filename
\$ <code>nano -m memo.txt</code>	Turn on mouse to move cursor (if supported)
\$ <code>nano +83 memo.txt</code>	Begin editing on line 83

The `-m` command line option turns on support for a mouse. You can use the mouse to select a position in the text, and the cursor moves to that position. After the first click, however, nano uses the mouse to mark a block of text, which may not be what you are expecting.

As with JOE, to **add text**, just begin typing. Use arrow keys to move the cursor left, right, up, or down. Use the Delete key to delete text under the cursor or the Backspace key to erase text to the left of the cursor. Press Enter to add a line break. Press Ctrl+g to read help text. The following list shows the control codes for nano that are described on the help screen.

Control Code	Function Key	Description
Ctrl+g	F1	Show help text. (Press Ctrl+x to exit help.)
Ctrl+x	F2	Exit nano (or close the current file buffer).
Ctrl+o	F3	Save the current file.

Ctrl+j	F4	Justify the current text in the current paragraph.
Ctrl+r	F5	Insert a file into the current file.
Ctrl+w	F6	Search for text.
Ctrl+y	F7	Go to the previous screen.
Ctrl+v	F8	Go to the next screen.
Ctrl+k	F9	Cut (and store) the current line or marked text.
Ctrl+u	F10	Uncut (paste) the previously cut line into the file.
Ctrl+c	F11	Display the current cursor position.
Ctrl+t	F12	Start spell checking.
Ctrl+-		Go to selected line and column numbers.
Ctrl+\		Search and replace text.
Ctrl+6		Mark text, starting at the cursor (Ctrl+6 to unset mark).
Ctrl+f		Go forward one character.
Ctrl+b		Go back one character.
Ctrl+Spacebar		Go forward one word.
Alt+Spacebar		Go backward one word.
Ctrl+p		Go to the previous line.
Ctrl+n		Go to the next line.
Ctrl+a		Go to the beginning of the current line.
Ctrl+e		Go to the end of the current line.
Alt+(Go to the beginning of the current paragraph.
Alt+)		Go to the end of the current paragraph.
Alt+\		Go to the first line of the file.
Alt+/		Go to the last line of the file.
Alt+]		Go to the bracket matching the current bracket.
Alt+=		Scroll down one line.
Alt+-		Scroll up the line.

Graphical Text Editors

Just because you are editing text doesn't mean you have to use a text-based editor. The main advantage of using a graphical text editor is that you can use a mouse to select menus, highlight text, cut and copy text, or run special plug-ins.

You can expect to have the GNOME text editor (gedit) if your Linux system has the GNOME desktop installed. Features in gedit enable you to check spelling, list document statistics, change display fonts and colors, and print your documents. The KDE desktop also has its own KDE text editor (kedit in the kdeutils package). It includes similar features to the GNOME text editor, along with a few extras, such as the ability to send the current document with KMail or another user-configurable KDE component.

Vim itself comes with an X GUI version. It is launched with the `gvim` command, which is part of the `vim-X11` package. If you'd like to turn GUI Vim into a more user-

friendly text editor, you can download a third-party configuration called Cream. You can install it by typing `sudo apt-get install cream`.

Other text editors you can install include `nedit` (with features for using macros and executing shell commands and aimed at software developers) and `leafpad` (which is similar to the Windows Notepad text editor). The Scribes text editor (`scribes`) includes some advanced features for automatic correction, replacement, indentation, and word completion.

Listing, Sorting, and Changing Text

Instead of just editing a single text file, you can use a variety of Linux commands to display, search, and manipulate the contents of one or more text files at a time.

Listing Text Files

The most basic method to display the contents of a text file is with the `cat` command. The `cat` command concatenates (in other words, outputs as a string of characters) the contents of a text file to your display (by default). You can then use different shell metacharacters to **direct the contents of that file in different ways**. For example:

<code>\$ cat myfile.txt</code>	Send entire file to the
<code>screen</code>	
<code>\$ cat myfile.txt > copy.txt</code>	Direct file contents to a
<code>file</code>	
<code>\$ cat myfile.txt >> myotherfile.txt</code>	Append file contents to a
<code>file</code>	
<code>\$ cat -s myfile.txt</code>	Show consecutive blanks as
<code>one</code>	
<code>\$ cat -n myfile.txt</code>	Show line numbers with
<code>output</code>	
<code>\$ cat -b myfile.txt</code>	Show line numbers on non-
<code>blanks</code>	

However, if your block of text is more than a few lines long, using `cat` by itself becomes impractical. That's when you need better tools to look at the beginning or the end, or page through the entire text.

To **view the top of a file**, use `head`:

```
$ head myfile.txt
$ cat myfile.txt | head
```

Both of these command lines use the `head` command to output the top ten lines of the file. You can specify the line count as a parameter to display any number of lines from

the beginning of a file. For example:

```
$ head -n 50 myfile.txt           Show the first 50 lines of a file
$ ps aux | head -n 15           Show the first 15 lines of ps output
```

This can also be done using this obsolete (but shorter) syntax:

```
$ head -50 myfile.txt
$ ps aux | head -15
```

You can use the `tail` command in a similar way to **view the end of a file**:

```
$ tail -n 15 myfile.txt           Display the last 15 lines in a
file
$ tail -15 myfile.txt           Display the last 15 lines in a
file
$ ps aux | tail -n 15           Display the last 15 lines of ps
output
```

The `tail` command can also be used to **continuously watch the end of a file** as the file is written to by another program. This is very useful for reading live log files when troubleshooting Apache, sendmail, or many other system services (some of these logs won't appear unless the application is installed):

```
# tail -f /var/log/messages       Watch system messages live
# tail -f /var/log/mail.log       Watch mail server messages
live
# tail -f /var/log/httpd/access_log Watch web server messages
live
```

You can press `Ctrl+c` to end the `tail -f` command.

Paging through Text

When you have a large chunk of text and need to get to more than just its beginning or end, you need a tool to **page through the text**. The original UNIX system pager was the `more` command:

```
$ ps aux | more                 Page through the output of ps (press spacebar)
$ more myfile.txt               Page through the contents of a file
```

However, `more` has some limitations. For example, in the line with `ps` above, `more` could not scroll up. The `less` command was created as a more powerful and user-friendly `more`. The common saying when `less` was introduced was: “What is `less`? `less` is `more`!” I recommend you no longer use `more`, and use `less` instead.

Note The `less` command has another benefit worth noting. Unlike text editors such as `vi`, it does not read the entire file when it starts. This results in faster start-up times when viewing large files.

The `less` command can be used with the same syntax as `more` in the previous examples:

```
$ ps auxx | less           Page through the output of ps
$ cat myfile.txt | less    Page through the contents of a file
$ less myfile.txt         Page through a text file
```

The `less` command enables you to **navigate** using the up and down arrow keys, PageUp, PageDown, and the spacebar. If you are using `less` on a file (not standard input), press `v` to open the current file in an editor. Which editor gets launched is determined by environment variables defined for your account. The editor is taken from the environment variable `VISUAL`, if defined, or `EDITOR` if `VISUAL` is not defined. If neither is defined, `less` invokes the JOE editor on Ubuntu.

Note Other versions of Linux invoke `vi` as the default editor in this case.

Press `Ctrl+c` to interrupt that mode. As in `vi`, while viewing a file with `less`, you can **search for a string** by pressing `/` (forward slash) followed by the string and Enter. To search for further occurrences, press `/` and Enter repeatedly.

To **scroll forward and back** while using `less`, use the `F` and `B` keys, respectively. For example, `10f` scrolls forward 10 lines and `15b` scrolls back 15 lines. Type `d` to scroll down half a screen and `u` to scroll up half a screen.

Paginating Text Files with `pr`

The `pr` command provides a quick way to format a bunch of text into a form where it can be printed. This can be particularly useful if you want to print the results of some commands without having to open up a word processor or text editor. With `pr`, you can **format text into pages with header information** such as date, time, filename, and page number. Here is an example:

```
$ dpkg-query -f='${Package} ${Version} ${Architecture}\n' | sort | pr --column=2 | less  Paginate list in 2
cols
```

In this example, the `dpkg-query -f='${Package} ${Version} ${Architecture}\n'` command lists all software packages installed on your system and pipes that list to the `sort` command, to be sorted alphabetically. Next, that list is piped to the `pr` command, which converts the single-column list into two columns (`--columns=2`) and paginates it. Finally, the `less` command enables you to page through the text.

Instead of paging through the output, you can **send the output to a file or to a printer**. Here are examples of that:

```
$ dpkg-query -f='${Package} ${Version} ${Architecture}\n' | sort | pr --column=2 > pkg.txt  Send pr output to
```

file

```
$ dpkg-query -f='${Package} ${Version} ${Architecture}\n' | sort | pr --column=2 | lpr
```

 Print pr output

Other **text manipulation** you can do with the `pr` command includes double-spacing the text (`-d`), showing control characters (`-c`), or offsetting the text a certain number of spaces from the left margin (for example, `-o 5` to indent five spaces from the left).

Searching for Text with grep

The `grep` command comes in handy when you need to **perform more advanced string searches in a file**. In fact, the phrase “to grep” has actually entered the computer jargon as a verb, just as “to Google” has entered the popular language. Here are examples of the `grep` command:

\$ <code>grep Remote /etc/services</code>	Show lines containing Remote
# <code>grep sudo /var/log/auth.log</code>	Show lines containing 404
\$ <code>ps auwx grep init</code>	Show init lines from ps
output	
\$ <code>ps auwx grep "\[*\]"</code>	Show bracketed commands
\$ <code>dmesg grep "[l]ata\ ^ata"</code>	Show ata kernel device
information	

These command lines have some particular uses, beyond being examples of the `grep` command. By searching `auth.log` for `sudo`, you can see when the `sudo` command was run, and what it was set to run. Displaying bracketed commands that are output from the `ps` command is a way to see commands for which `ps` cannot display options. The last command checks the kernel buffer ring for any ATA device information, such as hard disks and CD-ROM drives.

The `grep` command can also **recursively search a few or a whole lot of files at the same time**. If you have the `apache2` package installed, the following command recursively searches files in the `/etc/apache2/sites-enabled` and `/etc/apache2/conf.d` directories for the string `VirtualHost`:

```
$ grep -R VirtualHost /etc/apache2/conf.d /etc/apache2/sites-enabled
```

Add line numbers (`-n`) to your `grep` command to **find the exact lines** where the search terms occur:

```
# grep -Rn VirtualHost /etc/apache2/*conf*
```

By default, search terms are displayed in color on each line found. To explicitly ask to colorize the searched term in the search results, add the `--color` option:

```
# grep --color -Rn VirtualHost /etc/apache2/*conf*
```

By default, in a multi-file search, the filename is displayed for each search result. Use

the **-h** option to **disable the display of filenames**. This example searches for the string `sshd` in the file `auth.log`:

```
# grep -h sshd /var/log/auth.log
```

If you want to **ignore case** when you search messages, use the **-i** option:

```
$ grep -i acpi /var/log/dmesg    Search file for acpi (any case)
```

To **display only the name of the file** that includes the search term, add the **-l** option:

```
$ grep -Rl VirtualHost /etc/apache2
```

To **display all lines that do not match the string**, add the **-v** option:

```
$ grep -v " 200 " /var/log/apache2/access_*    Show lines without "
200 "
```

Checking Word Counts with `wc`

There are times when you need to know the number of lines that match a search string. The `wc` command can be used to **count the lines** that it receives. For example, the following command lists how many hits in an Apache log file come from a specific IP address:

```
$ grep 192.198.1.1 /var/log/apache2/access.log | wc -l
```

The `wc` command has other uses as well. By default, `wc` **prints the number of lines, words, and bytes in a file**:

```
$ wc /var/log/dmesg                List counts for a single file
```

```
436  3847 27984 /var/log/dmesg
```

```
$ wc /var/log/*.log                List single/totals for many
files
```

```
  305    3764   25772 /var/log/auth.log
  780    3517   36647 /var/log/bootstrap.log
  350    4405   39042 /var/log/daemon.log
10109   60654  669687 /var/log/dpkg.log
   71     419    4095 /var/log/fontconfig.log
1451   19860 135252 /var/log/kern.log
    0        0        0 /var/log/lpr.log
    0        0        0 /var/log/mail.log
    0        0        0 /var/log/pycentral.log
    0        0        0 /var/log/scrollkeeper.log
  108    1610   13864 /var/log/user.log
    0        0        0 /var/log/uucp.log
   12     43     308 /var/log/wvdialconf.log
  890    6717   46110 /var/log/Xorg.0.log
14076 100989 970777 total
```

Sorting Output with sort

It can also be useful to **sort the contents of a file or the output of a command**. This can be helpful in bringing order to disorderly output. The following examples list service names and numbers from the `/etc/services` file and sorts the results in alphanumeric order (forward and reverse):

```
$ cat /etc/services | sort           Sort in alphanumeric order
$ cat /etc/services | sort -r       Sort in reverse alphanumeric
order
```

The following command **sorts processes based on descending memory usage** (fourth field of `ps` output). The `-k` option specifies the key field to use for sorting. `4,4` indicates that the fourth field, and only the fourth field, is a key field.

```
$ ps aux | sort -r -k 4,4 | less
```

The following command line **sorts loaded kernel modules in increasing size order**. The `n` option tells `sort` to treat the second field as a number and not a string:

```
$ lsmod | sort -k 2,2n
```

Finding Text in Binaries with Strings

Sometimes you need to read the ASCII text that is inside a binary file. Occasionally, you can learn a lot about an executable that way. For those occurrences, use `strings` to **extract all the human-readable ASCII text**. The `strings` command is part of the `binutils` package and is installed by default on Ubuntu. Here are some examples:

```
$ strings /bin/ls | grep -i libc    Find occurrences of libc in ls
$ cat /bin/ls | strings            List all ASCII text in ls
$ strings /bin/ls                  List all ASCII text in ls
$ strings /usr/sbin/sshd | grep libwrap List TCP wrapper library
```

Replacing Text with sed

Finding text within a file is sometimes the first step toward replacing text. Editing streams of text is done using the `sed` command. The `sed` command is actually a full-blown scripting language. The examples in this chapter cover basic text replacement with the `sed` command.

If you are familiar with text replacement commands in `vi`, `sed` has some similarities. In the following example, you would **replace only the first occurrence per line** of `tcp` with `TEST`. Here, `sed` takes its input from a pipe, while sending its output to `stdout` (your screen):

```
$ cat /etc/services | sed s/tcp/TEST/ | head -n 20
```

```
TESTmux      1/tcp      # TCP port service multiplexer
echo         7/TEST
echo         7/udp
discard      9/TEST      sink null
```

Adding a `g` to the end of the substitution line, as in the following command, causes every occurrence of `tcp` to be changed to `TEST`. Also, in the following example, input is directed from the file `myfile.txt` and output is directed to `mynewfile.txt`:

```
$ sed s/tcp/TEST/g < /etc/services > mynewfile.txt
$ head -20 mynewfile.txt
TESTmux      1/TEST      # TCP port service multiplexer
echo         7/TEST
echo         7/udp
discard      9/TEST      sink null
```

To make the `sed` command case-insensitive, add the `i` option to the command line:

```
$ cat /etc/services | sed s/tcp/TEST/gi | head -n 20
TESTmux      1/TEST      # TEST port service multiplexer
```

The next example changes the first occurrences of the text `/home/chris` to `/home2/chris` from the `/etc/passwd` file. (Note that this command does not change that file, but outputs the changed text.) This is useful when user accounts are migrated to a new directory (presumably on a new disk), named with much deliberation, `home2`. Here, you have to use quotes and backslashes to escape the forward slashes so they are not interpreted as delimiters:

```
$ sed 's/\//home\//chris\//home2\//chris/g' < /etc/passwd | grep chris
chris:x:1000:1000:Chris Negus,,,:/home2/chris:/bin/bash
```

Although the forward slash is the `sed` command's default delimiter, you can **change the delimiter** to any other character of your choice. Changing the delimiter can make your life easier when the string contains slashes. For example, the previous command line that contains a path could be replaced with this command:

```
$ sed 's./home/chris./home2/chris.g' < /etc/passwd | grep chris
chris:x:1000:1000:Chris Negus,,,:/home2/chris:/bin/bash
```

In the line shown, a period (`.`) is used as the delimiter.

The `sed` command can **run multiple substitutions at once** by preceding each one with `-e`. Here, in the text streaming from `/etc/services`, all occurrences of `tcp` are changed to `LOWER` and occurrences of `TCP` are changed to `UPPER`:

```
$ sed -e s/tcp/LOWER/g -e s/TCP/UPPER/g /etc/services | head -n 20
LOWERMux     1/LOWER     # UPPER port service multiplexer
echo         7/LOWER
echo         7/udp
discard      9/LOWER     sink null
```


You can use `sed` to **add newline characters to a stream of text**. Where `Enter` appears, press the Enter key. The `>` on the second line is generated by `bash`, not typed in.

```
$ echo aaabccc | sed 's/b/\Enter
> /'
aaa
ccc
```

The trick just shown does not work on the left side of the `sed` substitution command. When you need to substitute newline characters, it's easier to use the `tr` command.

Translating or Removing Characters with `tr`

The `tr` command is an easy way to **do simple character translations on the fly**. In the following example, new lines are replaced with spaces, so all the files listed from the current directory are output on one line:

```
$ ls | tr '\n' ' '          Replace newline characters with
spaces
```

The `tr` command can be used to **replace one character with another**, but does not work with strings like `sed` does. The following command replaces all instances of the lowercase letter `f` with a capital `F`.

```
$ tr f F < /etc/services    Replace every f in the file with F
```

You can also use the `tr` command to simply **delete characters**. Here are two examples:

```
$ ls | tr -d '\n'          Delete new lines (resulting in one
line)
$ tr -d f < /etc/services  Delete every letter f from the file
```

The `tr` command can do some nifty tricks when you **specify ranges of characters** to work on. Here's an example of changing lowercase letters to uppercase letters:

```
$ echo chris | tr a-z A-Z    Translate chris into CHRIS
CHRIS
```

The same result can be obtained with the following syntax:

```
$ echo chris | tr '[:lower:]' '[:upper:]'    Translate chris into
CHRIS
```

Checking Differences between Two Files with `diff`

When you have two versions of a file, it can be useful **to know the differences between the two files**. For example, when upgrading a software package, you may save your old configuration file under a new filename, such as `config.old` or `config.bak`,

so you preserve your configuration. When that occurs, you can use the `diff` command to discover which lines differ between your configuration and the new configuration in order to merge the two. For example:

```
$ diff config config.old
```

You can change the output of `diff` to what is known as unified format. Unified format can be easier to read by human beings. It adds three lines of context before and after each block of changed lines that it reports, and then uses `+` and `-` to show the difference between the files. The following set of commands creates a file (`f1.txt`) containing a sequence of numbers (1–7), creates a file (`f2.txt`) with one of those numbers changed (using `sed`), and compares the two files using the `diff` command:

```
$ seq 1 7 > f1.txt           Send a sequence of numbers to f1.txt
$ cat f1.txt                 Display contents of f1.txt
1
2
3
4
5
6
7
$ sed s/4/FOUR/ < f1.txt > f2.txt   Change 4 to FOUR and send to
f2.txt
$ diff f1.txt f2.txt
4c4                               Shows line 4 was changed in
file
< 4
---
> FOUR
$ diff -u f1.txt f2.txt          Display unified output of diff
--- f1.txt 2007-09-07 18:26:06.000000000 -0500
+++ f2.txt 2007-09-07 18:26:39.000000000 -0500
@@ -1,7 +1,7 @@
1
2
3
-4
+FOUR
5
6
7
```

The `diff -u` output just displayed adds information such as modification dates and times to the regular `diff` output. The `sdiff` command can be used to give you yet another view. The `sdiff` command can **merge the output of two** files interactively, as shown in the following output:

```
$ sdiff f1.txt f2.txt
1
2
3
4
5
6
7
1
2
3
| FOUR
5
6
7
```

Another variation on the `diff` theme is `vimdiff`, which opens the two files side by side in Vim and outlines the differences in color. Similarly, `gvimdiff` opens the two files in gVim.

Note You need to install the vim-gnome package to run the `gvim` or `gvimdiff` program.

The output of `diff -u` can be fed into the `patch` command. The `patch` command takes an old file and a diff file as input and **outputs a patched file**. Following on the previous example, I use the `diff` command between the two files to generate a patch and then apply the patch to the first file:

```
$ diff -u f1.txt f2.txt > patchfile.txt
$ patch f1.txt < patchfile.txt
patching file f1.txt
$ cat f1.txt
1
2
3
FOUR
5
6
7
```

That is how many OSS developers (including kernel developers) distribute their code patches. The `patch` and `diff` commands can also be run on entire directory trees. For example, this set of commands recursively copies the `/etc/pam.d` directory to `/tmp/newpam.d` and `/tmp/oldpam.d`, and then makes changes to two files in the `newpam.d` directory:

```
# cp -r /etc/pam.d /tmp/oldpam.d
# cp -r /etc/pam.d /tmp/newpam.d
# echo hello >> /tmp/newpam.d/atd
# echo goodbye >> /tmp/newpam.d/sshd
# diff -r /tmp/newpam.d/ /tmp/oldpam.d/
diff -r /tmp/newpam.d/atd /tmp/oldpam.d/atd
10d9
< hello
```

```
diff -r /tmp/newpam.d/sshd /tmp/oldpam.d/sshd
40d39
< goodbye
```

By running a recursive `diff` (`diff -r`) on the directories, you can see that `hello` appears in the `/tmp/newpam.d/atd` file but not in the `/tmp/oldpam.d/atd` file (`< hello`). Likewise, the word `goodbye` is only in the `/tmp/newpam.d/sshd` file.

If the output is too much detail for you, and you only want to see which files changed and not how, you can add the `-q` option:

```
# diff -rq /tmp/newpam.d/ /tmp/oldpam.d/
Files /tmp/newpam.d/atd and /tmp/oldpam.d/atd differ
Files /tmp/newpam.d/sshd and /tmp/oldpam.d/sshd differ
```

Using `diff -r` is a good technique to keep track of changes to configuration files or software projects. For example, if you copied all your `/etc` files to another directory, and then later ran `diff -rq` on the two directories, you could see which configuration files have changed since you made the copy.

Using `awk` and `cut` to Process Columns

Another massive text processing tool is the `awk` command. The `awk` command is a full-blown programming language. Although there is much more you can do with the `awk` command, the following examples show you a few tricks related to **extracting columns of text**:

```
$ ps auxx | awk '{print $1,$11}'           Show columns 1, 11 of
ps
$ ps auxx | awk '/chris/ {print $11}'      Show chris' processes
$ ps auxx | grep chris | awk '{print $11}' Same as above
```

The first example displays the contents of the first column (username) and eleventh column (command name) from currently running processes output from the `ps` command (`ps auxx`). The next two commands produce the same output, with one using the `awk` command and the other using the `grep` command to find all processes owned by the user named `chris`. In each case, when processes owned by `chris` are found, column 11 (command name) is displayed for each of those processes.

By default, the `awk` command assumes the delimiter between columns is spaces. You can **specify a different delimiter** with the `-F` option as follows:

```
$ awk -F: '{print $1,$5}' /etc/passwd      Use colon delimiter to print
cols
```

You can get similar results with the `cut` command. As with the previous `awk` example, you can specify a colon (`:`) as the column delimiter to process information from the `/etc/passwd` file:

```
$ cut -d: -f1,5 /etc/passwd  
cols
```

Use colon delimiter to print

The `cut` command can also be **used with ranges of fields**. The following command prints columns 1 thru 5 of the `/etc/passwd` file:

```
$ cut -d: -f1-5 /etc/passwd
```

Show columns 1 through 5

Instead of using a dash (-) to indicate a range of numbers, you can use it to **print all columns from a particular column number and above**. The following command displays all columns from column 5 and above from the `/etc/passwd` file:

```
$ cut -d: -f5- /etc/passwd
```

Show columns 5 and later

I prefer to use the `awk` command when columns are separated by a varying number of spaces, such as the output of the `ps` command. And I prefer the `cut` command when dealing with files delimited by commas (,) or colons (:), such as the `/etc/passwd` file.

Converting Text Files to Different Formats

Text files in the UNIX world use a different end-of-line character (`\n`) than those used in the DOS/Windows world (`\r\n`). You can view these special characters in a text file with the `od` command:

```
$ echo hello > myunixfile.txt  
$ od -c -t x1 myunixfile.txt  
0000000  h   e   l   l   o   \n  
          68  65  6c  6c  6f  0a  
0000006
```

It is necessary to **convert the files** so they will appear properly when copied from one environment to the other. The `dos2unix` package contains several tools for converting files between formats (type `apt-get install dos2unix` to get the package). Here are some examples:

```
$ unix2dos < myunixfile.txt > mydosfile.txt  
$ cat mydosfile.txt | dos2unix > myunixfile.txt
```

The `unix2dos` example just shown converts a Linux or UNIX plain text file (`myunixfile.txt`) to a DOS or Windows text file (`mydosfile.txt`). The `dos2unix` example does the opposite by converting a DOS/Windows file to a Linux/UNIX file.

Summary

Linux and UNIX systems traditionally use plain text files for system configuration, documentation, output from commands, and many forms of stored information. As a

result, many commands have been created to search, edit, and otherwise manipulate plain text files. Even with today's GUI interfaces, the ability to manipulate plain text files is critical to becoming a power Linux user.

This chapter explores some of the most popular commands for working with plain text files in Linux. Those commands include text editors (such as `vi`, `nano`, and `JOE`), as well as commands that can edit streaming data (such as `sed` and `awk` commands). There are also commands for sorting text (`sort`), counting text (`wc`), and translating characters in text (`tr`).

Chapter 6

Playing with Multimedia

IN THIS CHAPTER

- Playing music with `play`, `ogg123`, and `mpg321`
- Adjusting audio with `alsamixer` and `aumix`
- Ripping music CDs with `cdparanoia`
- Encoding music with `oggenc`, `flac`, and `lame`
- Streaming music with `icecast` and `ices`
- Converting audio files with `sox`
- Transforming digital images with `convert`
- Playing video DVDs

There's no need to go to a GUI tool if all you need to do is play a song or convert an image or audio file to a different form. There are commands for working with multimedia files (audio or images) that are quick and efficient if you find yourself working from the shell. And if you need to manipulate batches of multimedia files, the same command you use to transform one file can be added to a script to repeat the process on many files.

This chapter focuses on tools for working with audio, digital image, and video files from the shell.

Working with Audio

There are commands available for Linux systems that can manipulate files in dozens of audio formats. Commands such as `ogg123`, `mpg321`, and `play` can be used to listen to audio files. There are commands for ripping songs from music CDs and encoding them to store efficiently. There are even commands to let you stream audio so anyone on your network can listen to your playlist.

Playing Music

Depending on the audio format you want to play, you can choose from several command line players for Linux. The `play` command (based on the `sox` facility, described later)

can play audio files in multiple, freely available formats. You can use `ogg123` to play popular open source music formats, including Ogg Vorbis, Free Lossless Audio Codec (FLAC), and Speex files. The `mpg321` player, which is available via third-party repositories, is popular for playing MP3 music files.

The `play` command requires the `sox` package. Install it with the following command:

```
$ sudo apt-get install sox
```

Type `sox -h` to see **audio formats and effects** available to use with `play`:

```
$ sox -h
```

...

Supported file formats: 8svx aif aifc aiff aiffc al alsa au auto
avr cdda cdr cvs cvsd dat dvms fssd gsm hcom ima ircam la lu maud
nist nul null ogg ossdsp prc raw s3 sb sf sl smp snd sndt sou sph
sw txw u3 u4 ub ul uw vms voc vorbis vox wav wve xa

Supported effects: allpass band bandpass bandreject bass chorus
comand dcshift deemph dither earwax echo echos equalizer fade
filter flanger highpass lowpass mcomand mixer noiseprof noised
pad pan phaser pitch polyphase repeat resample reverb reverse
silence speed stat stretch swap synth treble tremolo trim vibro vol

The `play` command uses the `sox` code to play sounds.

Here are some examples of playing files using `play`:

```
$ play inconceivable.wav      Play WAV file (may be ripped from CD)
$ play *.wav                  Play all WAV files in directory (up to
32)
$ play hi.au vol .6           AU file, lower volume (can lower
distortion)
$ play -r 14000 short.aiff    AIFF, sampling rate of 14000 hertz
```

To play Ogg Vorbis files, install the `vorbis-tools` package by typing the following:

```
$ sudo apt-get install vorbis-tools
```

Here are examples for playing Ogg Vorbis (www.vorbis.com/) files with `ogg123`:

```
$ ogg123 mysong.ogg           Play ogg file
$ cd /usr/share/example-content/Ubuntu_Free_Culture_Showcase/
$ ogg123 How\ fast.ogg        Play example file
$ ogg123 http://vorbis.com/music/Lumme-Badloop.ogg Play web
address
$ ogg123 -z *.ogg             Play files in pseudo-random
order
$ ogg123 /var/music/          Play songs in /var/music and sub
dirs
$ ogg123 -@ myplaylist       Play songs from playlist
```

A playlist is simply a list of directories or individual Ogg files to play. When a

directory is listed, all Ogg files are played from that directory or any of its subdirectories. When playing multiple files, press Ctrl+c to **skip to the next song**. Press Ctrl+c twice to **quit**.

To use the `mpg321` player to play MP3 files, you need to install the `mpg321` package. Because of patent claims associated with MP3 codecs, MP3 codecs are not included with most Linux distributions automatically. You need to choose to install MP3 codecs and players separately. To do that, type the following:

```
$ sudo apt-get install mpg321
```

Here are examples for playing MP3 audio files with `mpg321`:

\$ <u>mpg321 yoursong.mp3</u>	Play MP3 file
\$ <u>mpg321 -@ mp3list</u>	Play songs from playlist of MP3s
\$ <u>cat mp3list mpg321 -@ -</u>	Pipe playlist to mpg321
\$ <u>mpg321 -z *.mp3</u>	Play files in pseudo-random order
\$ <u>mpg321 -Z *.mp3</u>	Same as -z, but repeat forever

An `mpg321` playlist is simply a list of files. You can produce the list using a simple `ls` command and directing the output to a file. Use full paths to the files, unless you plan to use the list from a location from which relative paths make sense.

Adjusting Audio Levels

The command line audio tools you use to enable audio devices and adjust audio levels depend on the type of audio system you use. Advanced Linux Sound Architecture (ALSA) is the sound system used by most Linux systems these days. The Open Source Sound System (OSS) has been around longer and is still used on older hardware. In general, you can use `alsamixer` to adjust sound when ALSA is used and `aumix` with OSS.

ALSA is the default sound system for many Linux systems. By adding loadable modules that enable OSS device interfaces to work as well, audio applications that require the OSS device interface can work with ALSA as well. To see if **OSS modules are loaded**, such as `snd-pcm-oss` (emulates `/dev/dsp` and `/dev/audio`), `snd-mixer-oss` (emulates `/dev/mixer`), and `snd-seq-oss` (emulates `/dev/sequencer`), type the following:

```
# lsmod | grep snd
```

If they are not loaded, you can **install and load the OSS modules**:

```
$ sudo apt-get install oss4  
$ sudo modprobe snd-pcm-oss  
$ sudo modprobe snd-mixer-oss
```

```
$ sudo modprobe snd-seq-oss
```

If the modules are loaded, you can use `alsamixer` to adjust audio levels for OSS sound applications. **Start alsamixer** as follows:

```
$ alsamixer                Show alsamixer playback view (Esc to quit)
$ alsamixer -V playback    Show only playback channels (default)
$ alsamixer -V all         Show with playback and capture views
$ alsamixer -c 1          Use alsamixer on second (1) sound card
```

Volume bars appear for each volume channel:

- Move right and left arrow keys to **highlight different channels** (Master, PCM, Headphone, and so on).
- Use the up and down arrow keys to raise and lower the volume on each channel.
- With a channel highlighted, press **m** to **mute or unmute** that channel.
- Press the spacebar on a highlighted input channel (Mic, Line, and so on) to **assign the channel as the capture channel** (to record audio input).
- To **quit alsamixer**, press **Alt+q** or the **Esc** key.
- Press **Tab** to cycle through settings for Playback, Capture, and All.

The `aumix` audio mixing application (for which you need to install the `aumix` package) can operate in screen-oriented or plain command mode. In plain text, you use options to **change or display settings**. Here are examples of `aumix` command lines:

```
$ sudo apt-get install aumix
$ aumix -q                Show left/right volume and type for all
channels
vol 62, 62
pcm 100,100
igain 53,53
$ aumix -l q -m q         List current settings for line and mic only
$ aumix -v 80 -m 0        Set volume to 70% and microphone to 0
$ aumix -m 80 -m R -m q   Set mic to 80%, set it to record, list mic
$ aumix                  With no options, aumix runs screen-oriented
```

When run screen-oriented, `aumix` displays all available audio channels. In screen-oriented mode, **use keys to highlight and change displayed audio settings**:

- Use **PageUp**, **PageDown**, and the up arrow and down arrow keys to select channels.
- Use the right or left arrow key to increase or decrease volume.
- Type **m** to mute the current channel.
- Press the spacebar to select the current channel as the recording device.
- If a mouse is available, you can use it to select volume levels, balance levels, or the current recording channel.

Ripping CD Music

To be able to play your personal music collection from Linux, you can use tools such as `cdparanoia` to rip tracks from music CDs to WAV files on your hard disk. The ripped files can then be encoded to save disk space, using tools such as `oggenc` (Ogg Vorbis), `flac` (FLAC), or `lame` (MP3).

Note There are some excellent graphical tools for ripping and encoding CDs, such as `grip` and `sound-juicer`. Because they are CDDDB-enabled, those tools can also use information about the music on the CD to name the output files (artist, album, song, and so on). This section, however, describes how to use some of the underlying commands to rip and encode CD music manually.

Using `cdparanoia`, you can check that your CD drive is capable of ripping Compact Disc Digital Audio (CDDA) CDs, retrieve audio tracks from your CD drive, and copy them to hard disk. If `cdparanoia` is not installed, install it with `apt-get`. Then start by inserting a music CD in your drive and typing the following:

```
$ sudo apt-get install cdparanoia
$ cdparanoia -vsQ
...
Checking /dev/cdrom for cdrom...
Checking for SCSI emulation...
Checking for MMC style command set...
Verifying CDDA command set...
...
Table of contents (audio tracks only):
track          length          begin          copy pre ch
=====
  1.      18295 [04:03.70]          0 [00:00.00]    no  no  2
  2.      16872 [03:44.72]    18295 [04:03.70]    no  no  2
...
 11.      17908 [03:58.58]    174587 [38:47.62]    no  no  2
 12.      17342 [03:51.17]    192495 [42:46.45]    no  no  2
TOTAL  209837 [46:37.62]    (audio only)
```

The snipped output shows `cdparanoia` checking the capabilities of `/dev/cdrom`, looking for SCSI emulations and MMC command set support, and verifying that the drive can handle CDDA information. Finally, it prints information about each track. Here are examples of `cdparanoia` command lines for **ripping a CD to a hard drive**:

<code>\$ <u>cdparanoia -B</u></code>	Rip tracks as WAV files by track name
<code>\$ <u>cdparanoia -B -- "5-7"</u></code>	Rip tracks 5-7 into separate files
<code>\$ <u>cdparanoia -- "3-8" abc.wav</u></code> (abc.wav)	Rip tracks 3-8 to one file

\$ <u>cdparanoia -- "1:[40]-"</u>	Rip track 1 from 40 secs to end
of CD	
\$ <u>cdparanoia -f -- "3"</u>	Rip track 3 and save to AIFF
format	
\$ <u>cdparanoia -a -- "5"</u>	Rip track 5 and save to AIFC
format	
\$ <u>cdparanoia -w -- "1" my.wav</u>	Rip track 1 and name it my.wav

Encoding Music

After a music file is ripped from CD, encoding that file to save disk space is usually the next step. Popular encoders include `oggenc`, `flac`, and `lame`, for encoding to Ogg Vorbis, FLAC, and MP3 formats, respectively.

With `oggenc`, you can start with audio files or streams in WAV, AIFF, FLAC, or raw format and convert them to Ogg Vorbis format. Although Ogg Vorbis is a lossy format, the default encoding from WAV files still produces very good quality audio and can result in a file that's about one-tenth the size. Here are some examples of `oggenc`:

\$ <u>oggenc ab.wav</u>	Encodes WAV to Ogg (ab.ogg)
\$ <u>oggenc ab.flac -o new.ogg</u>	Encodes FLAC to Ogg (new.ogg)
\$ <u>oggenc ab.wav -q 9</u>	Raises encoding quality to 9

By default, the quality (`-q`) of the `oggenc` output is set to 3. You can **set the quality** to any number from `-1` to `10` (including fractions such as `5.5`).

```
$ oggenc NewSong.wav -o NewSong.ogg \
-a Bernstein -G Classical \
-d 06/15/1972 -t "Simple Song" \
-l "Bernsteins Mass" \
-c info="From Kennedy Center"
```

The command just shown converts `MySong.wav` to `MySong.ogg`. The artist's name is Bernstein and the music type is Classical. The date is June 15, 1972, the song title is "Simple Song," and the album name is Bernsteins Mass. A comment is "From Kennedy Center." The backslashes aren't needed if you just keep typing the whole command on one line. However, if you do add backslashes, make sure there are no spaces after the backslash.

The preceding example adds information to the header of the resulting Ogg file. You can **see the header information**, with other information about the file, using `ogginfo`:

```
$ ogginfo NewSong.ogg
Processing file "NewSong.ogg"...

...
Channels: 2
Rate: 44100
Nominal bitrate: 112.000000 kb/s
```

```

User comments section follows...
    info=From Kennedy Center
    title=Simple Song
    artist=Bernstein
    genre=Classical
    date=06/15/1972
    album=Bernsteins Mass
Vorbis stream 1:
    Total data length: 3039484 bytes
    Playback length: 3m:25.240s
    Average bitrate: 118.475307 kb/s
Logical stream 1 ended

```

Here you can see that comments were added during encoding. The `-c` option was used to set an arbitrary field (in this case, `info`) with some value to the header. In addition to the comment information, you can see that this file has two channels and was recorded at a 44100 bitrate. You can also see the data length, playback time, and average bitrate.

The `flac` command is an encoder similar to `oggenc`, except that the WAV, AIFF, RAW, FLAC, or Ogg file is encoded to a FLAC file. Because `flac` is a free lossless audio codec, it is a popular encoding method for those who want to save some space but still want top-quality audio output in an open source format. Using default values, our encoding from WAV to FLAC resulted in files one-half the size, as opposed to one-tenth the size with `oggenc`. Install the `flac` package to use the `flac` command. Here is how to install the `flac` package, followed by examples of the `flac` command:

```

$ sudo apt-get install flac
$ flac now.wav                                Encodes WAV to FLAC
(now.flac)
$ sox now.wav now.aiff                         Encodes WAV to AIFF
(now.aiff)
$ flac now.aiff -o now2.flac                   Encodes AIFF to FLAC
(now.flac)
$ flac -8 top.wav -o top.flac                 Raises compression level to
8

```

The compression level is set to `-5` by default. A range from `-0` to `-8` can be used, with the highest number giving the greatest compression and the lower number giving faster compression time. To **convert files to MP3 format** using the `lame` command, you must first install the `lame` package. Here are some examples of the `lame` command to encode from WAV and AIFF files:

```

$ sudo apt-get install lame
$ lame in.wav                                Encodes WAV to MP3
(in.wav.mp3)
$ lame in.wav --preset standard               Encodes to MP3 with std
presets

```

```
$ lame tune.aiff -o tune.mp3           Encodes AIFF to MP3
(tune.mp3)
$ lame -h -b 64 -m m in.wav out.mp3    High quality, 64-bit, mono
mode
$ lame -q 0 in.wav -o abcHQ.mp3        Encodes with quality set to 0
```

With `lame`, you can set the quality from 0 to 9 (5 is the default). Setting the quality to 0 uses the best encoding algorithms, while setting it to 9 disables most algorithms (but the encoding process moves much faster). As with `oggenc`, you can **add tag information to your MP3 file** that can be used later when you play back the file. Here's an example:

```
$ lame NewSong.wav NewSong.mp3      \
  --ta Bernstein --tg Classical      \
--ty 1972 --tt "Simple Song"      \
--tl "Bernsteins Mass"           \
  --tc "From Kennedy Center"
```

Like the wav-to-ogg example shown earlier in this chapter, the command just shown converts `MySong.wav` to `MySong.mp3`. As before, the artist's name is Bernstein and the music type is Classical. The year is 1972, the song title is "Simple Song," and the album name is Bernsteins Mass. A comment is "From Kennedy Center." The backslashes aren't needed if you just keep typing the whole command on one line. However, if you do add backslashes, make sure there are no spaces after the backslash.

The tag information appears on the screen in graphical MP3 players (such as Rhythmbox and Totem, when they have been enabled to play MP3 format). You can also see tag information when you use command line players, such as the following `mpg321` example:

```
$ mpg123 NewSong.mp3
High Performance MPEG 1.0/2.0/2.5 Audio Player for Layer 1, 2, and
3.

...
Title   : Simple Song                Artist: Bernstein
Album   : Bernsteins Mass            Year   : 1972
Comment: From Kennedy Center         Genre  : Classical

Playing MPEG stream from NewSong.mp3 ...
MPEG 1.0 layer III, 128 kbit/s, 44100 Hz joint-stereo
```

Streaming Music

If your music is on one machine, but you're working from another machine, **setting up a streaming music server** is a quick way to broadcast your music so it can be picked up from one or more computers on your network. The Icecast streaming media server (`icecast2` package) and Ices audio source client (`ices2` package) can be installed in

Ubuntu by typing the following:

```
$ sudo apt-get install icecast2 ices2
```

As you install icecast2, you have the choice of configuring it during installation or after (as described in the following procedure).

Here's a quick and dirty procedure for setting up Icecast and Ices to stream your music. Perform these steps on the computer that contains the music you want to serve, choosing to configure the service later:

1. Edit the `/etc/icecast2/icecast.xml` file to change all passwords listed. Search for `hackme` to find the current passwords. You probably want different user and administrative passwords, especially if you allow others to stream music to the server. Remember the passwords you set for later. You may want to change other settings in this file as well, such as the hostname:

```
$ sudo vi /etc/icecast2/icecast.xml
```

2. If you have a firewall, check that TCP port 8000 is accessible.
3. Edit the `/etc/default/icecast2` file and change the `ENABLE` line to read `ENABLE=true`.
4. Start the icecast2 server as the root user by typing the following (the server will actually run as the icecast2 user), and verify with the `netstat` command:

```
$ sudo /etc/init.d/icecast2 start  
$ sudo netstat -topnave1 | grep 8000  
tcp        0      0 0.0.0.0:8000          0.0.0.0:*            LISTEN  
115        35790      21494/icecast        off (0.00/0/0)
```

5. Create the directories needed by the ices2 program, which provides the playlist and music to the icecast2 server. Run the following commands:

```
$ sudo mkdir /var/log/ices  
$ sudo mkdir /etc/ices2  
$ sudo mkdir /etc/ices2/music
```

6. Create a playlist using any text editor or by directing a listing of your music to a file. For example, if all your Ogg music files are in `/var/music` subdirectories, type the following:

```
$ find /var/music -name *.ogg > playlist.txt
```

7. The `playlist.txt` file must contain full paths to every music file, and the files must be accessible to the icecast2 server. Then, copy the playlist file to the `/etc/ices2` directory:

```
$ sudo cp playlist.txt /etc/ices2
```

With the playlist file created, use any text editor to remove or add files or directories to make your playlist as you would like it. (If you want some files to try out for your playlist, download some from <http://vorbis.com/music>.)

8. As root user, edit the `/etc/ices2/ices-playlist.xml` file so it will play from your playlist and feed that music to your running icecast2 server. Start with the example configuration file and then edit it. Run the following commands:

```
$ sudo cp /usr/share/doc/ices2/examples/ices-playlist.xml /etc/ices2
$ sudo vi /etc/ices2/ices-playlist.xml
```

9. In particular, you want to modify the metadata, input, and instance modules (be sure to change `/etc/ices2/playlist.txt` to the path where you put your `playlist.txt` file):

```
<metadata>
  <name>My Music Server</name>
  <genre>Different music styles</genre>
  <description>Mix of my personal music</description>
</metadata>
<input>
  <module>playlist</module>
  <param name="type">basic</param>
  <param name="file">/etc/ices2/playlist.txt</param>
  <!-- random play -->
  <param name="random">1</param>
  ...
</input>
<instance>
  <hostname>localhost</hostname>
  <port>8000</port>
  <password>MIcePw</password>
  <mount>/mymusic.ogg</mount>
  ...
</instance>
```

Of the values just shown (in bold), the most critical are the location of your playlist and the information about the instance of your icecast2 server. The password must match the source password you added to your `/etc/icecast2/icecast.xml` file.

10. Launch the ices audio feed by typing the following:

```
$ sudo ices2 /etc/ices2/ices-playlist.xml &
```

11. Test that you can play music from the local computer as follows:

```
$ ogg123 http://localhost:8000/mymusic.ogg
```

12. If that test works, try playing the icecast2 stream from another computer on your network by replacing `localhost` with the server's IP address or hostname.

13. If there are problems, check `/var/log/icecast2` and `/var/log/ices` log files. Recheck your passwords and locations of configuration files.

14. When you are done, just kill the icecast2 service:

```
$ sudo /etc/init.d/icecast2 stop
```

When the icecast and ices servers are running, you should have access to that streaming music from any computer that can access your server computer. Use any music player that can play from an HTTP address (`ogg123`, `Rhythmbox`, `XMMS`, and so on). Windows music players that can support the type of content you are serving should work

as well.

Note If you want to skip a song, type this from the server: `killall -HUP ices`.

Converting Audio Files

The `sox` utility is an extremely versatile tool for working with audio files in different freely available formats. The following material provides a few examples of things you can do with `sox`.

The following command **concatenates two WAV files to a single output file**:

```
$ sox head.wav tail.wav output.wav
```

This command **mixes two WAV files**:

```
$ soxmix sound1.wav sound2.wav output.wav
```

To use `sox` to **display information about a file**, use the `stat` effect as follows:

```
$ sox sound1.wav -e stat
Samples read:          208512
Length (seconds):      9.456327
Scaled by:             2147483647.0
Maximum amplitude:     0.200592
Minimum amplitude:     -0.224701
Midline amplitude:     -0.012054
Mean   norm:           0.030373
Mean   amplitude:      0.000054
RMS    amplitude:      0.040391
Maximum delta:         0.060852
Minimum delta:         0.000000
Mean   delta:          0.006643
RMS    delta:          0.009028
Rough   frequency:      784
Volume adjustment:     4.450
```

Use `trim` to delete seconds of sound from an audio file. For example:

```
$ sox sound1.wav output.wav trim 4           Trim 4 seconds from start
$ sox sound1.wav output.wav trim 2 6         Keep from 2-6 seconds of
file
```

The first example deletes the first 4 seconds from `sound1.wav` and writes the results to `output.wav`. The second example takes `sound1.wav`, keeps the section between second 2 and second 6 and deletes the rest, and writes to `output.wav`.

Transforming Images

With directories full of digital images, the ability to manipulate images from the command line can be a huge timesaver. The ImageMagick package (use `apt-get install imagemagick` to install the package on Ubuntu) comes with some very useful tools for transforming your digital images into forms you can work with. This section shows some commands for manipulating digital images, and provides examples of simple scripts for making those changes in batches.

Getting Information about Images

To **get information about an image**, use the `identify` command, as follows:

```
$ identify p2090142.jpg
p2090142.jpg JPEG 2048x1536+0+0 DirectClass 8-bit 402.037kb
$ identify -verbose p2090142.jpg | less
Standard deviation: 61.1665 (0.239869)
Colors: 205713
Rendering intent: Undefined
Resolution: 72x72
Units: PixelsPerInch
Filesize: 402.037kb
Interlace: None
Background color: white
Border color: rgb(223,223,223)
Matte color: grey74
Transparent color: black
Page geometry: 2048x1536+0+0
Compression: JPEG
Quality: 44
...
```

The first command in the preceding example displays basic information about the image (its filename, format, geometry, class, channel depth, and file size). The second command shows every bit of information it can extract from the image. In addition to the information you see in the example, the verbose output also shows creation times, the type of camera used, aperture value, and ISO speed rating.

Converting Images

The `convert` command is a Swiss Army knife of file converters. Here are some ways to manipulate images using the `convert` command. The following examples **convert image files from one format to another**:

```
$ convert tree.jpg tree.png           Convert a JPEG to a PNG file
$ convert icon.gif icon.bmp          Convert a GIF to a BMP file
$ convert photo.tiff photo.pcx       Convert a TIFF to a PCX file
```

Image types that `convert` supports include `.jpg`, `.bmp`, `.pcx`, `.gif`, `.png`, `.tiff`, `.xpm`, and `.xwd`. Here are examples of `convert` being used to **resize images**:

```
$ convert -resize 1024x768 hat.jpg hat-sm.jpg
$ convert -sample 50%x50% dog.jpg dog-half.jpg
```

The first example creates an image (`hat-sm.jpg`) that is 1024 x 768 pixels. The second example reduced the image `dog.jpg` in half (50%x50%) and saves it as `dog-half.jpg`.

You can **rotate images** from 0 to 360 degrees. Here are examples:

```
$ convert -rotate 270 sky.jpg sky-final.jpg      Rotate image 270
degrees
$ convert -rotate 90 house.jpg house-final.jpg   Rotate image 90
degrees
```

You can **add text to an image** using the `-draw` option:

```
$ convert -fill black -pointsize 60 -font helvetica \
-draw 'text 10,80 "Copyright NegusNet Inc."' \
p10.jpg p10-cp.jpg
```

The previous example adds copyright information to an image, using 60-point black Helvetica font to write text on the image. The text is placed 10 points in and 80 points down from the upper-left corner. The new image name is `p10-cp.jpg` to indicate that the new image had copyright information added.

Here are some interesting ways to **create thumbnails** with the `convert` command:

```
$ convert -thumbnail 120x120 a.jpg a-a.png
$ convert -thumbnail 120x120 -border 8 a.jpg a-b.png
$ convert -thumbnail 120x120 -border 8 -rotate 8 a.jpg a-c.png
```

All three examples create a 120 x 120–pixel thumbnail. The second adds the `-border` option to put a border around the thumbnail so it looks like a Polaroid picture. The last example rotates the image. [Figure 6-1](#) shows the results of these three examples.

Figure 6-1: Use `convert` to create a thumbnail, add borders, and rotate images.



Besides the things you can do to make images useful and manageable, there are also ways of **making your images fun and even weird**. Here are some examples:

```
$ convert -sepia-tone 75% house.jpg oldhouse.png  
$ convert -charcoal 5 house.jpg char-house.png  
$ convert -colorize 175 house.jpg color-house.png
```

The `-sepia-tone` option gives the image an Old West sort of look. The `-charcoal` option makes the image look as if the picture was hand-drawn using charcoal. By using the `-colorize` option, every pixel in the image is modified using the colorize number provided (175 in this case). [Figure 6-2](#) shows the original house picture in the upper-left corner, the Sepia Tone in the upper right, the charcoal in the lower left, and the colorized house in the lower right.

If you are looking for one more example of weird image conversions, try swirling your image. For example:

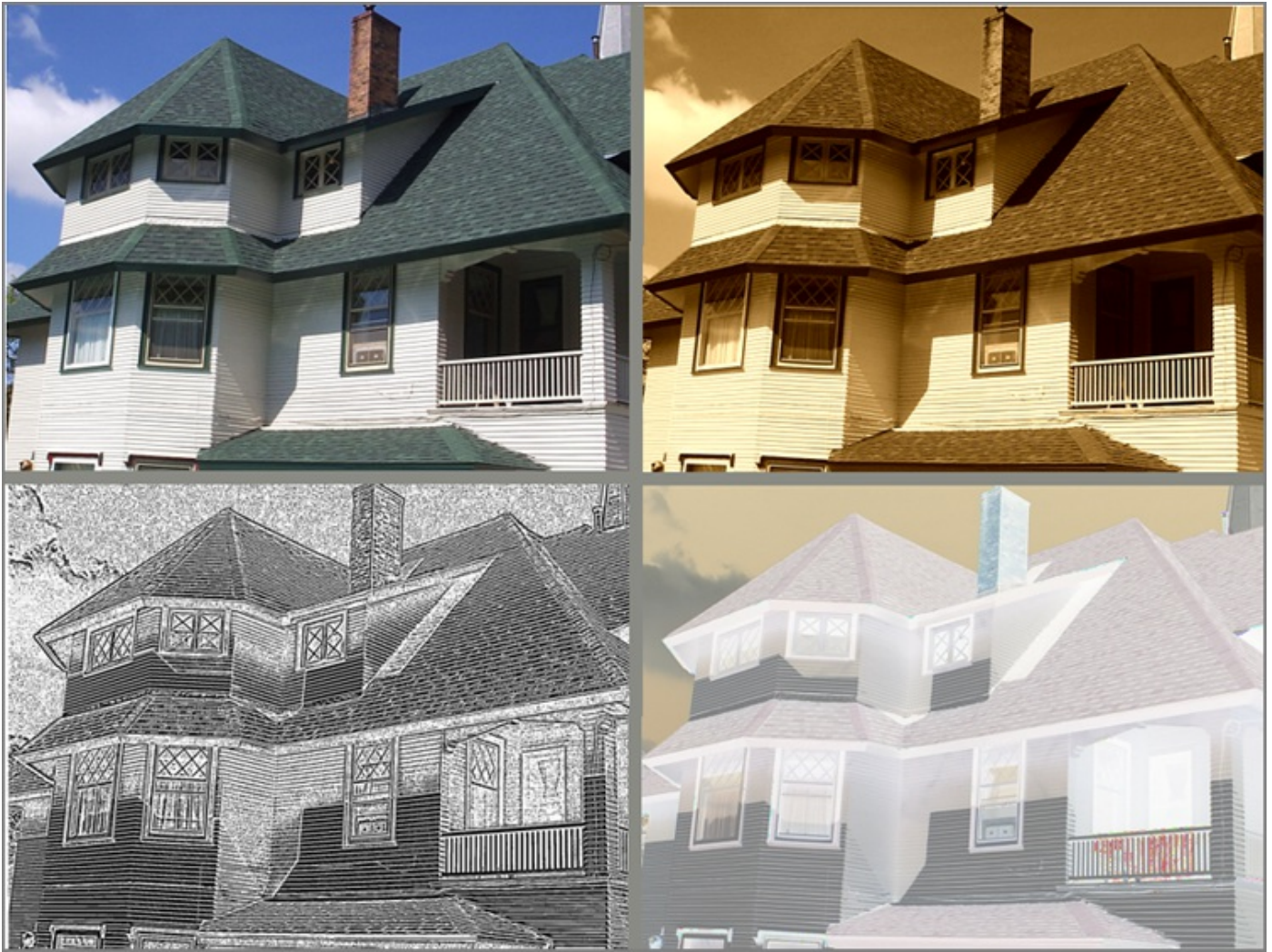
```
$ convert -swirl 300 photo.pcx weird.pcx
```

Converting Images in Batches

Most of the image conversions described in this chapter can be done quite easily using a graphical image manipulation tool such as The GIMP. However, the `convert` commands I described can really shine when you use them in scripts. So, instead of resizing, rotating, writing on, or colorizing a single file, you can do any (or all) of those things to a whole directory of files.

You may want to create thumbnails for your duck decoy collection images. Or perhaps you want to reduce all your wedding photos so they can play well on a digital photo frame. You might even want to add copyright information to every image in a directory before you share them on the web. All of these things can be done quite easily with the `convert` commands already described and some simple shell scripts.

Figure 6-2: Start with a normal image and sepia-tone, then charcoal, and then colorize it.



Here's an example of a script you can run to **resize an entire directory of photos** to 1024×768 pixels to play on a digital photo frame:

```
$ cd $HOME/myimages
$ mkdir small
$ for pic in `ls *.png`
do
    echo "converting $pic"
    convert -resize 1024x768 $pic small/sm-$pic
done
```

Before running the script, this procedure changes to the `$HOME/myimages` directory (which happens to contain a set of high-resolution images). Then it creates a subdirectory to hold the reduced images called `small`. The script itself starts with a `for` loop that lists each file ending in `.png` in the current directory (you might need to make that `.jpg` or another image suffix). Then, each file is resized to 1024×768 and copied to the `small` directory, with `sm-` added to each filename.

Using that same basic script, you can use any of the `convert` command lines shown earlier, or make up your own to suit your needs. You might be able to convert a whole

directory of images in a few minutes that would have taken you hours of clicking in the GUI.

Playing with Video

Playing, recording, copying, and otherwise working with video has long been a contentious issue in Linux. Although there were many open source tools early on for working with video, those who produced commercial video codecs and video content (particularly commercial movies) were not always keen on others being so free with their properties.

The capability to play and copy commercial DVD movies in Linux relies on special software for accessing and converting video into a viewable format. At first, decryption software called DeCSS was obtained from a cracked DVD player. Since then, libdvdcss was created to allow Linux users to decrypt commercial DVD content so it can be played or copied.

Even though libdvdcss has not faced the same legal challenges that DeCSS has, most Linux distributions leave it up to the individual to assess the legality of using libdvdcss for decrypting movies. All the other software needed for managing and playing commercial DVD movies is readily available and capable of playing movies once you add libdvdcss.

Just for fun, the following subsection tells you how to get the software you need to play your DVD movies.

Playing Video Files

There is a handful of high-quality video players available for Ubuntu. The following is a list of a few of the most popular Linux video players:

- **Totem**—This is the default movie player available with the GNOME desktop. It is based on the GStreamer project, which produces many plug-ins needed to provide audio and video codecs. In Ubuntu, the Movie Player icon launches Totem by default. So if you installed a desktop system, the basic Totem player is automatically installed.
- **Mplayer**—The Mplayer movie player for Linux is one of the oldest video projects for Linux. It plays dozens of audio and video types and is especially good at playing damaged media files that won't play on Windows players. You install the mplayer package to get the basic `mplayer` video player.

```
$ sudo apt-get install mplayer
```


- **Xine**—The Xine project is actually a media player engine that can be used in different audio/video player front-ends. You install the xine-ui package to get a basic front-end for playing videos with the xine engine.

```
$ sudo apt-get install xine-ui
```

- **VLC**—VLC is another popular media player. It supports a variety of audio and video codecs (www.videolan.org/vlc/features.html). To get it in Ubuntu, install the vlc package:

```
$ sudo apt-get install vlc
```

To avoid legal issues of playing DVD movies in Linux, you can buy the Fluendo DVD Player from the company that does the GStreamer project (www.fluendo.com/shop/product/fluendo-dvd-player/). It sells for about 20 Euros (That's about \$26 USD). The next section describes how to install the software needed for open source video players.

Installing Video Software

After you have installed any of the video players described in the previous section, you still need extra software to be able to play different audio and video formats. To play content that is not already built into a player, you can add codecs.

In addition to providing its own commercial DVD player, the GStreamer project creates a set of GStreamer plug-in packages that support a variety of audio/video formats. To see what codecs are supported by each GStreamer plug-in package, refer to the GStreamer plug-in documentation:

```
http://gstreamer.freedesktop.org/documentation/plugins.html
```

For a description of the Good, Bad, and Ugly sets of codecs available with GStreamer, refer to the Wikipedia description of GStreamer (<http://en.wikipedia.org/wiki/GStreamer>). If, for example, if you wanted to install the GStreamer plug-ins from the Good set, you could type the following:

```
$ sudo apt-get install gstreamer0.10-plugins-good
```

As I mentioned earlier, however, the most crucial (and contentious) piece of software for playing commercial DVD movies is libdvdcss. The Ubuntu community help pages describe how to get and install libdvdcss without enabling any special third-party repositories. It warns that you should make sure it is legal to use in your area:

```
https://help.ubuntu.com/community/RestrictedFormats/PlayingDVDs
```

For Ubuntu, the libdvdcss2 package provides the software for reading commercial DVD movies. You can get that package by installing the libdvdread4 package and running the following script:

```
$ sudo /usr/share/doc/libdvdread4/install-css.sh
```

At this point, you should be able to read and play most commercial DVD movies from your Linux system. If you need help setting the DVD region code on your Ubuntu system, you can refer again to the Ubuntu community Playing DVDs page (<https://help.ubuntu.com/community/RestrictedFormats/PlayingDVDs>).

Starting the DVD player

You can start any of the video players described earlier in this chapter from an Ubuntu desktop by simply inserting a DVD movie into your DVD drive and selecting Movie Player. This will start the Totem movie player, by default. If you prefer, you could choose Xine or other players instead.

Alternatively, you can start your video players at the command line. For example, to start the Mplayer movie player, type the following from the shell:

```
$ mplayer dvd://
```

Once the software is installed, playing DVD movies is actually quite intuitive. If, however, you want to refine your DVD playing or play back your movies in some special way, refer to the man pages for your DVD player. For example:

```
$ man mplayer
```

For example, say that you have installed Ubuntu, but you don't install a desktop, you can output your video from the Mplayer video player in ASCII art. The following is an example of a command line for directing video output to your local shell for viewing a movie in ASCII art:

```
$ mplayer dvd:// -vo aa -monitorpixelaspect 0.5
```

Press the spacebar to pause and unpauses the movie. When you are done, press Esc to exit the movie.

You may need to adjust the number you give the `monitorpixelaspect` option to fit your monitor. Refer to the `mplayer` man page for other useful settings to adjust other aspects of your movie.

Summary

The shell can provide a quick and efficient venue for working with your audio and digital image files. This chapter described ways of playing, ripping, encoding, converting, and streaming audio files from the command line. As for digital images, there are many examples of using the `convert` command for resizing, rotating, converting, writing on, and otherwise manipulating those images. For playing video,

once you get the right software installed, you have several choices, including displaying video in ASCII art in your shell.

Chapter 7

Administering Filesystems

IN THIS CHAPTER

- Understanding Linux filesystem types
- Partitioning disks with fdisk and parted
- Working with labels with e2label and findfs
- Creating filesystems with mkfs
- Viewing filesystem info with tune2fs/dumpe2fs
- Using swap areas with mkswap, swapon, and swapoff
- Using fstab, mount, and umount to mount and unmount filesystems
- Checking filesystems with badblocks and fsck
- Creating encrypted filesystems
- Viewing RAID information with mdadm
- Checking disk space with du and df
- Logical Volume Manager (LVM)

Filesystems provide the structures in which files, directories, devices, and other elements of the system are accessed from Linux. Linux supports many different types of filesystems (ext4, VFAT, ISO9660, NTFS, and so on) as well as many different types of media on which filesystems can exist (hard disks, CDs, USB flash drives, Zip drives, network drives, and so on).

Creating and managing disk partitions and the filesystems on those partitions are among the most critical jobs in administering a Linux system. That's because if you mess up your filesystem, you might very well lose the critical data stored on your computer's hard disk or removable media.

This chapter contains commands for partitioning storage media, creating filesystems, mounting and unmounting partitions, and checking filesystems for errors and disk space.

Understanding Filesystem Basics

Even though many different filesystem types are available in Linux, you will probably only assign a few different filesystem types on any given Linux system. For a basic

Linux system, your computer hard disk may contain only three partitions: a swap partition (used to handle the overflow of information in RAM), a boot partition that contains the boot loader and kernel, and a root filesystem partition. The boot and root filesystem partitions are usually the ext4 filesystem type.

Most of the examples in this chapter use ext4 filesystems to illustrate how a filesystem is created and managed. Ext filesystems have traditionally been used as the default filesystem type for Linux distributions. However, there are times when you might want to use other filesystem types. The following is a list of different filesystem types and descriptions of when you might want to use them.

- **ext4**—Adds performance and reliability features to ext3 filesystem types.
- **ext3**—Contains journaling features for safer data and fast reboots after unintended shutdowns.
- **ext2**—Predecessor of ext3, but doesn't contain journaling.
- **iso9660**—Evolved from the High Sierra filesystem (which was the original standard used on CD-ROM). May contain Rock Ridge extensions to allow iso9660 filesystems to support long filenames and other information (file permissions, ownership, and links).
- **Jffs2**—Journaling Flash filesystem, version 2 (JFFS2), which is designed for efficient operations on USB flash drives. Successor to JFFS.
- **jfs**—JFS filesystem, which IBM used for OS/2 Warp. Tuned for large filesystems and high-performance environments.
- **msdos**—MS-DOS filesystem. Can be used to mount older MS-DOS filesystems, such as those on old floppy disks.
- **ntfs**—Microsoft New Technology Filesystem (NTFS). Useful when filesystems need to share files with newer Windows systems (as with dual booting or removable drives).
- **reiserfs**—Journaling filesystem, which used to be used by default on some SUSE, Slackware, and other Linux systems. Reiserfs is not well-supported in Ubuntu.
- **squashfs**—Compressed, read-only filesystem used on many Linux live CDs.
- **swap**—Used on swap partitions to hold data temporarily when RAM is not currently available.
- **ufs**—Popular filesystem on Solaris and SunOS operating systems from Sun Microsystems.
- **vfat**—Extended FAT (VFAT) filesystem. Useful when filesystems need to share files with older Windows systems (as with dual booting or removable drives).
- **xfs**—Journaling filesystem for high-performance environments. Can scale up to systems that include multiple terabytes of data that transfer data at multiple gigabytes per second.

Besides the filesystem types listed in the table, there are also what are referred to as network shared filesystems. Locally, a network shared filesystem may be an ext4, ntfs, or other normal filesystem type. However, all or part of those filesystems can be shared with network protocols such as Samba (smbfs or cifs filesystem type), NFS (nfs), and NetWare (ncpfs).

Many available filesystem types are either not useful for creating new filesystems or not fully supported in every version of Linux. For example, filesystem types such as minix (for Minix systems), befs (for BeOS systems), and affs (for Amiga systems) are mostly useful if you need to mount and access very old backup media from those systems. Even popular filesystems may not be fully supported. For example, reiserfs filesystems aren't fully supported, as of this writing, by the Kubuntu variant of Ubuntu.

Creating and Managing Filesystems

Ubuntu gives you the option of either having the installer create a default partitioning and filesystem scheme or letting you set that all up manually when you first install Linux. The installer lets you choose to erase the entire hard disk, erase only Linux partitions, or use only free disk space to set up the partitions. To take the manual approach instead, you must choose to create a custom layout.

With the manual approach, the disk-partitioning tool lets you divide the hard disk into partitions as you choose. Later, there are a lot of command-line utilities you can use to change and work with your disk partitions and the filesystems created on those partitions.

Partitioning Hard Disks

Historically, PC hard drives have used a 32-bit PC-BIOS partition table with a Master Boot Record (MBR). This limits partition sizes to 2TB and allows only four primary partitions per drive. The use of extended partitions is a way to overcome the four primary partition limit. In order to overcome the 2TB limit, PC-BIOS partition tables are being replaced with GPT (GUID Partition Tables).

The old standard command for working with disk partitions is `fdisk`. Because `fdisk` cannot work with GPT partitions, however, it is slowly being deprecated. A more powerful and actively supported tool is the `parted` command.

To try examples of `fdisk` and `parted` in this section, I recommend you get a USB flash drive that you don't mind erasing. When you insert it, the drive will show up as the next available sd device, such as `/dev/sdb` or `/dev/sdc`. The non-destructive `fdisk -`

1 command will help you find the right drive, so you don't risk damaging the main hard disk used on your system.

Note If you prefer to use graphical tools for partitioning, resizing, and otherwise manipulating your hard disk, you can try the `gparted` partitioning tool. Install the `gparted` package to get that tool, which is not installed by default.

Changing Disk Partitions with `fdisk`

The `fdisk` command is a useful Linux tool for listing and changing disk partitions. Keep in mind that modifying or deleting partitions can cause valuable data to be removed, so be sure of your changes before writing them to disk. To use the `fdisk` command to **list information about the partitions on your hard disk**, type the following command as root user:

```
$ sudo fdisk -l                                List disk partitions for every
disk
Disk /dev/sda: 82.3 GB, 82348277760 bytes
255 heads, 63 sectors/track, 10011 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks    Id  System
/dev/sda1    *           1          13       104391    83  Linux
/dev/sda2                14         9881     79264710    83  Linux
/dev/sda3           9882        10011      1044225    82  Linux swap
```

This example is for an 80GB hard disk that is divided into three partitions. The first (`/dev/sda1`) is a small `/boot` partition that is configured as a Linux filesystem (`Id` 83), which is appropriate for any ext filesystem. Note the asterisk (*), indicating that the first partition is bootable. The next partition is assigned to the root filesystem and is also a Linux partition. The final partition is `Linux swap`.

If multiple disks are present, `fdisk -l` will list them all unless you **indicate the specific disk** you want:

```
$ sudo fdisk -l /dev/sdb                        List partitions for a specific
disk
```

To **work with a specific disk** with the `fdisk` command, simply indicate the disk you want with no other options:

```
$ sudo fdisk /dev/sda                            Start interactive fdisk session
Command (m for help): m                        Type m to list help text as shown
Command action
a      toggle a bootable flag
```

```

b   edit bsd disklabel
c   toggle the dos compatibility flag
d   delete a partition
l   list known partition types
m   print this menu
n   add a new partition
o   create a new empty DOS partition table
p   print the partition table
q   quit without saving changes
s   create a new empty Sun disklabel
t   change a partition's system id
u   change display/entry units
v   verify the partition table
w   write table to disk and exit
x   extra functionality (experts only)

```

Command (m for help):

With the prompt displayed, you can use any of the commands shown to work with your hard disk. In particular, you can use `p` (to print the same listing as `fdisk -l`), `n` (to create a new partition), `d` (to delete an existing partition), `l` (to list known filesystem types), or `t` (to change the filesystem type for a partition). The following examples show some of those `fdisk` commands in action:

```

Command (m for help): d           Ask to delete a partition
Partition number (1-4): 4         Type partition number to
delete
...
Command (m for help): n           Create a new disk partition
First cylinder (1-4983, default 1): 1 Select start (or Enter)
Last cylinder ... (default 4983): 4983 Select end (or Enter)
...
Command (m for help): a           Make a partition bootable
Partition number (1-3): 1         Type bootable partition
number
...
Command (m for help): t           Select a filesystem type
Partition number (1-3): 3         Select partition to change
Hex code (type L to list codes): 82 Assign partition as swap

```

Unless you tell it otherwise, `fdisk` assumes the new partition is a Linux partition (83). You could have typed `L` to see the same listing of filesystem types and hex codes produced from the `l` command. As noted, `82` can assign the partition as `swap`. Other Linux partitions that may interest you include Linux extended (85), Linux LVM (8e), Linux software raid (`fd`), and EFI/GTP (`ee`).

For Windows partitions, you can assign a partition as HPFS/NTFS (7), Windows 95 FAT32 (b), FAT 16 (6), or Windows 95 FAT32 LBA (c). Other Unix-type filesystems include Minix (`be` or `bf`), BSD/OS (`e4`), FreeBSD (`ee`), OpenBSD (`ef`), NeXTSTEP

(`ef0`), Darwin UFS (`ef1`), and NetBSD (`ef4`). Any of these filesystem types might be useful if you have old backup media from those filesystems you want to restore.

So far, you have not made any permanent changes to your partition table. If you are now very sure that your new settings are correct, type `w` to write those changes to the partition table. To abandon your changes (or quit after writing your changes), type `q` to quit your `fdisk` session.

Copying Partition Tables with `sfdisk`

To **back up or replicate a disk's partition table**, use `sfdisk`:

```
$ sudo sfdisk -d /dev/sda > sda-table           Back up partition table
to file
$ sudo sfdisk /dev/sda < sda-table             Restore partition table
from file
$ sudo sfdisk -d /dev/sda | sfdisk /dev/sdb     Copy part table from a
to b
```

Changing Disk Partitions with `parted`

As with `fdisk`, `parted` can be used to list or change disk partitions. However, `parted` has a few other useful features as well. Here's how to **list partitions for a given disk**, `/dev/sda`, with `parted`:

```
$ sudo parted /dev/sda print
Model: ATA ST3160815AS (scsi)
Disk /dev/sda: 160GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
```

Number	Start	End	Size	Type	File system	Flags
1	1049kB	256MB	255MB	primary	ext2	boot
2	257MB	160GB	160GB	extended		
5	257MB	160GB	160GB	logical		lvm

This listing shows you whether you have a classic `msdos` disk label (partition table), or a `gpt` one. In this case, the partition table is `msdos`.

To **run `parted` interactively**, type `parted` followed by the name of the storage device you want to work with (such as `/dev/sda`). Or, if you have only one storage device, simply type `parted`. But, if you have a USB flash drive you don't mind erasing, I recommend you use that to try `parted` for the first time. Here, the USB drive is located on `/dev/sdc`:

```
$ sudo parted /dev/sdc
GNU Parted 2.3
Using /dev/sdc
Welcome to GNU Parted! Type 'help' to view a list of commands.
```

(parted)

To use `parted` interactively, either type whole commands or start with a few letters and use the Tab key to complete the command (as you would in the bash shell). And if you're really efficient, you can just type enough letters to allow `parted` to guess your input, as you would with Cisco IOS: `p` for print, `mk1` for `mklabel`, and so on.

Warning Unlike `fdisk`, `parted` immediately incorporates changes you make to your partitions, without explicitly writing the changes to disk. So don't just assume you can back out of any changes by simply quitting `parted`.

With each command in a `parted` session, you also have the option to enter the command with all the arguments (for example, `mkpart logical ext4 1MB 2GB`) or just enter the command (`mkpart`) and `parted` will guide you interactively:

Warning The following commands will change the configuration of your disk. That's why I use a USB flash drive that contains no data that I mind losing. If you make a mistake on your primary drive, you could make the data inaccessible and your system unbootable.

```
(parted) mkpart                                Create a new partition
Partition type?  [logical]? primary
File system type?  [ext2]? ext4
Start? 1MB
End? 2GB
```

The `parted` command is not recommended for resizing partitions because it can make the filesystems on those disks unusable. If you do use it to resize a partition, `parted` will try to resize the filesystem located on that partition as well. Because `parted` does not handle journaling or other features required by `ext3` and `ext4` filesystems, the resizing will fail for most useful cases.

Because resizing is an iffy proposition with standard disk partitions, if you suspect you will need to resize a partition at some point, consider using Logical Volume Manager to create and manage logical partitions. LVM is covered later in this chapter.

To **resize NTFS partitions**, you can use the `ntfsresize` command. In Ubuntu, that command comes with the `ntfsprogs` package. That package also comes with commands for creating (`mkfs.ntfs`), fixing (`ntfsfix`), and getting information about (`ntfsinfo`) NTFS partitions.

Working with Filesystem Labels

The term label, in terms of disk partitions, can refer to two different things. A **disk label** can be used as another name for a partition table, as seen in `parted` output. A **partition label** can also be the name of an individual partition that contains an ext filesystem.

Labels provide several advantages:

- You can use a label as the permanent name to mount a filesystem. It's possible that `/dev/sda` might be detected as `/dev/sdb` the next time the system reboots. Mounting by label lets you give the partition a unique name.
- If a label is on a removable USB drive, when you plug in that drive, Linux will use the label name as the name of the directory to mount on. For example, if the label is `mydocs`, the USB drive is mounted on the `/media/mydocs` directory.

To **see a partition's label** (if it has one), use the `e2label` command:

```
$ sudo e2label /dev/sda2  
/home
```

To **set the label on a partition**:

```
$ sudo e2label /dev/sda2 mypartition
```

Bear in mind that `/etc/fstab` can be set up to use the partition label to mount the partition, as in the following example. Changing this label may render the system unbootable.

```
LABEL=/boot                /boot                ext4                defaults            1  
2
```

To **find a partition when you know only the label**, type the following:

```
$ sudo findfs LABEL=mypartition  
/dev/sdc2
```

Formatting a Filesystem

With your disk partitions in place, you can build a filesystem of your choice on each partition. Most Linux systems come with the commands needed to make and check filesystems that are commonly used in Linux. Commands for **formatting and checking** filesystems are `mkfs` and `fsck`, respectively.

The `mkfs` command serves as the front end for many different commands aimed at formatting particular filesystem types, such as `mkfs.ext2`, `mkfs.ext3`, `mkfs.ext4`, `mkfs.cramfs`, `mkfs.msdos`, `mkfs.ntfs`, and `mkfs.vfat`. By adding packages that support other filesystems, additional `mkfs` commands are available to seamlessly work with `mkfs`. These include `mkfs.bfs`, `mkfs.minix`, `mkfs.xfs`, and `mkfs.xiafs`. Use each command directly (as in `mkfs.vfat /dev/sdb1`) or via the `mkfs` command (as in `mkfs -t vfat /dev/sdb1`).

Creating a Filesystem on a Disk Partition

Basic software packages you need in Ubuntu to do filesystem creation and checking include `util-linux` (includes `mkfs` and other general utilities) and `e2fsprogs` (ext2/ext3/ext4-specific tools). Specific `mkfs` commands for different filesystem types are included in `ntfsprogs` (ntfs), `dosfstools` (msdos and vfat), `xfsprogs` (xfs), `jfsutils` (jfs), `mtd-utils` (jffs and jffs2), and `reiserfs-utils` (reiserfs). The basic tools get installed with Ubuntu.

Here are examples of the **mkfs command to create filesystems** (be sure to add the `-t` option first):

```
$ sudo mkfs -t ext4 /dev/sdb1          Create ext4 file system on  
sdb1  
$ sudo mkfs -t ext4 -v -c /dev/sdb1    More verbose/scan for bad  
blocks  
$ sudo mkfs.ext4 -c /dev/sdb1          Same result as previous  
command
```

If you would like to add a partition label to the new partition when the filesystem is created, use the `-L` option:

```
$ sudo mkfs.ext4 -c -L mypartition /dev/sdb1  Add mypartition label
```

Creating a Virtual Filesystem

If you want to try out different filesystem types or simply make a filesystem that is more portable (in other words, not tied to a physical disk), you can create a **virtual filesystem**. A virtual filesystem is one that sits within a file on an existing filesystem. You can format it as any filesystem type you like, move it around, and use it from different computers.

Virtual filesystems are useful for such things as creating live CDs or running dedicated virtual operating systems. In the example that follows, you create a blank 1GB disk image file, format it as a filesystem, and then mount it to access data on the filesystem:

```
$ dd if=/dev/zero of=mydisk count=2048000    Create zero-filled 1GB  
file  
$ du -sh mydisk                               Check virtual file size  
1001M    mydisk  
$ mkfs -t ext4 mydisk                           Create file system on  
mydisk  
mydisk is not a block special device  
Continue (y/n): y  
$ sudo mkdir /mnt/image                         Create a mount point  
$ sudo mount -o loop mydisk /mnt/image          Mount mydisk on
```

/mnt/image

In this procedure, the `dd` command creates an empty disk image file of 2048000 blocks (about 1GB). The `mkfs` command can create any filesystem type you choose (ext4 is done here). Because the file is not a block special device, as is the case when formatting disk partitions, `mkfs` will warn you before starting to make the filesystem. The only other trick, after creating the mount point, is to indicate that you are mounting the file (`mydisk`) as a loop device (`-o loop`). Note that the `mount` command is the only command shown in the preceding code that requires root privilege.

When the virtual filesystem is mounted, in this example under `/mnt/image`, you can access it as you would any filesystem. When you are done with the filesystem, leave it and unmount it:

\$ <code>sudo cd /mnt/image</code>	Change to the mount point
\$ <code>sudo mkdir test</code>	Create a directory on the file
system	
\$ <code>sudo cp /etc/hosts .</code>	Copy a file to the file system
\$ <code>cd</code>	Leave the file system
\$ <code>sudo umount /mnt/image</code>	Unmount the file system

With the virtual filesystem unmounted, you could move it to another system or burn it to a CD to use a filesystem in another location. If you don't want the filesystem any more, simply delete the file.

Viewing and Changing Filesystem Attributes

Using the `tune2fs` or `dumpe2fs` commands, you can **view attributes of ext2, ext3, and ext4 filesystems**. The `tune2fs` command can also be used to **change filesystem attributes**. Use the `swapfs` command to **create a swap partition**. Here are examples (both commands produce the same output):

\$ <code>sudo tune2fs -l /dev/sda1 less</code>	View tunable file system
attributes	
\$ <code>sudo dumpe2fs -h /dev/sda1</code>	Same as tune2fs output
dumpe2fs 1.42 (29-Nov-2011)	
Filesystem volume name:	<none>
Last mounted on:	<not available>
Filesystem UUID:	f5f261d3-3879-41d6-8245-f2153b003204
Filesystem magic number:	0xEF53
Filesystem revision #:	1 (dynamic)
Filesystem features:	filetype sparse_super
Default mount options:	(none)
Filesystem state:	clean
Errors behavior:	Continue
Filesystem OS type:	Linux
Inode count:	7914368

```

Block count:                7907988
Reserved block count:       395399
Free blocks:                5916863
Free inodes:                7752077
First block:                0
Block size:                 4096
Fragment size:              4096
Reserved GDT blocks:        1022
Blocks per group:           32768
Fragments per group:        32768
Inodes per group:           32704
Inode blocks per group:     1022
Filesystem created:         Fri Jun 15 12:13:17 2007
Last mount time:            Sat Mar 12:13:14 2013
Last write time:            Sat Mar 12:13:14 2013
Mount count:                2
Maximum mount count:        29

```

The output shows a lot of information about the filesystem. For example, if you have a filesystem that needs to create many small files (such as a news server), you can check that you don't run out of inodes. Setting the `Maximum mount count` ensures that the filesystem is checked for errors after it has been mounted the selected number of times. You can also find dates and times for when a filesystem was created, last mounted, and last written to.

To **change settings on an existing ext2, ext3, or ext4 filesystem**, you can use the `tune2fs` command. The following command changes the number of mounts before a forced filesystem check:

```

$ sudo tune2fs -c 31 /dev/sda1      Sets # of mounts before forced
check
tune2fs 1.42 (29-Nov-2011)
Setting maximal mount count to 31

```

If you'd like to switch to forced **filesystem checks based on time interval** rather than number of mounts, disable mount-count checking by setting it to negative 1 (-1):

```

$ sudo tune2fs -c -1 /dev/sda1
tune2fs 1.42 (29-Nov-2011)
Setting maximal mount count to -1

```

Use the `-i` option to **enable time-dependent checking**. Here are some examples:

```

$ sudo tune2fs -i 10 /dev/sda1      Check after 10 days
$ sudo tune2fs -i 1d /dev/sda1      Check after 1 day
$ sudo tune2fs -i 3w /dev/sda1      Check after 3 weeks
$ sudo tune2fs -i 6m /dev/sda1      Check after 6 months
$ sudo tune2fs -i 0 /dev/sda1      Disable time-dependent
checking

```

Be sure you always have either mount-count or time-dependent checking turned on. Use the `-j` option to **turn an ext2 filesystem into ext3** (by adding a journal):

```
$ sudo tune2fs -j /dev/sda1           Journaling changes ext2 to  
ext3
```

Creating and Using Swap Partitions

Swap partitions are needed in Linux systems to hold data that overflows from your system's RAM. If you didn't create a swap partition when you installed Linux, you can create it later using the `mkswap` command. You can **create your swap partition** either on a regular disk partition or in a file formatted as a swap partition. Here are some examples:

```
$ sudo mkswap /dev/sda6           Format sda6 as a swap partition  
Setting up swspace version 1, size = 205594 kB
```

To **check your swap area for bad blocks**, use the `-c` option to `mkswap`:

```
$ sudo mkswap -c /dev/sda1
```

If you don't have a spare partition, you can **create a swap area within a file**:

```
$ sudo dd if=/dev/zero of=/mnt/swapfile count=65536  
65536+0 records in  
65536+0 records out  
33554432 bytes (34 MB) copied, 1.56578 s, 21.4 MB/s  
$ sudo chmod 600 /mnt/swapfile  
$ sudo mkswap /mnt/swapfile  
Setting up swspace version 1, size = 67104 kB
```

The `dd` command in the preceding code creates a 32MB file named `swapfile`. The `chmod` command locks down the permissions on the file to avoid getting a warning from the `swapon` command down the road. The `mkswap` command formats the `/mnt/swapfile` file to be a swap partition.

After you have created a swap partition or swap file, you need to **tell the system to use the swap area** you made using the `swapon` command. This is similar to what happens at boot time. Consider the following examples:

```
$ sudo swapon /dev/sda1           Turn swap on for /dev/sda1  
partition  
$ sudo swapon -v /dev/sda1       More verbosity as swap is turned  
on  
swapon on /dev/sda1  
$ sudo swapon -v /mnt/swapfile   Turn swap on for /mnt/swapfile  
swapon on /mnt/swapfile
```

You can also use the `swapon` command to **see a list of your swap files and**

partitions:

```
$ swapon -s                                View all swap files and partitions that are on
```

Filename	Type	Size	Used	Priority
/dev/sda5	partition	1020088	142764	-1
/mnt/swapfile	file	65528	0	-6

Before you turn off a swap area, make sure no space is being used on it (look for a “0” under the Used column). Then, to **turn off a swap area**, you can use the `swapoff` command:

```
$ sudo swapoff -v /mnt/swapfile  
swapoff on /mnt/swapfile
```

Swap areas are prioritized. The kernel will swap first to areas of high priorities and then go down the list. Areas of the same priority get striped between. You can **specify the priority of your swap area** as you enable it using the `-p` option:

```
$ sudo swapon -v -p 1 /dev/sda1           Assign top swap priority to  
sda1
```

Mounting and Unmounting Filesystems

Before you can use a regular, non-swap filesystem, you need to attach it to a directory in your computer’s filesystem tree by **mounting** it. Your root filesystem (/) and other filesystems you use on an ongoing basis are typically mounted automatically based on entries in your `/etc/fstab` file. Other filesystems can be mounted manually as they are needed using the `mount` command.

Mounting Filesystems from the fstab File

When you first install Linux, the `/etc/fstab` file is usually set up automatically to contain information about your root filesystems and other filesystems. Those filesystems can then be set to mount at boot time or be ready to mount manually (with mount points and other options ready to use when a manual mount is done).

Here is an example of a `/etc/fstab` file:

proc	/proc	proc nodev,noexec,nosuid	0 0
/dev/mapper/abc-root	/	ext4 errors=remount-ro	0 1
LABEL=/boot	/boot	ext2 defaults	0 2
/dev/mapper/abc-swap	swap	swap defaults	0 0
/dev/sdb1	/mnt/windows	vfat noauto	0 0

Note For clarity, the UUID listing for the `/boot` filesystem was removed in the preceding example. For the `/boot` filesystem, you'll normally see an entry such as `UUID=da2dbc48-862e-4fbe-9529-a88b57b15bac` instead of the UUID. Either UUID or LABEL can be used to provide a way of uniquely identifying a disk partition.

All the filesystems are mounted automatically, except for `/dev/sdb1` (as indicated by the `noauto` option). The root (`/`) and `swap` disk partitions are configured as logical volume management (LVM) volumes. LVM volumes can make it easier to grow and shrink filesystems while still retaining the same device names. The `/dev/sdb1` disk partition was added manually in this example to mount the Windows partition located on the second hard disk device.

The `/etc/fstab` file no longer typically holds information about removable media. That's because the Hardware Abstraction Layer (HAL) facility automatically detects removable media and mounts those media in appropriate mount points in the `/media` directory (based on such things as volume ID on the media).

The following list describes each field in the `/etc/fstab` file.

1—The device name representing the filesystem. Originally, this contained the device name of the partition to mount (such as `/dev/sda1`). It can now also contain a LABEL, a universally unique identifier (UUID), or a remote filesystem (NFS or CIFS) instead of a device name.

2—The mount point in the filesystem. The filesystem contains all data from the mount point down the directory tree structure, unless another filesystem is mounted at some point beneath it.

3—The filesystem type. See the section “Understanding Filesystem Basics “ for a list of many common filesystem types.

4—The mount command options. Examples of `mount` options include `noauto` (to prevent the filesystem from mounting at boot time) and `ro` (to mount the filesystem read-only). To let any user mount a filesystem, you could add the `user` or `owner` option to this field. Commas must separate options. See the `mount` command manual page (under the `-o` option) for information on other supported options.

5—Dump filesystem? This field is only significant if you run backups with `dump` (which is rare). A number 1 signifies that the filesystem needs to be dumped. A zero means that it doesn't.

6—Filesystem check? The number in this field indicates whether or not the filesystem needs to be checked with `fsck`. A zero indicates that the filesystem should not be checked. A number 1 means that the filesystem needs to be checked

first (this is used for the root filesystem). A number 2 assumes that the filesystem can be checked at any point after the root filesystem is checked.

You can create your own entries for any hard disk or removable media partitions you want in the `/etc/fstab` file. Remote filesystems (NFS, Samba, and others) can also contain entries in the `/etc/fstab` file to automatically mount those filesystems at boot time or later by hand.

Mounting Filesystems with the mount Command

The `mount` command is used to view mounted filesystems, as well as mount any local (hard disk, USB drive, CD, DVD, and so on) or remote (NFS, Samba, and so on) filesystems immediately. Here is an example of the `mount` command for **listing mounted filesystems**:

```
$ mount                                List mounted remote and local file
systems
proc on /proc type proc (rw,noexec,nosuid,nodev)
/dev/mapper/abc-root on / type ext4 (rw,errors=remount-ro)
/dev/sda1 on /boot type ext2 (rw)
192.1.0.9:/volumel/books on /mnt/books type nfs (rw,addr=192.1.1.9)
/dev/sdb1 on /media/myusb type ext4
(rw,nosuid,nodev,uhelper=udisks)
```

Use the `-t` option to **list only mounts of a specific filesystem type**:

```
$ mount -t ext4                        List mounted ext4 file systems
/dev/mapper/abc-root on / type ext4 (rw,errors=remount-ro)
/dev/sdb1 on
/media/myusb type ext4 (rw,nosuid,nodev,uhelper=udisks)
```

To **display partition labels with mount information**, use the `-l` option:

```
$ mount -t ext4 -l                    List mounted ext4 file systems and
labels
/dev/sda7 on / type ext4 (rw) [/123]
/dev/sda6 on /mnt/debian type ext4 (rw) [/mnt/debian]
/dev/sda3 on /mnt/slackware type ext4 (rw) [/mnt/slackware]
```

Here is a simple `mount` command to mount the `/dev/sdb1` device on an existing directory named `/mnt/mymount`:

```
$ sudo mount /dev/sdb1 /mnt/mymount/    Mount a local file system
$ sudo mount -v /dev/sdb1 /mnt/mymount/ Mount /dev/sdb1 more
verbose
mount: you didn't specify a filesystem type for /dev/sdb1
I will try type ext4
/dev/sdb1 on /mnt/mymount type ext4 (rw)
```


In the preceding examples, the `mount` command will either look for an entry for `/dev/sdb1` in the `/etc/fstab` file or try to guess the type of filesystem.

Use `-t` to **explicitly indicate the type of filesystem to mount**:

```
$ sudo mount -v -t ext4 /dev/sdb1 /mnt/mymount/ Mount an ext4 file
system
/dev/sdb1 on /mnt/mymount type ext4 (rw)
```

You can also **display the label/name of the partition** that is mounted:

```
$ sudo mount -vl -t ext4 /dev/sdb1 /mnt/mymount/ Mount and show
label
```

If you're mounting something that is listed in your `fstab` file already, you need to specify only one item: mount point or device. For example, with the following `fstab` entry:

```
/dev/sdb1          /mnt/mymount      ext4      defaults      1 2
```

you can do either of the following to mount the filesystem:

```
$ sudo mount -v /dev/sdb1 Mount file system with device name
only
/dev/sdb1 on /mnt/mymount type ext4 (rw)
$ sudo mount -v /mnt/mymount/ Mount file system with mount point
only
/dev/sdb1 on /mnt/mymount type ext4 (rw)
```

You can **specify mount options** by adding `-o` and a comma-separated list of options. They are the same options you can add to field 4 of the `/etc/fstab` file. By default, partitions are mounted with read/write access. You can explicitly indicate to **mount a filesystem as read/write (rw) or read-only (ro)**:

```
$ sudo mount -v -t ext4 -o rw /dev/sdb1 /mnt/mymount/ Mount
read/write
/dev/sdb1 on /mnt/mymount type ext4 (rw)
$ sudo mount -v -t ext4 -o ro /dev/sdb1 /mnt/mymount/ Mount read-
only
/dev/sdb1 on /mnt/mymount type ext4 (ro)
```

A few other useful `mount` options you can use include:

- `noatime`—Does not update the access time on files. Good on filesystems with a lot of I/O, such as mail spools and logs.
- `noexec`—Prevents execution of binaries located on this filesystem. Can be used to increase security, for example for `/tmp` in environments with untrusted users.
- `remount`—Change options on a mounted filesystem. With `remount`, you can unmount the filesystem and remount it with the new options in a single command. In this example, we change a previous read/write mount to read-only:

```
$ sudo mount -v -o remount,ro /dev/sdb1  
/dev/sdb1 on /mnt/mymount type ext4 (ro)
```

- **--bind**—Mount an existing filesystem to another location in the tree. Assuming /dev/sdb1 is already mounted on /mnt/mymount, type the following:

```
$ sudo mount --bind -v /mnt/mymount/ /tmp/mydir/  
/mnt/mymount on /tmp/mydir type none (rw,bind)
```

Now the same filesystem is accessible from two locations. The new mount point has the same mount options as the original.

- **--move**—Move a filesystem from one mount point to another. Assuming /dev/sdb1 is already mounted on /mnt/mymount, this moves the filesystem to /tmp/mydir:

```
$ sudo mount -v --move /mnt/mymount/ /tmp/mydir/  
/mnt/mymount on /tmp/mydir type none (rw)
```

Just as you can swap to a file, you can create a filesystem in a file and then mount it in what is called a **loopback** mount. Creating and mounting such a file is described in the “Creating a Virtual Filesystem” section earlier in this chapter. A common situation where you might want to **mount a file in loopback** is after downloading a Linux install CD or live CD. By mounting that CD image in loopback, you can view its contents or copy files from that image to your hard disk.

In the following example, the `mount` command is allowed to automatically pick an existing loopback device when mounting a CD image file (filesystem type iso9660). The command output shows /dev/loop0 was selected:

```
$ sudo mount -v -t iso9660 -o loop /tmp/myimage.iso /mnt/mymount/  
mount: going to use the loop device /dev/loop0  
/tmp/myimage.iso on /mnt/mymount type ext4 (rw,loop=/dev/loop0)
```

In the following example, I downloaded a Linux USB flash drive boot image called `diskboot.img` to /tmp. Here is an example of how to **mount the boot image**:

```
$ sudo mount -v -o loop /tmp/diskboot.img /mnt/mymount  
mount: going to use the loop device /dev/loop0  
mount: you didn't specify a filesystem type for /dev/loop0  
I will try type vfat  
/tmp/diskboot.img on /mnt/mymount type vfat (rw,loop=/dev/loop0)
```

To see the status of the loopback devices, use the `losetup` command:

```
$ sudo losetup /dev/loop0  
List mounted loopback devices  
/dev/loop0: [0807]:1009045 (/tmp/diskboot.img)
```

If a loopback mount gets stuck and you have problems during unmount, try detaching it as follows:

```
$ sudo losetup -d /dev/loop1  
Force unmount mounted loopback
```

device

Note The `mount` command can also be used to attach to NFS, or Samba/Windows CIFS shares. See Chapter 12 for information on mounting those remote filesystem types.

Unmounting Filesystems with `umount`

To **unmount a filesystem**, use the `umount` command. You can `umount` the filesystem using the device name or the mount point. You're better off unmounting with the mount point, to avoid the confusion when using bind mounts (one device, multiple mount points). Here is an example of each, with verbosity on:

```
$ sudo umount -v /dev/sdb1           Unmount by device name
/dev/sdb1 unmounted
$ sudo umount -v /mnt/mymount/       Unmount by mount point
/tmp/diskboot.img unmounted
```

If the device is busy, the unmount will fail. A common reason for an unmount to fail is that you have a shell open with the current directory of a directory inside the mount:

```
$ sudo umount -v /mnt/mymount/
umount: /mnt/mymount: device is busy
umount: /mnt/mymount: device is busy
```

Sometimes, it's not obvious what makes the device busy. You can use `lsof` to list open files, and then **search that list for the mount point** that interests you:

```
$ sudo lsof | grep mymount           Find open files on mymount
partition
bash    11224  chris  cwd    DIR    8,17   4096    2 /mnt/mymount
```

You can see that a bash process run by chris with a PID of 9341 is preventing the mymount partition from being unmounted.

Another option when a filesystem is busy is to **perform a lazy unmount**:

```
$ sudo umount -vl /mnt/mymount/       Perform a lazy unmount
```

A **lazy unmount** unmounts the filesystem from the tree now, but waits for the device to no longer be busy before cleaning up everything. Unmounts of removable media can also be done with `eject`. This **unmounts a CD and ejects the CD** from the drive:

```
$ sudo eject /dev/cdrom               Unmount and eject a CD
```

Checking Filesystems

In Linux, instead of having the scandisk utility you have in Windows, you can scan a physical device for bad blocks at a physical level with the `badblocks` command and scan a filesystem for errors at the logical level with the `fsck` command. Here's how to **scan for bad blocks**:

```
$ sudo badblocks /dev/sdb1           Physically scan disk for bad
blocks
$ sudo badblocks -v /dev/sdb1       Add verbosity to hard disk scan
Checking blocks 0 to 200781
Checking for bad blocks (read-only test): done
Pass completed, 0 bad blocks found.
```

By default, `badblock` does a safe read-only test of the blocks. You can also perform a non-destructive read/write test. This is the slowest test, but the best one you can perform without destroying the data on the device. Add `-s` to **see the ongoing progress**:

```
$ sudo badblocks -vsn /dev/sdb1      Check bad blocks, non-
destructive
Checking for bad blocks in non-destructive read-write mode
From block 0 to 200781
Testing with random pattern: Pass completed, 0 bad blocks found.
```

To test a disk that has no data on it that you need to keep, the following command performs a **faster, destructive read-write test**:

Warning This will erase all the data on the partition.

```
$ sudo badblocks -vsw /dev/sda1      Destructive check for bad
blocks
Checking for bad blocks in read-write mode
From block 0 to 200781
Testing with pattern 0xaa: done
Reading and comparing: done
...
Pass completed, 0 bad blocks found.
```

You can **perform multiple badblocks passes**; for example, this command line can be used to burn in a drive and screen for hard drive infant mortality:

```
$ sudo badblocks -vswp 2 /dev/sda1
```

Like the `mkfs` command, the `fsck` command is just a front end to filesystem-specific utilities. You can **check an ext filesystem** by simply adding the device name of the disk partition you want to check to the `fsck` command:

```
$ sudo fsck /dev/sdb1
fsck from util-linux 2.20.1
mypart was not cleanly unmounted, check forced.
```

```
fsck 1.42 (29-Nov-2011)
e2fsck 1.42 (29-Nov-2011)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
mypart: 11/50200 files (9.1% non-contiguous), 12002/200780 blocks
```

You can **add other options to fsck**, such as **-T** (to not display the useless `fsck` version number) and **-v** (to be more verbose about what `fsck` actually does):

```
$ sudo fsck -TV /dev/sdb1          Check file system, verbose no
version
[/sbin/fsck.ext4 (1) -- /dev/sdb1] fsck.ext4 /dev/sdb1
e2fsck 1.42 (29-Nov-2011)
mypart: clean, 11/50200 files, 12002/200780 blocks
```

For any problem that `fsck` encounters, it will ask you if you want to repair it:

```
$ sudo fsck -TV /dev/sdb1          Prompting to correct problems
encountered
[/sbin/fsck.ext4 (1) -- /mnt/mymount] fsck.ext4 /dev/sda1
e2fsck 1.39 (29-Nov-2011)
Couldn't find ext2 superblock, trying backup blocks...
Resize inode not valid.  Recreate<y>? y
```

Unless you have a very in-depth knowledge of filesystems, you're better off answering yes. This can be done automatically with the **-y** option:

```
$ sudo fsck -TVy /dev/sdb1
[/sbin/fsck.ext4 (1) -- /mnt/mymount] fsck.ext4 -y /dev/sdb1
e2fsck 1.42 (29-Nov-2011)
Couldn't find ext2 superblock, trying backup blocks...
Resize inode not valid.  Recreate? yes
mypart was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
Free blocks count wrong for group #0 (3552, counted=3553).
Fix? yes
Free blocks count wrong (188777, counted=188778).
Fix? yes
mypart: ***** FILE SYSTEM WAS MODIFIED *****
mypart: 11/50200 files (0.0% non-contiguous), 12002/200780 blocks
```

Creating Encrypted Filesystems

Using components of the `cryptsetup` package, you can create encrypted filesystems to protect the data on your hard disks or removable media. A common practice is to encrypt the data on a separate `/home` partition on your laptop or on a USB drive you carry in your pocket. If those devices were lost or stolen, nobody could access your data without the passphrase.

Encrypting the `/home` partition on your Ubuntu system just means selecting the right options during the disk partition steps of installing Ubuntu. Encrypting a disk partition later, using the command line, requires several steps, as shown in the list that follows.

To **encrypt your USB drive** so that only someone who knows the password can access the encrypted data on the drive, you can do the following:

1. Watch the `/var/log/messages` file using the `tail` command. Once `tail` is running, insert your USB drive into your computer and you will see messages related to the USB drive being detected:

```
$ sudo tail -f /var/log/messages
Apr 11 05:54:09 myubuntu kernel: [4486653.002219] sd 11:0:0:0:
    Attached scsi generic sg3 type 0
Apr 11 05:54:09 myubuntu kernel: [4486653.004329] sd 11:0:0:0:
    [sd] 7831552 512-byte logical blocks: (4.00 GB/3.73 GiB)
Apr 11 05:54:09 myubuntu kernel: [4486653.005707] sd 11:0:0:0:
    [sd] Write Protect is off
Apr 11 05:54:09 myubuntu kernel: [4486653.014322] sd 11:0:0:0:
    [sd] Attached SCSI removable disk
```

2. Note information about the device. In this example, the USB storage device is assigned to the `sdc` device (`/dev/sdc`). It is a 4.00 GB device and it is not write protected. It is important to get the device name correct or you could erase your Ubuntu system or other critical data in the next step!

3. If you want to **erase all old data on the device**, you can use the `dd` command as follows:

```
$ sudo dd if=/dev/zero of=/dev/sdc
dd: writing to `/dev/sdc': No space left on device
7831553+0 records in
7831552+0 records out
4009754624 bytes (4.0 GB) copied, 1074.47 s, 3.7 MB/s
```

4. **Partition the USB drive** to prepare it for the encrypted filesystem. Because the disk is empty, first simply save a blank partition table to the disk:

```
$ sudo fdisk /dev/sdc
Command (m for help): w
```

5. Create a single partition on the USB drive as follows:

```
$ sudo fdisk /dev/sdc
Command (m for help): n
Partition type:
    p   primary (0 primary, 0 extended, 4 free)
```

```

    e    extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-78315, default 2048): <ENTER>
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-78315, default 78315):
<ENTER>
Using default value 78315
Command (m for help): w

```

6. Install encryption software on your Ubuntu system:

```
$ sudo apt-get install cryptsetup
```

7. Load the modules needed to encrypt your USB drive:

```

$ sudo modprobe dm-crypt
$ sudo modprobe aes
$ sudo modprobe sha512

```

8. Encrypt the USB partition with the `cryptsetup` command. Type **YES** when prompted. You must then be sure to remember the passphrase or you will not be able to access the data you put on the drive later:

```

$ sudo cryptsetup luksFormat /dev/sdc1
WARNING!
=====
This will overwrite data on /dev/sdc1 irrevocably.
Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase: *****
Verify passphrase: *****

```

9. Open the encrypted partition as follows (I named the device `mycrypt`, but you can use any name you like):

```

$ sudo cryptsetup luksOpen /dev/sdc1 mycrypt
Enter passphrase for /dev/sdc1: *****

```

10. Format the partition as you would any disk partition (in this case, an ext4 filesystem type). The device name of the new filesystem partition includes the name you assigned on the `luksOpen` line in the `/dev/mapper` directory (`/dev/mapper/mycrypt` in this example):

```
$ sudo mkfs -t ext4 /dev/mapper/mycrypt
```

11. You can mount the filesystem. Then change the ownership to the user you want to own the USB partition (the user named `chris` and the group named `chris`, in this case):

```

$ sudo mkdir /mnt/testing
$ sudo mount /dev/mapper/mycrypt /mnt/testing/
$ sudo chown chris:chris /mnt/testing

```

12. You can copy some files to the new partition to make sure it's working. When you are done, unmount the partition. Then you must close the partition using the `luksClose` option to `cryptsetup`, along with the name in the `/dev/mapper` directory that represents the encrypted partition:

```
$ sudo umount /dev/mapper/mycrypt
```

```
$ sudo cryptsetup luksClose \  
udisks-luks-uuid-9f8cb640-8a9b-4ae0-ac31-48aff59bf167-uid13597
```

You can now safely remove the USB drive. The next time you insert the drive in any Linux system that supports this type of encryption, what happens next depends on whether you have a desktop running or not:

- From a desktop, a pop-up window asks for the passphrase. Once you enter it, the unencrypted filesystem is mounted under the `/media` directory and a file manager window opens for you to access the data on the partition. When you are done using it, right-click the icon representing the partition on your desktop and select Safely Remove. Then you can safely remove your USB drive.
- From a shell console, you are prompted for the passphrase you entered when you created the encrypted partition. At that point, you may need to manually mount the filesystem (named something like `/dev/mapper/mycrypt`) to use it. Once you are done using the filesystem, unmount it (`umount`) and run `cryptsetup luksClose` on the encrypted device, as shown earlier in this procedure. Then you can safely remove the drive.

Checking RAID Disks

Redundant Array of Independent Drives (RAID) disks let you duplicate or distribute data across multiple hard drives. Using RAID can improve reliability and performance of your storage media. The `mdadm` command, which is part of the `mdadm` package, can be used to **check softraid devices** on your computer. Consider this example:

```
$ sudo apt-get install mdadm  
$ sudo mdadm -Q /dev/md1  
/dev/md1: 1498.13MiB raid1 2 devices, 0 spares.  
      Use mdadm --detail for more detail.  
/dev/md1: No md super block found, not an md component.
```

The message on the last line simply means that `/dev/md1` is not a member of a RAID array. That is normal because `md1` is the array itself. Similarly, if you query a member of a RAID array, your output will look like this:

```
$ sudo mdadm -Q /dev/sdb3  
/dev/sdb3: is not an md array  
/dev/sdb3: device 1 in 4 device active raid6 md0.  
Use mdadm --examine for more detail.
```

To obtain more detailed output, add the `--detail` option:

```
$ sudo mdadm -Q --detail /dev/md1  
/dev/md1:  
      Version : 00.90.01
```



```

Creation Time : Fri Apr 5 16:32:12 2013
Raid Level : raid1
Array Size : 1534080 (1498.38 MiB 1570.90 MB)
Device Size : 1534080 (1498.38 MiB 1570.90 MB)
Raid Devices : 2
Total Devices : 2
Preferred Minor : 1
Persistence : Superblock is persistent
Update Time : Mon Apr 15 02:06:01 2013
State : clean
Active Devices : 2
Working Devices : 2
Failed Devices : 0
Spare Devices : 0
    UUID : 49c564cc:2d3c9a14:d93ce1c9:070663ca
Events : 0.42

```

Number	Major	Minor	RaidDevice	State	
0	3	2	0	active sync	/dev/hda2
1	3	66	1	active sync	/dev/hdb2

The `mdadm` command can also be used to manage your softraid devices. For more info, run the following:

```

$ sudo mdadm --manage -help
$ man mdadm

```

Make sure that you have a RAID disk prior to installing the `mdadm` package. When you install `mdadm`, the installation program will attempt to configure your RAID drives as part of the installation. Furthermore, should you uninstall `mdadm`, it will likely leave behind a `mdadm.conf` file in `/etc/mdadm`. The presence of this file can cause compatibility issues with the `lvm2` package, described later.

Finding Out about Filesystem Use

Running out of disk space can be annoying on your desktop system and potentially a disaster on your servers. To determine how much disk space is available and how much is currently in use, you can use the `df` command. To check how much space particular files and directories are consuming, use the `du` command.

The `df` command provides **utilization summaries of your mounted filesystems**. Using the `-h` option, you can have the data (which is shown in bytes by default) converted to megabytes (M) and gigabytes (G), to make that output more human-readable:

```

$ df -h           Display space on filesystems in human-readable
form

```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda2	7.6G	3.4G	3.9G	47%	/
/dev/sda1	99M	14M	80M	15%	/boot
tmpfs	501M	0	501M	0%	/dev/shm
/dev/sda5	352G	197G	137G	59%	/home
//thompson/chris	9204796	5722608	3007068	66%	/mnt/mymount

Because ext filesystems have only so many inodes created at `mkfs` time, if you have lots of small files, you can possibly run out of inodes before you run out of actual space. To **check inode utilization**, use the `-i` option:

```
$ df -hi
Filesystem          Inodes    IUsed    IFree IUse% Mounted on
/dev/sda2           2.0M      108K     1.9M    6% /
```

If you have network mounts (such as Samba or NFS), these will show up, too, in your `df` output. To **limit `df` output to local filesystems**, type the following:

```
$ df -hl           Display disk space only for local file systems
```

To **add the filesystem type to the listing**, use the `-T` option:

```
$ df -hT           Add file system type information to listing
Filesystem      Type    Size    Used Avail Use% Mounted on
/dev/sda7       ext4    8.8G    5.5G    2.9G   66% /
```

To **check for disk space usage for particular files or directories in a filesystem**, use the `du` command. The following command was run as the user named `chris`:

```
$ du -h /home/      Show disk space usage for /home directory
du: `/home/joe': Permission denied
4.0K    /home/chris/Mail
52K     /home/chris
64K     /home/
```

The output shows that access to another home directory's disk use (in this case `/home/joe`) was denied for security reasons. So the next examples show how to **avoid permission issues and get totals that are correct by using the root user account**. This is clearly visible when you use `-s` to summarize:

```
$ du -sh /home      Regular user is denied space totals to others'
homes
du: `/home/joe': Permission denied
du: `/home/horatio199': Permission denied
64K     /home
$ sudo du -sh /home  You can display summary disk use as root
user
1.6G    /home
```

You can **specify multiple directories** with the `-c` option and total them up:

```
$ sudo du -sch /home /var    Show directory and total summaries
```

```
1.6G    /home
111M    /var
1.7G    total
```

You can **exclude files that match a pattern** from being counted using the `exclude` option. In the following example, disk image files (ending with the `.iso` suffix) are not used in totaling the disk space used:

```
$ sudo du -sh --exclude='*.iso' /home/chris    Exclude ISOs from
totals
588M    /home/chris
```

You can **specify what depth in the tree** you want to summarize. Set `--max-depth` to a number larger than the 1 value shown to dig deeper into disk space usage:

```
$ sudo du -h --max-depth=1 /home    Provide disk use to one level
deep
1.6G    /home/chris
52K     /home/joe
1.6G    /home
$ sudo du -h --max-depth=2 /home    Dig two-levels deep for disk
space use
...
4.0K    /home/joe/Mail
52K     /home/joe
1.6G    /home
```

Logical Volume Manager

Logical Volume Manager (LVM) is a feature designed to help you cope with the changing needs for disk space on your Linux systems. With your hard disks configured as LVM volumes, you have tremendous flexibility in growing, shrinking, and moving the storage space on your systems as your needs change. LVM also allows for snapshots, a feature typically found on expensive enterprise SANs, or Storage-Area Networks.

Ubuntu incorporates LVM2 into its releases and uses it to define how disk partitions are allocated when you first install Ubuntu. Using LVM2, you define and manage volume groups (`vg`), logical volumes (`lv`), and physical volumes (`pv`). Each logical volume and physical volume is divided up into logical extents and physical extents, respectively.

The basic business of using LVM is to create the volume groups and logical volumes you need, and then assign the extents (small chunks of disk space) to those areas where they are needed. Unlike older disk partitioning schemes, where you might have to back up your data, change your partitioning, and then return data to the resized partitions, you can simply add unused extents where they are needed. Likewise, if you run out of extents in a volume group, you can simply add more physical volumes so more space is

available.

To use LVM, you need to install the lvm2 package.

LVM comes with a set of commands that can be used to work with LVM volumes. Step through the procedure in the following section to learn about many of those LVM commands.

Warning To avoid messing up the hard disks your computer relies on as you learn LVM, I recommend you try the following examples on some non-critical storage device. For example, I used an inexpensive 4GB USB flash drive (on `/dev/sdc`) to run the commands shown in this section.

Creating LVM Volumes

To begin, use the `fdisk` command to **create physical partitions for the storage device** on which you want to create logical partitions. Here we have an 4GB USB flash drive, located on device `/dev/sdc`:

```
$ sudo fdisk /dev/sdc           Start command to manage disk
partitions
Command (m for help): p           Print current partitions (none
exist)
Disk /dev/sdc: 4009 MB, 4009754624 bytes
84 heads, 22 sectors/track, 4237 cylinders
Units = cylinders of 1848 * 512 = 946176 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xdba43801    Device Boot        Start          End
Blocks   Id   System
Command (m for help): n           Create a new partition
Command action
    e     extended
    p     primary partition (1-4)
p                               Make it a primary partition
Partition number (1-4): 1         Assigned to partition 1
First cylinder (2-4237, default 2): <Enter>
Using default value 2
Last cylinder or +size or +sizeM or +sizeK
(2-4237, default 4237): <Enter>
Using default value 4237
Command (m for help): t           Assign a partition type
Selected partition 1
Hex code (type L to list codes): 8E   Indicate 8E (LVM partition)
Changed system type of partition 1 to 8e (Linux LVM)
Command (m for help): p           Type p to see the new
```

partition

...

Device	Boot	Start	End	Blocks	Id	System
/dev/sdc1		1	4237	3914977	8e	Linux LVM

Before proceeding, make sure you have made the correct changes to the correct partition! If everything looks correct, write the new partition table, as follows:

Command (m for help): w

The partition table has been altered!

Calling ioctl() to re-read partition table.

Syncing disks.

Back at the shell prompt, use the `sfdisk` command to see the partitioning on the drive:

\$ sudo sfdisk -l /dev/sdc View the LVM partitions

Disk /dev/sdc: 4237 cylinders, 84 heads, 22 sectors/track

Units = cylinders of 946176 bytes, blocks of 1024 bytes, count from 0

Device	Boot	Start	End	#cyls	#blocks	Id	System
/dev/sdc1		0+	4236	4237-	3914977	8e	Linux LVM
/dev/sdc2		0	-	0	0	0	Empty
/dev/sdc3		0	-	0	0	0	Empty
/dev/sdc4		0	-	0	0	0	Empty

Next, make `/dev/sdc1` a new LVM physical volume and use the `pvs` command to view information about physical LVM volumes:

\$ sudo pvcreate /dev/sdc1 Make sdb1 an LVM physical volume

Physical volume "/dev/sdc1" successfully created

\$ sudo pvs View physical LVM partitions

PV	VG	Fmt	Attr	PSize	PFree
/dev/sdc1		lvm2	a--	3.73g	3.73g

Then use `vgcreate` to create the `vgusb` volume group and list the active current volume groups:

\$ sudo vgcreate vgusb /dev/sdc1 Create vgusb volume group

Volume group "vgusb" successfully created

\$ sudo vgs View current volume groups

VG	#PV	#LV	#SN	Attr	VSize	VFree
vgusb	1	0	0	wz--n-	3.73g	3.73g

Use `lvcreate` to create a new LVM partition of 1GB from the `vgusb` volume group. Then use `lvs` to see the logical volume and `vgs` to see that the amount of free space has changed:

\$ sudo lvcreate --size 1G --name lvm_u1 vgusb

Logical volume "lvm_u1" created

\$ sudo lvs View the logical volume information

LV	VG	Attr	LSize	Pool	Origin	Data%	Move	Log	Copy%
----	----	------	-------	------	--------	-------	------	-----	-------

```

Convert
lvm_u1 vgusb -wi-a--- 1.00g
$ sudo vg See that you still have 2.73GB free
  VG      #PV #LV #SN Attr   VSize  VFree
vgusb    1   1   0  wz--n- 3.73g  2.73g

```

To create an ext4 filesystem on the lvm partition, use the `mkfs.ext4` command as follows:

```

$ sudo mkfs.ext4 /dev/mapper/vgusb-lvm_u1
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
...
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 23 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.

```

The ext4 filesystem has now been created and the LVM volume is ready to use.

Using LVM Volumes

To use the new volume just created, represented by `/dev/mapper/vgusb-lvm_u1`, **create a mount point** (`/mnt/u1`) and **mount the volume**. Then use `df` to check the available space:

```

$ sudo mkdir /mnt/u1 Create mount point
$ sudo mount -t ext4 /dev/mapper/vgusb-lvm_u1 /mnt/u1 Mount volume
$ df -m /mnt/u1 Check disk space
Filesystem          1M-blocks      Used Available Use% Mounted on
/dev/mapper/vgusb-lvm_u1
1008                34          924      4% /mnt/u1

```

At this point, the filesystem contains only the `lost+found` directory:

```

$ ls /mnt/u1
lost+found

```

Copy a file to the new filesystem. For example, choose one of the kernel files from the `/boot` directory and copy it to `/mnt/u1`:

```

$ sudo cp /boot/vmlinuz-* /mnt/u1/ Copy a file to /mnt/u1
$ df -m /mnt/u1 See that 45MB is used on /mnt/u1
Filesystem          1M-blocks      Used Available Use% Mounted on
/dev/mapper/vgusb-lvm_u1
1008                45          913      5% /mnt/u1

```

Run `md5sum` on the file you copied and save the resulting checksum for later. For

example:

```
$ sudo md5sum /mnt/u1/vmlinuz-*                Check md5sum
56b9ca81bdfba6e563f407caf9a52a2b /boot/vmlinuz-3.2.0-38-generic
```

Growing the LVM Volume

Say that you are running out of space and you want to **add more space to your LVM volume**. To do that, unmount the volume and use the `lvresize` command. (Actually, it is not required that you unmount the volume to grow it, but it is done here as an extra precaution.) After that, you must also check the filesystem with `e2fsck` and run `resize2fs` to resize the ext4 filesystem on that volume:

```
$ sudo umount /mnt/u1                        Unmount volume
$ sudo lvresize --size 2G /dev/vgusb/lvm_u1    Resize volume
    Extending logical volume lvm_u1 to 2.00 GiB
    Logical volume lvm_u1 successfully resized
$ sudo e2fsck -f /dev/vgusb/lvm_u1
e2fsck 1.41.12 (17-May-2010)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/vgusb/lvm_u1: 14/65536 files (0.0% non-contiguous),
15546/262144 blocks$ sudo resize2fs /dev/vgusb/lvm_u1 2G  Resize
lvm_u1
resize2fs 1.41.12 (17-May-2010)
Resizing the filesystem on /dev/vgusb/lvm_u1 to 524288 (4k) blocks.
The filesystem on /dev/vgusb/lvm_u1 is now 524288 blocks long.
```

In the example just shown, the volume and the filesystem are both resized to 2GB. Next, mount the volume again and check the disk space and the md5sum you created earlier:

```
$ sudo mount -t ext4 /dev/mapper/vgusb-lvm_u1 /mnt/u1  Remount
volume
$ df -m /mnt/u1                                     See 45MB of 2016MB used
Filesystem          1M-blocks      Used Available Use% Mounted on
/dev/mapper/vgusb-lvm_u1
                    2016          45      1869    3% /mnt/u1
$ sudo md5sum /mnt/u1/vmlinuz-*                      Recheck md5sum
56b9ca81bdfba6e563f407caf9a52a2b /boot/vmlinuz-3.2.0-38-generic
```

The newly mounted volume is now 2GB instead of 1GB in size.

Shrinking an LVM Volume

You can also use the `lvresize` command if you want to **take unneeded space from an existing LVM volume**. As before, unmount the volume before resizing it and run `e2fsck` (to check the filesystem) and `resize2fs` (to resize it to the smaller size):

```
$ sudo umount /mnt/ul
$ sudo e2fsck -f /dev/vgusb/lvm_u1
e2fsck 1.41.12 (17-May-2010)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
...
/dev/vgusb/lvm_u1: 14/131072 files (0.0% non-contiguous),
19723/524288 blocks
$ sudo resize2fs /dev/vgusb/lvm_u1 1G      Resize fs
resize2fs 1.41.12 (17-May-2010)
Resizing the filesystem on /dev/vgusb/lvm_u1 to 262144 (4k) blocks.
The filesystem on /dev/vgusb/lvm_u1 is now 262144 blocks long.
$ sudo lvresize --size 1G /dev/vgusb/lvm_u1
WARNING: Reducing active logical volume to 1.00 GiB
THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce lvm_u1? [y/n]: y
Reducing logical volume lvm_u1 to 1.00 GiB
Logical volume lvm_u1 successfully resized
$ sudo mount -t ext4 /dev/mapper/vgusb-lvm_u1 /mnt/ul  Remount
volume
$ df -m /mnt/ul
See 4MB of 12MB used
Filesystem              1M-blocks      Used Available Use% Mounted on
/dev/mapper/vgusb-lvm_u1
                        1008           45       913      5% /mnt/ul
```

The newly mounted volume appears now as 1008MB instead of 2016MB in size.

Removing LVM Logical Volumes and Groups

To remove an LVM logical volume from a volume group, unmount it and then use the `lvremove` command as follows:

```
$ sudo umount /dev/vgusb/lvm_u1
$ sudo lvremove /dev/vgusb/lvm_u1
Do you really want to remove active logical volume "lvm_u1"? [y/n]:
y
Logical volume "lvm_u1" successfully removed
```

To remove an existing LVM volume group, use the `vgremove` command:

```
$ sudo vgremove vgusb
Volume group "vgusb" successfully removed
```

There are many more ways to work with LVM. Refer to the LVM HOWTO for further information (<http://tldp.org/HOWTO/LVM-HOWTO/>).

Summary

Creating and managing filesystems in Linux are critical parts of Linux system administration. Ubuntu contains support for many standard Linux filesystem types (`ext2`, `ext3`, `ext4`, `xfs`, `reiserfs`, and others). It can also create and manage Windows filesystem types (VFAT, NTFS, and so on) as well as legacy and specialty Linux and Unix filesystem types (such as `minix`, `jfs`, and `xfs`).

You can partition hard disks with commands such as `fdisk` and `parted`. Tools for working with filesystems include those that create filesystems (`mkfs`), view and modify filesystem attributes (`tune2fs` and `dumpe2fs`), mount/unmount filesystems (`mount` and `umount`), and check for problems (`badblocks` and `fsck`). To see how much space has been used in filesystems, use the `df` and `du` commands.

Chapter 8

Backups and Removable Media

IN THIS CHAPTER

- Creating backup archives with `tar`
- Compressing backups with `gzip`, `bzip2`, and `lzop`
- Backing up over the network with `SSH`
- Doing network backups with `rsync`
- Making backup ISO images with `mkisofs`
- Burning backup images to CD or DVD with `cdrecord` and `growisofs`

Data backups in Linux were originally done by running commands to archive and compress the files to back up, and then writing that backup archive to tape. Choices for archive tools, compression techniques, and backup media have grown tremendously in recent years. Tape archiving has, for many, been replaced with techniques for backing up data over the network to other hard disks, or to CDs, DVDs, or other low-cost removable media.

This chapter details some useful tools for backing up and restoring your critical data. The first part of the chapter details how to use basic tools such as `tar`, `gzip`, and `rsync` for backups.

Backing Up Data to Compressed Archives

If you are coming from a Windows background, you may be used to tools such as WinZip and PKZIP, which both archive and compress groups of files in one application. Linux offers separate tools for gathering groups of files into a single archive (such as `tar`) and compressing that archive for efficient storage (`gzip`, `bzip2`, and `lzop`). However, you can also do the two steps together by using additional options to the `tar` command.

Creating Backup Archives with `tar`

The `tar` command, which stands for **tape archiver**, dates back to early UNIX systems. Although magnetic tape was the common medium that `tar` wrote to originally, today `tar` is most often used to create an archive file that can be distributed to a variety of media.

The fact that the `tar` command is rich in features is reflected in the dozens of options available with `tar`. The basic operations of `tar`, however, are used to create a backup archive (`-c`), extract files from an archive (`-x`), compare differences between archives (`-d`), and update files in an archive (`-u`). You can also append files to (`-r` or `-A`) or delete files from (`-d`) an existing archive, or list the contents of an archive (`-t`).

As part of the process of creating a tar archive, you can add options that compress the resulting archive. For example, add `-j` to compress the archive in bzip2 format or `-z` to compress in gzip format. By convention, regular tar files end in `.tar`, while compressed tar files end in `.tar.bz2` (compressed with bzip2) or `.tar.gz` (compressed with gzip). If you compress a file manually with `lzop` (see www.lzop.org), the compressed tar file should end in `.tar.lzo`.

In addition to being used for backups, tar files are popular ways to distribute source code and binaries from software projects. That's because you can expect every Linux and UNIX-like system to contain the tools you need to work with tar files.

Note One quirk of working with the `tar` command comes from the fact that `tar` was created before there were standards regarding how options are entered. Although you can prefix `tar` options with a dash, it isn't always necessary. So you might see a command that begins `tar xvf` with no dashes to indicate the options.

A classic example for using the `tar` command might combine old-style options and pipes for compressing the output; for example:

```
$ tar c *.txt | gzip -c > myfiles.tar.gz
```

 Make archive, zip it and output

The example just shown illustrates a two-step process you might find in documentation for old UNIX systems. The `tar` command creates (`c`) an archive from all `.txt` files in the current directory. The output is piped to the `gzip` command and output to stdout (`-c`), and then redirected to the `myfiles.tar.gz` file. Note that `tar` is one of the few commands that doesn't require that options be preceded by a dash (`-`).

New tar versions, on modern Linux systems, can **create the archive and compress the output** in one step:

```
$ tar czf myfiles.tar.gz *.txt
```

 Create gzipped tar file of .txt files

```
$ tar czvf myfiles.tar.gz *.txt
```

 Be more verbose creating archive
textfile1.txt

```
textfile2.txt
```

In the examples just shown, note that the new archive name (`myfiles.tar.gz`) must immediately follow the `f` option to `tar` (which indicates the name of the archive). Otherwise, the output from `tar` will be directed to `stdout` (in other words, your screen). The `z` option says to do `gzip` compression, and `v` produces verbose descriptions of processing.

When you want to **return the files to a filesystem** (unzipping and untarring), you can also do that as either a one-step or two-step process, using the `tar` command and optionally the `gunzip` command:

```
$ gunzip -c myfiles.tar.gz | tar x           Unzips and untars  
archive
```

Or try the following command line instead:

```
$ gunzip myfiles.tar.gz ; tar xf myfiles.tar  Unzips then untars  
archive
```

To do that same procedure in one step, you could use the following command:

```
$ tar xzvf myfiles.tar.gz  
textfile1.txt  
textfile2.txt
```

The result of the previous commands is that the archived `.txt` files are copied from the archive to the current directory. The `x` option extracts the files, `z` uncompresses (unzips) the files, `v` makes the output, and `f` indicates that the next option is the name of the archive file (`myfiles.tar.gz`).

Using Compression Tools

Compression is an important aspect of working with backup files. It takes less disk space on your backup medium (CD, DVD, tape, and so on) or server to store compressed files. It also takes less time to transfer the archives to the media or download the files over a network.

While compression can save a lot of storage space and transfer times, it can significantly increase your CPU usage. You can consider using hardware compression on a tape drive (see www.amanda.org/docs/faq.html#id346016).

In the examples shown in the previous section, `tar` calls the `gzip` command. But `tar` can work with many compression tools. Out of the box on Ubuntu, `tar` will work with `gzip` and `bzip2`. A third compression utility you can add to your toolbox is the `lzop` command, which can be used with `tar` in a different way. The order of these tools from fastest/least compression to slowest/most compression is: `lzop`, `gzip`, and `bzip2`.

If you are archiving and compressing large amounts of data, the time it takes to

compress your backups can be significant. So you should be aware that, in general, `bzip2` may take about 10 times longer than `lzop` and only give you twice the compression. However, with each compression command, you can choose different compression levels, to balance the need for more compression with the time that compression takes.

To use the `tar` command with **bzip2 compression**, use the `-j` option:

```
$ tar cjvf myfiles.tar.bz2 *.txt    Create archive, compress with
bzip2
```

You can also **uncompress (-j) a bzip2 compressed file** as you extract files (`-x`) using the `tar` command:

```
$ tar xjvf myfiles.tar.bz2    Extract files, uncompress bzip2
compression
```

The `lzop` compression utility is a bit less integrated into `tar`. Before you can use `lzop`, you might need to install the `lzop` package. To do **lzop compression**, you need the `--use-compress-program` option:

```
$ sudo apt-get install lzop
$ tar --use-compress-program=lzop -cf myfiles.tar.lzo *.txt
$ tar --use-compress-program=lzop -xf myfiles.tar.lzo
```

In the previous examples, the command line reverses the old syntax of `tar` with a switch before the command. For normal use and in other examples, I used the modern syntax of `tar` with no switch.

Note You may encounter `.rar` compressed files in the RAR format. This format seems to be popular in the world of peer-to-peer networks. RAR is a proprietary format so there is no widespread compressing tool. On Ubuntu, you can install the `unrar` and `rar` packages to get commands to work with RAR-format files.

Compressing with `gzip`

As noted, you can use **any of the compression commands alone** (as opposed to within the `tar` command line). Here are some examples of the `gzip` command to create and work with `gzip`-compressed files:

```
$ gzip myfile                gzips myfile and renames it myfile.gz
```

The following command provides the same result, with verbose output:

```
$ gzip -v myfile            gzips myfile with verbose output
myfile: 86.0% -- replaced with myfile.gz
$ gzip -tv myfile.gz        Tests integrity of gzip file
```

```

myfile.gz:      OK
$ gzip -lv myfile.gz      Get detailed info about gzip file
method  crc      date  time      compressed      uncompressed  ratio
uc_name
deflate 0f27d9e4 Jul 10 04:48      46785           334045          86.0%
myfile

```

Use any one of the following commands to **compress all files in a directory**:

```

$ gzip -rv mydir      Compress all files in a directory
mydir/file1: 39.1% -- replaced with mydir/file1.gz
mydir/file2: 39.5% -- replaced with mydir/file2.gz
$ gzip -1 myfile      Fastest compression time, least compression
$ gzip -9 myfile      Slowest compression time, most compression

```

Add a dash before a number from 1 to 9 to set the compression level. As illustrated, -1 is the fastest (least) and -9 is the slowest (most) compression. The default for `gzip` is level 6. The `lzop` command has fewer levels: 1, 3 (default), 7, 8, and 9. Compression levels for `bzip2` behave differently.

To **uncompress a gzipped file**, you can use the `gunzip` command. Use either of the following examples:

```

$ gunzip -v myfile.gz      Unzips myfile.gz and renames it myfile
myfile.gz:      86.0% -- replaced with myfile
$ gzip -dv myfile.gz      Same as previous command line

```

Although the examples just shown refer to zipping regular files, the same options can be used to compress tar archives.

Compressing with bzip2

The **bzip2** command is considered to provide the highest compression among the compression tools described in this chapter. Here are some examples of `bzip2`:

```

$ bzip2 myfile      Compresses file and renames it
myfile.bz2
$ bzip2 -v myfile      Same as previous command, but more
verbose
  myfile:  9.529:1, 0.840 bits/byte, 89.51% saved, 334045 in, 35056
out.
$ bunzip2 myfile.bz2      Uncompresses file and renames it myfile
$ bzip2 -d myfile.bz2      Same as previous command
$ bunzip2 -v myfile.bz2      Same as previous command, but more
verbose
  myfile.bz2: done

```

Compressing with lzop

The `lzop` command behaves differently from `gzip` and `bzip2`. The `lzop` command is

best in cases where compression speed is more important than the resulting compression ratio. When `lzop` compresses the contents of a file, it leaves the original file intact (unless you use `-U`) but creates a new file with an `.lzo` suffix. Use either of the following examples of the **`lzop` command to compress a file called `myfile`**:

```
$ lzop -v myfile           Leave myfile, create compressed
myfile.lzo
compressing myfile into myfile.lzo
$ lzop -U myfile           Remove myfile, create compressed
myfile.lzo
```

With `myfile.lzo` created, choose any of the following commands to test, list, or uncompress the file:

```
$ lzop -t myfile.lzo       Test the compressed file's integrity
$ lzop --info myfile.lzo   List internal header for each file
$ lzop -l myfile.lzo       List compression info for each file
method  compressed  uncompr.  ratio  uncompressed_name
LZO1X-1    59008      99468    59.3%  myfile
$ lzop --ls myfile.lzo     Show contents of compressed file as ls
-l
$ cat myfile | lzop > x.lzo Compress stdin and direct to stdout
$ lzop -dv myfile.lzo      Leave myfile.lzo, make uncompressed
myfile
```

Unlike `gzip` and `bzip2`, `lzop` has no related command for unzipping. Always just use the `-d` option to `lzop` to uncompress a file. If fed a list of file and directory names, the `lzop` command will compress all files and ignore directories. The original filename, permission modes, and timestamps are used on the compressed file as were used on the original file.

Listing, Joining, and Adding Files to tar Archives

So far, all you've done with `tar` is create and unpack archives. There are also options for listing the contents of archives, joining archives, adding files to an existing archive, and deleting files from an archive.

To **list an archive's contents**, use the `-t` option:

```
$ tar tvf myfiles.tar      List files from uncompressed
archive
-rw-r--r--  root/root      9584 2007-07-05 11:20:33 textfile1.txt
-rw-r--r--  root/root      9584 2007-07-09 10:23:44 textfile2.txt
$ tar tzvf myfiles.tgz     List files from gzip compressed
archive
```

If the archive were a tar archive compressed with `lzop` and named `myfile.tar.lzo`, you could **list that tar/lzop file's contents** as follows:

```
$ tar --use-compress-program=lzo -tf myfiles.tar.lzo    List lzo
archives
```

To **concatenate one tar file to another**, use the `-A` option. The following command results in the contents of `archive2.tar` being added to the `archive1.tar` archive:

```
$ tar -Af archive1.tar archive2.tar
```

Use the `-r` option to **add one or more files to an existing archive**. In the following example, `myfile` is added to the `archive.tar` archive file:

```
$ tar rvf archive.tar myfile    Add a file to a tar archive
```

You can use wildcards to **match multiple files to add** to your archive:

```
$ tar rvf archive.tar *.txt    Add multiple files to a tar archive
```

Deleting Files from tar Archives

If you have a tar archive file on your hard disk, you can delete files from that archive. Note that you can't use this technique to delete files from tar output on magnetic tape. Here is an example of **deleting files from a tar archive**:

```
$ tar --delete file1.txt -f myfile.tar    Delete file1.txt in
myfile.tar
```

Backing Up over Networks

After you have backed up your files and gathered them into a tar archive, what do you do with that archive? The primary reason for having a backup is in case something happens (such as a hard disk crash) where you need to restore files from that backup. Methods you can employ to keep those backups safe include:

- **Copying backups to removable media** such as tape, CD, or DVD (as described later in this chapter)
- **Copying them to another machine over a network**

Fast and reliable networks, inexpensive high-capacity hard disks, and the security that comes with moving your data off-site have all made network backups a popular practice. For an individual backing up personal data or a small office, combining a few simple commands may be all you need to create efficient and secure backups. This approach represents a direct application of the UNIX philosophy: joining together simple programs that do one thing to get a more complex job done.

Although just about any command that can copy files over a network can be used to move your backup data to a remote machine, some utilities are especially good for the

job. Using OpenSSH tools such as `ssh` and `scp`, you can set up secure password-less transfers of backup archives and encrypted transmissions of those archives.

Tools such as the `rsync` command can save resources by backing up only files (or parts of files) that have changed since the previous backup. With tools such as `unison`, you can back up files over a network from Windows as well as Linux systems.

The following sections describe some of these techniques for backing up your data to other machines over a network.

Note A similar tool that might interest you is the `rsnapshot` command. The `rsnapshot` command (www.rsnapshot.org) can work with `rsync` to make configurable hourly, daily, weekly, or monthly snapshots of a filesystem. It uses hard links to keep a snapshot of a filesystem, which it can then sync with changed files.

Install this tool with the following commands:

```
$ sudo apt-get install rsnapshot  
$ sudo apt-get install sshfs
```

Backing Up tar Archives over ssh

OpenSSH (www.openssh.org) provides tools to securely do remote login, remote execution, and remote file copy over network interfaces. By setting up two machines to share encryption keys, you can transfer files between those machines without entering passwords for each transmission. That fact lets you create scripts to back up your data from an SSH client to an SSH server without any manual intervention.

From a central Linux system, you can **gather backups from multiple client machines** using OpenSSH commands. The following example runs the `tar` command on a remote site (to archive and compress the files), pipes the tar stream to standard output, and uses the `ssh` command to catch the backup locally (over `ssh`) with `tar`:

```
$ mkdir mybackup ; cd mybackup  
$ ssh chris@server1 'tar cf - myfile*' | tar xvf -  
chris@server1's password: *****  
myfile1  
myfile2
```

In the example just shown, all files beginning with `myfile` are copied from the home directory of `chris` on `server1` and placed in the current directory. Note that the left side of the pipe creates the archive and the right side expands the files from the archive to the current directory. (Keep in mind that `ssh` will overwrite local files if they exist, which is why you created an empty directory in the example.)

To reverse the process and **copy files from the local system to the remote system**, you can run a local `tar` command first. This example, however, adds a `cd` command to put the files in the `/home/chris/myfolder` directory on the remote machine:

```
$ tar cf - myfile* | ssh chris@server1 \  
    'cd /home/chris/myfolder; tar xvf -'  
chris@server1's password: *****  
myfile1  
myfile2
```

In this next example, you're not going to untar the files on the receiving end, but instead **write the results to tgz files**:

```
$ ssh chris@server1 'tar czf - myfile*' | cat > myfiles.tgz  
$ tar cvzf - myfile* | ssh chris@server1 'cat > myfiles.tgz'
```

The first example takes all files beginning with `myfile` from the `chris` user's home directory on `server1`, tars and compresses those files, and directs those compressed files to the `myfiles.tgz` file on the local system. The second example does the reverse by taking all files beginning with `myfile` in the local directory and sending them to a `myfiles.tgz` file on the remote system.

The examples just shown are good for copying files over the network. In addition to providing compression, they also enable you to use any `tar` features you choose, such as incremental backup features.

Backing Up Files with `rsync`

A more feature-rich command for doing backups is `rsync`. What makes `rsync` so unique is the `rsync` algorithm, which compares the local and remote files one small block at a time using checksums, and only transfers the blocks that are different. This algorithm is so efficient that it has been reused in many backup products.

The `rsync` command can work either on top of a remote shell (`ssh`) or by running an `rsyncd` daemon on the server end. The following example uses `rsync` over `ssh` to **mirror a directory**:

```
$ rsync -avz --delete chris@server1:/home/chris/pics/ chrispics/
```

The command just shown is intended to mirror the remote directory structure (`/home/chris/pics/`) on the local system. The `-a` says to run in archive mode (recursively copying all files from the remote directory), the `-z` option compresses the files, and `-v` makes the output verbose. The `--delete` tells `rsync` to delete any files on the local system that no longer exist on the remote system.

For ongoing backups, you can have `rsync` do seven-day incremental backups. Here's an example:

```
# mkdir /var/backups
# rsync --delete --backup \
  --backup-dir=/var/backups/backup-`date +%A` \
  -avz chris@server1:/home/chris/Personal/ \
  /var/backups/current-backup/
```

When the command just shown runs, all the files from `/home/chris/Personal` on the remote system `server1` are copied to the local directory `/var/backups/current-backup`. All files modified today are copied to a directory named after today's day of the week, such as `/var/backups/backup-Monday`. Over a week, seven directories will be created that reflect changes over each of the past seven days.

Another trick for rotated backups is to **use hard links instead of multiple copies** of the files. This two-step process consists of rotating the files, and then running `rsync`:

```
# rm -rf /var/backups/backup-old/
# mv /var/backups/backup-current/ /var/backups/backup-old/
# rsync --delete --link-dest=/var/backups/backup-old -avz \
  chris@server1:/home/chris/Personal/ /var/backups/backup-current/
```

In the previous procedure, the existing `backup-current` directory replaces the `backup-old` directory, deleting the two-week-old full backup with last week's full backup. When the new full backup is run with `rsync` using the `--link-dest` option, if any of the files being backed up from the remote `Personal` directory on `server1` existed during the previous backup (now in `backup-old`), a hard link is created between the file in the `backup-current` directory and `backup-old` directory.

You can save a lot of space by having hard links between files in your `backup-old` and `backup-current` directory. For example, if you had a file named `file1.txt` in both directories, you could check that both were the same physical file by listing the files' inodes as follows:

```
$ ls -li /var/backups/backup*/file1.txt
260761 /var/backups/backup-current/file1.txt
260761 /var/backups/backup-old/file1.txt
```

Backing Up with unison

Although the `rsync` command is good to back up one machine to another, it assumes that the machine being backed up is the only one where the data is being modified. What if you have two machines that both modify the same file and you want to sync those files? Unison is a tool that will let you do that.

It's common for people to want to work with the same documents on their laptop and desktop systems. Those machines might even run different operating systems. Because `unison` is a cross-platform application, it can let you **sync files** that are on both Linux

and Windows systems. To use unison in Ubuntu, you must install the unison package (type the `sudo apt-get install unison` command). Both systems you are syncing must run the same version of unison.

With unison, you can define two roots representing the two paths to synchronize. Those roots can be local or remote over ssh. For example:

```
$ unison /home/chris ssh://chris@server1//home/cnegus
$ unison /home/chris /mnt/backups/chris-homedir
```

Unison contains both graphical and command-line tools for doing unison backups. It will try to run the graphical version by default. This may fail if you don't have a desktop running or if you're launching unison from within `screen`. To **force unison to run in command line mode**, add the `-ui text` option as follows:

```
$ unison /home/chris ssh://chris@server1//home/cnegus -ui text
Contacting server...
chris@server1's password:
Looking for changes
    Waiting for changes from server
Reconciling changes
local          server1
newfile ---->          memo.txt    [f] y
Propagating updates
...
```

The unison utility will then compare the two roots and for each change that occurred since last time, ask you what you want to do. In the preceding example, there's a new file called `memo.txt` on the local system. You are asked if you want to proceed with the update (in this case, copy `memo.txt` from the local machine to `server1`). Type `y` to do the updates.

If you trust unison, add `-auto` to make it **take default actions without prompting you**:

```
$ unison /home/chris ssh://chris@server1//home/cnegus -auto
```

For more information, see the man page for unison. In addition, you can view unison options using the `-help` option. You can also display and page through the unison manual using the `-doc all` option as shown here:

```
$ unison -help          See unison options
$ unison -doc all | less  Display unison manual
```

If you find yourself synchronizing two roots frequently, you can **create a profile**, which is a series of presets. In graphical mode, the default screen makes you create profiles. Profiles are stored in `.prf` text files in the `~/unison/` directory. They can be as simple as the following:

```
root = /home/chris
root = ssh://chris@server1//home/cnegus
```

If this is stored in a profile called `cn-home.prfl`, you can invoke it simply with the following command line:

```
$ unison cn-home
```

Backing Up to Removable Media

The capacity of CDs and DVDs, and the low costs of those media, has made them attractive options as computer backup media. Using tools that commonly come with Linux systems, you can gather files to back up into CD or DVD images and burn those images to the appropriate media.

Command line tools such as `mkisofs` (for creating CD images) and `cdrecord` (for burning images to CD or DVD) once provided the most popular interfaces for making backups to CD or DVD. Now there are many graphical front-ends to those tools you could also consider using. For example, GUI tools for mastering and burning CDs/DVDs include K3b (the KDE CD and DVD Kreator) and Nautilus (GNOME's file manager that offers a CD-burning feature). Other GUI tools for burning CDs include `gcombust`, `X-CD-Roast`, and `graveman`.

The commands for creating filesystem images to back up to CD or DVD, as well as to burn those images, are described in this section.

Creating Backup Images with `mkisofs`

Most data CDs and DVDs can be accessed on both Windows and Linux systems because they are created using the ISO9660 standard for formatting the information on those discs. Because most modern operating systems need to save more information about files and directories than the basic ISO9660 standard includes, extensions to that standard were added to contain that information.

Using the `mkisofs` command, you can back up the file and directory structure from any point in your Linux filesystem and produce an ISO9660 image. That image can include the following kinds of extensions:

- **System Use Sharing Protocol (SUSP)** extensions are records identified in the Rock Ridge Interchange Protocol. SUSP records can include UNIX-style attributes, such as ownership, long filenames, and special files (such as character devices and symbolic links).
- **Joliet** directory records store longer filenames in a form that makes them usable to Windows systems.

- **Hierarchical File System (HFS)** extensions allow the ISO image to appear as an HFS filesystem, which is the native filesystem for Macintosh computers. Likewise, Data and Resource forks can be added in different ways to be read by Macs.

When you set out to create your ISO image, consider where you will ultimately need to access the files you back up using `mkisofs` (Linux, Windows, or Mac). Once the image is created, it can be used in different ways, the most obvious of which is to burn the image to a CD or DVD.

Besides being useful in producing all or portions of a Linux filesystem to use on a portable medium, `mkisofs` is also useful for creating live CDs/DVDs. It does this by adding boot information to the image that can launch a Linux kernel or other operating system, bypassing the computer's hard drive.

Note Although you can still use the `mkisofs` command in Ubuntu, `mkisofs` is now a pointer to `genisoimage`. The `genisoimage` command was derived from `mkisofs`, which was part of the `cdrtools` package (see <http://cdrecord.berlios.de>). Development of `genisoimage` is part of the `cdrkit` project (www.cdrkit.org). Type `apt-get install genisoimage` to install that package.

Because most Linux users store their personal files in their home directories, a common way to use `mkisofs` to back up files is to back up everything under the `/home` directory. Here are some examples of using `mkisofs` to **create an ISO image from all files and directories under the `/home` directory**:

```
$ cd /tmp
$ sudo mkisofs -o home.iso /home           Create basic ISO9660
image
$ sudo mkisofs -o home2.iso -J -R /home     Add Joliet Rock Ridge
extns
$ sudo mkisofs -o home3.iso -J -R -hfs /home Also add HFS
extensions
```

With the last command, you will see a warning message like the following:

```
genisoimage: Warning: no Apple/Unix files will be decoded/mapped
```

In each of the three examples, all files and directories beneath the `/home` directory are added to the ISO image (`home.iso`). The first example has no extensions, so all filenames are converted to DOS-style naming (8.3 characters). The second example uses Joliet and Rock Ridge extensions, so filenames and permissions should appear as they did on the original Linux system when you open the ISO on a Linux or Windows system. The last example also makes the files on the image readable from a Mac filesystem.

Note You can also read Rock Ridge and Joliet extensions on Mac OS X.

You can have **multiple sources added to the image**. Here are some examples:

```
$ mkisofs -o home.iso -R -J music/ docs/ \ Multiple
directories/files
      chris.pdf /var/spool/mail
$ mkisofs -o home.iso -J -R \ Graft files on to the image
-graft-points Pictures/= /usr/share/pixmaps/ \
      /home/chris
```

The first example in the preceding code shows various files and directories being combined and placed on the root of the ISO image. The second example grafts the contents of the `/usr/share/pixmaps` directory into the `/home/chris/Pictures` directory. As a result, on the CD image the `/Pictures` directory will contain all content from the `/usr/share/pixmaps` directory.

Adding information into the header of the ISO image can help you identify the contents of that image later. This is especially useful if the image is being saved or distributed online, without a physical disc you can write on. Here are some examples:

```
$ mkisofs -o /tmp/home.iso -R -J \ Add header info to
ISO
      -p www.handsonhistory.com \
      -publisher "Swan Bay Folk Art Center" \
      -V "WebBackup" \
      -A "mkisofs" \
      -volset "1 of 4 backups, July 30, 2013" \
      /home/chris
```

In the preceding example, `-p` indicates the preparer ID, which could include a phone number, mailing address, or website for contacting the preparer of the ISO image. With the option `-publisher`, you can indicate a 128-character description of the preparer (possibly the company or organization name).

The `-v` indicates the volume ID. Volume ID is important because in many Linux systems this volume ID is used to mount the CD when it is inserted. For example, in the command line just shown, the CD would be mounted on `/media/WebBackup` in Ubuntu and other Linux systems. The `-A` option can be used to indicate the application used to create the ISO image. The `-volset` option can contain a string of information about a set of ISO images.

When you have created your ISO image, and before you burn it to disc, you **can check the image** and make sure you can access the files it contains. Here are ways to check it out:

```
$ volname home.iso Display volume name
```

WebBackup

```
$ isoinfo -d -i home.iso           Display header information
CD-ROM is in ISO 9660 format
System id: LINUX
Volume id: WebBackup
Volume set id: All Website material on November 2, 2013
Publisher id: Swan Bay Folk Art Center
Data preparer id: www.handsonhistory.com
Application id: mkisofs
Copyright File id:
Abstract File id:
Bibliographic File id:
Volume set size is: 1
Volume set sequence number is: 1
Logical block size is: 2048
Volume size is: 23805
Joliet with UCS level 3 found
Rock Ridge signatures version 1 found
```

You can see a lot of the information entered on the `mkisofs` command line when the image was created. If this had been an image that was going to be published, you might also have indicated the locations on the CD of a copyright file (`-copyright`), abstract file (`-abstract`), and bibliographic file (`-biblio`). Provided that the header is okay, you can next try **accessing files on the ISO image by mounting it**:

```
$ sudo mkdir /mnt/myimage           Create a mount point
$ sudo mount -o loop home.iso /mnt/myimage Mount the ISO in
loopback
$ ls -l /mnt/myimage                 Check the ISO
contents
$ sudo umount /mnt/myimage           Unmount image when
done
```

Note The ISO image is mounted read-only. If you want to work with the contents, you need to copy the files and directories you want to another directory.

Besides checking that you can access the files and directories on the ISO, make sure that the date/time stamps, ownership, and permissions are set as you would like. That information might be useful if you need to restore the information at a later date.

Burning Backup Images with `cdrecord`

The `cdrecord` command is the most popular Linux command line tool for burning CD and DVD images. After you have created an ISO image (as described earlier) or obtained one otherwise (such as downloading an install CD or live CD from the

Internet), `cdrecord` makes it easy to put that image on a disc.

Note In Ubuntu, `cdrecord` has been replaced with the `wodim` command. The `wodim` command was created from the `cdrecord` code base and still supports most of the same options. If you run `cdrecord`, you will actually be running `wodim` in this Ubuntu release. If you have problems with that utility, contact the CDRkit project (<http://cdrkit.org>). Type `apt-get install wodim` to install the package, if it isn't already installed.

There is no difference in making a CD or DVD ISO image, aside from the fact that a DVD image can obviously be bigger than a CD image. Check the media you have for their capacities. A CD can typically hold 650MB, 700MB, or 800MB, whereas mini CDs can hold 50MB, 180MB, 185MB, or 193MB. Single-layer DVDs hold 4.7GB, while double-layer DVDs can hold 8.4GB.

Note Keep in mind, however, that CD/DVD manufacturers list their capacities based on 1000KB per 1MB, instead of 1024KB. Type `du --si home.iso` to list the size of your ISO, instead of `du -sh` as you would normally, to check if your ISO will fit on the media you have.

Before you begin burning your image to CD or DVD, **check that your drive supports CD/DVD burning** and determine the address of the drive. Use the `--scanbus` option to `cdrecord` to do that:

```
$ cdrecord --scanbus                               Shows a drive that cannot do burning
scsibus0:
    0,0,0  0) 'SAMSUNG ' 'DVD-ROM SD-616E ' 'F503' Removable CD-
ROM
    0,0,0  1) *
    0,0,0  2) *
```

```
    ...
$ cdrecord --scanbus                               Shows a drive that can burn CDs or
DVDs
scsibus0:
    0,0,0  0) 'LITE-ON ' 'DVDRW SOHW-1633S' 'BS0C' Removable CD-
ROM
    0,0,0  1) *
    0,0,0  2) *
    ...
```

In the two examples shown, the first indicates a CD/DVD drive that only supports reading and cannot burn CDs (DVD-ROM and CD-ROM). The second example shows a drive that can burn CDs or DVDs (DVDRW). Insert the medium you want to record on. Assuming your drive can burn the media you have, here are some simple `cdrecord`

commands for burning a CD or DVD images:

```
$ cdrecord -dummy home.iso           Test burn without actually burning
$ cdrecord -v home.iso                 Burn CD (default settings) in
verbose
$ cdrecord -v speed=24 home.iso       Set specific speed
$ cdrecord -pad home.iso              Can't read track, add 15 zeroed
sectors
$ cdrecord -eject home.iso            Eject CD/DVD when burn is done
$ cdrecord /dev/cdrw home.iso         Identify drive by dev name (may
differ)
$ cdrecord dev=0,2,0 home.iso         Identify drive by SCSI name
```

The `cdrecord` command can also **burn multi-session CDs/DVDs**. Here is an example:

```
$ cdrecord -multi home.iso           Start a multi-burn session
$ cdrecord -msinfo                   Check the session offset for next
burn
Using /dev/cdrom of unknown capabilities
0,93041
$ mkisofs -J -R -o new.iso \          Create a second ISO to burn
-C 0,93041 /home/chris/more         Indicate start point and new data for
ISO
$ cdrecord new.iso                   Burn new data to existing CD
```

You can use multiple `-multi` burns until the CD is filled up. For the final burn, don't use `-multi` so that the CD will be closed.

Making and Burning DVDs with growisofs

Using the `growisofs` command, you can **combine the two steps of gathering files into an ISO image (mkisofs) and burning that image to DVD (cdrecord)**. Besides saving a step, the `growisofs` command also offers the advantage of keeping a session open by default until you close it, so you don't need to do anything special for multi-burn sessions.

Here is an example of some `growisofs` commands for a **multi-burn session**:

```
$ growisofs -Z /dev/dvd -R -J /home/chris   Master and burn to
DVD
$ growisofs -Z /dev/dvd -R -J /home/chris   Add to burn
$ growisofs -M /dev/dvd=/dev/zero          Close burn
```

If you want to add options when creating the ISO image, you can simply add `mkisofs` options to the command line. (For example, see how the `-R` and `-J` options are added in the preceding examples.)

If you want to **burn a DVD image using growisofs**, you can use the `-dvd-compat`

option. Here's an example:

```
$ growisofs -dvd-compat -Z /dev/dvd=image.iso Burn an ISO image to  
DVD
```

The `-dvd-compat` option can improve compatibility with different DVD drives over some multi-session DVD burning procedures.

Summary

Linux and its predecessor UNIX systems handled data backups by combining commands that each handled a discrete set of features. Backups of your critical data can still be done in this way. In fact, many of the tools you can use will perform more securely and efficiently than ever before.

The tape archiver utility (`tar` command) has expanded well beyond its original job of making magnetic tape backups of data files. Because nearly every Linux and UNIX system includes `tar`, it has become a standard utility for packaging software and backing up data to compressed archives. Those archives can then be transported and stored in a variety of ways.

To move backed up data to other machines over a network, you can use remote execution features of OpenSSH tools (such as `ssh`). You can also use an excellent utility called `rsync`. With `rsync`, you can save resources by only backing up files (or parts of files) that have changed.

Inexpensive CDs and DVDs have made those media popular for doing personal and small-office backups. The `mkisofs` command can create filesystems of backed up data in ISO9660 format that can be restored on a variety of systems (Linux, Windows, or Mac). Once the `mkisofs` command has created an ISO image, the image can be burned to CD or DVD using the `cdrecord` or `growisofs` command.

Chapter 9

Checking and Managing Running Processes

IN THIS CHAPTER

- Viewing active processes with `ps` and `top`
- Searching for processes with `pgrep`
- Adjusting CPU priority with `nice` and `renice`
- Moving processes to the background or foreground
- Killing and signaling processes with `kill` and `killall`
- Using `at` and `batch` to run commands
- Scheduling commands to run repeatedly with `cron`

When an executable program starts up, it runs as a process that is under the management of your Linux system's process table. Linux provides all the tools you need to view and change the processes running on your system.

The `ps` and `top` commands are great for viewing information on your running processes. There are literally dozens of options to `ps` and `top` to help you view process information exactly the way you want to. The `pgrep` command can further help find the process you want.

There are commands such as `nice` and `renice` for raising and lowering processor priority for a process. You can move processes to run in the background (`bg` command) or back to the foreground (`fg` command). On rare occasions, you can use the `chrt` command to run processes in realtime.

Sending signals to a process is a way of changing its behavior or killing it altogether. Using the `kill` and `killall` commands, you can send signals to processes by PID or name, respectively. You can also send other signals to processes to do such things as reread configuration files or continue with a stopped process.

To run commands at scheduled times or so they are not tied to your shell session, you can use the `at` and `batch` commands. To run commands repetitively at set times, there are the `cron` and `anacron` facilities. Or you can drop scripts (or symbolic links to scripts) into `/etc/cron.hourly` (or `cron.daily`, `cron.weekly`, or `cron.monthly`).

Listing Active Processes

To see which processes are currently running on a system, most people use the `ps` and `top` commands. The `ps` command gives you a snapshot (in a simple list) of processes running at the moment. The `top` command offers a screen-oriented, constantly updated listing of running commands, sorted as you choose (by CPU use, memory use, UID, and so on).

Viewing Active Processes with `ps`

Every Linux system (as well as every system derived from UNIX, such as BSD, Mac OS X, and others) includes the `ps` command. Over the years, however, many slightly different versions of `ps` have appeared, offering slightly different options. Because `ps` dates back to the first UNIX systems, it also supports nonstandard ways of entering some options (for example, allowing you to drop the dash before an option in some cases).

The different uses of `ps` shown in this chapter will work on Ubuntu and most other Linux systems. Here are some examples you can run to **show processes running for the current user**. ([Table 9-1](#) contains column descriptions of `ps` output.)

```
$ ps                                List processes of current user at current shell
```

PID	TTY	TIME	CMD
2552	pts/0	00:00:00	bash
3438	pts/0	00:00:00	ps

```
$ ps -u chris                        Show chris's running processes (simple output)
```

PID	TTY	TIME	COMMAND
2678	tty1	0:00	startx
2689	tty1	0:00	xinit
2710	tty1	0:06	gnome-session

...

```
$ ps -u chris u                      Show all chris's running processes (with CPU/MEM)
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
chris	2678	0.0	0.0	4328	852	tty1	S+	Aug14	0:00	/bin/sh
chris	2689	0.0	0.1	2408	488	tty1	S+	Aug14	0:00	xinit
chris	2710	0.0	1.1	22016	5496	tty1	S	Aug14	0:06	gnome-session

...

```
$ ps -fu chris                      Show all chris's running processes (with PPID)
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
-----	-----	------	---	-------	-----	------	-----

```

chris  2678  2645  0 Aug14 tty1  00:00:00 /bin/sh
/usr/X11R6/bin/startx
chris  2689  2678  0 Aug14 tty1  00:00:00 xinit
/etc/X11/xinit/xinitrc
chris  2710  2689  0 Aug14 tty1  00:00:09 /usr/bin/gnome-session
...
$ ps -fu chris          Show chris's running processes (with SZ and
PSR)
UID      PID  PPID  C    SZ    RSS  PSR  STIME  TTY      TIME  CMD
chris  2678  2645  0   1082   852    0 Aug14  tty1    00:00:00 /bin/sh
startx
chris  2689  2678  0    602   488    0 Aug14  tty1    00:00:00 xinit
chris  2710  2689  0   5504  5440    0 Aug14  tty1    00:00:09 gnome-
session
...

```

These examples illustrate some of the processes from a user running a GNOME desktop session. The first example in the preceding code shows `ps` alone being run from a Terminal window, so you see only the processes for the current shell running in that window. Other examples let you display different information for each process (see later examples for ways of producing custom output).

Here are `ps` examples showing output for **every process currently running on the system**:

```

$ ps -e                  Show every running process
  PID TTY          TIME CMD
    1 ?            00:00:01 init
    2 ?            00:00:00 migration/0
    3 ?            00:00:00 ksoftirqd/0
...
$ ps -el                 Show every running process, long listing
F S      UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S      0         1     0  0  75   0 -   534 -        ?    00:00:01 init
1 S      0         2     1  0 -40   - -     0 -        ?    00:00:00
migration/0
1 S      0         3     1  0  94  19 -     0 -        ?    00:00:00
ksoftirqd/0
...
$ ps -ef                 Show every running process, full-format
listing
UID      PID  PPID  C  STIME  TTY          TIME CMD
root         1     0  0 Aug05 ?           00:00:01 init [5]
root         2     1  0 Aug05 ?           00:00:00 [migration/0]
root         3     1  0 Aug05 ?           00:00:00 [ksoftirqd/0]
...
$ ps -eF                 Show every running process, extra full-format
list
UID      PID  PPID  C    SZ    RSS  PSR  STIME  TTY      TIME  CMD

```

```

root      1      0  0    534    556    0 Aug05 ?    00:00:01 init [5]
root      2      1  0      0      0    0 Aug05 ?    00:00:00
[migration/0]
root      3      1  0      0      0    0 Aug05 ?    00:00:00
[ksoftirqd/0]

```

...

\$ ps ax Show every running process, short BSD style

```

  PID TTY      STAT   TIME COMMAND
    1 ?        Ss      0:01 init [5]
    2 ?        S        0:00 [migration/0]
    3 ?        SN       0:00 [ksoftirqd/0]

```

...

\$ ps aux Show every running process, long BSD style

```

USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0   0.0   2136    556 ?        Ss     Aug05    0:01 init [5]
root         2   0.0   0.0      0      0 ?        S       Aug05    0:00
[migration/0]
root         3   0.0   0.0      0      0 ?        SN     Aug05    0:00
[ksoftirqd/0]

```

...

\$ ps auwx Show every running process, BSD style, wide format

\$ ps auwxw Show every running process, BSD style, unlimited width

Some processes start up other processes. For example, a web server (httpd daemon) will spin off multiple httpd daemons to wait for requests to your web server. You can **view the hierarchy of processes (in a tree view)** using various options with `ps`:

\$ ps -ejH Show process hierarchy with process/session IDs

```

  PID  PGID    SID TTY      TIME CMD
16267 16267 16267 ?        00:00:00 sshd
16462 16267 16267 ?        00:00:00 sshd
16463 16463 16463 pts/2    00:00:00 bash
16563 16563 16463 pts/2    00:00:00 firefox
16606 16606 16463 pts/2    00:00:00 sudo
16607 16606 16463 pts/2    00:00:00 su
16615 16615 16463 pts/2    00:00:00 bash
16673 16673 16463 pts/2    00:00:00 ps

```

\$ ps axjf Show process hierarchy in BSD-style output

```

  PPID  PID  PGID  SID TTY  TPGID STAT  UID  TIME  COMMAND
    1    957   957   957 ?    -1 Ss      0 0:00 /usr/sbin/sshd -D
    957 16267 16267 16267 ?    -1 Ss      0 0:00 \_ sshd: chris [priv]
16267 16462 16267 16267 ?    -1 S    1000 0:00 \_ sshd: chris@pts/
16462 16463 16463 16463 pts/2 16688 Ss 1000 0:00 \_ -bash
16463 16563 16563 16463 pts/2 16688 Sl 1000 0:02 \_ /usr/lib
16463 16606 16606 16463 pts/2 16688 S      0 0:00 \_ sudo su

```

\$ ps -ef --forest Show process hierarchy in forest format

```

UID      PID PPID  C STIME TTY      TIME CMD
root      957   1  0 Feb18 ?        00:00:00 /usr/sbin/sshd -D

```

```

root  16267   957   0 19:03 ?      00:00:00  \_ sshd: chris [priv]
chris 16462 16267  0 19:03 ?      00:00:00  \_ \_ sshd: chris@pts/2
chris 16463 16462  0 19:03 pts/2  00:00:00  \_ \_ -bash
chris 16563 16463  0 19:03 pts/2  00:00:02  \_ \_
/usr/lib/firefox/
root  16606 16463  0 19:03 pts/2  00:00:00  \_ sudo su -
root  16607 16606  0 19:03 pts/2  00:00:00  \_ \_ su -
$ pstree          Show processes alphabetically in tree format
|-rsyslogd---3*[{rsyslogd}]
  |rtkit-daemon---2*[{rtkit-daemon}]
  |sound-juicer---3*[{sound-juicer}]
  |sshd---sshd---sshd---bash-T-firefox---25*[{firefox}]
  |                             L-sudo---su---bash---pstree
...

```

The “tree” examples just shown illustrate different ways of displaying the hierarchy of processes. The output was snipped to compare several of the same processes with different output. Note that the PPID (Parent Process ID) is the ID of the process that started each child process shown. The `sshd` processes show a running Secure Shell Daemon with a user logging in over the network, resulting in a `bash` shell starting. From that shell, the user starts a `firefox` command in the background. Then the user opens a shell as super user (`sudo su`). The last example shows the `ps` command, which is specifically used for displaying tree views of processes.

If you prefer personalized views of `ps` output, you can use the `-o` option to select exactly which columns of data to display with `ps`. You can then use the `--sort` option to sort the output by any of those data. [Table 9-1](#) shows available column output and the options to add to `-o` to have each column print with `ps`.

Table 9-1: Selecting and Viewing `ps` Column Output

Option	Column Head	Description
%cpu	%CPU	CPU utilization of process’s lifetime in 00.0 format
%mem	%MEM	Percentage of system’s physical memory use (resident set size)
args	COMMAND	Command with all arguments
bsdstart	START	Start time of command started: “HH:MM” or “MonthDay” (for example, Jan 03)
bsdtime	TIME	Total (user and system) CPU time
comm	COMMAND	Command name only (no arguments shown)
cp	CP	CPU utilization in tenth-of-a-percentage
cputime	TIME	Total CPU time in [DD-]HH:MM:SS format
egid	EGID	Effective group ID of the process (as integer)
egroup	EGROUP	Effective group ID of the process (as name)
etime	ELAPSED	Time since process was started, in [[DD-]HH:]MM:SS format

euid	EUID	Effective user ID of the process (as integer)
euser	EUSER	Effective user ID of the process (as name)
fgid	FGID	Filesystem access group ID (as number)
fgroup	FGROUP	Filesystem access group ID (as name)
fname	COMMAND	First eight characters of command name
fuid	FUID	Filesystem access user ID (as number)
fuser	FUSER	Filesystem access user ID (as name)
lstart	STARTED	Date and time the command started
nice	NI	Nice value, from 19 (nicest) to -20 (CPU hog)
pgid	PGID	Process group ID of process
pid	PID	Process ID number of process
ppid	PPID	Parent process ID of process
psr	PSR	Processor process is assigned to (first CPU is 0)
rgid	RGID	Real group ID (as number)
rgroup	RGROUP	Real group (as name)
rss	RSS	Non-swapped physical memory (resident set size) in KB
rtprio	RTPRIO	Realtime priority
ruid	RUID	Real user ID (as number)
ruser	RUSER	Real user (as name)
s	S	One-character state display (D : sleep, no interrupt; R : running; S : sleep, can interrupt; T : stopped; W : paging; X : dead; Z : zombie)
sess	SESS	Session ID of session leader
sgi_p	P	Processor that process is currently running on
size	SZ	Rough amount of swap space needed if process were to swap out
start	STARTED	Time command started: HH:MM:SS or MonthDay
start_time	START	Time command started: HH:MM or MonthDay
stat	STAT	Multi-character state: One-character “s” state plus other state characters (<: High priority; N: Low priority; L: Has pages locked in memory; s: Is session leader; l: Multi-threaded; +: in foreground process group)
sz	SZ	Size of process’s core image (physical pages)
tname	TTY	Controlling tty (terminal)
user	USER	Effective user ID of process (as name)
vsiz	VSZ	Process’s virtual memory (1024-byte units)

Note that some values that are meant to print usernames may still print numbers (UIDs) instead, if the name is too long to fit in the given space.

Using a comma-separated list of column options, you can produce your custom output. Here are some examples of **custom views of running processes**:

```
$ ps -eo ppid,user,%mem,size,vsize,comm --sort=-size mem          Sort by
mem
  PPID USER      %MEM    SIZE      VSZ COMMAND
    1 root        0.0  1004292  1043060 console-kit-dae
 6901 chris       2.3   713248  1125172 compiz
$ ps -eo ppid,user,bsdstart,bsdtime,%cpu,args --sort=-%cpu        Sort by
CPU
  PPID USER      START  TIME %CPU COMMAND
    6901 chris Feb 23 11:23  0.1 compiz
 16463 chris  19:03  0:03  0.0 /usr/lib/firefox/firefox
   1101 root   Feb 18 14:31  0.0 /usr/bin/X :0 -auth ...
$ ps -eo ppid,user,nice,cputime,args --sort=-nice                Sort by low
priority
  PPID USER      NI      TIME COMMAND
    2  root       19 00:00:00 [khugepaged]
    1 chris      10 00:00:31 /usr/bin/python /usr/bin/update-manager
    2  root       5 00:00:00 [ksmd]
$ ps -eo ppid,user,stat,tname,sess,cputime,args --sort=user      By
user
  PPID USER      STAT TTY  SESS      TIME COMMAND
    1 avahi       S    ?    851 00:00:09 avahi-daemon: running
 1640 chris      Ssl  ?    6901 00:00:00 gnome-session --
session=ubuntu
    1 daemon     Ss   ?    1088 00:00:00 atd
   846 lp        S    ?    846 00:00:00 /usr/lib/cups/notifier/dbus
    0 root       Ss   ?    1 00:00:00 /sbin/init
```

Here are a few other examples of the **ps** command:

```
$ ps -C httpd          Display running httpd
processes
  PID TTY          TIME CMD
 1493 ?           00:00:00 httpd
 1495 ?           00:00:00 httpd
```

Note that you need to install an HTTP server, such as Apache, to run an httpd process.

```
$ ps -p 16563 -o pid,ppid,bsdtime,args          Display info for PID
16565
  PID  PPID    TIME COMMAND
16563 16463    0:04 /usr/lib/firefox/firefox
$ ps -U chris,avahi -o pid,ruser,TTY,stat,args  See info for 2 users
  PID RUSER    TT      STAT COMMAND
   852 avahi     ?        S    avahi-daemon: running [ubuntutb.local]
   853 avahi     ?        S    avahi-daemon: chroot helper
 6890 chris    ?        Sl   /usr/bin/gnome-keyring-daemon
 6901 chris    ?        Ssl  gnome-session --session=ubuntu
```

Watching Active Processes with top

If you want to **see the processes running on your system on an ongoing basis**, you can use the `top` command. The `top` command runs a screen-oriented view of your running processes that is updated continuously. If you start the `top` command with no options, it displays your system's uptime, tasks, CPU usage, and memory usage, followed by a list of your running processes, sorted by CPU usage. Here's an example:

```
$ top
top - 2:37:18 up 2 days, 14:00, 3 users, load average: 0.00, 0.01, 0.05
Tasks: 178 total, 1 running, 177 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.2%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.2%si, 0.0%st
Mem: 4014504k total, 2208972k used, 1805532k free, 161628k buffers
Swap: 4157436k total, 0k used, 4157436k free, 1388516k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  6967 chris    20   0 1213m  93m  29m S   0   2.4   11:24.95 compiz
 16748 root     20   0 17340 1332  932 R   0   0.0    0:00.09 top
```

Here are examples of other options you can use to **start `top` to continuously display running processes**:

```
$ top -d 5           Change update delay to 5 seconds (from
default 3)
$ top -u chris       Only see processes of effective user name
chris
$ top -p 190,2690    Only display processes 190 and 2690
$ top -n 10          Refresh the screen 10 times before quitting
$ top -b             Run in batch mode
```

The last example (`top -b`) formats the output of `top` in a way that is suitable for output to a file, as opposed to redrawing the same screen for interactive viewing. This can be used to create a log of processes, for example when hunting down that runaway process that eats up all your resources in the middle of the night. Here's how to **run `top` and log the output 50 times**:

```
$ top -b -n 50 > myprocesslog
```

When `top` is running, you can press keys to update and sort the process list in different ways:

- **Press Space or Enter**—Immediately updates the process list
- **Press Shift+n**—sorts by PID
- **Press Shift+p**—sorts by CPU usage
- **Press Shift+m**—sorts by memory usage
- **Press Shift+t**—sorts by CPU time consumed
- **Press <**—sorts the column to left characters
- **Press >**—sorts the column to right characters

- **Press f plus the letter of the column**—sorts by when the list of columns appears
- There are several ways to change the behavior of `top` as it's running:
- **Press d and type a number**—delays between refreshes in seconds represented by the number entered.
 - **Press u and a username**—displays only processes for the selected user.
 - **Type n plus the number you want to see**—displays only a select number of processes.
 - **Press =**—returns to the original `top` display. You can do this at any point.

You can act on any of the running processes in different ways:

- **Type k followed plus the PID**—signals (kills) a running process.
- **Type 9 (after the k)**—ends the process.
- **Type a different signal number (after k)**—sends that signal to the process.
- **Type n**—gives a process higher or lower run priority. You type `n` and then add a negative number (to increase priority) or a positive number (to reduce priority).

If you want to **find more information about how to use** `top`, type `?` during a `top` session. The man page also has a lot of information about how to use `top`:

```
$ man top           View the top man page
```

When you are done using `top`, type `q` to exit.

Finding and Controlling Processes

Changing a running process first means finding the process you want to change, and then modifying the processing priority or sending the process a signal to change its behavior. If you are looking for a particular process, you might find it tough to locate it in a large list of processes output by `ps` or `top`.

The `pgrep` command offers ways of searching through your active processes for the ones you are looking for. The `renice` command lets you change the processing priority of running processes. The `kill`, `pkill`, and `killall` commands let you send signals to running processes (including signals to end those processes).

Using pgrep to Find Processes

In its most basic form, you can use `pgrep` to search for a command name (or part of one) and produce the process ID of any process that includes that name. For example:

```
$ pgrep init           Show PID for any process including 'init'
string
1
```

2689

Because you know there is only one `init` command running, you next use the `-l` option to see each process's command name (to learn why two processes showed up):

```
$ pgrep -l init          Show PID and name for processes including init
1 init
2689 xinit
```

You can also search for processes that are associated with a particular user:

```
$ pgrep -lu chris        List all processes owned by user chris
16462 sshd
16463 bash
16563 firefox
```

Probably the most useful way to use `pgrep` is to have it **find the process IDs of the running processes and pipe those PIDs to another command** to produce the output. Here are some examples (look for other commands if `sshd` or `firefox` aren't running):

```
$ ps -p `pgrep sshd`      Search for sshd and run ps (short)
  PID TTY          STAT       TIME COMMAND
   957 ?            Ss          0:00 /usr/sbin/sshd -D
16267 ?            Ss          0:00 sshd: chris [priv]
16462 ?            R           0:00 sshd: chris@pts/2
$ ps -fp $(pgrep firefox) Search for firefox and run ps
(full)
UID      PID  PPID  C  STIME TTY      TIME    CMD
chris 16563 16463  0 19:03 pts/2    00:00:05 /usr/lib/firefox/firefox
$ sudo renice -5 $(pgrep firefox) Search for firefox, improve
priority
16563 (process ID) old priority 0, new priority -5
```

Any command that can take a process ID as input can be combined with `pgrep` in these ways. As the previous example of `pgrep` illustrates, you can use commands such as `renice` to change how a process behaves while it is running.

Using `fuser` to Find Processes

Another way to locate a particular process is by what the process is accessing. The `fuser` command can be used to find which processes have a file or a socket open at the moment. After the processes are found, `fuser` can be used to send signals to those processes.

The `fuser` command is most useful for finding out if files are being held open by processes on mounted filesystems (such as local hard disks or Samba shares). Finding those processes allows you to close them properly (or just kill them if you must) so the filesystem can be unmounted cleanly. (If the `fuser` command is not found, install the

psmisc package.)

Here are some examples of the `fuser` command for **listing processes that have files open on a selected filesystem**:

```
$ sudo fuser -mauv /boot    Verbose output of processes with /boot open
```

	USER	PID	ACCESS	COMMAND
/boot:	root	kernel	mount	(root)/boot
	root	16615	..c..	(root)bash
	root	17289	..c..	(root)vi

The example just shown displays the process ID for running processes associated with `/boot`. They may have a file open, a shell open, or be a child process of a shell with the current directory in `/boot`. Specifically in this example, there is a bash shell open in the `/boot` filesystem and one `vi` command with files open in `/boot`. The `-a` shows all processes, `-u` indicates which user owns each process, and `-v` produces verbose output.

Here are other examples using `fuser` to **show processes with files open**:

```
$ fuser -m /boot            Show all PIDs for processes opening /boot
/boot:          16615c 17289c 17312c
$ sudo fuser -um /boot      Show PIDs/user for this shell open in /boot
/boot:          16615c(root) 17289c(root) 17312c(root)
```

After you know which processes have files open, you can close those processes manually or kill them. Close processes manually if at all possible because simply killing processes can leave files in an unclean state! Here are examples of using `fuser` to **kill or send other signals to all processes with files open to a filesystem**:

```
$ sudo fuser -k /boot      Kill processes with /boot files open
$ fuser -l                List supported signals
HUP INT QUIT ILL TRAP ABRT IOT BUS FPE KILL USR1 SEGV USR2 PIPE
ALRM
TERM
    STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF
WINCH IO
PWR SYS UNUSED
$ sudo fuser -k -HUP /boot Send HUP signal to processes with /boot open
```

Changing Running Processes

Even after a process is running, you can change its behavior in different ways. With the `renice` command, shown earlier, you can adjust a running process's priority in your

system's scheduler. With the `nice` command, you can determine the default priority and also set a higher or lower priority at the time you launch a process.

Another way you can change how a running process behaves is to send a signal to that process. The `kill` and `killall` commands can be used to send signals to running processes. Likewise, the `pkill` command can send a signal to a process.

Adjusting Processor Priority with `nice`

Every running process has a nice value that can be used to tell the Linux process scheduler what priority should be given to that process. Positive values of niceness actually give your process a lower priority. The concept came about during the days of large, multi-user UNIX systems where you could be “nice” by running a non-urgent process at a lower priority so other users had a shot at the CPU.

Niceness doesn't enforce scheduling priority, but is merely a suggestion to the scheduler. To **see your current default nice value**, you can type the `nice` command with no options:

```
$ nice                                Run nice to determine current niceness
0
```

The default nice value is 0. You can use the `nice` command to run a process at a higher or lower priority than the default. The priority number can range from -20 (most favorable scheduling priority) to 19 (least favorable scheduling priority). Although the root user can raise or lower any user's `nice` value, a regular user can only lower the priorities of a process (setting a higher nice value).

Here are a few examples of starting a command with `nice` to **change a command's nice value**:

```
$ nice -n 12 nroff -man a.roff | less  Format man pages at low
priority
$ sudo nice -n -10 gimp                Launch gimp at higher
priority
```

When a process is already running, you can **change the process's nice value using the `renice` command**. Here are some examples of the `renice` command:

```
$ renice +2 -u chris                  Renice chris's processes
+2
$ renice +5 4737                      Renice PID 4737 by +5
$ sudo renice -3 `pgrep -u chris spamd` Renice chris's spamd
processes
9688: old priority -1, new priority -3
20279: old priority -1, new priority -3
2: old priority -1, new priority -3
```

The backticks are used in the previous command line to indicate that the output of the

`pgrep` command (presumably PIDs of `spamd` daemons run by chris) is fed to the `renice` command.

The niceness settings for your processes are displayed by default when you run `top`. You can also see niceness settings using `-o nice` when you produce custom output from the `ps` command.

Running Processes in the Background and Foreground

When you run a process from a shell, it is run in the foreground by default. That means that you can't type another command until the first one is done. By adding an ampersand (&) to the end of a command line, you can run that command line in the background. Using the `fg`, `bg`, and `jobs` commands, along with various control codes, you can move commands between background and foreground.

In the following sequence of commands, you can start the GIMP image program from a Terminal window. After that is a series of control keys and commands to **stop and start the process and move it between foreground and background**:

<code>\$ gimp</code>	Run gimp in the foreground
<code><Ctrl+z></code>	Stop process and place in background
<code>[1]+ Stopped gimp</code>	
<code>\$ bg 1</code>	Start process running again in
<code>background</code>	
<code>\$ fg 1</code>	Continue running process in
<code>foreground</code>	
<code>gimp</code>	
<code><Ctrl+c></code>	Kill process

Note that processes placed in the background are given a job ID number (in this case, 1). By placing a percentage sign in front of the number (for example, `%1`) you can identify a particular background process to the `bg` and `fg` commands, or you can simply type the number with the command (as in `fg 1`). With one or more background jobs running at the current shell, you can **use the `jobs` command to manage your background jobs**:

<code>\$ jobs</code>	Display background jobs for current shell
<code>[1] Running gimp &</code>	
<code>[2] Running xmms &</code>	
<code>[3]- Running gedit &</code>	
<code>[4]+ Stopped gtali</code>	
<code>\$ jobs -l</code>	Display PID with each job's information
<code>[1] 31676 Running gimp &</code>	
<code>[2] 31677 Running xmms &</code>	
<code>[3]- 31683 Running gedit &</code>	
<code>[4]+ 31688 Stopped gtali</code>	
<code>\$ jobs -l %2</code>	Display information only for job %2

The processes running in the `jobs` examples might have been started while you were logged in (using `ssh`) to a remote system. Some of those processes you might want to **run as remote GUI applications on your local desktop**. By running those processes in the background, you can have multiple applications running at once while still having those applications associated with your current shell. Once a process is running, you can **disconnect the process from the current shell using the `disown` command**:

```
$ disown %3           Disconnect job %3 from current shell
$ disown -a          Disconnect all jobs from current shell
$ disown -h          Protect all jobs from HUP sent to current
shell
```

After you have disowned a process, you can close the shell without also killing the process.

The `fg` and `bg` commands manipulate running processes by moving those processes to the foreground or background. Another way to manipulate running commands is to send signals directly to those processes. A common way to send signals to running processes is with the `kill` and `killall` commands.

Killing and Signaling Processes

You can stop or change running processes by sending signals to those processes. Commands such as `kill` and `killall` can send signals you select to running processes, which as their names imply, is often a signal to kill the process.

Signals are represented by numbers (9, 15, and so on) and strings (`SIGKILL`, `SIGTERM`, and so on). [Table 9-2](#) shows standard signals you can send to processes in Linux.

Table 9-2: Standard Signals to Send to Processes

Signal Number	Signal Name	Description
1	SIGHUP	Hang up from terminal or controlling process died
2	SIGINT	Keyboard interrupt
3	SIGQUIT	Keyboard quit
4	SIGILL	Illegal instruction
6	SIGABRT	Abort sent from abort function
8	SIGFPE	Floating point exception
9	SIGKILL	Kill signal
11	SIGSEGV	Invalid memory reference
13	SIGPIPE	Pipe broken (no process reading from pipe)
14	SIGALRM	Timer signal from alarm system call

15	SIGTERM	Termination signal
30, 10, 16	SIGUSR1	User-defined signal 1
31, 12, 17	SIGUSR2	User-defined signal 2
20, 17, 18	SIGCHLD	Child terminated or stopped
19, 18, 25	SIGCONT	Continue if process is stopped
17, 19, 23	SIGSTOP	Stop the process
18, 20, 24	SIGTSTP	Stop typed at terminal
21, 21, 26	SIGTTIN	Terminal input for background process
22, 22, 27	SIGTTOU	Terminal output for background process

The `kill` command can send signals to processes by process ID or job number while the `killall` command can signal processes by command name. Here are some examples:

```
$ kill 28665           Send SIGTERM to process with PID 28665
$ kill -9 4895        Send SIGKILL to process with PID 4895
$ kill -SIGCONT 5254  Continue a stopped process (pid 5254)
$ kill %3            Kill the process represented by job %3
$ killall spamd       Kill all spamd daemons currently running
$ killall -SIGHUP sendmail Have sendmail processes reread config
files
```

The `SIGKILL` (-9) signal, used generously by trigger-happy novice administrators, should be reserved as a last resort. It does not allow the targeted process to exit cleanly but forces it to end abruptly. This can potentially result in loss or corruption of data handled by that process. The `SIGHUP` signal was originally used on UNIX systems to indicate that a terminal was being disconnected from a mainframe (such as from a hang-up of a dial-in modem). However, daemon processes, such as `sendmail` and `httpd`, were implemented to catch `SIGHUP` signals as an indication that those processes should reread configuration files.

Running Processes Away from the Current Shell

If you want a process to continue to run, even if you disconnect from the current shell session, there are several ways to go about doing that. You can use the `nohup` command to **run a process in a way that it is impervious to a hang-up signal**:

```
$ nohup updatedb &           Run updatedb with no ability to
interrupt
# nohup nice -9 gcc hello.c & Run gcc uninterrupted and higher
priority
```

Using `nohup` is different than running the command with an ampersand alone because with `nohup` the command will keep running, even if you exit the shell that launched the

command.

The `nohup` command was commonly used in the days of slow processors and dial-up connections (so you didn't have to stay logged in to an expensive connection while a long compile completed). Also, today using tools such as `screen` (described in Chapter 13) you can keep a shell session active, even after you disconnect your network connection to that shell. So `nohup` isn't used as often as it once was.

Scheduling Processes to Run

Commands associated with the cron facility can be used to set a command to run at a specific time (including now) so that it is not connected to the current shell. The `at` command **runs a command at the time you set**:

```
$ at now +1 min           Start command running in one minute
at> updatedb
at> <Ctrl+d> <EOT>
job 5 at Sat Mar 2 20:37:00 2013
$ at teatime             Start command at 4pm today
$ at now +5 days         Start a command in five days
$ at 06/25/13           Start a command at current time on June 25,
2013
```

Another way to run a command that's not connected with the current shell is with the `batch` command. With `batch`, you can **set a command to start as soon as the processor is ready** (load average below .8):

```
$ batch                 Start command running immediately
at> find /mnt/isos | grep jpg$ > /tmp/mypics
at> <Ctrl+d> <EOT>
```

Note that after the `at` or `batch` commands you see a secondary `at>` prompt. Type the command you want to run at that prompt and press Enter. After that, you can continue to enter commands. When you are done, press `Ctrl+d` on a line by itself to queue the commands you entered to run.

After the commands are entered, you can **check the queue of at jobs that are set to run** by typing the `atq` command:

```
$ atq
11          Thu Sep  5 21:10:00 2013 a chris
10          Sat Aug 24 21:10:00 2013 a chris
8           Fri Aug 23 20:53:00 2013 a chris
```

Regular users can only see their own `at` jobs that are queued. The root user can see everyone's queued `at` jobs. If you want to **delete an at job from the queue**, use the `atrm` command:

```
$ atrm 11                Delete at job number 11
```

The `at` and `batch` commands are for queuing up a command to run as a one-shot deal. You can use the cron facility to **set up commands to run repeatedly**. These commands are scripted into cron jobs, which are scheduled in crontab files. There is one system crontab file (`/etc/crontab`). Also, each user can create a personal crontab file that can launch commands at times that the user chooses. To **create a personal crontab file**, type the following:

```
$ crontab -e                                Create a personal crontab file
```

The `crontab -e` command opens your crontab file (or creates a new one) using the `vi` text editor. Here are examples of several entries you could add to a crontab file:

```
15 8 * * Mon,Tue,Wed,Thu,Fri mail chris < /var/project/stats.txt
* * 1 1,4,7,10 * find / | grep .doc$ > /var/sales/documents.txt
```

The first crontab example shown sends a mail message to the user named `chris` by directing the contents of `/var/project/stats.txt` into that message. That mail command is run Monday through Friday at 8:15 a.m. In the second example, on the first day of January, April, July, and October, the `find` command runs to look for every `.doc` file on the system and sends the resulting list of files to `/var/sales/documents.txt`.

The last part of each crontab entry is the command that is run. The first five fields represent the time and date the command is run. The fields from left to right are: minute (0 to 59), hour (0 to 23), day of the month (0 to 31), month (0 to 12 or Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), and day of the week (0 to 7 or Sun, Mon, Tue, Wed, Thu, Fri, or Sat). An asterisk (*) in a field means to match any value for that field.

If there is output from the command that is run, and you don't direct that output to a file or another command, by default the output is sent to the user running the `cron` command. To set a specific e-mail address to receive the output, you can add a `MAILTO` line to the crontab file. For example:

```
MAILTO=chris@example.com
```

Here are some **other options with the `crontab` command**:

```
# crontab -eu chris                        Edit another user's crontab (root
only)
$ crontab -l                                List contents of your crontab file
15 8 * * Mon,Tue,Wed,Thu,Fri mail chris < /var/project/stats.txt
* * 1 1,4,7,10 * find / | grep .doc$ > /var/sales/documents.txt
$ crontab -r                                Delete your crontab file
```

The traditional way to configure system cron jobs was to add them to the system crontab. Although this is still an option, Ubuntu provides an easier way to create hourly, daily, weekly, and monthly cron jobs, by **associating the command you want to run with a cron directory**. Simply create a script that you want to run. Then copy the script

to the `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly`, or `/etc/cron.monthly` directory. The command will then run in the time frame indicated by the directory (hourly, daily, weekly, or monthly).

An alternative to the cron facility is the anacron facility. With anacron, as with cron, you can configure commands to run periodically. However, anacron is most appropriate for machines that are not on all the time. If a command is not run because the computer was off during the scheduled time, the next time the computer is on, the anacron facility makes sure that the commands that were missed during the downtime are run after the system resumes.

Scheduling Realtime Processes

The order processes are run, and the amount of attention they get from the CPUs is determined by the process scheduler. In most cases, adjusting the `nice` value on regular processes (those running under the standard `SCHED_OTHER` policy) is the only change you need to give a process higher or lower priority to the processor. (See the section “Adjusting Processor Priority with `nice`” earlier in this chapter.)

In some rare cases, however, you might want to assign a special policy to a process. While some of those policies reduce the amount of attention processes get from the CPUs (for example, some long-running batch jobs that are not time critical), other policies are what is referred to as realtime policies, which get a process ahead of any `SCHED_OTHER` processes running on the system.

Note Setting realtime priority to a process should not be done on production systems unless you are sure of what you are doing. These processes will preempt system processes and can make your system unstable or even unusable if done incorrectly.

Processor policies in Linux include the following:

- `SCHED_OTHER`—All processes receive this scheduling on basic Linux time-sharing systems unless another scheduling type is assigned. Order of processing is based on `nice` value.
- `SCHED_BATCH`—Processes set to this priority get less favor in scheduling, assuming the processes demand a lot of CPU but can wait for attention. Processes within this policy are scheduled based on `nice` value (although they would get less attention than processes of the `SCHED_OTHER` policy).
- `SCHED_IDLE`—Processes in this policy get very low priority (after even `SCHED_OTHER` and `SCHED_BATCH` processes set to 19 `nice` value).
- `SCHED_FIFO`—Any processes set to this realtime processor policy will preempt

any running `SCHED_OTHER`, `SCHED_BATCH`, and `SCHED_IDLE` process. Within the `SCHED_FIFO` policy, processes are scheduled first in, first out.

- `SCHED_RR`—This realtime process is like `SCHED_FIFO`, except that processes in this policy share the CPU with each other. In other words, one `SCHED_RR` process need not complete before another `SCHED_RR` process can run.

Using the `chrt` command, you can see which of the five Linux scheduling policies is assigned to a command. You can also use `chrt` to change the scheduling policy on a running process or start a process with a particular scheduling policy.

To read more about process scheduling, see the `chrt` and `sched_setschedule` man pages.

Change a Process to a Realtime Process

The following commands find the process ID of a regular process you want to make into a realtime process, list its scheduling policy, and change its policy:

```
$ pidof firefox                Find process ID of the firefox process
16563
$ chrt -p 16563                List scheduling policy of process
pid 16563's current scheduling policy: SCHED_OTHER
pid 16563's current scheduling priority: 0
$ chrt -m                      List scheduling policies and ranges
SCHED_OTHER min/max priority      : 0/0
SCHED_FIFO min/max priority       : 1/99
SCHED_RR min/max priority         : 1/99
SCHED_BATCH min/max priority      : 0/0
SCHED_IDLE min/max priority       : 0/0
$ sudo chrt -r -p 20 16563      Change round robin realtime process
$ chrt -p 16563                View new round robin realtime process
pid 16563's current scheduling policy: SCHED_RR
pid 16563's current scheduling priority: 20
$ sudo chrt -f -p 10 16563     Change to fifo realtime process
```

Start a Process as a Realtime Process

To start a process at a particular priority, you can use `chrt` when you first start the process.

```
$ sudo chrt -r 50 firefox      Start at round-robin priority 50
$ sudo chrt -f 99 gcc hello.c  Start at fifo priority 99
```

Summary

Watching and working with the processes that run on your Linux system are important activities to make sure that your system is operating efficiently. Using commands such as `ps` and `top`, you can view the processes running on your system. You can also use `pgrep` to search for and list particular processes.

With commands such as `nice` and `renice`, you can adjust the recommended priorities at which selected processes run. When a process is running, you can change how it is running or kill the process by sending it a signal from the `kill` or `killall` command.

After launching a command from the current shell, you can set that command's process to run in the background (`bg`) or foreground (`fg`). You can also stop and restart the process using different control codes.

To schedule a command to run at a later time, you can use the `at` or `batch` command. To set up a command to run repeatedly at set intervals, you can use the `cron` or `anacron` facilities. To run commands at different scheduling priorities, use the `chrt` command.

Chapter 10

Managing the System

IN THIS CHAPTER

- Checking memory use with `free`, `top`, `vmstat`, and `slabtop`
- Viewing CPU use with `iostat`, `dstat`, and `top`
- Monitoring storage devices with `iostat`, `vmstat`, and `lsdf`
- Working with dates/time using `date`, `hwclock`, `cal`, and NTP
- Changing GRUB boot loader behavior
- Rebuilding the initial ramdisk
- Dealing with run levels with `runlevel` and `init`
- Adding, removing, and listing services with `chkconfig` and `service`
- Shutting down the system with `reboot`, `halt`, and `shutdown`
- Checking and changing kernel driver settings with `lsmod`, `modinfo`, and `modprobe`
- Watching hardware settings with `lspci`, `dmidecode`, and `hdparm`

Without careful management, the demands on your Linux system can sometimes exceed the resources you have available. Being able to monitor your system's activities (memory, CPU, and device usage) over time can help you make sure that your machine has enough resources to do what you need it to. Likewise, managing other aspects of your system, such as the device drivers it uses and how the boot process works, can help avoid performance problems and system failures.

This chapter is divided into several sections that relate to ways of managing your Ubuntu or other Linux system. The first section can help you monitor the resources (processing power, devices, and memory) on your Linux system. The next section describes how to check and set your system clock. Descriptions of the boot process and subsequent runlevels follow. The last sections describe how to work with the kernel and related device drivers, as well as how to view information about your computer's hardware components.

Monitoring Resources

Ubuntu, Debian, and other Linux systems do a wonderful job of keeping track of what

they do. If you care to look, you can find lots of information about how your CPU, hard disks, virtual memory, and other computer resources are being used.

You can go to where the Linux kernel stores realtime information about your system by directly viewing the contents of files in the `/proc` filesystem (see Appendix C). An alternative, however, is to use commands to view information about how your computer's virtual memory, processor, storage devices, and network interfaces are being used on your system.

There are commands that can monitor several different aspects of your system's resources. Because this book is not just a man page, however, I have divided the following sections by topic (monitoring memory, CPU, and storage devices) rather than by the commands that do them (`top`, `vmstat`, and `iostat`).

Note Some applications described in this section are installed by default in Ubuntu, in packages such as the `procps` package. To use `iostat` or `sar`, however, you need to install the `sysstat` package. Install the `sysstat` package with the following command:

```
$ sudo apt-get install sysstat
```

Monitoring Memory Use

Few things will kill system performance faster than running out of memory. Commands such as `free` and `top` enable you to see basic information about how your RAM and swap are used. The `vmstat` command gives detailed information about memory use and can run continuously. The `slabtop` command can show how much memory the kernel (slab cache) is consuming.

The `free` command provides the quickest way to **see how much memory is being used** on your system. It shows the total amount of RAM (`Mem:`) and swap space (`Swap:`), along with the amount currently being used. The following are examples of the `free` command:

```
$ free                                List memory usage in kilobytes (-k default)
$ free -m                             List memory usage in megabytes
      total      used      free      shared      buffers      cached
Mem:    3920      2105      1815           0          550          942
-/+ buffers/cache:    612      3307
Swap:    4059           0      4059
$ free -b                             List memory usage in blocks
      total      used      free      shared      buffers      cached
Mem:    4110852096 2207576064 1903276032           0 576901120 987951104
-/+ buffers/cache: 642723840 3468128256
```

```

Swap: 4257214464          0 4257214464
$ free -mt                List memory usage with totals (Swap + Mem)
      total      used      free      shared      buffers      cached
Mem:    3920      2075      1845          0         550         942
-/+ buffers/cache:    582      3337
Swap:    4059          0      4059
Total:    7980      2075      5905
$ free -g                List memory usage in gigabytes (rounding down)
      total      used      free      shared      buffers      cached
Mem:         3         2         1          0          0          0
-/+ buffers/cache:         0         3
Swap:         3         0         3
$ free -s 5              Continuously display memory usage every 5 seconds

```

To avoid wasting RAM and speed up applications, Linux uses as much otherwise unused RAM as possible for the disc cache. For that reason, the first line of output from `free` that often shows little free RAM can be misleading. I recommend you pay closer attention to the second line of output, which shows the amount of RAM actually available for applications. That amount is 3307MB in this example:

```

-/+ buffers/cache:    612      3307

```

One way to guess how much memory you need on a system is to go to another computer running Ubuntu, and then open every application you think you may be running at once. Run `free` with the total option (`free -t`) to see how much memory is being used. Then make sure that your new system has at least that much total memory (with most or all of it preferably being available in RAM).

The `top` command provides a means of watching the currently running processes, with those processes sorted by CPU usage or memory (see Chapter 9 for a description of `top` for watching running processes). However, you can also use `top` to **watch your memory usage in a screen-oriented way**. Here is an example:

```

$ top
top - 8:43:04 up 3 days, 5:00, 3 users,
      load average: 0.21, 0.18, 0.11
Tasks: 188 total, 1 running, 187 sleeping  0 stopped,  0 zombie
Cpu(s): 17.4%us, 3.3%sy, 0.0%ni, 79.3%id,
        0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 4014504k total, 1950936k used, 2063568k free, 248148k buffers
Swap: 4157436k total,      0k used, 4157436k free, 1016924k cached
  PID USER  PR  NI  VIRT  RES  SHR  S  %CPU  %MEM   TIME+  COMMAND
 11062 chris  20   0 1636m 47m   23m  S   26   1.2  0:20.19 totem
   2646 chris  20   0 1198m 88m   28m  S    7   2.3  6:36.98 compiz
   1750 root   20   0  209m 29m   12m  S    4   0.8  4:33.56 Xorg

```

To exit `top`, press `q`. Like the output for `free`, `top` shows the total of memory usage for RAM (`Mem:`) and swap space (`Swap:`). However, because `top` is screen-oriented

and provides ongoing monitoring, you can watch memory usage change every three seconds (by default). With `top` running, press `Shift+m` and the running processes will be displayed in memory-use order (so you can watch which processes are consuming the most memory). The most useful column to analyze a process's memory usage is `RES`, which shows the process's actual physical RAM usage, also known as **resident size**. The `%MEM` column is based on this resident size.

Because most new computers today have multiple CPUs, the `Cpu(s) :` line shows the average CPU usage. To see CPU usage for each individual CPU, press the number `1` key. The `Cpu` lines will look something like the following:

```
Cpu0 : 17.7%us, 5.3%sy, 0.0%ni, 44.7%id,
      31.7%wa, 0.0%hi, 0.7%si, 0.0%st
Cpu1 : 16.4%us, 4.4%sy, 0.0%ni, 46.4%id,
      32.8%wa, 0.0%hi, 0.0%si, 0.0%st
```

For a more detailed view of your virtual memory statistics, use the `vmstat` command. With `vmstat`, you can **view memory use over a given time period**, such as since the previous reboot or using a sample period. The following example shows `vmstat` redisplaying statistics every three seconds:

```
$ vmstat 3
procs -----memory----- --swap-- ----io---- --system-- --
cpu--
 r  b  swpd  free  buff  cache  si   so  bi   bo   in   cs us  sy id
wa st
 1  0   7740 32488 3196 148360   0    0   0    1   26 3876 85 15   0
0  0
 1  1   8388  7428 3204 151472   0  216   0  333   30 3200 82 18   0
0  0
 1  0 13316  8148 2980 146968   0 4980   4 5121   79 3846 77 23   0
0  0
 2  0 32648  7472 2904 148488   0 6455   3 6455   90 3644 83 17   0
0  0
 2  0 47892  8088 2732 144208   0 5085   9 5220   79 3468 84 16   0
0  0
 1  0 57948  7680 2308 134812   0 3272  12 3296   69 3174 77 23   0
0  0
 3  0 58348  7944 1100 123888  21  144  25  275   26 3178 86 14   0
1  0
 2  0 66116  7320  568  10   11 2401  20 2403   51 3175 84 16   0   0
0
 3  0 81048  7708  648 119452  53 4852 796 4984  123 1783 86 13   0
1  0
```

To exit `vmstat`, press `Ctrl+c`. The `vmstat` example shows a 30-second time period where more than 100 applications are started. Notice that when the free space goes from

32488 kilobytes to 7428 kilobytes (RAM is filling up), data begins moving to the swap area (see the 216 under the `so` column). Because the swap area resides on the hard disk, you can see that the blocks written to a disk device (`bo`) increases as the swap out increases. You can see the amount of swap space being used increasing under the `swpd` column.

The CPU is also straining in the example, with no idle time showing (`id 0`). Notice also that when some of the applications need to be swapped back in (see the last three lines of output), the processor has to wait on two occasions for input/output to complete (`wa 1`).

Here are some other options for using `vmstat`:

```
$ vmstat -S m           Display output in 1000k megabytes
$ vmstat -S M          Display output in 1024k megabytes
$ vmstat -S k          Display output in 1000-byte kilobytes
$ vmstat -S K          Display output in 1024-byte kilobytes
$ vmstat -n 2 10       Output every two seconds, repeat 10 times
$ vmstat -s | less     Display event counters and memory stats
$ vmstat -S M -s | less Display statistics in megabytes
    3920 M total memory
    1953 M used memory
     972 M active memory
     726 M inactive memory
    1966 M free memory
     261 M buffer memory
     993 M swap cache
    4059 M total swap
       0 M used swap
    4059 M free swap
  137803 non-nice user cpu ticks
    9412 nice user cpu ticks
    52746 system cpu ticks
  62828527 idle cpu ticks
    ...
```

The previous example shows various memory statistics (`-s`) output in megabytes (`-S M`), which you will find more convenient to get a general view of memory usage. The other examples show how to display `vmstat` output in megabytes and kilobytes (in both marketing and technical terms). After that, the `-n 2 10` option tells `vmstat` to repeat every set number of seconds (2) for a limited number of times (10).

With commands, such as `ps` and `top`, you can see how much memory each application is consuming on your system. The kernel itself, however, has its own memory cache to keep track of its resources, called the **kernel slab**. You can use the `vmstat` command to **display kernel slab memory cache statistics** (from `/proc/slabinfo`) as follows:

```
# vmstat -m | less           Page through kernel slab memory cache
```

Cache	Num	Total	Size	Pages
udf_inode_cache	120	120	664	24
dm_crypt_io	0	0	152	26
nf_conntrack_expect	0	0	240	17
nfsd4_openowners	0	0	392	20
nfs_read_data	0	0	768	21
nfs_inode_cache	0	0	1008	16
fscache_cookie_jar	51	51	80	51
rpc_inode_cache	19	19	832	19
ext2_inode_cache	105	105	752	21
ip6_dst_cache	50	50	320	25
UDPLITEv6	0	0	1024	16
...				

The slab memory cache information shows each cache name, the number of objects active for that cache type, the total number of objects available for that cache type, the size of the cache (in bytes), and the number of pages for each cache. You can **display kernel slab memory cache information in a screen-oriented view** (similar to the `top` command) using `slabtop`:

```
# slabtop
Active / Total Objects (% used) : 562520 / 566736 (99.3%)
Active / Total Slabs (% used)   : 20103 / 20103 (100.0%)
Active / Total Caches (% used)  : 74 / 110 (67.3%)
Active / Total Size (% used)    : 152005.08K / 153123.88K (99.3%)
Minimum / Average / Maximum Object: 0.01K / 0.27K / 8.00K
  OBJS ACTIVE  USE OBJ SIZE SLABS OBJ/SLAB CACHE SIZE NAME
144768 144400  99%   0.10K  3712    39   14848K buffer_head
142170 141903  99%   0.19K  6770    21    7080K dentry
  57168  57168 100%   0.86K  3176    18   50816K ext4_inode_cache
  40898  40726  99%   0.61K  1573    26   25168K proc_inode_cache
  23713  23611  99%   0.17K  1031    23    4124K vm_area_struct
  19824  19824 100%   0.14K   708    28    2832K sysfs_dir_cache
```

The `slabtop` output updates every three seconds. By default, slab caches are sorted by the number of objects (first column) in each cache. By pressing `c` you can sort by cache size instead (as shown in the previous example).

Monitoring CPU Usage

An overburdened CPU is another obvious place to look for performance problems on your system. The `vmstat` command, shown earlier, can produce basic statistics relating to CPU usage (user activity, system activity, idle time, I/O wait time, and time stolen from a virtual machine). The `iostat` command (from the `sysstat` package), however, can generate more detailed reports of CPU utilization.

Here are two examples of using `iostat` to **display a CPU utilization report**:

```

$ iostat -c 3          CPU stats every 3 seconds (starting apps)
Linux 3.2.0-38-generic (ubuntutb)    05/20/2013    _x86_64_ (2 CPU)
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.26    0.01    0.09    0.05    0.00   99.58
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           22.64    0.00    8.18    2.36    0.00   66.82
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           32.01    0.00    4.14   19.90    0.00   43.95

$ iostat -c 3          CPU stats every 3 seconds (copying files)
Linux 3.2.0-38-generic (ubuntutb)    05/20/2013    _x86_64_ (2
CPU)
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.37    0.01    0.11    0.15    0.00   99.35
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           21.69    0.00    6.10   65.42    0.00    6.78
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           21.48    0.00    5.87   63.76    0.00    8.89
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           19.35    0.00    7.30   68.76    0.00    4.58

```

The first `iostat` example just shown starts with a quiet system, and then several applications start up. You can see that, after the first line which shows average use since the system was last booted, most of the processing to start the applications is being done in user space.

The second `iostat` example shows a case where several large files are copied from one hard disk to another. The result is a higher percentage of time being spent waiting for I/O requests to complete (`%iowait`).

The following examples use `iostat` to **print CPU utilization reports with timestamps**:

```

$ iostat -c -t          Print time stamp with CPU report
Linux 3.2.0-38-generic (ubuntutb)    05/20/2013    _x86_64_ (2
CPU)
05/20/2013 07:08:20 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.40    0.01    0.12    0.21    0.00   99.26

$ iostat -c -t 2 10     Repeat every 2 seconds for 10 times

```

The `dstat` command (`dstat` package) is available as an alternative to `iostat` for **viewing information about your CPU usage** (as well as other performance-related items). One advantage of `dstat` over other tools is that it more precisely shows the units of measurement it is displaying (such as kilobytes or megabytes) and also uses colors to differentiate the data. To install the `dstat` package, type the following:

```
$ sudo apt-get install dstat
```

Here is an example of `dstat` for displaying CPU information:

```
$ dstat -t -c 3           View CPU usage continuously with time stamps
----system---- ----total-cpu-usage----
      time      |usr sys idl wai hiq siq
20-05 19:14:38| 27   5  59   9   0   1
20-05 19:14:41| 34   7  55   3   0   1
20-05 19:14:44| 35   6  54   4   0   0
20-05 19:14:47| 21   4  74   1   0   0
20-05 19:14:50| 19   3  76   1   0   0
20-05 19:14:53| 19   4  77   0   0   0
20-05 19:14:56| 18   3  79   0   0   0
```

In this example, the output includes date/time values based on the start of the epoch (-t) for the CPU report (-c) that is produced every three seconds (3). This report runs continuously until you stop it (Ctrl+c).

If you want to find out specifically which processes are consuming the most processing time, you can use the `top` command. Type `top`, and then press Shift+p to sort by CPU usage (this is the default sorting order):

```
$ top           Display running processes and sort by CPU usage
top - 9:18:29 up 3 days, 16:36, 3 users, load avg: 0.29, 1.05, 1.73
Tasks: 192 total, 1 running, 191 sleeping, 0 stopped, 0 zombie
Cpu(s): 14.6%us,2.2%sy,0.0%ni,82.7%id,0.3%wa,0.0%hi,0.2%si,0.0%st
Mem: 4014504k total, 3305876k used, 708628k free, 360896k buffers
Swap: 4157436k total, 264k used 4157172k free,2229972k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 11338 chris    20   0 1637m  48m  24m S   22   1.2 10:48.97 totem
   2646 chris    20   0 1214m  88m  28m S    6   2.3 17:55.98 compiz
   1750 root      20   0  206m  32m  13m S    4   0.8  8:53.63 Xorg
```

The full output would show many more processes, all sorted by current CPU usage (%CPU column). In this example, Totem Movie Player (22%), the compiz window manager (6%), and the X display server (4%) are consuming most of the CPU. If you decided you wanted to kill the totem process, you could type `k` followed by the process ID of totem (11338) and the number 9 signal (if for some reason you couldn't just close the Totem window normally).

If you want **information about the processor itself**, you can view information directly from the `/proc/cpuinfo` file. Here is an example:

```
$ cat /proc/cpuinfo           View CPU information from /proc
Processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 23
model name      : Pentium(R) Dual-Core CPU           E6300   @ 2.80GHz
...
Processor       : 1
vendor_id       : GenuineIntel
```

```

cpu family      : 6
model          : 23
model name     : Pentium(R) Dual-Core CPU           E6300   @ 2.80GHz
...
Flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht
tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts rep_good
nopl aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16
xtpr pdcm xsave lahf_lm dtherm tpr_shadow vnmi flexpriority
bogomips       : 5585.95
clflush size   : 64
...

```

This particular machine has two processors, so information is repeated for each processor. For each processor, you can see the vendor's name, the model name, and its speed (2.80GHz) in this case. Interesting things to note about your CPU are the flags that represent features that it supports. Some features in Ubuntu require that particular CPU extensions associated with those flags be on for the Ubuntu feature to work. For example, to use the Xen virtualization para-virtualized guests, the `pae` flag must be set. To run fully virtualized guests, the CPU must have either the `vmx` flag (for Intel processors) or `svm` flag (for AMD processors) extension support.

Similar information about your processor(s) is collected by the system at the very beginning of the boot process and can be obtained by looking at the beginning of the output of the `dmesg` command.

Monitoring Storage Devices

Basic information about storage space available to your Linux filesystems can be seen using commands such as `du` and `df` (as described in Chapter 7). If you want details about how your storage devices are performing, however, commands such as `vmstat` and `iostat` can be useful.

Some of the same kind of output from the `iostat` command shown earlier can be used to **tell if bottlenecks occur while doing disk reads and writes**. Here's an example:

```

$ iostat 3          Check disk reads and writes per disk
Linux 3.2.0-38-generic (ubuntutb)          05/20/2013    _x86_64_   (2
CPU)
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.48    0.02   0.13   0.23    0.00   99.15

Device:            tps    kB_read/s    kB_wrtn/s    kB_read  kB_wrtn
sda                 0.45         4.38         26.63    1400105   8517944
dm-0                0.68         4.37         26.62    1396897   8517196
dm-1                0.00         0.00         0.00       1152      736
scd0                0.39        24.94         0.00    7978782      0

```



```

sdb          0.01          0.01          1.49          2073      476784
Device:      tps      kB_read/s      kB_wrtn/s      kB_read      kB_wrtn
sda          17.00      1909.33          0.00          5728          0
dm-0         17.00      1909.33          0.00          5728          0
dm-1          0.00          0.00          0.00           0          0
scd0          0.00          0.00          0.00           0          0
sdb          46.33          0.00      5462.67           0      16388
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00   0.34   74.41    0.00   25.25
Device:      tps      kB_read/s      kB_wrtn/s      kB_read      kB_wrtn
sda          11.00      1257.33          0.00          3772          0
dm-0         11.00      1257.33          0.00          3772          0
dm-1          0.00          0.00          0.00           0          0
scd0          0.00          0.00          0.00           0          0
sdb          42.33          0.00      5021.33           0      15064
...

```

The first part of the output of `iostat` shows averages of CPU usage since the last reboot. The next part reflects processing that occurs when a large amount of data is copied from the first disk (`sda`) to the second disk (`sdb`). The `dm-0` device is actually an LVM logical volume on `sda` that is connected to the root filesystem.

High `iowait` values indicate that disk input/output is the bottleneck on the system. In other words, faster disk writing would improve performance more than a faster CPU.

The `vmstat` command can also list statistics about your disks. Here's an example of using `vmstat` to list information about disk reads and writes:

```

$ vmstat -d          Display disk read, write, and input/output stats
disk- -----reads----- --writes----- --
IO---
      total merged sectors      ms  total merged sectors      ms
cur sec
...
sda      80736  21757 9133194 1075864 101513 67798 27543352  97896  0
702
dm-0     101740      0 9127290 1568064 153265      0 27522840  24100  0
702
dm-1       288      0   2304   1680   2561      0   20488   6688  0
3
sr0      195504 697318 24962380 7629216      0      0      0      0  0
2129
sdb       457    126   4714    848  31786   307  7492448 075688  0
767

```

The Linux system in this example has two hard disks (`sda` and `sdb`). You can see the total number of sectors successfully read and written from those hard disks. You can also see how many seconds were spent on input/output (IO) for those disks. Furthermore, you can see if there are any I/O operations in progress, and you can also

list read/write information for selected disk partitions. Here is an example:

```
$ vmstat -p sda1          Display read/write stats for a disk partition
sda1                reads   read sectors   writes   requested writes
                   296      2172           10         24
```

Unfortunately the preceding command does not work with softraid `md` partitions, `lvm` partitions, and some hardware RAID driver-specific devices.

If you want to find out **what files and directories are currently open on your storage devices**, you can use the `lsof` command. This command can be particularly useful if you are trying to unmount a filesystem that keeps telling you it is busy. You can check what open file is preventing the unmount and decide if you want to kill the process holding that file open and force an unmount of the filesystem. Here is an example of `lsof`:

```
$ lsof | less              List processes holding files/directories open
COMMAND PID    USER   FD   TYPE    DEVICE  SIZE/OFF  NODE NAME
init      1             root   cwd    DIR     252,0    4096     2 /
init      1             root   rtd    DIR     252,0    4096     2 /
init      1             root   txt    REG     252,0   16719 3445 /sbin/init
...
bash     166          chris   cwd    DIR      8,17    4096     2 /mnt/a
```

The first files shown as being open are those held open by the `init` process (the first running process on the system). Files held open by system processes (such as `udev`) and daemons (such as `sshd` and `syslogd`) follow `init`. Eventually, you will see files held open by individual users (which are probably the ones you are interested in if you are unable to unmount a disk partition).

Note You may see permission restrictions unless you run the `sudo` command first:

```
$ sudo lsof | less
```

When you are looking at the `lsof` output, you want to see the name of the file or directory that is open (`NAME`), the command that has it open (`COMMAND`), and the process ID of that running command (`PID`). In the preceding example, the current directory of a running bash shell is holding a directory open in the filesystem I want to unmount (`/mnt/a`). When a filesystem won't unmount, it is often a bash shell that has its current directory on that filesystem that is preventing the unmount. Instead of piping `lsof` output to `less` or `grep`, here are a few other ways to find what you are looking for from `lsof` output:

```
$ lsof -c bash              List files open by bash shells
$ lsof -d cwd               List directories open as current directory
```

```
$ lsuf -u chris           List files and directories open by user  
chris  
$ lsuf /mnt/sda1         List anything open on /mnt/sda1 file system  
$ lsuf +d /mnt/sda1/dx/ List anything open under /mnt/sda1/dx/
```

As noted previously, you may need to use the `sudo` command to acquire the root permissions to view all the output of the `lsuf` command.

Mastering Time

Keeping correct time on your Linux system is critical to the system's proper functioning. Your computer running Linux keeps time in two different ways: a system clock (which Linux uses to keep track of time) and a hardware clock (which sets the system time when Linux boots up).

The system time is used to set timestamps for file creation, process run times, and anything else where date and time are used. System time can be viewed and set manually (with the `date` command) or automatically (with the `ntpd` service).

The hardware clock is part of the motherboard's CMOS and runs on a battery attached to the motherboard when the system is powered off. You set the hardware clock with the `hwclock` command.

There are many other tools that can be used to work with time in Linux systems. For example, there are tools for checking time in different ways, such as using `clockdiff` (to measure clock difference between computers) and `uptime` (to see how long your system has been up).

Changing Time/Date with Graphical Tools

Graphical tools in Ubuntu and other Linux systems for changing the date, time, and time zone used on your system include the Time & Date window (accessible by selecting Time & Date from the Ubuntu Dashboard). That window can also be used to enable the **Network Time Protocol (NTP)**, to automatically synchronize your Linux system's date and time over the network.

The Time & Date Properties window saves the settings and choices you make. During Ubuntu startup, the system reads these settings to set your time zone and whether your system is using UTC time.

Your Linux system's time zone is set based on the contents of the `/etc/localtime` file. You can set a new time zone immediately by copying the file representing your time zone from a subdirectory of `/usr/share/zoneinfo`. For example, to change the current time zone to that of America/Chicago, you could do the following:

```
$ sudo cp /usr/share/zoneinfo/America/Chicago /etc/localtime
```

This can also be accomplished by creating a symlink:

```
$ sudo ln -s /usr/share/zoneinfo/America/Chicago /etc/localtime
```

To change the time zone permanently, set it in the Time & Date window to the time zone you want—for example, New York, United States.

Displaying and Setting Your System Clock

The `date` command is the primary command-based interface for viewing and changing date and time settings, if you are not having that done automatically with NTP. Here are examples of `date` commands for **displaying dates and times** in different ways:

```
$ date                                Display current date/time/time zone
Sun Jul  7 20:13:00 EDT 2013
$ date '+%A %B %d %G'                  Display day, month, day of month,
year
Sunday July 7 2013
$ date '+The date today is %F.'      Add words to the date output
The date today is 2013-07-07.
$ date --date='4 weeks'                Display date four weeks from today
Sun Aug  4 20:16:09 EDT 2013
$ date --date='8 months 3 days'       Display date 8 months 3 days from
today
Mon Mar 10 20:27:46 EDT 2014
$ date --date='4 Jul' +%A              Display day on which July 4 falls
Thursday
```

Although the primary interest in this section is time, while I'm on the subject of dates, the `cal` command is also a quick way to **display dates by month**. Here are examples:

```
$ cal                                Show current month calendar (today is highlighted)
      July 2013
Su Mo Tu We Th Fr Sa
      1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
$ cal 2013                            Show whole year's calendar
                                2013
      January                  February                  March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
      1  2  3  4  5              1  2              1  2
  6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19    10 11 12 13 14 15 16    10 11 12 13 14 15 16
20 21 22 23 24 25 26    17 18 19 20 21 22 23    17 18 19 20 21 22 23
```

27 28 29 30 31

24 25 26 27 28

24 25 26 27 28 29 30
31

```
...
$ cal -j                Show Julian calendar (numbered from January 1)
      July 2013
Su   Mo   Tu   We   Th   Fr   Sa
      182 183 184 185 186 187
188 189 190 191 192 193 194
195 196 197 198 199 200 201
202 203 204 205 206 207 208
209 210 211 212
```

The `date` command can also be used to **change the system date and time**. For example:

```
$ sudo date 081215212013          Set date Aug. 12, 2:21PM, 2013
Mon Aug 12 15:21:00 EDT 2013
$ sudo date --set='+7 minutes'    Set time to 7 minutes later
Mon Aug 12 15:28:39 EDT 2013
$ sudo date --set='-1 month'      Set date/time to a month earlier
Fri Jul 12 15:29:11 EDT 2013
```

The next time you boot Ubuntu, the system time will be reset based on the value of your hardware clock (or your NTP server, if NTP service is enabled). And the next time you shut down, the hardware clock will be reset to the system time in order to preserve that time while the machine is powered off. To change the hardware clock, you can use the `hwclock` command.

Displaying and Setting Your Hardware Clock

Anyone can use the `hwclock` command to view hardware clock settings; however, you must have root privileges to change those settings. To use `hwclock` to **view the current time from your computer's hardware clock**, type the following:

```
$ hwclock -r                Display current hardware clock settings
Mon 20 May 2013 08:14:55 PM EDT -0.312931 seconds
```

Even if your hardware clock is set to UTC time, `hwclock` displays local time by default. If your system time strays from your hardware clock (for example, if you tried some of the `date` commands shown previously), you can **reset your system clock from your hardware clock** as follows:

```
$ sudo hwclock --hctosys      Reset system clock from hardware clock
```

Likewise, if your hardware clock is set incorrectly (for example, if you replaced the CMOS battery on your motherboard), you can **set the hardware clock from your system clock** as follows:

```
# hwclock --systohc      Reset hardware clock from system clock
```

Over time your hardware clock can drift. Because the clock tends to drift the same amount each day, `hwclock` can keep track of this drift time (which it does in the `/etc/adjtime` file). You can **adjust the hardware clock time** based on the `adjtime` file as follows:

```
$ sudo hwclock --adjust    Adjust hardware clock time for drift
```

To **set the hardware clock to a specific time**, you can use the `--set` option. Here is an example:

```
$ sudo hwclock --set --date="8/12/13 18:22:00"  Set new date/time
```

In this example, the hardware clock is set to August 12, 2013 at 6:22 p.m. This update does not immediately affect the system clock.

Using Network Time Protocol to Set Date/Time

If you didn't configure Network Time Protocol (NTP) to set time for your system when you first installed your Linux system, you can do so later by **turning on the `ntpd` service**. You can install and configure the `ntpd` service (`ntpd` daemon) with just a few commands.

Use the following command to enable the service from the shell:

```
$ sudo apt-get install ntp    Install ntp package if necessary,  
                             start the service
```

The `ntpd` service uses information in the `/etc/ntp.conf` file. You can specify the name or IP address of the time server to query for the time, for example.

The resulting NTP setup turns your machine into a time server, listening on UDP port 123. Unless you have very specific needs (and your own GPS or atomic clock), running `ntpd` on your machine can be both a waste of resources and a security risk. For that reason, some system administrators prefer using `ntpdate` (often in a daily cronjob) to set their system time via NTP:

```
$ sudo ntpdate pool.ntp.org  
08 July 20:21:19 ntpdate[16121]:  
step time server 72.8.140.200 offset 31565336.613857 sec
```

If you try running `ntpdate` while `ntpd` is running, you will get the following error:

```
$ sudo ntpdate pool.ntp.org  
08 July 00:37:00 ntpdate[9695]: the NTP socket is in use, exiting
```

Note that the `ntpdate` command has been marked as deprecated and will disappear in the future.

Managing the Boot Process

When a computer first starts up, the basic input/output system (BIOS) looks to its boot order settings to determine where to find the operating system to boot. Typically, if a bootable medium has not been inserted into a removable drive (CD, DVD, floppy disk, and so on), the BIOS looks to the master boot record (MBR) on the first bootable hard disk. At this point, for most Linux systems, control of the boot process is handed to the **boot loader**.

For Ubuntu, and, in fact, most Linux systems these days, the **Grand Unified Boot Loader (GRUB)** is the boot loader that is used by default. GRUB can be set up to boot not only your Linux system, but also any other operating systems installed on your hard disks (Windows, BSD, or others). GRUB can also include boot options with each bootable operating system to refine the boot process, such as to turn on or off support for a particular type of hardware.

Ubuntu 9.10 and beyond use the second version of GRUB (GRUB 2) as its boot loader. GRUB 2 is much faster and configurable than the previous version (often called GRUB legacy). It is also harder to learn.

Once a Linux system is selected to boot from the boot loader, the boot loader loads the kernel. The following dilemma then occurs: The kernel needs to mount the root filesystem on the hard drive. This requires the appropriate storage drivers (block device kernel modules). And those drivers are located on the hard drive itself!

To break that vicious cycle, a small initial ramdisk (initrd) containing the block device modules (along with various tools and configuration files) is mounted by the boot loader. This allows the Linux kernel to read the root filesystem. After that, the `init` process takes over and begins starting the system services, based on the runlevel that is set for the system.

The following sections describe commands for modifying the boot loader, startup scripts, and run levels associated with your Linux system.

Understanding the GRUB Boot Loader

If GRUB was set up when you first installed Ubuntu, the settings for your boot loader are stored in the `/boot/grub/grub.cfg` file. Unlike the main configuration files in GRUB Legacy (`menu.lst` or `grub.cfg`), you should not edit the `grub.cfg` file directly. The `grub.cfg` file is overwritten from the contents of files in the `/etc/grub.d` directory any time a new kernel is installed or when you run the `update-grub` command.

The `/boot/grub/grub.cfg` file contains settings to control how GRUB behaves and

how modules must be loaded, as well as other information required to start the operating system. Of most interest, however, are the entries that launch the kernel to start Ubuntu (or other operating system). The following example shows entries in the `grub.cfg` file that are used to create menu entries on your GRUB boot menu:

```
menuentry 'Ubuntu, with Linux 3.2.0-38-generic' --class ubuntu
    --class gnu-linux --class gnu --class os
{
    recordfail
    gfxmode $linux_gfx_mode
    insmod gzio
    insmod part_msdos
    insmod ext2
    set root='(hd0,msdos1)'
    search --no-floppy --fs-uuid
        --set=root 0d2494ad-62e6-43d4-8aff-780630dd65d
    linux /vmlinuz-3.2.0-38-generic root=/dev/mapper/ubuntub-root ro
    initrd /initrd.img-3.2.0-38-generic
}
```

This example shows a `menuentry` for a bootable operating system ('Ubuntu, with Linux 3.2.0-38-generic') to identify the version of the kernel that runs when this menu item is selected. Information on the `menuentry` line identifies the entry as an Ubuntu Linux operating system. Within that section, the screen resolution of the Terminal window is set (`gfxmode`) and modules are loaded to uncompress zip files (`gzio`), handle DOS partition structure (`part_msdos`), and support ext2 filesystem types needed to use the /boot partition (`ext2`).

The root partition is on the first hard disk (`hd0`) and its partition table is formatted as a DOS partition (`part_msdos`). The most critical data, however, is the identity of the kernel (`/vmlinuz-3.2.0-38-generic`), the device representing the root (`/`) filesystem (`/dev/mapper/ubuntub-root`), and the location of the initial RAM disk (`/initrd.img-3.2.0-38-generic`).

Modifying the GRUB Boot Loader

Because the `grub.cfg` file is created from files in the `/etc/grub.d` directory, to change your grub configuration you can do one of the following instead of editing the file directly:

- **Custom file**—Add entries to the `/etc/grub.d/40_custom` file that you want included in your `grub.cfg` file.
- **Default file**—The `/etc/default/grub` file contains settings, such as how long GRUB should pause before starting the boot process (to give you time to interrupt

it).

Once you have changed Custom or Default files, you can make those changes take effect by either installing a new kernel or by running the `update-grub` command.

Other activities that didn't happen previously with GRUB Legacy now automatically take place with GRUB 2. For example, if GRUB detects a Windows system or other operating system on any of your disks, it will add an entry to the GRUB menu so that when you reboot, you have the option of selecting that operating system to boot from.

Controlling Startup and Run Levels

After the kernel has started up, it hands control of the system to the `init` process. The `init` process becomes the first running process on the system (PID 1), directing the startup of other processes based on the contents of the `/etc/init/` directory, the default run level, and the `init` scripts set to run at that run level.

The default run level is typically set to 2 for Ubuntu systems (based on the value of the `DEFAULT_RUNLEVEL` line in the `/etc/init/rc-sysinit.conf` file). That value can be overridden by adding the runlevel you want to the `env RUNLEVEL=` line in the `rc-sysinit.conf` file.

Ubuntu 12.04 LTS supports both the old style System V `init` scripts (stored in the `/etc/init.d/` directory) and the newer Upstart-style configuration files (stored in the `/etc/init` directory). As an administrator, you can change the default runlevel, change the current runlevel, or change which services start and stop at each runlevel.

Most Linux administrators leave the basic startup features alone and focus on which services are turned on or off at the selected run level. This section contains commands for working with system initialization scripts and changing run levels.

As root, you can use the `runlevel` command to **see the current run level**:

```
$ runlevel                Display current and previous run levels
N 2
```

Because the system in this example booted directly to run level 2, there is no previous run level (N). To **change the current run level**, you can use the `init` command. For example, from the console, if you want to shut down the GUI and network and then go into single-user mode for system maintenance, you can type the following:

```
$ sudo init 1             Change run level to 1 (single user mode)
```

In this example, the current run level changes from the previous level (in this case, 2) to run level 1 (which puts the system in single-user mode). To manage services, you can use the `update-rc.d` and `service` commands. For example, to **start the NTP service**

immediately, you can type this:

```
$ sudo /etc/init.d/ntp start           Start NTP service immediately
Starting NTP server ntpd                [ OK ]
```

Each System V service comes with a shell script in the `/etc/init.d` directory. You can pass `start` or `stop` options to each service. The format is as follows:

```
sudo /etc/init.d/service_to_control start
sudo /etc/init.d/service_to_control stop
```

Most of the scripts in the `/etc/init.d` directory support `start` and `stop` options, while some support other features as well. Here's how to use `service` to **start and stop services**:

```
$ /etc/init.d/ntp                     Show usage statement (no
options)
Usage: /etc/init.d/ntp {start|stop|restart|try-restart|
force-reload|status}
$ sudo /etc/init.d/ntp restart          Restart NTP service
* Stopping NTP server ntpd                [ OK ]
* Starting NTP server ntpd                [ OK ]
$ sudo /etc/init.d/ntp try-restart      Restart NTP if already running
* Stopping NTP server ntpd                [ OK ]
* Starting NTP server ntpd                [ OK ]
$ sudo /etc/init.d/ntp force-reload    Reload config file to daemon
* Stopping NTP server ntpd                [ OK ]
* Starting NTP server ntpd                [ OK ]
$ sudo /etc/init.d/ntp status          Check if the NTP is running
* NTP server is running.
$ sudo /etc/init.d/ntp stop            Stop NTP service
* Stopping NTP server ntpd                [ OK ]
```

Any of the `init` scripts contained in `/etc/init.d` can be started in this way, but not all scripts support all the features just shown. Most `init` scripts, however, will show their usage statement with no option (as shown in the first example in the preceding code).

Although the previous commands start the run level script service immediately, to have a service start automatically at boot time or during a run level change, you can use the `update-rc.d` command. In addition, most installation scripts for services will automatically turn a service on for the next time you boot. With `update-rc.d`, you can **turn services on, or turn them off** on a per-runlevel basis. For a service that is not yet configured to run at all, you can use the `defaults` option:

```
$ sudo update-rc.d ntp defaults        Turn on the NTP service
```

Although you can use the `init` command to change to any run level, including `init 0` (shut down) and `init 6` (reboot), there are also specific commands for stopping Linux.

The advantage of commands such as `halt`, `reboot`, `poweroff`, and `shutdown` is that they include options that enable you to **stop some features before shutdown occurs**. For example:

Warning Don't try the following commands if you don't intend to actually turn off your system, especially on a remote system.

\$ <code>sudo reboot</code>	Reboot the computer
\$ <code>sudo halt -n</code>	Don't sync hard drives before shutdown
\$ <code>sudo halt -h</code>	Put hard drives in standby before halting
\$ <code>sudo shutdown 10</code>	Shutdown in ten minutes after warning users
\$ <code>sudo shutdown -r 10</code>	Reboot ten minutes after warning users
\$ <code>sudo shutdown 10 'Bye!'</code>	Send 'Bye' to users before shutdown

Besides the `reboot` and `init 6` commands, you can also use the old PC keystrokes `Ctrl+Alt+Del` to reboot your computer.

Straight to the Kernel

In general, when the kernel starts up on your Linux system, you shouldn't have to do too much with it. However, there are tools for checking the kernel that is in use and for seeing information about how the kernel started up. Also, if something goes wrong or if there is some extra support you need to add to the kernel, there are tools to do those things.

To find out **what kernel is currently running on your system**, type the following:

```
$ uname -r          Display name of kernel release
3.2.0-38-generic
$ uname -a          Display all available kernel info
Linux ubuntutb 3.2.0-38-generic #61-Ubuntu SMP
Tue Feb 19 12:18:21 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux
```

When the kernel starts, messages about what occurs are placed in the kernel ring buffer. You can **display the contents of the kernel ring buffer** using the `dmesg` command:

```
$ dmesg |less
[    0.000000] Linux version 3.2.0-38-generic (buildd@akateko)
(gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5) ) #61-Ubuntu SMP
Tue Feb 19 12:18:21 UTC 2013 (Ubuntu 3.2.0-38.61-generic 3.2.37)
```

```
[    0.000000] Command line: BOOT_IMAGE=/vmlinuz-3.2.0-38-generic
root=/dev/mapper/ubuntutb-root ro
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Centaur CentaurHauls
...
```

If that buffer fills up, it may no longer contain the beginning of the recorded information. In that case, you can use `less /var/log/dmesg`.

You can find additional information about kernel processing in the `/var/log` files, in particular the `messages` file. You can page through those files as follows:

```
$ sudo less /var/log/messages      Page through messages file
May 23 21:36:05 ubuntutb kernel: [12567.769587] Inbound
IN=eth0 OUT= MAC=40:61:86:c3:35:f8:48:5b:39:e7:95:72:08:00
SRC=192.168.0.131 DST=192.168.0.141 LEN=60 TO
S=0x00 PREC=0x00
TTL=63 ID=58788 DF PROTO=TCP SPT=49304 DPT=22 WINDOW=14600
RES=0x00 SYN URGP=0
```

In the best circumstances, all the hardware connected to your computer should be detected and configured with the proper Linux drivers. In some cases, however, either the wrong driver is detected or the necessary driver may not be available on your system. For those cases, Linux offers ways of listing loadable kernel modules and adding new ones to your system.

The `lsmod` command enables you to **view the names of the loaded modules**, their size, and what other modules are using them. Consider the following example:

```
$ lsmod
Module                               Size  Used by
...
ext2                                 73795  1
dm_crypt                            23125  0
iptables_nat                        13229  0
nf_nat                              25891  2 ipt_MASQUERADE, iptable_nat
nf_conntrack_ipv4                   19716  9 iptable_nat, nf_nat
...
snd_hda_codec_realtek               224173  1
snd_hda_intel                       33773  3
snd_hda_codec                       127706  2 snd_hda_codec_realtek, snd_hda_intel
snd_hwdep                           17764  1 snd_hda_codec
lp                                  17799  0
parport                             46562  3 parport_pc, ppdev, lp
```

If you want to **find out more information about a particular module**, you can use the `modinfo` command. Here's an example:

```
$ modinfo parport_pc
filename:
    /lib/modules/3.2.0-38-
generic/kernel/drivers/parport/parport_pc.ko
license:      GPL
description:   PC-style parallel port driver
author:       Phil Blundell, Tim Waugh, others
...
```

If you decide you need to **add or remove a loadable module** to get some hardware item on your system working properly, you can use the `modprobe` command. You can also use `modprobe` to **list all available modules and remove modules**. Consider the following examples:

```
$ modprobe -l | grep c-qcam      Show c-qcam from module list
kernel/drivers/media/video/c-qcam.ko
$ sudo modprobe c-qcam          Load module for Color QuickCam
$ sudo modprobe -r c-qcam       Remove module for Color QuickCam
```

Note You may hear about the command `insmod` for loading modules. While `insmod` loads an individual module, `modprobe` loads any additional modules required to get the requested module working as well. Therefore, `modprobe` is the preferred command in most cases.

You can **control kernel parameters with the system running** using the `sysctl` command. You can also add parameters permanently to the `/etc/sysctl.conf` file, so they can load as a group or at each reboot. Consider the following examples:

```
$ sudo sysctl -a | less          List all kernel parameters
kernel.panic = 0
kernel.core_uses_pid = 0
...
$ sudo sysctl kernel.hostname    List value of particular parameter
$ sudo sysctl -p                 Load parms from /etc/sysctl.conf
$ sudo sysctl -w kernel.hostname=joe Set value of kernel.hostname
```

As noted earlier, if you want to change any of your kernel parameters permanently, you should add them to the `/etc/sysctl.conf` file. Parameter settings in that file are in the form `parameter = value`. Parameters set with `sysctl -w` will return to their default values the next time you reboot your system.

Poking at the Hardware

If you just generally want to find out more about your computer's hardware, you can use the following commands. The `lspci` command **lists information about PCI devices** on

your computer:

```
$ lspci                      List PCI hardware items
00:00.0 Host bridge: Intel Corporation 4 Series Chipset
      DRAM Controller (rev 03)
00:02.0 VGA compatible controller: Intel Corporation 4 Series
      Chipset Integrated Graphics Controller (rev 03)
00:02.1 Display controller: Intel Corporation 4 Series Chipset
      Integrated Graphics Controller (rev 03)
00:1b.0 Audio device: Intel Corporation N10/ICH 7 Family
      High Definition Audio Controller (rev 01)
...
$ lspci -v                  List PCI hardware items with more details
$ lspci -vv                 List PCI hardware items, even more details
```

Using the `dmidecode` command, you can **display information about your computer's hardware components**, including information about what features are supported in the BIOS. Consider the following example:

```
$ sudo dmidecode | less      List hardware components
SMBIOS 2.5 present.
41 structures occupying 1535 bytes.
Table at 0x0009F400Handle 0x0000, DMI type 0, 24 bytes.
BIOS Information
  Vendor: MSI
  Version: 130
  Release Date: 10/06/2013
...
Processor Information
  Socket Designation:
  SOCKET 775_M/B
    Type: Central Processor
    Family: Unknown
    Manufacturer: Intel
    ID: 7A 06 01 00 FF FB EB BF
    Version: Pentium(R) Dual-Core CPU           E6300   @ 2.80GHz
```

You can use the `hdparm` command to **view and change information relating to your hard disk**.

Warning Although it's safe to view information about features of your hard disks, it can potentially damage your hard disk to change some of those settings.

Here are some examples of printing information about your hard disks:

```
$ sudo hdparm /dev/sda      Display hard disk settings (SATA or SCSI)
/dev/sda:
multcount          = 16 (on)
```

```
IO_support      = 1 (32-bit)
readonly        = 0 (off)
readahead       = 256 (on)
geometry        = 19457/255/63, sectors = 312581808, start = 0
$ sudo hdparm /dev/hda      Display hard disk settings (IDE drive)
$ sudo hdparm -I /dev/sda    Display detailed drive information
/dev/sda:
ATA device, with non-removable media
    Model Number:    ST3160815AS
    Serial Number:   6RA5EGGJ
    Firmware Revision: 3.AAD
...
```

Summary

Ubuntu and other Linux systems make it easy for you to watch and modify many aspects of your running system to make sure it is operating at peak performance. Commands such as `free`, `top`, `vmstat`, `slabtop`, `iostat`, and `dstat` enable you to see how your system is using its memory, CPU, and storage devices. Using commands such as `date`, `hwclock`, and `cal`, as well as services such as NTP, you can watch and manage your system's date and time settings.

To manage the features that are set and services that come up when you boot your system, you can modify features associated with your GRUB boot loader and system run levels. You can start, stop, list, add, and remove individual system services using commands such as `service` and `update-rc.d`. Commands such as `reboot`, `halt`, and `shutdown` enable you to safely stop or reboot your computer.

When it comes to managing your computer's hardware, commands such as `lsmod`, `modinfo`, and `modprobe` enable you to work with loadable modules. You can view information about your hardware with such commands as `lspci`, `dmidecode`, and `hdparm`.

Chapter 11

Managing Network Connections

IN THIS CHAPTER

- Using `ethtool` and `mii-tool` to work with network interface cards
- Getting network statistics with `netstat`
- Starting network devices
- Viewing Ethernet information with `ifconfig` and `ip`
- Managing wireless cards with `iwconfig`
- Checking DNS name resolution with `dig`, `host`, and `hostname`
- Checking connectivity with `ping` and `arp`
- Tracing connections with `traceroute`, `route`, and `ip`
- Watching the network with `netstat`, `tcpdump`, and `nmap`

Connecting to a network from Linux is often as easy as turning on your computer. Your wired or wireless network interfaces should just start up and immediately let you connect to other computers on your local network or the Internet. However, if your network interface doesn't come up or requires some manual setup, there are many commands available for configuring network interfaces, checking network connections, and setting up special routing.

This chapter covers many useful commands for configuring and working with your network interface cards (NICs), such as `ethtool`, `mii-tool`, and `ifconfig`. More specifically, it covers ways of configuring wired and wireless Ethernet connections. With your hardware connected and network interfaces in place, the chapter describes commands such as `netstat`, `dig`, `ip`, and `ping` for getting information about your network.

Configuring Networks from the GUI

When you first install Ubuntu, the installer lets you configure any wired Ethernet cards attached to your computer with the use of a DHCP server detected on your network. The DHCP server can assign an IP address to your computer's network interface, as well as assign a default gateway (to access the Internet or other remote networks), DNS server

(for name to address resolution), and possibly a hostname.

Alternatively, you can set a static IP address, along with your hostname and IP addresses for your gateway machine and name servers. After installation, there are also graphical tools for configuring your network interfaces.

For Ubuntu desktop systems, the NetworkManager application manages your network interfaces. To change your network connections, either type **Network Connections** from the Dashboard or select the network icon (it looks like a pie slice) from the top bar. Then select Edit Connections from the menu. From the Network Connections window that appears, you can configure both wired and wireless network connections. Select the connection you are interested in and, if you choose, you can change the dynamic (DHCP) configuration to static IP addresses.

In some cases, however, your network interfaces may not be working, or you may want to work with your network interfaces in ways that are not supported from the GUI. For those cases, the following sections describe how to work with your network interfaces from the command line.

Managing Network Interface Cards

If the network hardware on your computer didn't immediately come up and let you connect to the Internet, there are some steps you should go through to troubleshoot the problem:

- For a wired NIC, verify that it is properly installed and that the cable is connected to your network (ISP's DSL, switch, and so on).
- After the cable is connected, make sure you have a link with no speed or duplex mismatches.
- Make sure that the cable is firmly seated in the NIC (it should click when it goes in). For wireless NICs, if there is no indication that the wireless card exists, but you know that your laptop has a wireless card, check for a small switch on the side of the laptop. On several occasions, I've seen that switch turned off by mistake, which prevents the wireless card from appearing at all on your list of available network interface cards.

To check your link from Linux, and to set speed and duplex, there are two commands you can use: the older `mii-tool` (`net-tools` package) and the newer `ethtool` (`ethtool` package). Use `ethtool` unless you have a very old NIC and NIC driver that are not compatible with the `ethtool` command.

To install the `ethtool` package and then **view the syntax of the `ethtool` command**, type the following:

```
$ sudo apt-get remove ethtool
```

```
$ ethtool -h | less           View options to the ethtool command
```

The `ethtool` command outputs its built-in help to `stderr`. To be able to page through that help with `less`, you redirect `stderr` to `stdout`.

To **display settings for a specific Ethernet card**, add the interface name to the command. For example, to view card information for `eth0`, type the following:

```
$ sudo ethtool eth0           See settings for NIC at eth0
```

```
Settings for eth0:
```

```
Supported ports: [ TP ]
```

```
Supported link modes:   10baseT/Half 10baseT/Full
                        100baseT/Half 100baseT/Full
                        1000baseT/Full
```

```
Supported pause frame use: No
```

```
Supports auto-negotiation: Yes
```

```
Advertised link modes:  10baseT/Half 10baseT/Full
                        100baseT/Half 100baseT/Full
                        1000baseT/Full
```

```
Advertised auto-negotiation: Yes
```

```
Speed: 100Mb/s
```

```
Duplex: Full
```

```
Port: Twisted Pair
```

```
PHYAD: 1
```

```
Transceiver: internal
```

```
Auto-negotiation: on
```

```
MDI-X: off
```

```
Supports Wake-on: plumbg
```

```
Wake-on: g
```

```
Current message level: 0x00000001 (1)
                        drv
```

```
Link detected: yes
```

You will need root permissions to acquire information about the Ethernet interface, hence the use of the `sudo` command in the previous example.

To **find out about the driver being used for a particular network card**, use the `-i` option:

```
$ sudo ethtool -i eth0       Display driver information for NIC
```

```
driver: e1000e
```

```
version: 1.5.1-k
```

```
firmware-version: 0.5-7
```

```
bus-info: 0000:01:00.0
```

Use the `-s` option to **display detailed statistics for a NIC**:

```
$ sudo ethtool -S eth0       Show statistics for NIC at eth0
```

```
NIC statistics:
```

```
rx_packets: 1326384
```

```
tx_packets: 773046
rx_bytes: 1109944723
tx_bytes: 432773480
rx_errors: 5
tx_errors: 2
rx_dropped: 0
tx_dropped: 0
multicast: 0
collisions: 0
rx_length_errors: 0
rx_over_errors: 0
rx_crc_errors: 5
rx_frame_errors: 0
rx_fifo_errors: 0
rx_missed_errors: 0
tx_aborted_errors: 0
tx_carrier_errors: 2
...
```

The `ethtool` command can be used to **change NIC settings** as well as display them. To turn off auto-negotiation and hard-set the NIC to 100 Mbps, full duplex, type this:

```
$ sudo ethtool -s eth0 speed 100 duplex full autoneg off  Change
NIC
```

To turn off auto-negotiation and hard-set the speed to 10 Mbps, half-duplex, type this:

```
$ sudo ethtool -s eth0 speed 10 duplex half autoneg off  Change
NIC
```

The changes just made to your NIC settings are good for the current session. When you reboot, however, those setting will be lost. To **make these settings stick at the next reboot or network restart**, you need to create a new script to be executed at boot time. The following steps describe how to do this.

1. Choose a name for your new script, such as `eth_options`, and then create the script in the `/etc/init.d` directory:

```
$ sudo vi /etc/init.d/eth_options
```

2. Insert the following text into this new script:

```
#!/bin/sh
ETHTOOL="/usr/sbin/ethtool"
ETHTOOL_OPTS="speed 10 duplex half autoneg off"
DEV="eth0"
case "$1" in
start)
    echo -n "Setting $DEV options to $ETHTOOL_OPTS...";
    $ETHTOOL -s $DEV $ETHTOOL_OPTS;
    echo " done.";;
stop)
;;
esac
exit 0
```

3. The specific settings you desire should be placed into the variable `ETHTOOL_OPTS`. For example:

```
ETHTOOL_OPTS="speed 10 duplex half autoneg off"
```

You can also change the `DEV` variable, which points to the first Ethernet interface, `eth0`.

4. Set up the script as an executable file:

```
$ sudo chmod +x /etc/init.d/eth_options
```

5. Set up the symbolic links to run your new script under the different runlevels:

```
$ sudo update-rc.d eth_options defaults  
Adding system startup for /etc/init.d/eth_options ...  
/etc/rc0.d/K20eth_options -> ../init.d/eth_options  
/etc/rc1.d/K20eth_options -> ../init.d/eth_options  
/etc/rc6.d/K20eth_options -> ../init.d/eth_options  
/etc/rc2.d/S20eth_options -> ../init.d/eth_options  
/etc/rc3.d/S20eth_options -> ../init.d/eth_options  
/etc/rc4.d/S20eth_options -> ../init.d/eth_options  
/etc/rc5.d/S20eth_options -> ../init.d/eth_options
```

You can run your script with the following command:

```
$ sudo /etc/init.d/eth_options start
```

Note You can find tips similar to this at the nixCraft site at www.cyberciti.biz/tips/.

As mentioned earlier, `ethtool` may not work on some older NICs. So if you have an older NIC, try using `mii-tool` as follows:

```
$ sudo mii-tool          Show negotiated speed, link status of old  
NIC  
eth0: negotiated 100baseTx-FD flow-control, link ok
```

This example was taken from the same machine as the preceding examples, with the NIC auto-negotiating at 1000 Mbps, full-duplex. The `mii-tool` command is misreading the speed setting. This is why I recommend using `mii-tool` only as a last resort if `ethtool` doesn't work with your old NIC.

To display the `mii-tool` output with more verbosity, use the `-v` option:

```
$ sudo mii-tool -v      Show verbose settings output for old NIC  
eth0: negotiated 100baseTx-FD flow-control, link ok  
product info: Yukon-EC 88E1111 rev 0  
basic mode:   autonegotiation enabled  
basic status: autonegotiation complete, link ok  
capabilities: 1000baseT-FD 100baseTx-FD 100baseTx-HD  
              10baseT-FD 10baseT-HD  
advertising:  100baseTx-FD 100baseTx-HD 10baseT-FD  
              10baseT-HD flow-control  
link partner: 1000baseT-FD 100baseTx-FD 100baseTx-HD
```

```
10baseT-FD 10baseT-HD flow-control
```

In the example just shown, you can see that each mode (100baseTx and 10baseT) supports both half-duplex (HD) and full duplex (FD). The 1000baseT, however, supports only full duplex. To **disable auto-negotiation and force a particular setting**, use the `-F` option as follows:

```
$ sudo mii-tool -F 10baseT-FD eth0    Force speed/duplex to 10baseT-  
FD
```

If you change your mind and later want to **re-enable auto-negotiation**, use the `-r` option:

```
$ sudo mii-tool -r eth0                Enable auto-negotiation for an old  
NIC  
restarting autonegotiation...
```

`mii-tool` does not provide a capability to save settings like `ethtool` does, so you have to run it after every reboot. This can be done by adding it at the end of `/etc/rc.local`.

The `netstat` command provides another way to **get network interface statistics**:

```
$ netstat -i                            Get network interface statistics for eth0  
Kernel Interface table  
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR  
Flg  
eth0 1500 0 1757208      6      0      0 996834      4      0      0  
BMRU
```

Use the `-c` option to get `netstat` to **refresh network interface statistics every second**:

```
$ netstat -ic                            Refresh network statistics every second
```

You can **get cleaner (screen-oriented) refreshed output** from `netstat` by combining it with the `watch` command as follows:

```
$ watch netstat -i                        Refresh network stats (screen oriented)  
Every 2.0s: netstat -i                        Wed May 29 01:55:48  
2013  
  
Kernel Interface table  
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR  
Flg  
eth0 1500 0 1757208      6      0      0 996834      4      0      0  
BMRU
```

As the output indicates, the `netstat` statistics are updated every 2.0 seconds.

Managing Network Connections

Starting and stopping the network interfaces for your wired Ethernet connections to your LAN or the Internet are usually handled automatically at the time you boot and shut down your Ubuntu system. However, you can use the commands in `/etc/init.d` to start and stop your network interfaces any time you want or `update-rc.d` to configure whether your network starts automatically.

The `ifconfig` and `ip` commands can also be used to configure, activate, and deactivate interfaces. However, on Ubuntu and other Debian derivatives, the commands in the `/etc/init.d` directory provide simpler tools to start and stop network interfaces. Therefore, in most cases, you should only use `ifconfig` and `ip` commands to gather information about your Ethernet interfaces and NICs (as shown later in this section).

Starting and Stopping Ethernet Connections

Your wired Ethernet interfaces just come up in many cases when you boot Ubuntu because the `network` service is set to be on when the system enters the common boot run levels. There is a set of underlying configuration files and scripts that make that happen and a few simple commands that enable you to control it.

For Ubuntu, control scripts and configuration files are located in the `/etc/network/` directory. NICs are configured by editing `/etc/network/interfaces`. The file looks like the following:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp

auto eth2
iface eth2 inet dhcp

auto ath0
iface ath0 inet dhcp

auto wlan0
iface wlan0 inet dhcp
```

To get more information on this file, type the following:

```
$ less /usr/share/doc/network-manager/README.Debian
```

If you change the interfaces file, and are using NetworkManager to manage your network interfaces, you need to run the following command:

```
$ sudo service network-manager restart
```

The script that starts the configured network-scripts files is /etc/init.d/network. As with other Linux services, you can start and stop the network service using the /etc/init.d/networking command.

To **take all NICs offline and then bring them back online**, allowing any change to the network scripts to take effect, type the following:

```
$ sudo /etc/init.d/networking restart    Restart network interfaces
* Reconfiguring network interfaces...
...
```

You may see errors for extra interfaces defined but not available on your system, such as wireless interfaces. You can ignore any error that refers to a networking device you have not installed.

Use the start and stop options to **start and stop your network interfaces**, respectively:

```
$ sudo /etc/init.d/networking stop        Shutdown network
interfaces
$ sudo /etc/init.d/networking start        Bring up network
interfaces
```

To **check the status of your network interfaces**, type the following:

```
$ ifconfig    Check network interface status
eth0          Link encap:Ethernet  HWaddr 00:19:D1:5A:A9:E2
    inet addr:192.168.1.106  Bcast:192.168.1.255  Mask:255.255.255.0
    inet6 addr: fe80::219:d1ff:fe5a:a9e2/64 Scope:Link
    UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
    RX packets:14442 errors:0 dropped:0 overruns:0 frame:0
    TX packets:13080 errors:0 dropped:0 overruns:0 carrier:0
    collisions:434 txqueuelen:1000
    RX bytes:3732823 (3.5 MiB)  TX bytes:1142020 (1.0 MiB)
    Interrupt:16 Memory:fe9e0000-fea00000

lo            Link encap:Local Loopback
    inet addr:127.0.0.1  Mask:255.0.0.0
    inet6 addr: ::1/128 Scope:Host
    UP LOOPBACK RUNNING  MTU:16436  Metric:1
    RX packets:35 errors:0 dropped:0 overruns:0 frame:0
    TX packets:35 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:0
    RX bytes:2121 (2.0 KiB)  TX bytes:2121 (2.0 KiB)
```

If you have multiple network interfaces, you may want to just **bring one interface up or down**. To do that, use the `ifup` and `ifdown` commands:

```
$ sudo ifdown eth0           Take the eth0 network interface offline
$ sudo ifup eth0             Bring the eth0 network interface online
```

When your network interfaces are up, there are tools you can use to view information about those interfaces and associated NICs.

Viewing Ethernet Connection Information

To view the media access control (MAC) address for your NIC and IP address for your TCP/IP connections, you can use the `ifconfig` command. The following command line **shows the address information and status of your eth0 Ethernet interface**:

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:D0:B7:79:A5:35
          inet addr:10.0.0.155  Bcast:10.0.0.255
Mask:255.255.255.0
          inet6 addr: fe80::2d0:b7ff:fe79:a535/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1413382 errors:6 dropped:0 overruns:0 frame:6
          TX packets:834839 errors:4 dropped:0 overruns:0 carrier:4
          collisions:0 txqueuelen:1000
          RX bytes:1141608691 (1.0 GiB)  TX bytes:470961026 (449.1
MiB)
```

In this example, the `eth0` interface is the first Ethernet interface on the computer. The MAC address (HWaddr) of the NIC is `00:D0:B7:79:A5:35`. You can see `eth0`'s IP address (`10.0.0.155`), broadcast address (`10.0.0.255`), and subnet mask (`255.255.255.0`). Other information includes the number of packets received and transmitted, as well as problems (errors, dropped packets, and overruns) that occurred on the interface.

To get information on both active and inactive NICs, use the `-a` option:

```
$ ifconfig -a
```

Instead of using `ifconfig` (and several other commands described in this chapter), you can use the newer `ip` command. The `ip` command was made to show information about your network interfaces, as well as to change settings for network devices, routing, and IP tunnels. Here, the `ip` command is used to **show information about the eth0 interface**:

```
$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP>
    mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:d0:b7:79:a5:35 brd ff:ff:ff:ff:ff:ff
```



```
inet 10.0.0.155/24 brd 10.0.0.255 scope global eth0
inet6 fe80::2d0:b7ff:fe79:a535/64 scope link
valid_lft forever preferred_lft forever
```

The `ip` command allows for shorthand syntax. If you're familiar with the Cisco IOS command line interface, the `ip` command works the same way. For example, instead of typing `ip addr show`, you could type the following to **see information on all interfaces**:

```
$ ip a
```

The `ip` command can operate on multiple network components, known as objects. One of these objects is `addr`, which allows `ip` to configure network addresses. I cover other objects of the `ip` command in the next examples.

To **see how the `ip` command is used**, use the `help` option. Along with the `help` option, you can identify an `ip` object to get information on using that object:

```
$ ip help          View ip usage statement
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] [-batch filename]
where  OBJECT := { link | addr | route | rule | neigh | ntable |
                  tunnel| tuntap | maddr | mroute | mrule |
                  monitor | xfrm | netns }
       OPTIONS := { -V[ersion] | -s[tatistics] | -r[esolve] |
                   -f[amily] { inet | inet6 | ipx | dnet | link } |
                   -l[oops] { maximum-addr-flush-attempts } |
                   -o[neline] | -t[imestamp] | -b[atch] [filename] |
                   -rc[vbuf] [size] }

$ ip addr help     View help for the addr object
$ ip route help    View help for the route object
$ ip tunnel help   View help for the tunnel object
```

Subnetwork masks can be confusing if you're not used to them. You may find `ipcalc` (from the `ipcalc` package) useful to **calculate a host computer's netmask from its CIDR IP address**:

```
$ ipcalc -bn 192.168.1.100/27
Address:    192.168.1.100
Netmask:    255.255.255.224 = 27
Wildcard:   0.0.0.31
=>
Network:    192.168.1.96/27
HostMin:    192.168.1.97
HostMax:    192.168.1.126
Broadcast:  192.168.1.127
Hosts/Net:  30                               Class C, Private Internet
```

In the example just shown, the netmask (which indicates which part of an IP address represents the network and which represents the host) is 255.255.255.224. That was

derived from the /27 value at the end of the IP address 192.168.1.100.

Using Wireless Connections

Setting up wireless connections in Linux has been tricky in the past, primarily because open source drivers were not available for many of the first wireless LAN cards. More recent releases of Ubuntu have shown a marked improvement.

For basic wireless configuration, I suggest you use the GUI tools (in particular, the Network Configuration window described earlier in this chapter, or Network Manager).

In rare cases, you may need to add wireless tools packages to get your wireless interfaces to work, such as wireless-tools and bcm43xx-fwcutter packages, which are available from the Ubuntu repositories. Likewise, you may need firmware that is available in the following packages: ipw2100-source, ipw2200-firmware, and zd1211-firmware.

If you are not able to configure your wireless LAN card using the Network Configuration window, you might be able to get your wireless card working using drivers and tools available from Atheros (www.atheros.com), the MadWifi (www.madwifi.org) project, or the Ndiswrapper project (ndiswrapper.sourceforge.net). Many packages of software from those projects are available from the standard Ubuntu repositories, described in Chapter 2.

If you need help **determining exactly what wireless card you have**, type the following:

```
$ dmesg | grep -i wireless          Search for wireless PCI cards
Intel(R) Wireless WiFi Link AGN driver for Linux, in-tree:
```

Assuming that your wireless card is up and running, there are some useful commands in the wireless-tools package you can use to view and change settings for your wireless cards. In particular, the `iwconfig` command can help you work with your wireless LAN interfaces. The following **scans your network interfaces for supported wireless cards** and lists their current settings:

```
$ iwconfig
eth0      no wireless extensions.
eth1      IEEE 802.11-DS  ESSID:"Mylan"
          Mode:Managed Frequency:2.437 GHz Access Point:
          43:5A:29:E7:95:75
          Bit Rate:54 Mb/s   Tx-Power=15 dBm
          Retry long limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
```

Wireless interfaces may be named wlanX or ethX, depending on the hardware and

driver used. You may be able to obtain more information after setting the link up on the wireless interface:

```
$ ip link set eth1 up
$ iwconfig eth1
eth1      IEEE 802.11abgn  ESSID:"Mylan"
        Mode:Managed Frequency:2.437 GHz Access Point: 43:5A:29:E7:95:7
        Bit Rate:54 Mb/s   Tx-Power=15 dBm
        Retry long limit:7   RTS thr:off   Fragment thr:off
        Power Management:off
        Link Quality=70/70  Signal level=-39 dBm
        Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
        Tx excessive retries:0  Invalid misc:0  Missed beacon:0
```

The settings just shown can be modified in a number of ways. Here are some ways to **use iwconfig to modify your wireless interface settings**. The following examples operate on a wireless interface named wlan0. These operations may or may not be supported, depending on which wireless card and driver you are using.

```
$ sudo iwconfig wlan0 essid "MyWireless"    Set essid to MyWireless
$ sudo iwconfig wlan0 channel 3             Set the channel to 3
$ sudo iwconfig wlan0 mode Ad-Hoc           Change Managed to Ad-Hoc
mode
$ sudo iwconfig wlan0 ap any                 Use any access point
$ sudo iwconfig wlan0 sens -50              Set sensitivity to -50
$ sudo iwconfig wlan0 retry 20              Set MAC retransmissions
to 20
$ sudo iwconfig wlan0 key 1234-5555-66      Set key to 1234-5555-66
```

The `essid` is sometimes called the Network Name or Domain ID. You use it as the common name that identifies your wireless network. Setting the `channel` lets your wireless LAN operate on that specific channel.

With `Ad-Hoc` mode, the network is composed of only interconnected clients with no central access point. In `Managed/Infrastructure` mode, by setting `ap` to a specific MAC address, you can force the card to connect to the access point at that address, or you can set `ap` to `any` and allow connections to any access point.

If you have performance problems, try adjusting the sensitivity (`sens`) to either a negative value (which represents dBm) or positive value (which is either a percentage or a sensitivity value set by the vendor). If you get retransmission failures, you can increase the `retry` value so your card can send more packets before failing.

You use the `key` option to set an encryption key. You can enter hexadecimal digits (XXXX-XXXX-XXXX-XXXX or XXXXXXXXX). By adding an `s:` in front of the key, you can enter an ASCII string as the key (as in `s:My927pwd`).

Checking Name Resolution

Because IP addresses are numbers, and people prefer to address things by name, TCP/IP networks (such as the Internet) rely on DNS to resolve hostnames into IP addresses. Ubuntu provides several tools for looking up information related to DNS name resolution.

When you first installed Ubuntu, you either identified Domain Name System (DNS) servers to do name resolution or had them assigned automatically from a DHCP server. That information is then stored in the `/etc/resolv.conf` file, looking something like the following:

```
domain example.com
search example.com example.net
nameserver 11.22.33.44
nameserver 22.33.44.55
```

If present, the `domain` line identifies the local domain. This allows you to identify a machine by its base name and the DNS lookup assumes you mean the local domain. So, if you request a host named `abc`, your system will look up abc.example.com. The `search` line lets you identify several domains to be searched.

The numbers just shown in the `/etc/resolv.conf` file are replaced by real IP addresses of computers that serve as DNS name servers. When you can connect to working DNS servers, there are commands you can use to query those servers and look up host computers.

The `dig` command (which should be used instead of the deprecated `nslookup` command) can be used to look up information from a DNS server. The `host` command can be used to look up address information for a hostname or domain name.

To **search your DNS servers for a particular hostname** (www.turbosphere.com in the following examples), use the `dig` command as follows:

```
$ dig www.turbosphere.com    Search DNS servers in /etc/resolv.conf
```

Instead of using your assigned name server, you can **query a specific name server**. The following example queries the DNS server at `4.2.2.1`:

```
$ dig www.turbosphere.com @4.2.2.1
```

Using `dig`, you can also **query for a specific record type**:

```
$ dig turbosphere.com mx    Queries for the mail exchanger
$ dig turbosphere.com ns    Queries for the authoritative name
servers
```

Use the `+trace` option to **trace a recursive query** from the top-level DNS servers down to the authoritative servers:

```
$ dig +trace www.turbosphere.com    Recursively trace DNS servers
```

If you just want to **see the IP address of a host computer**, use the `+short` option:

```
$ dig +short www.turbosphere.com    Display only name/IP address  
pair  
turbosphere.com.  
66.113.99.70
```

You can use `dig` to **do a reverse lookup to find DNS information based on an IP address**:

```
$ dig -x 66.113.99.70                Get DNS information based on IP  
address
```

You can use `host` to **do a reverse DNS lookup** as well:

```
$ host 66.113.99.70  
70.99.133.66.in-addr.arpa domain name pointer  
boost.turbosphere.com.
```

To **get hostname information for the local machine**, use the `hostname` and `dnsdomainname` commands:

```
$ hostname                            View the local computer's full DNS host  
name  
boost.turbosphere.com
```

You can also use `hostname` to **set the local hostname temporarily** (until the next reboot). Here's an example:

```
$ sudo hostname server1.example.com    Set local hostname
```

Changing the hostname of a running machine may adversely affect some running daemons. Instead, I recommend you **set the local hostname so it is set each time the system starts up**. Edit the first line in the `/etc/hostname` file. Here is an example:

```
server1.example.com
```

Troubleshooting Network Problems

Troubleshooting networks is generally done from the bottom up. As discussed at the beginning of the chapter, the first step is to make sure that the physical network layer components (cables, NICs, and so on) are connected and working. Next, check that the links between physical nodes are working. After that, there are lots of tools for checking the connectivity to a particular host.

Checking Connectivity to a Host

When you know you have a link and no duplex mismatch, the next step is to ping your default gateway. You should have either configured the default gateway (gw) in the `/etc/network/interfaces` file or let the system set up the default gateway from a service such as DHCP. To **check your default gateway in the actual routing table**, use the `ip` command as follows:

```
$ ip route
10.0.0.0/24 dev eth0 proto kernel scope link src 10.0.0.155
169.254.0.0/16 dev eth0 scope link
default via 10.0.0.1 dev eth0
```

The gateway for the default route in this example is 10.0.0.1. To **make sure there is IP connectivity to that gateway**, use the `ping` command as follows, passing the address for your default gateway:

```
$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.382 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.313 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.360 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=1.43 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.313/0.621/1.432/0.469 ms
```

By default, `ping` continues until you press `Ctrl+c`. Other `ping` options include the following:

<code>\$ ping -a 10.0.0.1</code>	Add an audible ping as ping progresses
<code>\$ ping -c 4 10.0.0.1</code>	Ping 4 times and exit (default in Windows)
<code>\$ ping -q -c 5 10.0.0.1</code>	Show summary of pings (works best with -c)
<code>\$ sudo ping -f 10.0.0.1</code>	Send a flood of pings (must be root)
<code>\$ ping -i 3 10.0.0.1</code>	Send packets in 3-second intervals
<code>\$ sudo ping -I eth0 10.0.0.1</code>	Set source to eth0 (use if multiple NICs)
PING 10.0.0.1 (10.0.0.1) from 10.0.0.155 eth0: 56(84) bytes of data.	
<code>\$ sudo ping -I 10.0.0.155 10.0.0.1</code>	Set source to 10.0.0.155
PING 10.0.0.1 (10.0.0.1) from 10.0.0.155 : 56(84) bytes of data.	
<code>\$ ping -s 1500 10.0.0.1</code>	Set packet size to 1500 bytes
PING 10.0.0.1 (10.0.0.1) 1500(1528) bytes of data.	

Use the `pingflood` option with caution. By default, `ping` sends small packets (56

bytes). Large packets (such as the 1500-byte setting just shown) are good to make faulty NICs or connections stand out.

Checking Address Resolution Protocol

If you're not able to `ping` your gateway, you may have an issue at the Ethernet MAC layer. The Address Resolution Protocol (ARP) can be used to find information at the MAC layer. To view and configure ARP entries, use the `arp` or `ip neighbor` command. This example shows `arp` listing computers in the ARP cache by hostname:

```
$ arp -v                                List ARP cache entries by name
Address      HWtype  HWaddress      Flags Mask    Iface
ritchie      ether    00:10:5A:AB:F6:A7  C             eth0
einstein     ether    00:0B:6A:02:EC:98  C             eth0
Entries: 1   Skipped: 0   Found: 1
```

In this example, you can see the names of other computers that the local computer's ARP cache knows about and the associated hardware type and hardware address (MAC address) of each computer's NIC. You can **disable name resolution to see those computers' IP addresses** instead:

```
$ arp -vn                                List ARP cache entries by IP address
Address      HWtype  HWaddress      Flags Mask    Iface
10.0.0.1     ether    00:10:5A:AB:F6:A7  C             eth0
10.0.0.50    ether    00:0B:6A:02:EC:98  C             eth0
Entries: 1   Skipped: 0   Found: 1
```

To delete an entry from the ARP cache, use the `-d` option:

```
$ sudo arp -d 10.0.0.50                  Delete address 10.0.0.50 from ARP
cache
```

Instead of just letting ARP dynamically learn about other systems, you can **add static ARP entries to the cache** using the `-s` option:

```
$ sudo arp -s 10.0.0.51 00:0B:6A:02:EC:95 Add IP/MAC addresses to
ARP
```

To do the same actions with the `ip` command that you just did with the `arp` command, use the `neighbor` object (note that `neighbor`, `nei`, and `n` objects can be used interchangeably):

```
$ ip neighbor
10.0.0.1 dev eth0 lladdr 00:10:5a:ab:f6:a7 DELAY
10.0.0.50 dev eth0 lladdr 00:0b:6a:02:ec:98 REACHABLE
# ip nei del 10.0.0.50 dev eth0
```

```
# ip netns exec n add 10.0.0.51 lladdr 00:0B:6A:02:EC:95 dev eth0
```

To **query a subnet to see if an IP is already in use**, and to find the MAC address of the device using it, use the `arping` command. The `arping` command is used by `ifup` to avoid IP conflicts when bringing an Ethernet NIC up. Here are examples:

```
$ arping 10.0.0.50          Query subnet to see if 10.0.0.50
is in use
ARPING 10.0.0.50 from 10.0.0.195 eth0
Unicast reply from 10.0.0.50 [00:0B:6A:02:EC:98]  0.694ms
Unicast reply from 10.0.0.50 [00:0B:6A:02:EC:98]  0.683ms
$ sudo arping -I eth0 10.0.0.50 Specify interface to query from
```

Like the `ping` command, the `arping` command (from the `iputils-arping` package) continuously queries for the address until the command is ended when you type `Ctrl+c`. Typically, you just want to know if the target is alive, so you can run one of the following commands:

```
$ arping -f 10.0.0.50      Query 10.0.0.50 and stop at the first
reply
$ arping -c 2 10.0.0.51    Query 10.0.0.50 and stop after 2 counts
```

Tracing Routes to Hosts

After verifying that you can `ping` your gateway and even reach machines that are outside of your network, you may still have issues reaching a specific host or network. If that's true, you can **use `traceroute` (from the `traceroute` package) to find the bottleneck or point of failure**:

```
$ traceroute boost.turbosphere.com Follow the route taken to a
host
traceroute to boost.turbosphere.com (66.113.99.70),
 30 hops max, 40 byte packets
 1  10.0.0.1 (10.0.0.1)  0.281 ms  0.289 ms  0.237 ms
 2  tl-03.hbci.com (64.211.114.1)  6.213 ms  6.189 ms  6.083 ms
 3  172.17.2.153 (172.17.2.153)  14.070 ms  14.025 ms  13.974 ms
 4  so-0-3-2.ar2.MIN1.gblx.net (208.48.1.117)  19 ms  19 ms  19 ms
 5  sol-0-0-2488M.ar4.SEA1.gblx.net (67.17.71.210) 94.6 ms 94.6 ms
94.6ms
 6  64.215.31.114 (64.215.31.114)  99.643 ms  101.647 ms  101.577
ms
 7  dr02-v109.tac.opticfusion.net (209.147.112.50) 262.301ms
233.316ms 233.153 ms
 8  dr01-v100.tac.opticfusion.net (66.113.96.1) 99.3 ms 99.4 ms
99.3 ms
 9  boost.turbosphere.com (66.113.99.70) 99.25 ms 96.21 ms
100.22 ms
```


As you can see, the longest hop is between 4 (Global Crossing probably in Minneapolis) and 5 (GC in Seattle). That gap is not really a bottleneck; it just reflects the distance between those hops. Sometimes, the last hops look like this:

```
28  * * *
29  * * *
30  * * *
```

The lines of asterisks (*) at the end of the trace can be caused by firewalls that block traffic to the target. However, if you see several asterisks before the destination, those can indicate heavy congestion or equipment failures and point to a bottleneck.

By default, `traceroute` uses UDP packets, which provide a more realistic performance picture than ICMP. That's because some Internet hops will give lower priority to ICMP traffic. If you'd still like to **trace using ICMP packets**, try the following command:

```
$ traceroute -I boost.turbosphere.com  Use ICMP packets to trace a
route
```

By default, `traceroute` connects to port 80. You can **set a different port** using the `-p` option:

```
$ traceroute -p 25 boost.turbosphere.com  Connect to port 25 in
trace
```

You can **view IP addresses instead of hostnames** by disabling name resolution of hops:

```
$ traceroute -n boost.turbosphere.com  Disable name resolution in
trace
```

An alternative to `traceroute` is the `tracepath` command, which also uses UDP to perform the trace:

```
$ tracepath boost.turbosphere.com      Use UDP to trace the route
```

To view and manipulate the kernel's routing table, the `route` command used to be the tool of choice. This is slowly being replaced by the `ip route` command. For the most part, the Ubuntu network scripts rely on `ip route`. But it doesn't hurt to be familiar with both commands because `route` is still quite commonly used.

You can use the old `route` command to **display your local routing table**. Here are two examples of the `route` command, with and without DNS name resolution:

```
$ route          Display local routing table information
Kernel IP routing table
Destination    Gateway        Genmask         Flags Metric Ref  Use  Iface
10.0.0.0       *              255.255.255.0   U      0      0    0  eth0
default        ritchie        0.0.0.0         UG     0      0    0  eth0
$ route -n       Display routing table without DNS lookup
```

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.0.0.0	*	255.255.255.0	U	0	0	0	eth0
0.0.0.0	10.0.0.1	0.0.0.0	UG	0	0	0	eth0

You can **add a default gateway** using the `gw` option:

```
$ sudo route add default gw 10.0.0.2      Add 10.0.0.2 as default gateway
```

You can **add a new route to your network** by specifying either the interface (`eth0`) or IP address of the gateway (such as `gw 10.0.0.100`):

```
$ sudo route add -net 192.168.0.0 netmask 255.255.255.0 eth0  
$ sudo route add -net 192.168.0.0 netmask 255.255.255.0 gw 10.0.0.100
```

You can **delete a route** using the `del` option:

```
$ sudo route del -net 192.168.0.0 netmask 255.255.255.0      Delete a route
```

Using the newer `ip` command, you can do the same activities just shown with the `route` command. Here are three different ways to show the same basic routing information:

```
$ ip route show          Display basic routing information  
10.0.0.0/24 dev eth0  proto kernel  scope link    src 10.0.0.195  
169.254.0.0/16 dev eth0  scope link  
default via 10.0.0.1 dev eth0  
$ ip route              Display basic routing (example #2)  
$ ip r                  Display basic routing (example #3)
```

The following are some examples of **adding and deleting routes with `ip`**:

```
$ sudo ip r add 192.168.0.0/24 via 10.0.0.100 dev eth0 Add route to eth0  
$ sudo ip r add 192.168.0.0/24 via 10.0.0.100      Add route no interface  
$ sudo ip r del 192.168.0.0/24                    Delete route
```

To **make a new route permanent**, edit the `/etc/network/interfaces` file and place the information about the new route in that file. For example, to add the route added with the preceding `ip` command, add the following lines to `/etc/network/interfaces`:

```
iface eth0 inet static  
address 192.168.0.0  
netmask 255.255.255.0  
gateway 10.0.0.100
```

Displaying netstat Connections and Statistics

The tools shown in the preceding sections cover network troubleshooting mostly at the network layer (layer 3). To **display information about packets sent between transport-layer protocols (TCP and UDP), and ICMP**, you can use the `netstat` command:

```
$ netstat -s | less           Show summary of TCP, ICMP, UDP activities
```

You can see a **list of all TCP connections**, including which process is handling the connection:

```
$ sudo netstat -tanp           View active TCP connections
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program
name
tcp      0      0 127.0.0.1:631 0.0.0.0:*      LISTEN 2039/cupsd
tcp      0      0 127.0.0.1:25  0.0.0.0:*      LISTEN
2088/sendmail
...
```

You can also **view active UDP connections** as follows:

```
$ sudo netstat -uanp           View active UDP connections
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program
name
udp      0      0 0.0.0.0:631  0.0.0.0:*      2039/cupsd
udp      0      0 192.168.122.1:123 0.0.0.0:*      2067/ntpd
...
```

To **narrow your output from netstat to daemons bound to a TCP port**, look for the word `listen`. For example:

```
$ sudo netstat -tanp | grep -i listen       View daemons on a port
```

The command just shown is a great way to resolve port usage conflicts between daemons.

Other Useful Network Tools

If you'd like to **see header information about packets** as they are sent and received by your system, use `tcpdump`. The `tcpdump` command has a lot of advanced features, most of which revolve around filtering and finding a needle in a haystack of packets. If you run `tcpdump` on a remote machine, your screen will be flooded with all the SSH traffic between your client and the remote machine. To get started without having to learn too much about how `tcpdump` filtering works, run the following command:

```
$ sudo tcpdump | grep -v ssh      Find packets not associated with  
ssh
```

If you'd like to **dig deeper into packet-level traffic**, use `wireshark` (formerly known as `ethereal`). Install the `wireshark` package. You can run `wireshark` with X over SSH on a remote machine. Wireshark is a very powerful packet sniffer that rivals the best commercial tools.

To **explore networks and remote machines and see what services they offer**, use `nmap`. The `nmap` command (from the `nmap` package) is the most common port scanner. It was even featured in the movie *The Matrix Reloaded*! Make sure that you are explicitly authorized to scan the systems or networks you are scanning. The `nmap` command is part of the `nmap` package and can be run as a user, but several scan types require root privileges.

Here's how to **do a basic host scan** with `nmap`:

```
$ sudo nmap 10.0.0.1              Scan ports on computer at 10.0.0.1
```

To **get maximum verbosity** from `nmap`, use the `-vv` option:

```
$ sudo nmap -vv 10.0.0.1          Show maximum verbosity from nmap  
output
```

To use `nmap` to **scan an entire network**, use the network address as an argument. In the following example, you can add the `-sP` option to tell `nmap` to perform a simple ping sweep:

```
$ sudo nmap -vv -sP 10.0.0.0/24    Scan hosts on an entire network
```

You can be very specific about the information that `nmap` gathers for you. In the following example, the `-P0` option tells `nmap` not to use ping (this is good for scanning machines that don't respond to ping). The `-O` option displays OS fingerprinting for the machine you are scanning. The `-p 100-200` option tells `nmap` to scan only ports 100 through 200:

```
$ sudo nmap -vv -P0 -O -p 100-200 10.0.0.1  No ping, OS fp, ports  
100-200
```

The `nmap` command has many more options for advanced usage. Refer to the `nmap` man page (`man nmap`) for further information.

Summary

Nearly every aspect of the network connections from your Ubuntu system can be configured, checked, and monitored using command-line tools. You can view and change settings of your NICs using `ethtool` and `mii-tool` commands. You can view

network statistics with `netstat`.

To start and stop your network, commands such as `service`, `ifup`, and `ifdown` are easy to manage. When a connection is established, you can see statistics about that connection using `ifconfig` and `ip` commands.

Besides using wired Ethernet cards, other network hardware such as wireless LAN cards are supported in Linux. Use commands such as `iwconfig` to work with wireless interfaces.

To check DNS name resolution, use the `dig`, `host`, and `hostname` commands. Commands for checking connectivity and routes to a host include `ping`, `arp`, `traceroute`, and `ip`.

Chapter 12

Accessing Network Resources

IN THIS CHAPTER

- Web browsing with `elinks`
- Transferring files with `wget`, `curl`, `lftp`, and `scp`
- Sharing directories with NFS, Samba, and SSHFS
- Chatting with `irssi` IRC client
- Managing email with `mutt` and `mail`

In the time it takes to fire up a graphical FTP client, you could already have downloaded a few dozen files from a remote server using command line tools. Even when a GUI is available, commands for transferring files, web browsing, sharing directories, and reading mail can be quick and efficient to use. When no GUI is available, they can be lifesavers.

This chapter covers commands for accessing resources (files, e-mail, shared directories, and online chats) over the network.

Running Commands to Browse the Web

Text-mode web browsers provide a quick way to check that a web server is working or to get information from a web server when a usable GUI isn't available. The once-popular `lynx` text-based browser was supplanted in most Linux systems by the `links` or `elinks` browsers.

To use a command line browser, you need to install one of these programs, with package names shown in parens: `lynx` (`lynx-cur` package), `links` (`links` package), and `elinks` (`elinks` package). In most cases, if you want a command line web browser, install the `elinks` package.

The `elinks` browser runs in a terminal window. Aside from not displaying images in the terminal, `elinks` can handle most basic HTML content and features: tables, frames, tabbed browsing, cookies, history, MIME types, and simple cascading style sheets. You can even use your mouse to follow links and select menu items.

Because `elinks` supports multiple colors, as long as the terminal you are using

supports multiple colors, it's easy to spot links and headings in the text. (Colors may not work within a `screen` session.) Here are some examples of `elinks` command lines:

<code>\$ elinks</code>	Prompts for file name or URL
<code>\$ elinks www.handsonhistory.com</code>	Opens file name or URL you
request	

If you have a mouse available, click near the top of the terminal window to see the menu. Select the menu name or item you want. Select a link to go to that link. The following list shows `elinks` keyboard navigation keys.

- **Esc (or F9/F8)**—Toggle the menu on and off (then use arrow keys or mouse to navigate menus).
- **Down arrow**— Go to the next link or editable field on the page.
- **Up arrow**— Go to the previous link or editable field on the page.
- **Right arrow or Enter**— Go forward to the highlighted link. Enter text in the highlighted form field.
- **Left arrow**— Go back to the previous page.
- **/**— Search forward.
- **?**— Search backwards.
- **n**— Find next.
- **N**— Find previous.
- **PageUp**—Scroll one page up.
- **PageDown**— Scroll one page down.
- **g**— Go to a URL.
- **q or Ctrl+c**— Exit `elinks`.
- **=**— View page information.
- **Ctrl+r**— Reload page.
- **a**—Bookmark the current page.
- **t**—Open a new browser tab.
- **>**—Go to the next tab.
- **<**—Go to the previous tab.
- **c**—Close the current tab.
- **d**—Download the current link.
- **D**—View downloads.
- **A**—Add the current link to bookmarks.
- **s**—View bookmarks.
- **v**—View the current image.
- **h**—View the global history manager.

You can add global settings for `elinks` to `/etc/elinks.conf`. Per-user settings are stored in each user's `$HOME/.elinks` directory. Type **man elinkskeys** to see available

settings.

Transferring Files

Commands in Linux for downloading files from remote servers (HTTP, HTTPS, FTP, or SSH) are plentiful and powerful. You might choose one command over another because of the specific options you need. For example, you may want to perform a download over an encrypted connection, resume an aborted download, or do recursive downloads. This section describes how to use `wget`, `ftp`, `lftp`, `scp`, and `sftp`.

Downloading Files with `wget`

Sometimes you need to download a file from a remote server using the command line. For example, you find a link to an RPM software package, but the link goes through several HTTP redirects that prevent `rpm` from installing straight from HTTP. Or you may want to script the automated download of a file, such as a log file, every night.

The `wget` command can download files from web servers (HTTP and HTTPS) and FTP servers. With a server that doesn't require authentication, a `wget` command can be as simple as the `wget` command and the location of the download file:

```
$ wget http://design.ubuntu.com/wp-content/uploads/ubuntu-  
logo14.png
```

If, for example, an **FTP server requires a login and password**, you can enter that information on the `wget` command line in the following forms:

```
$ wget ftp://user:password@ftp.example.com/path/to/file  
$ wget --user=usr --password=passwd  
ftp://ftp.example.com/pathtofile
```

For example:

```
$ wget ftp://chris:mykuulpwd@ftp.linuxtoys.net/home/chris/image.jpg  
$ wget --user=chris --password=mykuulpwd \  
ftp://ftp.linuxtoys.net/home/chris/image.jpg
```

You can use `wget` to **download a single web page as follows**:

```
$ wget http://www.wiley.com Download only the Web page
```

If you open the resulting `index.html` file, you'll see links to images that are gathered from the website but would have all sorts of broken links if you didn't have an Internet connection. To download all the images and other elements required to render the page properly (if the site allows it, which it doesn't in this case), use the `-p` option:

```
$ wget -p http://www.wiley.com Download Web page and other
```


elements

But if you open the resulting `index.html` file in your browser, chances are you will still have all the broken links even though all the images were downloaded. That's because the links need to be translated to point to your local files. So instead, do this:

```
$ wget -pk http://www.wiley.com      Download pages, use local file
names
```

And if you'd like `wget` to keep the original file and also do the translation, type this:

```
$ wget -pkK http://www.wiley.com      Rename to local names, keep
original
```

Sometimes an HTML file you download does not have a `.html` extension but ends in `.asp` or `.cgi` instead. That may result in your browser not knowing how to open your local copy of the file. You can have `wget` append `.html` to those files using the `-E` option:

```
$ wget -E http://cgi-app.org/index.cgi?Examples      Add .html to
files
```

With the `wget` command, you can recursively mirror an entire website. While copying files and directories for the entire depth of the server's file structure, the `-m` option adds timestamping and keeps FTP directory listings (provided that the server allows it). (Use this with caution because it can take a lot of time and space.)

```
$ wget -m http://www.linuxtoys.net
```

Using some of the options just described, the following command line results in the most usable local copy of a website:

```
$ wget -mEkK http://www.linuxtoys.net
```

If you have ever had a large file download (such as a CD or DVD image file) disconnect before it completed, you may find the `-c` option to `wget` to be a lifesaver. Using `-c`, `wget` resumes where it left off, **continuing an interrupted file download**. For example:

```
$ wget http://example.com/DVD.iso      Begin downloading large file
...
95%[=====  ] 685,251,583 55K/s      Download killed before
completion
$ wget -c http://example.com/DVD.iso    Resume download where stopped
...
HTTP request sent, awaiting response... 206 Partial Content
Length: 699,389,952 (667), 691,513 (66M) remaining [text/plain]
```

Because of the continue feature (`-c`), `wget` can be particularly useful for those with slow Internet connections who need to download large files. If you have ever had a several-hour download get killed just before it finished, you'll know what I mean. (Note

that if you don't use the `-c` when you mean to resume a file download, the file will be saved to a different file: the original name with a `.1` appended to it.)

Transferring Files with cURL

The client for URLs application (`curl` command) provides similar features to `wget` for transferring files using web and FTP protocols. However, the `curl` command can also transfer files using other popular protocols, including SSH protocols (SCP and SFTP), LDAP, DICT, Telnet, and File.

Instead of supporting large, recursive downloads (as `wget` does), `curl` is designed for **single-shot file transfers**. It does, however, support more protocols (as noted) and some neat advanced features. To use this command, you need to install the `curl` package. Here are a few interesting examples of **file transfers with curl**:

```
$ curl -O ftp://kernel.org/pub/linux/kernel/v1.0/patch[6-8].sign
$ curl -OO ftp://kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.
{1,4}
$ curl -O ftp://chris:MyPasswd@ftp.example.com/home/chris/fileA \
  -O '-DELETE fileA'
$ curl -T install.log ftp://chris:MyPasswd@ftp.example.com/tmp/ \
  -O "-RNFR install.log" -O "-RNTO Xinstall.log"
$ curl ftp://ftp.kernel.org/pub//                List /pub
contents
```

The first two commands show how to use square brackets to indicate a range `[6-8]` and curly brackets for a list `{1,4}` of characters or numbers that are used to match files.

The third command line illustrates how to add a username and password (`chris:MyPasswd`), download a file (`fileA`) from the server, and then delete the file on the server once the download is done (`-O '-DELETE fileA'`).

The fourth example uploads (`-T`) the file `install.log` to an FTP server. Then it renames the remote file to `Xinstall.log`. The last example tells `curl` to list the contents of the `/pub/` directory at `ftp.kernel.org`.

Transferring Files with FTP Commands

Ubuntu comes with the standard FTP client (`ftp` command), that works the same way it does on most UNIX and Windows systems. I recommend you use the full-featured, user-friendly `lftp` instead.

With these FTP clients, you open a session to the FTP server (as opposed to just grabbing a file, as you do with `wget` and `curl`). Then you navigate the server much as you would a local filesystem, getting and putting documents across the network

connection.

To get the `lftp` command, type **apt-get install lftp**. Here are examples of how to **connect to an FTP server with lftp**:

\$ <code>lftp mirrors.kernel.org</code>	Anonymous connection
<code>lftp mirrors.kernel.org:~></code>	
\$ <code>lftp chris@example.com</code>	Authenticated connection
<code>lftp example.com:~></code>	
\$ <code>lftp -u chris example.com</code>	Authenticated connection
Password: <code>*****</code>	
<code>lftp example.com:~></code>	
\$ <code>lftp -u chris,Mypwd example.com</code>	Authentication with
password	
<code>lftp example.com:~></code>	
\$ <code>lftp</code>	Start lftp with no
connection	
<code>lftp :~> open mirrors.kernel.org</code>	Start connection in lftp
session	
<code>lftp mirrors.kernel.org:~></code>	

Warning The fourth example should be avoided in real life. Passwords that are entered in a command line end up stored in clear text in your `~/.bash_history`. They may also be visible to other users in the output of `ps auxx`.

When a connection is established to an FTP server, you can use a set of commands during the FTP session. FTP commands are similar to shell commands. Just like in a bash shell, you can press Tab to autocomplete filenames. In a session, `lftp` also supports sending multiple jobs to the background (Ctrl+z) and returning them to the foreground (`wait` or `fg`). These are useful if you want to continue traversing the FTP site while files are downloading or uploading. Background jobs run in parallel. Type **jobs** to see a list of running background jobs. Type **help** to see a list of `lftp` commands.

The following sample `lftp` session illustrates **useful commands when downloading**:

\$ <code>lftp mirrors.kernel.org</code>	
<code>lftp mirrors.kernel.org:~> pwd</code>	Check current
directory	
<code>ftp://mirrors.kernel.org</code>	
<code>lftp mirrors.kernel.org:~> ls</code>	List current
directory	
<code>drwxr-xr-x 8 ftp ftp 4096 Mar 08 21:02 debian</code>	
<code>...drwxr-xr-x 6 ftp ftp 4096 Mar 09 00:11 ubuntu</code>	
...	
<code>lftp mirrors.kernel.org:~> cd ubuntu/dists/quantal</code>	Change
directory	
<code>lftp mirrors.kernel.org:...> get Contents-amd64.gz</code>	Download a file

```

24772197 bytes transferred in 37 seconds (646.1K/s)
lftp mirrors.kernel.org:...> <Ctrl+z>           Background the
download
lftp mirrors.kernel.org:...> mget main/i18n/*      Get all in
main/i18n/
lftp mirrors.kernel.org:...> !ls                 Run local ls
lftp mirrors.kernel.org:...> bookmark add quantal  Bookmark location
lftp mirrors.kernel.org:...> quit                Close lftp

```

This session logs in as the anonymous user at `mirrors.kernel.org`. After changing to the directory containing the file I was looking for, I downloaded it using the `get` command. By typing `Ctrl+z`, the download could continue while I did other activities. Next, the `mget` command (which allows wildcards such as `*`) downloaded all files from the `main/i18n/` directory.

Any command preceded by an exclamation mark (such as `!ls`) is executed by the local shell. The `bookmark` command saves the current location (in this case, `ftp://mirrors.kernel.org/ubuntu/dists/`) under the name `quantal`, so next time I can run `lftp quantal` to return to the same location. The `quit` command ends the session.

Here are some **useful commands during an authenticated lftp upload session**. This assumes you have the necessary file permissions on the server:

```

$ lftp chris@example.com
Password: *****
lftp example.com:~> lcd /home/chris/songs      Change to a local
directory
lftp example.com:~> cd pub/uploads             Change to server
directory
lftp example.com:~> mkdir songs                Create directory on
server
lftp example.com:~> chmod 700 songs            Change remote directory
perm
lftp example.com:~> cd songs                   Change to the new
directory
lftp example.com:~> put song.ogg tune.ogg      Upload files to server
3039267 bytes transferred
lftp example.com:~> mput /var/songs/*          Upload matched files
lftp example.com:~> quit                       Close lftp

```

The `lftp` session illustrates how you can use shell command names to operate on remote directories (provided you have permission). The `mkdir` and `chmod` commands create a directory and leave permissions open only to your user account. The `put` command uploads one or more files to the remote server. The `mput` command can use wildcards to match multiple files for download. Other commands include `mirror` (to download a directory tree) and `mirror -R` (to upload a directory tree).

`lftp` also provides a shell script for non-interactive download sessions: `lftpget`. The syntax of `lftpget` is similar to that of the `wget` command:

```
$ lftpget ftp://mirrors.kernel.org/ubuntu/dists/quantal/Release
```

Keep in mind that standard FTP clients are insecure because they do all their work in clear text. So your alternative, especially when security is a major issue, is to use SSH tools to transfer files.

Using SSH Tools to Transfer Files

Because SSH utilities are among the most important tools in a system administrator's arsenal of communications commands, some of the more complex uses of configuring and using SSH utilities are covered in Chapter 13. However, in their most basic form, SSH utilities are the tools you should use most often for basic file transfer.

In particular, the `scp` command will do most of what you need to get a file from one computer to another, while making that communication safe by encrypting both the password stage and data transfer stage of the process. The `scp` command replaces the `rcp` command as the most popular tool for host-to-host file copies.

Warning You do not get a warning before overwriting existing files with `scp`, so be sure that the target host doesn't contain any files or directories you want that are in the path of your `scp` file copies.

Copying Remote Files with `scp`

To use `scp` to transfer files, the SSH service (usually the `sshd` server daemon) must be running on the remote system. Here are some examples of **useful `scp` commands**:

```
$ scp myfile chris@server1:/tmp/          Copy myfile to server1
Password: *****
$ scp server1:/tmp/myfile .               Copy remote myfile to local
dir
Password: *****
```

Use the `-p` option to **preserve permissions and timestamps** on the copied files:

```
$ scp -p myfile server1:/tmp/
```

If the SSH service is configured to listen on a port other than the default port 22, use `-P` to **indicate that port** on the `scp` command line:

```
$ scp -P 12345 myfile server1:/tmp/      Connect to a particular
port
```

To do **recursive copies**, from a particular point in the remote filesystem, use the `-r`

option:

```
$ scp -r mydir francois@server1:/tmp/      Copy all in mydir to remote  
/tmp
```

Although `scp` is most useful when you know the exact locations of the file(s) you need to copy, sometimes it's more helpful to browse and transfer files interactively.

Copying Files Using `rsync`

Like `scp`, the `rsync` command lets you copy files between remote systems over the `ssh` facility. However, `rsync` has some features that make it particularly useful for doing backups and for syncing directories over a network.

In the `rsync` example ahead, replace `localhost` with the hostname or IP address of the remote system where you want to back up your files. Testing `rsync` by copying files to `localhost` makes it easier to see the results of `rsync` and see how it works.

The `rsync` command can **recursively copy files from a local directory to another system (local or remote)**. The `-a` (archive) option requests a recursive copy and tries to maintain file timestamps and permission. The `-v` is for verbose:

```
$ cd /usr/share/doc  
$ sudo rsync -av anacron/ chris@localhost:/tmp/anacron      Recursive  
copy  
chris@localhost's password: *****  
sending incremental file list  
created directory /tmp/anacron  
README.Debian  
...
```

Type `ls -l /usr/share/doc/anacron /tmp/anacron` and notice that date and timestamps were maintained on the files copied by `rsync`. However, because the user `chris` was used to perform the `rsync`, all files copied to the target system are owned by `chris`.

One of the nice features of `rsync` is that it **only copies files that have changed**. So if you ran the same `rsync` command again after adding a few files, only the new files are copied.

```
$ cd /usr/share/doc  
$ sudo touch anacron/testing      Create a  
new file  
$ sudo rsync -av anacron/ chris@localhost:/tmp/anacron/      Repeat  
rsync  
chris@localhost's password: *****  
sending incremental file list  
./  
testing
```

Notice that only the new file is copied to the remote target. This makes it very efficient to use `rsync` as a backup tool. If instead of using `rsync` for backup, you wanted to just keep two directory structures in sync, you could add the `--delete` option to delete any files not in the original directory from the target directory. For example:

```
$ sudo rm anacron/testing
$ sudo rsync -av --delete anacron/ chris@localhost:/tmp/anacron/
chris@localhost's password: *****
./
deleting testing
```

When copying files with `rsync`, you should also think about what to do with symbolically linked files. With the `--links` option, symbolic link files are copied; with `--copy-links` the file that the symbolic link ultimately points to is copied to the remote location. With `--links`, you risk not having a copy of the file the link points to at all. With `--copy-links`, you risk having too many copies of a file.

Copying Remote Files in `sftp` and `lftp` Sessions

The `sftp` command lets you use an FTP-like interface to **find and copy files over SSH protocols**. Here's an example of how to start an `sftp` session:

```
$ sftp chris@server1
chris@server1's password: *****
sftp>
```

Use `sftp` in the same manner as you use regular FTP clients. Type `?` for a list of commands. You can change remote directories (`cd`), change local directories (`lcd`), check current remote and local directories (`pwd` and `lpwd`), and list remote and local contents (`ls` and `lls`). Depending on the permission of the user you logged in as, you may be able to create and remove directories (`mkdir` and `rmdir`), and change permissions (`chmod`) and ownership/group (`chown` and `chgrp`) of files and directories.

You can also use `lftp` (discussed earlier in this chapter) as an `sftp` client. Using `lftp` adds some user-friendly features such as **path completion** using the Tab key:

```
$ lftp sftp://chris@server1
Password: *****
lftp chris@server1:~>
```

Using Windows File Transfer Tools

In many cases, people need to get files from Linux servers using Windows clients. If your client operating system is Windows, you can use one of the following open source tools to get files from Linux servers:

- **WinSCP** (<http://winscp.net>)—Graphical scp, sftp, and FTP client for Windows over SSH1 and SSH2 protocols
- **FileZilla** (<http://filezilla.sourceforge.net>)—Provides graphical client FTP and SFTP services in Windows, as well as offering FTP server features
- **PSCP** (www.chiark.greenend.org.uk/~sgtatham/putty/)—Command line scp client that is part of the PuTTY suite
- **PSFTP** (www.chiark.greenend.org.uk/~sgtatham/putty/)—Command line sftp client that is part of the PuTTY suite

Sharing Remote Directories

Tools described to this point in the chapter provide atomic file access, where a connection is set up and files are transferred in one shot. In times where more persistent, ongoing access to a remote directory of files is needed, services for sharing and mounting remote filesystems can be most useful. Such services include Network File System (NFS), Samba, and SSHFS.

Sharing Remote Directories with NFS

Assuming a server is already running the NFS service (part of the `nfs-kernel-server` package), you can use `exportfs` and `showmount` commands to see available and mounted shared directories. Mounting a shared directory is done with special options to the standard `mount` command. If you install the `nfs-kernel-server` package, Ubuntu will start the NFS service.

Viewing and Exporting NFS Shares

Run from the NFS server, the `exportfs` command shows all shared directories available from that server:

```
$ sudo /usr/sbin/exportfs -v
/export/mysh      client.example.com(ro,root_squash,no_subtree_check)
/mnt/public      *(rw,wdelay,root_squash,no_subtree_check)
```

The two directories being shared are `/export/mysh` and `/mnt/public`. The first is only available to host computer `client.example.com`, whereas the second is available to everyone. Options for each share are shown in parentheses. The first share is available read-only (`ro`), and requests from the root user on the client are mapped into the anonymous UID (`root_squash`). Also, a less thorough check of filesystem permission is done (`no_subtree_check`). The second share allows read-write

mounting (`rw`) and writes to the share are delayed to improve performance when more writes are expected (`wdelay`).

You can **add and modify shared NFS directories** by making changes to the `/etc/exports` file. To get changes to take effect, type any of the following as root:

```
$ sudo /etc/init.d/nfs-kernel-server reload    Reload exported
directories
$ sudo exportfs -r                            Reload exported
directories
$ sudo exportfs -rv                            Verbose reload of
shares
exporting client.example.com:/export/myshare
exporting */mnt/public
```

From the Linux server system, you can use the `showmount` command to **see what shared directories are available from the local system**. For example:

```
$ sudo /usr/sbin/showmount -e
Export list for server.example.com
/export/myshare  client.example.com
/mnt/public      *
```

From a client Linux system, you can use the `showmount` command to **see what shared directories are available from a selected computer**. For example:

```
$ sudo /usr/sbin/showmount -e server.example.com
/export/myshare client.example.com
/mnt/public      *
```

Note If you are unable to see the NFS shared directories from another system, it is possible that the firewall (iptables) is blocking access to the NFS service on the server side.

Mounting NFS Shares

Use the `mount` command to mount a remote NFS share on the local computer. Here is an example:

```
$ sudo mkdir /mnt/server-share
$ sudo mount server.example.com:/export/myshare /mnt/server-share
```

This example notes the NFS server (`server.example.com`) and the shared directory from that server (`/export/myshare`). The local mount point, which must exist before mounting the share, appears at the end of the command (`/mnt/server-share`).

Pass NFS-specific options to the `mount` command by adding them after the `-o` option:

```
$ sudo mount -o rw,hard,intr server.example.com:/export/myshare
/mnt/server-share
```

The `rw` option mounts the remote directory with read-write permissions, assuming that permission is available. With `hard` set, someone using the share will see a `server not responding` message when a read or write operation times out. If that happens, having set the `intr` option lets you **interrupt a hung request to a remote server** (press Ctrl+c to send an interrupt).

On older systems, NFS version 3 (`nfs3`) protocol is used to connect to the share. To use NFS version 4, which is designed to work over the Internet and through firewalls, **indicate that protocol as the filesystem type** on the command line as follows:

```
$ sudo mount -t nfs4 server.example.com:/ /mnt/server-share
```

Note Depending on which version of Ubuntu you are using, the implementation of NFS v4 may not be robust enough for production. It may be safer and/or more reliable to tunnel earlier versions of NFS over SSH. You can find more information on this topic with an Internet search for “nfs ssh”, and especially look at www.howtoforge.com/nfs_ssh_tunneling. In addition, look at <http://tldp.org/HOWTO/NFS-HOWTO/security.html> for more on NFS security.

Sharing Remote Directories with Samba

Samba is the open source implementation of the Windows file and print sharing protocol originally known as **Server Message Block** (SMB) and now called **Common Internet File System**(CIFS). There is an implementation of Samba in Linux, as well as in many other operating systems. To use Samba, install the packages `samba` and `samba-doc`.

Graphical tools for sharing, querying, and mounting shared SMB directories from Windows include the Samba SWAT web-based administration tool. To **use the SWAT tool** in Linux, install the `swat` package. Next, read the instructions at <https://help.ubuntu.com/community/Swat> for details on how you can start SWAT.

Commands for working with Samba shares can be used to query SMB servers, mount directories, and share directories.

Viewing and Accessing Samba Shares

To **scan your network for SMB hosts**, type the following:

```
$ findsmb
```

*=DMB

+=LMB

IP ADDR	NETBIOS NAME	WORKGROUP/OS/VERSION
---------	--------------	----------------------

```
-  
192.168.1.1      SERVER1      +[MYWORKGROUP] [Unix] [Samba 3.6.3]
```

To view a text representation of your network neighborhood (shared directories and printers), use `smbtree`:

```
$ sudo smbtree  
Password: *****  
MYGROUP  
  \\THOMPSON                thompson server (Samba, Ubuntu)  
    \\THOMPSON\hp2100 HP LaserJet 2100M Printer  
    \\THOMPSON\IPC$ IPC Service (thompson server (Samba,  
Ubuntu))  
  \\EINSTEIN                Samba Server  
    \\EINSTEIN\hp5550 HP DeskJet 5550 Printer  
    \\EINSTEIN\IPC$ IPC Service (Samba Server)
```

To add an existing Linux user as a Samba user, use the `smbpasswd` command:

```
$ sudo smbpasswd -a chris  
New SMB password: *****  
Retype new SMB password: *****  
Added user chris
```

Note You need to set up a Samba password for yourself to perform any of the commands that ask for a password.

To list services offered by a server to an anonymous user, type the following (press Enter to skip the password):

```
$ smbclient -L server  
Enter chris's password:  
Anonymous login successful  
Domain=[MYGROUP] OS=[Unix] Server=[Samba 3.6.3]  
      Server                Comment  
      -----              -  
      EINSTEIN  
      THOMPSON                thompson server (Samba, Ubuntu)
```

Here's the output from `smbclient` for a specific user named `chris`:

```
$ smbclient -L server -U chris  
Enter chris's password:  
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.6.3]  
  Sharename      Type  Comment  
  -----      -  
  homes          Disk  Home Directories  
  print$         Disk  Printer Drivers  
  IPC$           IPC   IPC Service (thompson server (Samba, Ubuntu))  
  chris          Disk  Home Directories  
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.6.3]
```

Server	Comment
-----	-----
EINSTEIN	
THOMPSON	thompson server (Samba, Ubuntu)
Workgroup	Master
-----	-----
DATAGROUP	MYSERVER
WORKGROUP	THOMPSON

To connect to a Samba share **FTP-style**, type the following:

```
$ smbclient //thompson/homes -U chris
Password: *****
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.6.3]
smb: \>
```

As with most FTP clients, type **help** or **?** to see a list of available commands. Likewise, you can use common shell-type commands, such as **cd**, **ls**, **get**, **put**, and **quit**, to **get around on the SMB host**.

Mounting Samba Shares

You can **mount remote Samba shares on your local filesystem** much as you would mount a local hard disk partition or remote NFS filesystem. To mount the share:

```
$ sudo mount -t cifs -o username=chris,password=MySecret \
//192.168.1.1/homes /mnt/mymount/
```

You can **see the current connections and file locks** on a server using the `smbstatus` command. This will tell you if someone has mounted your shared directories or is currently using an `smbclient` connection to your server:

```
$ sudo smbstatus
Samba version 3.6.3
PID      Username      Group      Machine
-----
 5466    chris        chris      192.168.1.1  (192.168.1.1)
Service  pid          machine    Connected at
-----
IPC$     30180        192.168.0.145  Sat Mar  9 12:19:28 2013
chris    30180        192.168.0.145  Sat Mar  9 12:19:28 2013
```

Looking Up Samba Hosts

NetBIOS names are used to identify hosts in Samba. You can **determine the IP address of a computer** using the `nmblookup` command to broadcast for a particular NetBIOS name on the local subnet as follows:

```
$ nmblookup thompson
```

```
querying thompson on 192.168.1.255
192.168.1.1 thompson<00>
```

To find the IP address for a server on a specific subnet, use the `-U` option:

```
$ nmblookup -U 192.168.1.255 einstein
querying einstein on 192.168.1.255
192.168.1.1 einstein<00>
```

Checking Samba Configuration

If you are unable to use a Samba share or if you have other problems communicating with your Samba server, you can test the Samba configuration on the server. The `testparm` command can be used to **check your main Samba configuration file** (`smb.conf`):

```
$ testparm
Load smb config files from /etc/samba/smb.conf
Processing section "[homes]"
Processing section "[printers]"
Processing section "[myshare]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press Enter to see a dump of your service definitions
```

After pressing Enter as instructed, you can see the settings from your `smb.conf` file. Here's how an entry for the `myshare` shared directory, used earlier in an example, might appear in the [smb.conf](#) file:

```
[myshare]
    path = /home/chris
    username = chris
    valid users = chris
    hosts allow = einstein
    available = yes
```

This entry allows the Samba user `chris` to access the `/home/chris` directory (represented by the `myshare` share name) from the host computer named `einstein`. The share is shown as being currently available.

The previous example of `testparm` showed the entries you set in the `smb.conf` file. However, it doesn't **show all the default entries you didn't set**. You can view those using the `-v` option. Pipe it to the `less` command to page through the settings:

```
$ testparm -v | less
```

If you want to **test a configuration file before it goes live**, you can tell `testparm` to use a file other than `/etc/samba/smb.conf`:

```
$ testparm /etc/samba/test-smb.conf
```

Sharing Remote Directories with SSHFS

Another magical trick you can do over the SSH protocol is to mount remote filesystems. Using the SSH filesystem (`sshfs`), you can mount any directory from an SSH server that your user account can access from your local Linux system. `sshfs` provides encryption of the mount operation as well as of all the data being transferred. Another cool aspect of `sshfs` is that it requires no setup on the server side (other than having SSH service running).

Here is a quick procedure for **mounting a directory of documents from a remote server to a local directory**. Doing this only requires that the remote server is running SSH and that the directory you want is accessible to your user account on the server. Here, you mount a directory named `/var/docs` from the host at `10.0.0.50` to a mount point called `/mnt/docs` on the local system:

```
$ sudo apt-get install sshfs           Install sshfs
software
$ mkdir /var/tmp/chris                 Create mount
point
$ sshfs chris@10.0.0.5:/home/chris /var/tmp/chris Mount remote
directory
```

When you are done using the remote directory, you can **unmount** it with the `fusermount` command (part of the `fuse-utils` package):

```
$ fusermount -u /var/tmp/chris         Unmount remote
directory
```

Chatting with Friends in IRC

Despite the emergence of instant messaging, **Internet Relay Chat** (IRC) is still used by a lot of people today. With global companies and more employees working remotely, IRC has become a popular technology for helping groups of people to work together from different locations.

[Freenode.net](http://freenode.net) has tons of chat rooms dedicated to supporting major open source software projects. In fact, many people stay logged into them all day and just watch the discussions of their favorite Linux projects scroll by. This is known as **lurking**.

The `xchat` utility is a good graphical, multi-operating system IRC client. You can install just the `xchat` package or the GNOME bindings in the `xchat-gnome` package. To run either version of `xchat` from Ubuntu, type **xchat** from the dashboard and select the XChat IRC or XChat GNOME IRC icon.

The elite way to do IRC, however, is to run a text-mode client in `screen` on an

always-on machine, such as an old server. Another similar option is to use an IRC proxy client, also known as a **bouncer**, such as `dirproxy` (part of the `dirproxy` package).

The original IRC client was `ircII`. It allowed the addition of scripts—in some ways similar to macros found in productivity suites—that automated some of the commands and increased usability. The most popular was `PhoEniX` by Vassago. Then came `BitchX`, which started as an `ircII` script and then became a full-blown client. Today, many people use `irssi`. To **install and launch irssi** from Ubuntu, type:

```
$ sudo apt-get install irssi  
$ irssi -n JayJoe199x Open an irssi chat session
```

In this example, the username (referred to as the person's nick) is set to `JayJoe199x` (you should choose your own). You should see a blue status bar at the bottom of the screen indicating that you are in Window 1, the status window. If this is the first time you've run `irssi`, the program displays help messages pointing you to the documentation. IRC commands are preceded with a `/` character. For example, to **connect to the freenode server** (after accepting policies at <http://freenode.net>) type the following:

```
/connect chat.freenode.net
```

If you didn't add your username on the command line, you are connected to `chat.freenode.net` with the username you are logged in under. On IRC, a chat room is called a **channel** and has a pound sign (`#`) in front of the name. Next, **try joining the #ubuntu IRC channel**:

```
/join #ubuntu
```

You are now in the channel in Window 2, as indicated in the status bar. Switch among the `irssi` windows by pressing `Ctrl+n` and `Ctrl+p`. Alternatively, you could select `Alt+1` or `Alt+2`, but those keys won't work inside a `gnome-terminal` window, because the `gnome-terminal` eats those keystrokes.

To **get help** at any time, type **/help command**, where **command** is the name of the command you want more information on. Help text will output in the status window, not necessarily the current window.

To add to the IRC chat, simply type a message and press `Enter` to **send the message** to those in the channel. To **direct a message to a specific user in the chat**, type the first few characters of that user's nick, and press the `Tab` key to autocomplete the nick. Then type the message on the rest of the line and press `Enter`. The message appears in the chat window and appears highlighted to that user.

You can **change your nick** at any time using the `/nick` command. For example, to change your nick to `joe`, type `/nick joe`. Type `/part` to leave a channel. Type `/quit` to exit the program.

There is a lot more to `irssi`. You can customize it and improve your experience significantly. Refer to the `irssi` documentation (www.irssi.org/documentation) for more information about how to use `irssi`.

Using Text-Based E-mail Clients

Most Mail User Agents (MUAs) are GUI-based these days. So if you began using e-mail in the past decade or so, you probably think of browser-based e-mail clients (such as Gmail) or stand-alone graphical applications such as Evolution, Kmail, Thunderbird, or (on Windows systems) Outlook when it comes to e-mail clients. On the first UNIX and Linux systems, however, reading e-mail was handled by text-based applications.

If you find yourself needing to check e-mail on a remote server or other text-based environment, venerable text-based mail clients are available and still quite useful. In fact, some hard core geeks still use text-based mail clients exclusively, touting their efficiency and scoffing at HTML-based messages.

The mail clients described in this chapter expect your messages to be stored in standard MBOX format on the local system. That means that you are either logged into the mail server or you have already downloaded the messages locally (for example, by using POP3 or similar).

As an alternative, there are some text-based e-mail readers, such as the `mutt` command, that let you connect to POP and IMAP mail servers (including encrypted protocols) to read your e-mail.

Note Text-based mail clients can be used to read mail already downloaded by other mail clients. For example, you could open your Evolution mail Inbox file by typing `mail -f $HOME/.evolution/mail/loc/Inbox`.

Managing E-mail with mail

The oldest command, and easiest to use when you just want a quick check for messages in the root user's mailbox on a remote server, is the `mail` command (`/bin/mail`), part of the `mailutils` package.

Before you can use the command line mail program, you must configure the package. There are many possible issues with mail servers that depend on your Internet service provider, or ISP. The configuration process is started as part of the installation when you run the following command:

```
$ sudo apt-get install mailutils
```


Although `mail` can be used interactively, it is often used for **sending script-based e-mails**. Here are some examples:

```
$ mail -s 'My Linux version' chris@localhost < /etc/lsb-release
$ ps auxx | mail -s 'My Process List' root@localhost
```

The two `mail` examples just shown provide quick ways to mail off some text without having to open a GUI mail application. The first example sends the contents of the `/etc/lsb-release` file to the user `chris@localhost`. The subject (`-s`) is set to `'My Linux Version'`. In the second example, a list of currently running processes (`ps auxx`) is sent to the root user with a subject of `'My Process List'`.

Used interactively, by default the `mail` command opens the mailbox set by your current shell's `$MAIL` value. For example:

```
$ echo $MAIL
/var/mail/chris
```

Note You may need to set this environment variable. The value should be `/var/mail/username` (where `chris` or some other name replaces `username`). On Ubuntu, the `MAIL` variable is not set by default, as the `mail` command is not installed by default.

To read the mail for the root user, run the following command:

```
$ sudo mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/root": 25 messages 25 new
>U  1 logwatch@a.1  Sat Jun 15 20:03  44/1667  "Logwatch for a
(Linux)"
  U  2 logwatch@a.1  Sun Jun 16 04:32  87/2526  "Logwatch for a
(Linux)"
    3 logwatch@a.1  Mon Jun 17 04:32  92/2693  "Logwatch for a
(Linux)"
  N  4 logwatch@a.1  Sat Jun 22 09:28  44/1667  "Logwatch for a
(Linux)"
  N  5 MAILER-DAEMON@a  Sat Jun 22 09:28  93/3348  "Warning: not
sent "
&
```

The current message has a greater-than sign (`>`) next to it. New messages have an `N` at the beginning, unread (but not new) messages have a `U`, and if there is no letter, the message has been read. The prompt at the bottom (`&`) is ready to accept commands.

At this point, you are in command mode. You can use simple commands to **move around** and **perform basic mail functions** in `mail`. Type the following to enact these functions:

- `?`—To see a list of commands

- The number of the message—To see that message
- **v3**—Opens the third message in the vi editor
- **h18**—To see a list of message headers that begins with message 18.
- **r7**—To reply to message 7 (type your message, and then put a dot on a line by itself to send the message)
- **d4**— To delete the fourth message (or **d4-9** to delete messages four through nine)
- **!bash**—To escape to the shell (then `exit` to return to `mail`)

Before you exit `mail`, know that any messages you view will be copied from your mailbox file to your `$HOME/mbox` file when you exit, unless you preserve them (`pre*`). To have all messages stay in your mailbox, exit by typing **x**. To save any messages you have already viewed to the mailbox, type **q** to exit.

You can open any file that is in MBOX format when you use `mail`. For example, if you are logged in as one user, but want to open the mailbox for the user `chris`, type this:

```
$ sudo mail -f /var/mail/chris
```

Managing E-mail with mutt

If you want to use a command line mail client on an ongoing basis, I recommend you use `mutt` instead of `mail`. The `mail` command has many limitations, such as not being able to send attachments without encoding them in advance (such as with the `uuencode` command), while `mutt` has many features for handling modern e-mail needs. The `mutt` command is part of the `mutt` package, which you need to install to use this command. Configure `mutt` by editing `/etc/Muttrc`. You also need to configure `sendmail` to allow for sending e-mail.

Like `mail`, `mutt` can also be used to send a message from a script. `mutt` also adds the capability to **send attachments**. For example:

```
$ mutt -s "My Linux Version" -a /etc/lsb-release \  
chris@example.com < email-body.txt  
$ mutt -s "My Linux Version" -a /etc/lsb-release \  
chris@example.com < /dev/null
```

The first example just shown includes the file `email-body.txt` as the body of the message and attaches the file `/etc/lsb-release` as an attachment. The second example sends the attachment, but has a blank message body (`< /dev/null`).

You can **begin your mutt mail session** (assuming your default mailbox is `$MAIL`) by simply typing `mutt`:

```
$ mutt  
/home/chris/Mail does not exist. Create it? ([yes]/no): y  
q:Quit d:Del u:Undel s:Save m:Mail r:Reply g:Group ?:Help
```

```

1 O Jun 16 logwatch@a ( 69) Logwatch for a (Linux)
2 O Jun 18 logwatch@a ( 171) Logwatch for a (Linux)
3 O Jun 18 Mail Delivery S ( 219) Warning: could not send
message
4 O Jun 19 logwatch@a ( 33) Logwatch for a (Linux)
--Mutt: /var/mail/root [Msgs:22 New:2 Old:20 63K]--(date/date)--
(all)--

```

Because `mutt` is screen-oriented, it is easier to use than `mail`. As with `mail`, you **use key commands to move around in mutt**. Type the following to navigate around your mailbox:

- **?**—As usual, you types this to get help. Hints appear across the top bar to help you with your mail.
- **Up and down arrow keys**—To highlight the messages you want to read.
- **Enter**—To view the highlighted message.
- **PageUp and PageDown**—To page through each message.
- **i**—To return to the message headers.
- **slash (/)**—Search forward for text.
- **Esc-/**—Search backwards.
- **n**—To search again.
- **Tab**—To jump to the next new or unread message.
- **Esc+Tab**—To go to the previous message.
- **s**—To save the current message to a file.
- **d**—To delete a message.
- **u**—To undelete a message.

You can type the following to write and send e-mails:

- **m**—To send a new mail message. After adding the recipient and subject, a blank message opens in `joe` (or whatever you have your `$EDITOR` set to).
- **a**—After exiting the message body, type **a** to add an attachment, if you like.
- **?**—To see other ways of manipulating your message, headers, or attachments.
- **y**—To send the message or **q** to abort the send.
- **x**—To exit without changing your mailbox when you are done.
- **q**—To exit and incorporate the changes you made (messages read, deleted, and so on).

Although, by default, `mutt` reads mail from your local mail spool file, it can also attach to mail servers using POP, POPS, IMAP, and IMAPS protocols. If you know the address of your mail server, here are examples of how to connect to a POPS or IMAPS e-mail server from `mutt`:

```

$ mutt -f imaps://chris@imapsserver.example.com
$ mutt -f pops://chris@popsserver.example.com

```

For mail protocols (SMTP) that require certificates (such as secure POP and IMAP protocols) you are presented with a certificate to accept or not after you connect to the server. Once you accept the certificate and provide the correct username and password, you can read, write, and reply to mail as you would with mail from a local mail spool file.

Summary

Network access commands provide quick and efficient ways to get content you need over a network. The `elinks` web browser is a popular screen-oriented command for browsing the web or taking a quick look at any HTML file. Dozens of commands are available to download files over FTP, HTTP, SSH, or other protocols, including `wget`, `curl`, `lftp`, and `scp`.

For more ongoing access to remote directories of files, this chapter covers how to use NFS, Samba, and SSHFS command tools. You can do IRC chats, which are popular among open source projects, using the `irssi` command. For text-based e-mail clients, you have choices such as the `mail` and `mutt` commands.

Chapter 13

Doing Remote System Administration

IN THIS CHAPTER

- Configuring SSH
- Using SSH for remote login
- Using SSH to do tunneling
- Using SSH to provide proxy service
- Using SSH with private keys
- Using screen and byobu remote multiplexing terminals
- Accessing remote Windows desktops
- Sharing remote Linux desktops with VNC

Most professional Linux administrators do not run a graphical interface on their Internet servers. As a result, when you need to access other computers for remote administration, you will almost surely need to work from the command line at some time. Fortunately, there are many feature-rich Linux commands to help you do so.

Tools associated with the Secure Shell (SSH) service not only allow remote login and file transfer, but they also offer encrypted communication to keep your remote administration work secure. With tools such as Virtual Network Computing (VNC), you can have a server's remote desktop appear on your local client computer. These and other features for doing remote systems administration are described in this chapter.

Doing Remote Login and Tunneling with SSH

Linux's big brother UNIX grew up on university networks. At a time when the only users of these networks were students and professors, and with networks mostly isolated from each other, there was little need for security.

Applications and protocols that were designed in those times (the 1970s and 1980s) reflect that lack of concern for encryption and authentication. SMTP is a perfect example of that. This is also true of the first generation of UNIX remote tools: `telnet`,

`ftp` (file transfer protocol), `rsh` (remote shell), `rcp` (remote copy), `rexec` (remote execution), and `rlogin` (remote login). These tools send user credentials and traffic in clear text. For that reason, they are very dangerous to use on the public, untrusted Internet, and have become mostly deprecated and replaced with the Secure Shell (SSH) commands (`ssh`, `scp`, `sftp` commands and related services).

Although there are still some uses for the legacy remote commands (see the section “Using Legacy Communications Tools”), most of this section describes how to use SSH commands to handle most of your needs for remote communications commands.

Using Legacy Communications Tools

Despite the fact that SSH provides better tools for remote communications, legacy communications commands, sometimes referred to as “r” commands, are still included with most major Linux distributions. Some of these tools will perform faster than equivalent SSH commands because they don’t need to do encryption. So some old-school UNIX administrators may use them occasionally on private networks or still include them in old scripts. Although for the most part you should ignore these legacy remote commands, one of these commands in particular can be useful in some cases: `telnet`.

The `telnet` command is still used to communicate with some network appliances (routers, switches, UPSes, and so on) that do not have the horsepower to run an `ssh` daemon. Even though it poses a security risk, some appliance manufacturers include `telnet` support anyway.

One good way to use the `telnet` command, however, is for troubleshooting many Internet protocols such as POP3, SMTP, HTTP, and others. Under the hood, these plain-text protocols are simply automated telnet sessions during which a client (such as a browser or mail user agent) exchanges text with a server. The only difference is the TCP port in use. Here is an example of how you could telnet to the HTTP port (80) of a web server:

```
$ telnet www.example.com 80
Trying 208.77.188.166...
Connected to www.example.com.
Escape character is '^]'.
GET / HTTP/1.0
Enter a second carriage return here
HTTP/1.1 200 OK
```

Similarly, you can telnet to a mail server on port 25 (SMTP) and 110 (POP3) and issue the proper commands to troubleshoot e-mail problems.

Note For more complete descriptions of using the `telnet` command to troubleshoot network protocols, refer to *Linux Troubleshooting Bible* (Christopher Negus and Thomas Weeks, Wiley Publishing, 2004), pages 505–508.

If you need to forcibly exit your telnet session, type the escape sequence (Ctrl+] by default). This will stop sending your keyboard input to the remote end and bring you to `telnet`'s command prompt where you can type `quit` or `?` for more options.

Configuring SSH

Nowadays, the Swiss Army knife of remote system administration is Secure Shell (SSH). SSH commands and services replace all the old remote tools and add strong encryption, public keys, and many other features. The most common implementation of SSH in the Linux world is OpenSSH (www.openssh.com), maintained by the OpenBSD project. OpenSSH provides both client and server components.

If it's not already installed, install the OpenSSH server by running the following command:

```
$ sudo apt-get install openssh-server
```

Here are a few facts about SSH:

- For Windows, you can use the Linux SSH tools within Cygwin (www.cygwin.com). But unless you're already using Cygwin (a Linux-like environment for Windows), we recommend PuTTY (www.chiark.greenend.org.uk/sgatatham/putty). PuTTY is a powerful open source Telnet/SSH client.
- Use SSH version 2 whenever possible because it is the most secure. Some SSH-enabled network appliances may only support older, less secure versions. OpenSSH supports all versions. Some older versions of Ubuntu accepted SSH v1 and v2 connections. Newer releases accept version 2 by default.
- In Ubuntu, run `service ssh start` to **start the SSH service** (`sshd` daemon). To **configure the service**, edit the `/etc/ssh/sshd_config` file.
- To **configure the ssh client**, edit the `/etc/ssh/ssh_config` file.

If you prefer to use graphical tools to administer your remote Linux system, you can enable **X11 Tunneling** (also called **X11 Port Forwarding**). With X11 Tunneling enabled (on both the SSH client and server), you can start an X application on the server and have it displayed on the client. All communication across that connection is encrypted.

Ubuntu comes with X11 forwarding turned on (`X11Forwarding yes`) for the server (`sshd` daemon). You still need to enable it on the client side. To **enable X11 forwarding**

on the client for a one-time session, connect with the following command:

```
$ ssh -X chris@myserver
```

To **enable X11 forwarding permanently for all users**, add `ForwardX11 yes` to `/etc/ssh/ssh_config`. To enable it permanently for a specific user only, add the line to that user's `~.ssh/config`. To test that the tunneling is working, run `xclock` after `ssh`'ing into the remote machine, and it should appear on your client desktop. To get these commands, you must install the `x11-apps` package.

SSH tunneling is an excellent way to securely use remote graphical tools!

Logging in Remotely with ssh

To **securely log in to a remote host**, you can use either of two different syntaxes to specify the username:

```
$ ssh -l chris myserver  
$ ssh chris@myserver
```

However, `scp` and `sftp` commands (discussed in Chapter 12) support only the `user@server` syntax, so we recommend you get used to that one. If you don't specify the username, `ssh` will attempt to log in using the same user you are logged in as locally. Once connected, if you need to **forcibly exit your ssh session**, type the escape sequence of a tilde followed by a period (`~.`). If that doesn't work, try `Ctrl+c`.

Accessing SSH on a Different Port

For security purposes, a remote host may have its **SSH service listening on a different port** than the default port number 22. If that's the case, use the `-p` option to `ssh` to contact that service:

```
$ ssh -p 12345 chris@turbosphere.com    Connect to SSH on port 12345
```

Using SSH to Do Tunneling (X11 Port Forwarding)

With SSH tunneling configured as described earlier, the SSH service forwards X Window System clients to your local display. However, tunneling can be used with other TCP-based protocols as well.

Tunneling for X11 Clients

The following sequence of commands illustrates **starting an SSH session**, and then **starting a few X applications so they appear on the local desktop**:

```
$ ssh -X chris@myserver                Start ssh
```



```

connection to myserver
chris@myserver's password: *****
[chris@myserver ~]$ echo $DISPLAY           Show the current X
display entry
localhost:10.0                               SSH sets display to
localhost:10.0
[chris@myserver ~]$ xeyes&                   Show moving desktop
eyes
[chris@myserver ~]$ gnome-calculator&       Use the GNOME
calculator
[chris@myserver ~]$ gksu system-config-printer& Configure printers

```

Tunneling for CUPS Printing Remote Administration

X11 is not the only protocol that can be tunneled over SSH. You can **forward any TCP port** with SSH. This is a great way to configure secure tunnels quickly and easily. No configuration is required on the server side.

For example, `myserver` is a print server with the CUPS printing service's web-based user interface enabled (running on port 631). That GUI is accessible from the local machine only. On the following client PC, you tunnel to that service using `ssh` with the following options:

```
$ ssh -L 1234:localhost:631 myserver
```

This example forwards port 1234 on the client PC to localhost port 631 on the server. You can now browse to `http://localhost:1234` on the client PC. This will be redirected to `cupsd` listening on port 631 on the server.

Tunneling to an Internet Service

When your local machine is blocked from connecting to the Internet, but you can get to another machine (`myserver`) that has an Internet connection, SSH tunneling can **provide Internet access to the local machine**. The following example lets you visit the Google.com website (HTTP, TCP port 80) across an SSH connection to a computer named `myserver` that has a connection to the Internet:

```
$ ssh -L 12345:google.com:80 chris@myserver
```

With this example, any connection to the local port 12345 is directed across an SSH tunnel to `myserver`, which in turn opens a connection to `Google.com` port 80. The command tries to log into `myserver` as the user `chris` (if a user name is not provided, the current user name is used on the remote system).

You can now browse to `http://localhost:12345` and use `myserver` as a relay to the Google.com website. Because you're using `ssh` only to forward a port and not to obtain a shell on the server, you can add the `-N` option to **prevent the execution of**

remote commands:

```
$ ssh -L 12345:google.com:80 -N myserver
```

Using SSH as a SOCKS Proxy

The previous example demonstrates that you can forward a port from the client to a machine other than the server. In the real world, the best way to **get your browser traffic out of your local network** via an encrypted tunnel is to use the SSH built-in SOCKS proxy feature. For example:

```
$ ssh -D 12345 myserver
```

The dynamic (`-D`) option of `ssh` enables you to log in to `myserver` (as usual). As long as the connection is open, all requests directed to port `12345` are then forwarded to `myserver`. Next, set your browser of choice to use `localhost` port `12345` as a SOCKS v5 proxy and you're good to go. Do not enter anything on the fields for HTTP and other protocols. They all work over SOCKS. See the Firefox Connections Settings window in [Figure 13-1](#).

[Figure 13-1](#): Use the Firefox Connections Settings window for proxy configuration.

Configure Proxies to Access the Internet

☐ No proxy

☐ Auto-detect proxy settings for this network

☐ Use system proxy settings

☒ Manual proxy configuration:

HTTP Proxy: Port:

☐ Use this proxy server for all protocols

SSL Proxy: Port:

FTP Proxy: Port:

SOCKS Host: Port:

☐ SOCKS v4 ☒ SOCKS v5

No Proxy for:

Example: .mozilla.org, .net.nz, 192.168.1.0/24

☐ Automatic proxy configuration URL:

To test your setup, try disconnecting your ssh session and browsing to any website. Your browser should give you a proxy error.

From a Windows client, the same port forwarding can be accomplished in Putty by selecting Connection ⇒ SSH ⇒ Tunnels.

Using ssh with Public Key Authentication

Up to this point, we've only used `ssh` with the default password authentication. The `ssh` command also supports public key authentication. This offers several benefits:

- **Automated logins for scripts and cron jobs**—By assigning an empty passphrase, you can use `ssh` in a script to log in automatically. Although this is convenient, it is also dangerous, because anybody who gets to your key file can connect to any machine you can. Configuring for automatic login can also be done with a passphrase and a key agent. This is a compromise between convenience and security, as explained in the “Using Public Key Logins” section.
- **A two-factor authentication**—When using a passphrase-protected key for interactive logins, authentication is done using two factors (the key and the

passphrase) instead of one.

Using Public Key Logins

Here's the process for **setting up key-based communications** between two Linux systems. In the following example, we use empty passphrases for no-password logins. If you prefer to protect your key with a passphrase, simply enter it when prompted during the first step (key pair creation).

On the client system, run the following `ssh-keygen` command to **generate the key pair** while logged in as the user that needs to initiate communications:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/chris/.ssh/id_rsa):
<Enter>
Enter passphrase (empty for no passphrase): <Enter>
Enter same passphrase again: <Enter>
Your identification has been saved in /home/chris/.ssh/id_rsa.
Your public key has been saved in /home/chris/.ssh/id_rsa.pub.
The key fingerprint is:
ac:db:a4:8e:3f:2a:90:4f:05:9f:b4:44:74:0e:d3:db
chris@host.example.com
```

Note that at each prompt, you pressed the Enter key to create the default key filename and to enter (and verify) an empty passphrase. You now have a private key that you need to keep very safe, especially because in this procedure you didn't protect it with a passphrase.

You also now have a public key (`id_rsa.pub`), which was created by the previous command. This public key needs to be installed on hosts you want to connect to. The content of `~/.ssh/id_rsa.pub` needs to be copied (securely) to `~/.ssh/authorized_keys` for the user you want to `ssh` to on the remote server. The `authorized_keys` file can contain more than one public key, if multiple users use `ssh` to connect to this account.

Although it used to be a manual process to copy your public key to a user account on another system, now you can do it quite easily with the `ssh-copy-id` command. While you are logged into the user account where you created the key, you would run the following command to copy your public key to the `authorized_keys` file of a user on another system:

```
$ ssh-copy-id chris@host1.example.com
chris@host1's password:
*****
Now try logging into the machine, with "ssh 'chris@host1'",
and check in:
```

```
~/.ssh/authorized_keys  
to make sure we haven't added extra keys that you weren't  
expecting.
```

In this example, you are prompted for the remote user chris's password. Once you type it in, that should be the last time you have to type that password to log in from your local user account to that remote user account.

As the message instructs, log in to the remote user account and check the `authorized_keys` file. You will be automatically logged in without being prompted to type a password. Here's an example:

```
$ ssh chris@host1  
Last login: Wed Mar 20 11:48:58 2013  
  
$ cat ~/.ssh/authorized_keys  
ssh-rsa  
AAAAB3NzaC1yc2EAAAADAQABABAQCdinPlzlSjW5IfsbfGBk05VDLb+5hN1  
PaJhTJLv+8478yox4N1Ub0KhEhjVZqcEd8YFpMgUjJqkEQ0IVO+TneGyFPDW5666k5nK  
  
cgAlw2lVELGfHZtTMS1Aaq0vwK/bUE2+rrDgwo+YlKhaA+/2DPaU4XRcnI86yS4NEdEN  
  
PAfoYIZj8DTMI+G0KnHqunIfkiP9g8wnvKIrJsFOJSi62ZeQ2CbXp3PQCAASNSt2r/o0  
  
tXI9rpdIwLPuwKkSj+FDJaFX5cbbecG5m6uH5IcGbTjeZgf1UpWNZEK/AkhA2E8HEta/  
  
4HqtbqfeqNq48x5zGEgyQphwqn8cDlnEYc7 chris@mydesktop
```

At this point, any of the `ssh` commands (`scp`, `sftp`, `rsync`, and so on) will also work without being prompted for a password from the local to the remote system. Keep in mind, however, that if you set a passphrase on your key, you will be asked for it before your key can be used to connect to the remote system.

Saving Private Keys to Use from a USB Flash Drive

If you'd like to **store your private key** somewhere safer than your hard drive, you can use a **USB flash drive** (sometimes called a thumbdrive or pen drive):

```
$ mv ~/.ssh/id_rsa /media/THUMBDRIVE1/myprivatekey
```

And then, when you want to use the key, insert the USB drive and type the following:

```
$ ssh -i /media/THUMBDRIVE1/myprivatekey chris@myserver
```

Using keys with passphrases is more secure than simple passwords but also more cumbersome. To make your life easier, you can use `ssh-agent` to **store unlocked keys for the duration of your session**. When you add an unlocked key to your running `ssh-agent`, you can run `ssh` using the key without being prompted for the passphrase each time.

To see what the `ssh-agent` command does, run the command with no option. A three-line bash script appears, as follows:

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-SkEQZ18329/agent.18329; export
SSH_AUTH_SOCK;
SSH_AGENT_PID=18330; export SSH_AGENT_PID;
echo Agent pid 18330;
```

The first two lines of the output just shown need to be executed by your shell. Copy and paste those lines into your shell now. You can avoid this extra step by starting `ssh-agent` and having the bash shell evaluate its output by typing the following:

```
$ eval `ssh-agent`
Agent pid 18408
```

You can now unlock keys and add them to your running agent. Assuming you have already run the `ssh-keygen` command to create a default key, let's **add that default key** using the `ssh-add` command:

```
$ ssh-add
Enter passphrase for /home/chris/.ssh/id_rsa: *****
Identity added: /home/chris/.ssh/id_rsa (/home/chris/.ssh/id_rsa)
```

Next, you can add the key you stored on the USB thumbdrive:

```
$ ssh-add /media/THUMBDRIVE1/myprivatekey
```

Use the `-l` option to `ssh-add` to **list the keys stored in the agent**:

```
$ ssh-add -l
2048 f7:b0:7a:5a:65:3c:cd:45:b5:1c:de:f8:26:ee:8d:78
/home/chris/.ssh/id_rsa
(RSA)
2048 f7:b0:7a:5a:65:3c:cd:45:b5:1c:de:f8:26:ee:8d:78
/media/THUMBDRIVE1/myprivatekey (RSA)
```

To **remove one key from the agent**, for example the one from the USB thumbdrive, run `ssh-add` with the `-d` option as follows:

```
$ ssh-add -d /media/THUMBDRIVE1/myprivatekey
```

To **remove all the keys stored in the agent**, use the `-D` option:

```
$ ssh-add -D
```

Using byobu and screen for Remote Shells

The `ssh` command gives you only one screen (shell). If you close that screen, you lose all you were doing on the remote computer. That can be very bad if you were in the middle of something important, such as a 12-hour compile. And if you want to do three things at once—for example, `vi httpd.conf`, `tail -f error_log`, and `service httpd reload`—you need to open three separate `ssh` sessions.

Traditionally, the `screen` command has been used to provide multiple, active shell sessions through a single login session. With `screen`, you can also detach from that `screen` session and then reattach to that session to pick up where you left off the next time you log in.

Instead of using `screen` directly, the `byobu` command offers a simplified interface and more powerful features to `screen`. The next section describes how to use the `screen` command directly, followed by a description of the `byobu` command.

Managing Remote Shells with Screen

Essentially, `screen` is a terminal multiplexer. If you are a system administrator working on remote servers, `screen` is a great tool for managing a remote computer with only a command line interface available. Besides allowing multiple shell sessions, `screen` also lets you disconnect from it, and then reconnect to that same `screen` session later.

The `screen` software package is installed by default with Ubuntu.

To **use `screen`**, run the `ssh` command from a client system to connect to the Linux server where `screen` is installed. Then simply type the following command:

```
$ screen
```

If you ran `screen` from a Terminal window, you should first see a welcome message asking for pizza and beer, and then see a regular bash prompt in the window. The next message suggests that you might want to use `byobu` instead (as described in the “Using `byobu` to Manage Remote Shells” section).

To control `screen`, press the `Ctrl+a` key combination, followed by another keystroke. For example, `Ctrl+a` followed by `?` (noted as `Ctrl+a, ?`) displays the help screen. With `screen` running, here are some commands and control keys you can use to operate `screen`.

\$ <u>screen -ls</u>	List active screens
There is a screen on:	
7089.pts-2.myserver (Attached)	Shows screen is attached
1 Socket in /var/run/screen/S-chris.	
\$ <u>Ctrl+a, Shift+a</u>	Change window title
Set window's title to: <u>My Server</u>	Type a new title
\$ <u>Ctrl+a, c</u>	Create a new window
\$ <u>Ctrl+a, "</u>	Show active window

```

titles
Num Name                               Flags                               Up/down arrows change
  0 My Server
windows
  1 bash
$ Ctrl+a, d                             Detach screen from
terminal
$ screen -ls                             List active screens
There is a screen on:
      7089.pts-2.myserver      (Detached)  Shows screen is detached
1 Socket in /var/run/screen/S-chris.

```

The `screen` session just shown resulted in two windows (each running a bash shell) being created. You can create as many as you like and name them as you choose. Also, instead of detaching from the `screen` session, you could have just closed it by exiting the shell in each open window (type **exit** or Ctrl+d).

When the `screen` session is detached, you are returned to the shell that was opened when you first logged into the server. You can reconnect to that `screen` session as described in the following section, “Reconnecting to a screen Session.”

The following list shows some other useful control key sequences available with `screen`:

- **Ctrl+a, ?**—Show help screen.
- **Ctrl+a, c**—Create new window.
- **Ctrl+a, d**—Detach `screen` from terminal. The `screen` session and its windows keep running.
- **Ctrl+a, "**—View list of windows.
- **Ctrl+a, '**—Prompt for number or name of window to switch to.
- **Ctrl+a, n**—View next window.
- **Ctrl+a, p**—View previous window.
- **Ctrl+a, [**—Terminal’s vertical scroll is disabled in `screen`. These keys turn on `screen`’s scrollbar mode. Press Enter twice to exit.
- **Ctrl+a, Shift+a**—Rename the current window.
- **Ctrl+a, w**—Show the list of window names in the title bar.

Reconnecting to a screen Session

After you detach from a `screen` session, you can return to that screen again later (even after you log out and disconnect from the server). To **reconnect when only one screen is running**, type the following:

```
$ screen -r
```

If there are several `screen` sessions running, `screen -r` won’t work. For example,

this shows what happens when two detached `screen` sessions are running:

```
$ screen -r
There are several suitable screens on:
    7089.pts-2.myserver      (Detached)
    7263.pts-2.myserver      (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
```

As the output suggests, you can identify the `screen` session you want by its name (which, by default, is a combination of the session's process ID, tty name, and hostname). For example:

```
$ screen -r 7089.pts-2.myserver
```

Naming screen Sessions

Instead of using the default names, you can **create more descriptive names for your screen sessions** when you start `screen`. For example:

```
$ screen -S mysession
$ screen -ls
There is a screen on:
    26523.mysession (Attached)
```

Sharing screen Sessions

The `screen` command also allows the **sharing of screens**. This feature is great for tech support because each person connected to the session can both type into and watch the current session. Creating a named screen, as in the preceding section, makes this easier. Then another person on a different computer can `ssh` to the server (using the same username) and type the following:

```
$ screen -x mysession
```

As with `screen -r`, if there's only one screen running, you don't need to specify which screen you're connecting to:

```
$ screen -x
```

Using byobu to Manage Remote Shells

The `byobu` command was developed by the Ubuntu community to provide a configuration layer to `screen`. It offers more features and an easier interface that relies more on function keys than Ctrl keys. It also has some nice extras, such as the ability to show status information of the machine you are logged into.

To start `byobu` on a remote shell session, log in to the remote system using `ssh`, and

then either type **byobu** to **start a session immediately** or type **byobu-enable** to have byobu enabled as your default shell session when you open any text-based login session from the system (type **byobu-disable** to disable this behavior).

When byobu first starts, you have the choice of operating in **screen mode** or **emacs mode** (see `man 1 tmux`). For our purposes, I selected **byobu mode**:

```
Configure Byobu's ctrl-a behavior...
```

```
When you press ctrl-a in Byobu, do you want it to operate in:
```

- (1) Screen mode (GNU Screen's default escape sequence)
- (2) Emacs mode (go to beginning of line)

```
Note that:
```

- F12 also operates as an escape in Byobu
- You can press F9 and choose your escape character
- You can run 'byobu-ctrl-a' at any time to change your selection

```
Select [1 or 2]: 1
```

Once byobu has started, a status bar appears at the bottom of the page. You can use function keys (F1 through F12) on the top of the keyboard to **create new shell windows and manage them**. Here are examples of how to use function keys to make and manage shell windows in byobu:

```
F2 Opens a new shell window (try opening a few)
```

```
F3 Goes to the previous window
```

```
F4 Goes to the next window
```

To refresh the status of notifications, use the F5 function key:

```
F5 Status notifications refresh
```

When you are done with your byobu session, you can **disconnect from byobu**, leaving it active to return to later. Type Shift+F6 keys to not log out of the login session when you disconnect:

```
F6 Disconnect (detach) from byobu session and logout
```

```
Shift-F6 Disconnect (detach) from byobu session, don't logout
```

While in the byobu session, you can **go back to up to 10,000 lines of text** on the screen. With a bunch of text on the screen, press F7 to go into scrollback mode. Then use the PgUp and PgDn keys to refer back to the text on the screen. Press Enter to exit this mode:

```
$ find / Run some command to fill the screen with text
```

```
F7 Go into the scrollback/search mode
```

```
PgUp Page back through the text on the screen
```

```
PgDn Page forward through the text on the screen
```

You can change the byobu configuration in different ways. Use F8 to change the name of the current window. Use F9 to see the Byobu Configuration Menu, which allows you to change many settings.

F8 Change the name of the current window

F9 Open up the Byobu Configuration Menu

With the Byobu Configuration Menu open, you can change several settings. Here's what that menu looks like:

```
Byobu Configuration Menu
Help -- Quick Start Guide
Toggle status notifications
Change escape sequence
Byobu currently does not launch at login (toggle on)
<Exit>
```

Use arrow keys, the Tab key, Enter, and PgUp or PgDn keys to **navigate the menus**. Press Esc to return to earlier menus. The Help selection shows descriptions of keystrokes to use in `byobu`. Select Toggle status notifications to change the status that is shown in the bottom status bar. Use the other selections to change escape sequences and change whether or not `byobu` is used by default when you log into a text-based login session.

To **close the current byobu window**, simply exit the shell (type `exit` or Ctrl+d). Closing the last shell associated with a `byobu` session ends that session.

For more information on `byobu`, type `man byobu` or visit the Byobu documentation page at the Ubuntu Community site: <https://help.ubuntu.com/community/Byobu>.

Using a Remote Windows Desktop

Many system administrators who become comfortable using a Linux desktop prefer to do administration of their Windows systems from Linux whenever possible. Linux provides tools such as `rdesktop` and `tsclient`, which allow you to connect to a Windows system running Windows Terminal Services.

- To be able to **connect to your Windows system desktop** from Linux, you have to enable Remote Desktop from your Windows system. How that is done differs between different Windows systems: To enable remote desktop access to a Windows XP system (and others) right-click My Computer and select Properties. Then choose the Remote tab from the System Properties window and select the Allow users to connect remotely to this computer checkbox. Select which users you want to let connect to the Windows box and click OK.
- To enable remote desktop access from Ubuntu to a Windows 7 system, select System and Security from the Windows Control Panel. Click Remote settings in the left pane. From the System Properties dialog box on the Remote tab, choose Allow Connections from Computers Running Any Version of Remote Desktop. Next,

choose Select Users to choose which user accounts are allowed access.

- In Windows 8, to gain remote access to an Ubuntu remote desktop, right-click on the Windows 8 Style Menu, select All Apps, then right-click on the Computer menu entry. Then select the Properties entry. In the Properties dialog box, click Remote settings link.

Now, from Linux, you can use either `rdesktop` or `remmina` to connect to the Windows system using Remote Desktop Protocol (RDP). Ubuntu comes with both of these applications installed.

Connecting to a Windows Desktop with Remmina

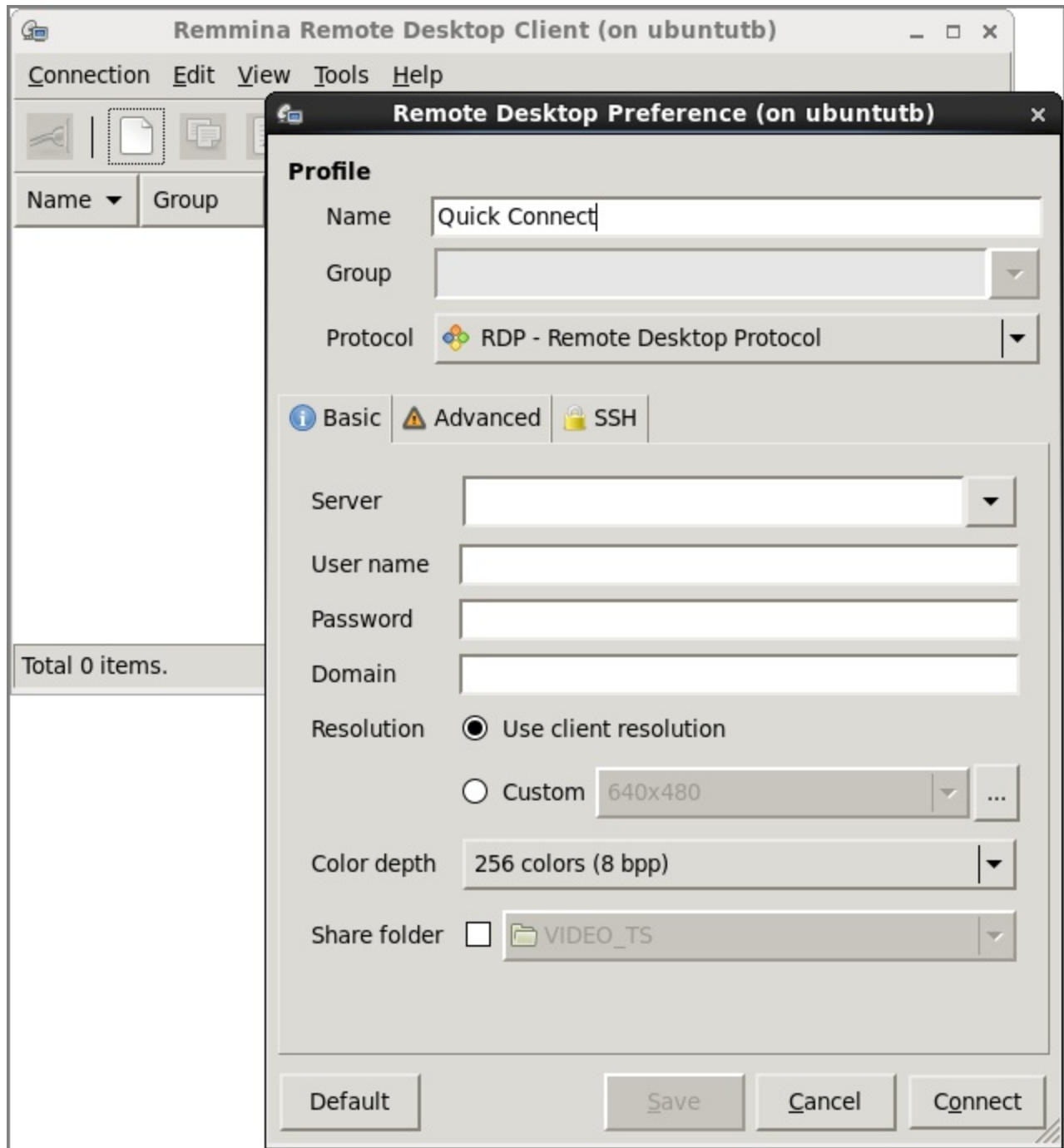
If you are used to using Windows' Remote Desktop Connection (formerly known as Terminal Services Client) to connect from one Windows box to another, you will probably find the Remmina Remote Desktop (`remmina`) tool a good way to connect to a Windows desktop from Linux. Running `remmina` opens a Remmina Remote Desktop Client window, which provides access to a Windows remote desktop client's user interface.

When the `remmina` package is installed (`apt-get install remmina`), **launch `remmina`** by typing **Remmina Remote Desktop** from the Ubuntu Dashboard or by typing the following from the shell:

```
$ remmina &
```

When the window appears, select Connection \Rightarrow New. [Figure 13-2](#) shows the Remmina Remote Desktop Client and Preferences windows.

[Figure 13-2](#): Remmina Remote Desktop Client (`remmina`) connects to Windows desktops.



Probably all you need to enter on this screen is the hostname or IP address of the Windows system. Next enter the username and password. Select different tabs to further refine your connection to the remote Windows desktop.

Note that `remmina` can also be used as a client for VNC, SSH, and SFTP services.

Connecting to a Windows Desktop with `rdesktop`

If you prefer not to use the `remmina` described previously, you can **log in to a remote Windows desktop** using the `rdesktop` command. To install `rdesktop`, type the

following:

```
$ sudo apt-get install rdesktop
```

The `rdesktop` command requests a login to the Windows machine, and then opens the Windows desktop for the user after you log in. Here are examples of the `rdesktop` command:

\$ <u>rdesktop 172.16.18.66</u>	Login to desktop at IP address
\$ <u>rdesktop -u chris -p M6pyXX win1</u>	Identify user/password for host
win1	
\$ <u>rdesktop -f win1</u>	Run rdesktop in full-screen
mode	
\$ <u>rdesktop -0 -r sound:local win1</u>	Direct sound from server to
client	
\$ <u>rdesktop -E win1</u>	Disable client/server
encryption	

If you disable client/server encryption, the login packet is encrypted, but everything after that is not. Although this can improve performance greatly, anyone sniffing your LAN would be able to see your clear-text communications (including any interactive logins after the initial login packet). Other `rdesktop` options that can improve performance or your Windows desktop include `-m` (don't send mouse motion events), `-D` (hide window manager's decorations), and `-K` (don't override window manager key bindings).

Using Remote Linux Desktop and Applications

The X Window System (X) should not be run on typical production servers for security and performance reasons. But thanks to the client-server nature of X, you can run an X-enabled program on a remote machine with its graphical output directed to your local desktop. In that relationship, the application running from the remote machine is referred to as the X client, and your desktop is the X server. When running remote X applications on untrusted networks or the Internet, use SSH forwarding as described earlier. On trusted LANs, do it without SSH, as described here.

By default, your X desktop will not allow remote X applications to connect (pop up) on your desktop. You can **allow remote apps on your desktop** using the `xhost` command. On your local Linux display, use the `xhost` command to control which remote machines can connect to X and display applications on your desktop. Here are examples of `xhost`:

```

$ xhost                                List allowed hosts
access control enabled, only authorized clients can connect
SI:localuser:chris
$ xhost +                               Disable access control (dangerous)
access control disabled, clients can connect from any host
$ xhost -                               Re-enable access control
access control enabled, only authorized clients can connect
$ xhost remotemachine                   Add an allowed host
remotemachine being added to access control list

```

Access control should be completely disabled only for troubleshooting purposes. However, with access enabled for a particular host machine (remotemachine in this case), you can do the following from a shell on the remote computer to have X applications from that machine appear on the local desktop (in this case called localmachine):

```

$ export DISPLAY=localmachine:0         Set the DISPLAY to
localmachine:0
$ xterm &                               Open remote Terminal on
local
$ xclock &                               Open remote clock on local
$ gnome-calculator &                     Open remote calculator game
on local

```

After setting the `DISPLAY` variable on remotemachine to point to localmachine, any application run from that shell on remotemachine should appear on Desktop 0 on localmachine. In this case, we started the Terminal window, clock, and calculator applications.

Note In the past few versions of Ubuntu, the X server doesn't listen for TCP connections by default. To allow remote X connections, edit the `/etc/lightdm/lightdm.conf` file on the X server to add the line `xserver-allow-tcp=true`, so it appears as follows:

```

[SeatDefaults]
    greeter-session=unitygreeter
    user-session=ubuntu
    xserver-allow-tcp=true

```

Then restart the X Window server or the window manager (`sudo restart lightdm`).

Sharing X applications in this way between Linux and UNIX hosts is pretty easy. However, running X applications so they appear on other desktop computer platforms takes a little more setup. If your desktop runs Windows, you have to run an X server. A free solution is Cygwin, which includes an X server. There are also feature-rich

commercial X servers, but they can be very expensive. To share remote desktops across different operating system platforms, we suggest you use Virtual Network Computing (VNC).

Sharing Desktops Using VNC

Virtual Network Computing (VNC) consists of server and client software that lets you **assume remote control of a full desktop display from one computer on another**. In Ubuntu, the `vncviewer` command comes preinstalled to access a remote desktop on your display (client), but you need the `vncserver` package to share a desktop from your computer (server). To install the `vncserver` package, type the following:

```
$ sudo apt-get install vncserver
```

VNC clients and servers are available for, and interoperable with, many different operating systems. VNC servers are available on Linux, Windows (32-bit), Mac OS X, and UNIX systems. VNC clients are offered on those and many other types of systems (including OS/2, PalmOS, and even as a Java application running in a web browser).

Setting Up the VNC Server

From your Linux desktop, we'll assume you are using the default display (`DISPLAY=:0`) as your local desktop. So we'll set out to **create independent displays accessible via VNC**.

Then, as each user, run the `vncpasswd` command to create the password each of those users will need to connect to their own desktops on the VNC server. In our example, we run the following as the user `chris`:

```
$ vncpasswd  
Password: *****  
Verify: *****
```

Finally, you can start the VNC server (`vncserver`). Type the following as a regular user (in our example, we used `chris`):

```
$ vncserver  
New 'myserver:1 (chris)' desktop is myserver:1  
Starting applications specified in /home/chris/.vnc/xstartup  
Log file is /home/chris/.vnc/myserver:1.logs
```

If you are using the `iptables` firewall built into your system, make sure you open the port(s) for VNC. Each display runs on its own port. Display number `N` is accessed on TCP port `5900+N`. For example, display 1 is accessible on port 5901. Refer to Chapter

14 for more details on iptables.

To change how your VNC servers are configured, edit the `xstartup` file in your `$HOME/.vnc` directory. Also in that directory are a `passwd` file (to store the user password protecting the VNC session) and log files for each VNC session started.

When you are done with a VNC session, use the `vncserver` command to kill that session. For example, to end session `:1`, type:

```
$ vncserver -kill :1  
Killing Xvnc4 process ID 11651
```

Starting Up the VNC Client

With the VNC server running, you can connect to a desktop on that server from any of the client systems mentioned earlier (Windows, Linux, Mac OS X, UNIX, and so on). For example, assuming your VNC server is on a system named `myserver`, you could type the following command to **start that remote desktop** from another Linux system:

```
$ apt-get install vncviewer  
$ vncviewer myserver:1           Connect as chris on display 1  
Connected to RFB server, using protocol version 3.8  
Performing standard VNC authentication  
Password: *****  
$ vncviewer myserver:2           Connect as joe on display 2
```

Unless you define some commands to start up, you will only see the background screen for an X Window System display, with cross-hatching. To get beyond this, you need to run X Window programs on the server system, or from your client system, pointing to the VNC X display. For example:

```
$ xterm -display myserver:1 &  
$ metacity --display myserver:1 &
```

Note Most X Window programs specify which X server to use (the VNC server in this example) with a `-display` option. The `metacity` window manager, however, uses two leading dashes for `--display`.

You can also use `remmina` to connect; for this example, you would just specify `myserver:1` as the computer and VNC as the protocol. You can also type in the username and password or wait to be prompted for it.

Using VNC on Untrusted Networks with SSH

VNC is considered to be an insecure protocol. The password is sent using fairly weak encryption, and the rest of the session is not encrypted at all. For that reason, when using

VNC over an untrusted network or the Internet, we recommend you tunnel it over SSH.

For a general description of how the SSH service works, refer to the “Doing Remote Login and Tunneling with SSH” section earlier in this chapter. To forward VNC display 2 (port 5902) on the computer named `myserver` to the same local port, type the following:

```
$ ssh -L 5902:localhost:5902 myserver
```

Note If you start using VNC routinely, you may want to look at `tightvnc` (in the package of the same name). Although it’s not included with Ubuntu, `tightvnc` is another open source implementation of the VNC protocol, under active development and with newer features and optimizations. These features include built-in SSH tunneling.

Summary

If you ever find yourself in a position where you need to administer multiple Linux systems, you have a rich set of commands with Linux for doing remote system administration. The Secure Shell (SSH) facility offers encrypted communications between clients and servers for remote login, tunneling, and file transfer.

Virtual Network Computing (VNC) lets one Linux system share its desktop with a client system so that the remote desktop appears right on the client’s desktop. Both graphical tools (such as the Remmina Remote Desktop Client) and commands (such as `vncserver` and `vncviewer`) can be used to share remote desktops in Linux.

Chapter 14

Locking Down Security

IN THIS CHAPTER

- Adding user accounts and changing user settings with `useradd`
- Changing user accounts with `usermod`
- Deleting users with `userdel`
- Adding, changing, and managing passwords with `passwd`
- Managing groups with `groupadd`, `groupmod`, and `groupdel`
- Seeing who's logged in with `last`, `lastb`, and `who`
- Configuring firewalls with `iptables`
- Checking out advanced security with SELinux, tripwire, and APT

Securing your Linux system means first restricting access to the user accounts and services on that system. After that, security means checking that no one has gotten around the defenses you have set up.

Ubuntu, Debian, and other systems based on those Linux distributions are designed to be secure by default. That means that there are no user accounts with blank passwords, and that most network services (web, FTP, and so on) are off by default (even if their software is installed).

Although many of the commands covered in this book can be used to check and improve the security of your Linux system, some basic Linux features are particularly geared toward security. For example, secure user accounts with good password protection, a solid firewall, and consistent logging (and log monitoring) are critical to having a secure Linux system. Commands related to those features, plus some advanced features, such as SELinux and tripwire, are covered in this chapter.

Working with Users and Groups

During most Linux installation procedures, you are asked to assign a password to the root user (for system administration). Then you might be asked to create a username of your choice and assign a password to that as well (for everyday computer use). We encourage you to always log in as a regular user and only `su` or `sudo` to the root account

when necessary. Once Linux is installed, you can use commands or graphical tools to add more users, modify user accounts, and assign and change passwords.

Ubuntu enhances security by disabling logins as root, by default. Instead, during installation, you set up a user, with a password, who can perform administrative functions. On Ubuntu, use the `sudo` command to perform individual commands as the root user. The `sudo` command prompts for the administrative password, typically your password. This way, you avoid most issues of performing some command as root that you did not intend.

Managing Users the GUI Way

For an Ubuntu desktop system with X, you can manage users and groups with the User Manager window (System ⇒ Administration ⇒ Users and Groups).

When managing user accounts for servers, one option is to use web-based GUIs. The most commonly used general-purpose tool is Webmin (www.webmin.com). Make sure you do not run Webmin on its default port (10000) for security reasons. You can also use special-purpose web interfaces. For example, there are many web hosting automation GUIs, such as cPanel (www.cpanel.com), Plesk (www.swsoft.com/plesk), and Ensim (www.ensim.com).

Adding User Accounts

To add new users, you can use the `useradd` command. The only option that is required to add a new user is the username you are adding. You can see some of the default settings for adding a new user by entering the `-D` option:

<code>\$ useradd -D</code>	Show useradd default values
<code>GROUP=100</code>	Set group ID to 100 (users)
<code>HOME=/home</code>	Set base home directory to /home
<code>INACTIVE=-1</code>	Password expiration is disabled (-1)
<code>EXPIRE=</code>	Don't set date to disable user account
<code>SHELL=/bin/sh</code>	Set the default shell to /bin/bash
<code>SKEL=/etc/skel</code>	Copy config files from /etc/skel to \$HOME
<code>CREATE_MAIL_SPOOL=no</code>	Create a mail spool directory

Ubuntu and other Debian systems override the default group (100) and create a new group for every user. By default, the user ID assigned to the first user created is 1000 and the group ID is also 1000. The group name is the same as the username. The home directory is the username appended to `/home`. So, for example, if you created the first regular user account on the system as follows:

```
$ sudo useradd -m willz
```

The result would be a new user account with a `willz` username (UID 1001) and `willz` group name (GID 1001). The `-m` option tells `useradd` to create a home directory of `/home/willz`, and copy a set of configuration files (each beginning with a “.”) to the home directory from `/etc/skel`. The account would remain active indefinitely (no expiration date). **Add a password** as follows, and in most cases that’s all you need to do to have a working user account.

```
$ sudo passwd horatio
Changing password for user horatio
New UNIX password: *****
Retype new UNIX password: *****
passwd: all authentication tokens updated successfully.
```

Note Remember to use strong, non–dictionary-based passwords.

There are many options you can enter to **override the defaults** when you create a user. Combine the different options as you choose. Here are some examples:

\$ <u>sudo useradd -u 1101 -g 60 -m skolmes</u>	Use specific UID and GID
\$ <u>sudo useradd -m -d /home/jj jones</u>	Create /home/jj home
directory	
\$ <u>sudo useradd -m -G games,man timd</u>	Add user to games and man
groups	
\$ <u>sudo useradd -c "Tom G. Lotto" tlot</u>	Add full name to comment
field	
\$ <u>sudo useradd -s /bin/sh joeq</u>	Assign a new default shell
(tcsh)	
\$ <u>sudo useradd -e 2014-04-01 jerry</u>	New account expire April
01, 2014	
\$ <u>sudo useradd -f 0 jdoe</u>	Create a disabled account
\$ <u>sudo useradd -s /sbin/nologin billt</u>	Keep user from shelling in
\$ <u>sudo useradd billyq</u>	Prevent creation of home
directory	

Before you can add a user to a group, that group must exist (see the `groupadd` command in the “Adding Groups” section later in this chapter). The one exception is that, if no group is indicated, a group is created with the same name of the new user and the user is assigned to it. A user must belong to one initial group that can be overridden with `-g` and can also belong to supplementary groups, defined with `-G`.

To **list the group(s) that a user belongs to**, use the `groups` command:

```
$ groups timd                                List the groups that a user belongs to
timd : timd man games
```

The `-e` example, **sets an expiration date** for a user account that you know to be temporary. Change the default shell to `/usr/sbin/nologin` when you want a user to be able to access the computer (via FTP, POP3, and so on), but you don’t want to allow

access to a regular Linux login shell. Likewise, the last example, with no `-m` to create a home directory, might allow a user to access a machine but not have a home directory. Note that in all the examples, unless you provide the `-m` option, the `useradd` command will not create the home directory for the user.

Changing useradd Defaults

The default values you get when you create a new user account with `useradd` (default shell, GID, expiration dates, and so on) are set by values in the `/etc/login.defs` and `/etc/default/useradd` files. You can edit those files to change defaults or run the `useradd` command with the `-D` option to list or selectively change values:

```
$ useradd -D                                List default useradd
settings
$ sudo useradd -D -b /home2 -s /bin/csh      Set default base dir and
shell
$ sudo useradd -D -e 2014-01-01             Set all new users to
expire in 2014
```

As noted earlier, files and directories from the `/etc/skel` directory are copied to the new user's home directory when the account is created when you pass the `-m` option. Those files include some bash shell files and a link to an example directory. You can add other files and directories to `/etc/skel` so that each new user gets them. For example, if you are configuring a web server, you might create `public_html` and `public_ftp` directories for users to add web pages and files they want to share.

Modifying User Accounts

After a user account is created, you can **change values for that account with the `usermod` command**. Most options are the same ones you would use with `useradd`. For example:

```
$ sudo usermod -c "Thomas Lotto" tlot      Change user name in comment
field
$ sudo usermod -s /bin/sh joeq              Change default shell to sh
$ sudo usermod -L swanson                   Lock the swanson user
account
$ sudo usermod -U travis                     Unlock user account named
travis
```

Note that the last two examples lock and unlock a user account, respectively. Locking a user account does not remove the user's account from the system or delete any of the user's files and directories. However, it does keep the user from logging in. Once unlocked, the user's password is also restored, so a new password doesn't have to be

assigned if the same user is to resume using the account.

Locking an account can be useful if an employee is leaving the company, but the work in that employee's files needs to be passed to another person. Under those circumstances, locking the user instead of deleting it prevents the files owned by that user from appearing as belonging to an unassigned UID.

Because a regular user can't use the `useradd` or `usermod` command, there are special commands for **changing personal account information**. Consider the following examples:

```
$ chsh -s /bin/sh Change current user's shell to /bin/sh
$ sudo chsh -s /bin/sh chris Change a user's shell to /bin/sh
$ sudo chfn \
-h "212-555-1212" \ Change home phone number
-w "212-555-1957" Change office phone number
$ finger chris
Login: chris Name: Chris Negus
Office Phone: 919-555-1957 Home Phone: 919-555-1212
Directory: /home/chris Shell: /bin/bash
On since Sat Mar 9 14:43 (EST) on pts/0 from :0
10 seconds idle
(messages off)
Mail last read Sat Mar 9 16:26 2013 (EST)
No Plan.
```

The information changed with the previous `chfn` command (home and office phone numbers) and displayed with `finger` are stored in the fifth field of the `/etc/passwd` file for the selected user. (The `/etc/passwd` file can only be edited directly by the root user, and should only be edited using the `vipw` command and with extreme caution.)

On other versions of Linux, you can use the `-f` option to `chfn` to change your real or full name. On Ubuntu, the permission to do this is turned off by default. You can change this by editing `/etc/login.defs`. Look for the following line:

CHFN RESTRICT rwh

and change this to:

```
CHFN  RESTRICT                                frwh
```

Deleting User Accounts

With the `userdel` command, you can **remove user accounts** from the system as well as other files (home directories, mail spool files, and so on) if you choose. Consider the following examples:

```
$ sudo userdel jimbo          Delete user, not user's home directory
```

```
$ sudo userdel -r lily      Delete user, home directory, and mail
spool
```

Keep in mind that removing a user's home directory with `userdel` does not remove other files owned by that user in other locations on the system. Keep in mind the following issues:

- When you do a long listing of files that were owned by a deleted user (`ls -l`), the former user's UID and GID appear as the owners of the file.
- If you were to add a new user that was assigned to that UID and GID, the new user would become the owner of those files.

You can run the command `find / -nouser -ls` to **find files anywhere on the system that are not owned by any user**.

Managing Passwords

Adding or changing a password is usually done quite simply with the `passwd` command. However, there are additional options available with `passwd` that let an administrator manage such things as user account locking, password expiration, and warnings to change passwords. In addition to `passwd`, there are commands such as `chage`, `chfn`, and `vipw` for working with user passwords.

Regular users can **change only their own passwords**, whereas the root user can change the password for any user. For example:

```
$ passwd                                Change regular user's
password
```

```
Changing password for user chris.
```

```
Changing password for chris.
```

```
(current) UNIX password: *****
```

```
New UNIX password: *
```

```
BAD PASSWORD: it's WAY too short
```

```
New UNIX password: *****
```

```
Retype new UNIX password: *****
```

```
passwd: password updated successfully
```

```
$ sudo passwd joseph                    Root can change any user
password
```

```
Changing password for user joseph.
```

```
New UNIX password: *
```

```
Retype new UNIX password: *
```

```
passwd: password updated successfully
```

In the first example, a regular user (chris) changes his own password. Even while logged in, the user must type the current password before entering a new one. Also, `passwd` keeps a regular user from setting a password that is too short, based on a dictionary word, doesn't have enough different characters, or is otherwise easy to

guess. In the second example, the root user can change any user password without the old password.

Passwords should be at least eight characters, be a combination of letters and other characters (numbers, punctuation, and so on), and not include real words. Make passwords easy to remember but hard to guess.

A system administrator can use `passwd` to **lock and unlock user accounts**. For example:

```
$ sudo passwd -l carl           Lock the user account (carl)
Locking password for user carl.
passwd: Success
$ sudo passwd -u carl           Unlock a locked user account
(carl)
Unlocking password for user carl.
passwd: Success
$ sudo passwd -u jordan         Unlock fails with blank password
Unlocking password for user jordan.
passwd: Warning: unlocked password would be empty.
passwd: Unsafe operation (use -f to force)
```

Locking a user account with `passwd` causes an exclamation mark (!) to be placed at the front of the password field in the `/etc/shadow` file (where user passwords are stored). When a user account is unlocked, the exclamation mark is removed and the user's previous password is restored.

An administrator can use the `passwd` command to **require users to change passwords regularly**, as well as **warn users when passwords are about to expire**. To use the password expiration feature, the user account needs to have had password expiration enabled. The following examples use `passwd` to modify password expiration:

```
$ sudo passwd -n 2 vern         Set minimum password life to 2 days
$ sudo passwd -x 300 vern       Set maximum password life to 300 days
$ sudo passwd -w 10 vern        Warn of password expiration 10 days
before
$ sudo passwd -i 14 vern        Days after expiration account is
disabled
```

In the first example, the user must wait at least two days (`-n 2`) before changing to a new password. In the second, the user must change the password within 300 days (`-x 300`). In the next example, the user is warned 10 days before the password expires (`-w 10`). In the last example, the user account is disabled 14 days after the password expires (`-i 14`).

To **view password expiration**, you can use the `chage` command as follows:

```
$ sudo chage -l vern           View password expiration info
```

```

Last password change           : Mar 04, 2013
Password expires               : May 31,
2014
Password inactive             : Jun 14,
2014
Account expires               : never
Minimum number of days between password change : 2
Maximum number of days between password change : 300
Number of days of warning before password expires : 10

```

As system administrator, you can also use the `chage` command to manage password expiration. Besides being able to set minimum (`-m`), maximum (`-M`), and warning (`-W`) days for password expiration, `chage` can also set the day when a user must set a new password or a particular date the account becomes inactive:

```

$ sudo chage -I 40 frank           Make account inactive in 40 days
$ sudo chage -d 5 perry           Force user password to expire in 5
days

```

Instead of five days (`-d 5`), you can set that option to `0` and cause the user to have to set a new password the next time he or she logs in. For example, the next time the user `perry` logs in, if `-d 0` has been set, `perry` will be prompted for a new password as follows:

```

login: perry
Password: *****
You are required to change your password immediately (root
enforced)
Changing password for perry.
(current) UNIX password:
New UNIX password: *****
Retype new UNIX password: *****

```

Adding Groups

Each new user is assigned to one or more groups. You can create groups at any time and add users to those groups. The permissions that each group has to use files and directories in Linux depend on how the group permission bits are set on each item. Assigning users to a group allows you to attach ownership to files, directories, and applications so that those users can work together on a project or have common access to resources.

Commands similar to those for working with users are available for managing your groups. You can add groups (`groupadd`), change group settings (`groupmod`), delete groups (`groupdel`), and add and delete members from those groups (`groupmems`). Here are some examples for **adding new groups** with the `groupadd` command:

```
$ sudo groupadd marketing           Create new group with next GID
$ sudo groupadd -g 1701 sales       Create new group with GID of
1701
$ sudo groupadd -o -g 74 myssh      Create group with existing GID
```

With the `groupmod` command, you can **change the name or group ID** of an existing group. Consider the following examples:

```
$ sudo groupmod -g 491 myadmin      Modify myadmin to use GID 491
$ sudo groupmod -n myad myadmin    Change name of myadmin group to
myad
```

To remove an existing group, use the `groupdel` command. Here is an example:

```
$ sudo groupdel myad               Remove existing myad group
```

Keep in mind that removing a group or user doesn't remove the files, directories, devices, or other items owned by that group or user. If you do a long listing (`ls -l`) of a file or directory assigned to a user or group that was deleted, the UID or GID of the deleted user or group is displayed.

Checking on Users

After you have created user accounts, and let those users loose on your computer, there are several different commands you can use to keep track of how they are using your computer. The following commands for checking on user activity on your Linux system are covered in other chapters:

- Use the `find` command (see Chapter 4) to search the system for files anywhere on the system that are owned by selected users.
- Use the `du` command (see Chapter 7) to see how much disk space has been used in selected users' home directories.
- Use commands such as `fuser`, `ps`, and `top` (see Chapter 9) to find out which processes users are running.

Aside from the commands just mentioned, there are commands for checking such things as who is logged into your system and getting general information about the users with accounts on your system. The following are examples of commands for **getting information about people logging into your system**:

```
$ last                               List the most recent successful logins
greek      tty3                      Tue Mar  5 18:05      still logged
in
chris      tty1                      Mon Mar  4 13:39      still logged
in
root      pts/4                      thompson          Tue Mar  5 14:02      still logged
```

```

in
chris    pts/1            :0.0            Mon Mar  4 15:47    still logged
in
jim      pts/0            10.0.0.50       Sun Mar  3 13:46 - 15:40
(01:53)
james    pts/2            Sat Mar  2 11:14 - 13:38
(2+02:24)
$ last -a                Makes it easier to read the remote client
hostname
$ sudo lastb             List the most recent unsuccessful logins
julian    ssh:notty        ritchie         Wed Mar  6 12:28 - 12:28
(00:00)
morris    ssh:notty        thompson        Thu Feb 28 13:08 - 13:08
(00:00)
baboon    ssh:notty        10.0.0.50       Thu Feb  8 09:40 - 09:40
(00:00)
james    ssh:notty        000db9034dce.cli Fri Jun 22 17:23 - 17:23
(00:00)
$ who -u                 List who is currently logged in (long
form)
greek     tty3        2013-08-05 18:05 17:24        18121
jim       pts/0        2013-08-06 12:29 .            20959
(server1.example.com)
root      pts/3        2013-08-04 18:18 13:46        17982
(server2.example.com)
james     pts/2        2013-07-31 23:05 old          4700
(0a0d9b34x.example.com)
chris     pts/1        2013-08-04 15:47 old          17502 (:0.0)
$ users                 List who is currently logged in (short
form)
chris james greek jim root

```

With the `last` command, you can see when each user logged in (or opened a new shell) and either how long they were logged in or a note that they are “still logged in.” The `tty1` and `tty3` terminal lines show users working from virtual terminals on the console. The `pts` lines indicate a person opening a shell from a remote computer (`thompson`) or local X display (`:0.0`). We recommend you use the `-a` option for improved readability. The `lastb` command shows failed login attempts and where they are from. The `who -u` and `users` commands show information on currently logged-in users.

Here are some commands for **finding out more about individual users** on your system:

```

$ id                    Your identity (UID, GID and group for current
shell)
uid=1000(chris) gid=1000(chris)
groups=4(adm),20(dialout),24(cdrom),25(floppy),

```

```

29(audio),
30(dip),44(video),46(plugdev),104(scanner),112(netdev),
113(lpadmin),
115(powerdev),117(admin),1000(chris)
$ who am i           Your identity (user, tty, login date, location)
chris      pts/0       Mar 3 21:40 (:0.0)
$ finger -s chris    User information (short)
Login      Name        Tty      Idle      Login Time      Office      Office
Phone
chris      Chris Negus  tty1      1d       Mar  4 13:39    A-111      555-1212
$ finger -l chris    User information (long)
Login: chris                               Name: Chris Negus
Directory: /home/chris                     Shell: /bin/bash
Office: A-111, 555-1212                     Home Phone: 555-2323
On since Mon Mar  4 13:39 (CDT) on tty1      2 days idle
New mail received Wed Mar  6 13:46 2013 (CDT)
      Unread since Mon Mar  4 09:32 2013 (CDT)
No Plan.

```

In addition to displaying basic information about the user (login, name, home directory, shell, and so on), the `finger` command will also display any information stored in special files in the user's home directory. For example, the contents of the user's `~/.plan` and `~/.project` files, if those files exist, are displayed at the end of the `finger` output. With a one-line `.project` file and multi-line `.plan` file, output can appear as follows:

```

$ finger -l chris    User information (long, .project and .plan
files)

...
Project:
My project is to take over the world.
Plan:
My grand plan is
to take over the world
by installing Linux on every computer

```

Configuring the Built-In Firewall

A firewall is a critical tool for keeping your computer safe from intruders over the Internet or other network. It can protect your computer by checking every packet of data that comes into, out of, or through your computer's network interfaces, and then making a decision about what to do with that packet based on the parameters you set.

The firewall facility built into the current Linux kernel is netfilter, although it is more commonly referred to by the name of the command and service that represents it:

iptables. When you install Ubuntu, the iptables command is installed and iptables features are built into the kernel. So you are ready to immediately begin creating firewall rules for your system.

The iptables facility (www.netfilter.org) is extraordinarily powerful, yet complex to use from the command line. For that reason, many people set up their basic firewall rules using a graphical interface. To get a graphical interface, install the firestarter package (`sudo apt-get install firestarter`).

Firestarter provides a wizard to configure and set up your firewall. To run Firestarter, type the following from a Terminal window on your desktop:

```
# sudo firestarter
```

You can also try add-on packages such as FWBuilder (fwbuilder package) and Shorewall (shorewall package) for graphically configuring firewalls.

Note Before you go much further, read the IpTables HowTo document for Ubuntu, at <https://help.ubuntu.com/community/IptablesHowTo>. This document provides a lot of useful information for using iptables on Ubuntu, as this usage differs a lot from other versions of Linux such as Fedora.

Understanding iptables Firewalls

The iptables facility works by examining every packet that traverses your Linux system's network interfaces and acting on those packets based on rules you set up. Rules are added to one of the **tables** associated with iptables. Available tables are filter, nat, and mangle.

The filter table is, by far, the one you will work with the most. Based on attributes you choose (such as the port requested, the host that sent the packet, or network interface receiving the packet) an action is taken. The action may be to allow (ACCEPT), deny (REJECT or DROP), or log and continue (LOG) the request from each packet.

Within each table, each rule is also assigned a particular **chain**. The filter table has three chains:

- **INPUT**—For packets coming into the system
- **OUTPUT**—For packets leaving the system
- **FORWARD**—For packets being forwarded through the system

Probably most of your packet filter rules will be created for the INPUT chain. The INPUT chain filters packets coming into your system, essentially blocking or allowing access to services on your system. A common practice is to allow access to packets coming from your local system (lo interface), associated with established connections,

and requesting one of a handful of services (such as ssh or http). Then the system might drop all other packets destined for the system.

Besides the filter table, you have the nat table which is used to do such things as network address translation and packet forwarding. There is also the mangle table, which can be used to change packet information (such as to appear to come from a different address than it is actually from).

The steps for configuring a firewall are basically the following:

- Create rules that can be injected into the running kernel using the `iptables` command.
- Save those rules to a file.
- Load those rules before your network interfaces come up.

The `iptables` examples shown in this section illustrate a good sample set of filter firewall rules you can start with on a system that allows access to a few services on the local system but blocks all other access.

Note The order of the rules is important. Rules are read in order. When a packet matches a rule, it leaves the chain (being accepted, dropped, or rejected). Using `-A` adds a rule to the end of the chain. With `-I #` you insert a rule into a particular line number (#) in the chain.

First, flush all the current `iptables` rules in the kernel:

```
$ sudo iptables -F
```

This rule allows any packets coming into the system that are associated with existing connections (for example, to return data from a web page requested from a local browser):

```
$ sudo iptables -A INPUT -m state \  
--state ESTABLISHED,RELATED -j ACCEPT
```

This allows ICMP requests (such as ping requests):

```
$ sudo iptables -A INPUT -p icmp -j ACCEPT
```

This allows any requests from the local system:

```
$ sudo iptables -A INPUT -i lo -j ACCEPT
```

This allows outside requests to TCP port 22 (SSH service) to enable such things as remote login and remote copy services (repeat this command with other port numbers to allow access to other services):

```
$ sudo iptables -A INPUT -m state --state NEW \  
-m tcp -p tcp --dport 22 -j ACCEPT
```

These two rules allow printing service requests (port 631) for both TCP and UDP

protocols:

```
$ sudo iptables -A INPUT -m state --state NEW \
-m udp -p udp --dport 631 -j ACCEPT
$ sudo iptables -A INPUT -m state --state NEW
-m tcp -p tcp --dport 631 -j ACCEPT
```

This rule drops any packet that is not already explicitly accepted:

```
$ sudo iptables -A INPUT -j REJECT \
--reject-with icmp-host-prohibited
```

This command saves the current rules from the kernel to a file:

```
$ sudo iptables-save > /root/myiptables
```

This command restores rules from a file to the kernel:

```
$ sudo iptables-restore /root/myiptables
```

You can add this command to a startup script, as described in the section "Saving and Reloading Firewall Rules," to load the iptables firewall rules before network interfaces are started.

Listing iptables Rules

To check how the firewall is set up on your system, use the `-L` option of `iptables`. Here is how to **list the current rules set** on your Linux system's firewall (for the default filter table):

```
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source      destination
ACCEPT    all  --  anywhere   anywhere    state RELATED,ESTABLISHED
ACCEPT    icmp --  anywhere   anywhere
ACCEPT    all  --  anywhere   anywhere
ACCEPT    tcp  --  anywhere   anywhere    state NEW tcp dpt:ssh
ACCEPT    udp  --  anywhere   anywhere    state NEW udp dpt:ipp
ACCEPT    tcp  --  anywhere   anywhere    state NEW tcp dpt:ipp
REJECT    all  --  anywhere   anywhere    reject-with icmp-host-prohibited
Chain FORWARD (policy ACCEPT)
target    prot opt source      destination
Chain OUTPUT (policy ACCEPT)
target    prot opt source      destination
```

Add a `-v` option to see a more verbose output of rules. Notice here that `-v` shows that the "all anywhere anywhere" rule actually accepts all packets from the local system (lo). You can also see the port numbers instead of names and the number of packets and bytes that have matched each rule:


```
$ sudo iptables -vnL Verbose, numeric services, packet counts
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target prot opt in  out source      destination
    85  6514 ACCEPT all  --  *   *   0.0.0.0/0    0.0.0.0/0
        state RELATED,ESTABLISHED
     0     0 ACCEPT icmp --  *   *   0.0.0.0/0    0.0.0.0/0
     0     0 ACCEPT all  --  lo  *   0.0.0.0/0    0.0.0.0/0
     0     0 ACCEPT tcp  --  *   *   0.0.0.0/0    0.0.0.0/0
        state NEW tcp dpt:22
    24  5144 ACCEPT udp  --  *   *   0.0.0.0/0    0.0.0.0/0
        state NEW udp dpt:631
     0     0 ACCEPT tcp  --  *   *   0.0.0.0/0    0.0.0.0/0
        state NEW tcp dpt:631
```

The example illustrates the filter iptables firewall table. It shows that for packets coming into the computer's network interfaces, packets for Internet Printing Protocol (ipp) on udp and tcp protocols are allowed. Likewise, tcp packets matching the Secure Shell (ssh) are accepted. Packets are also accepted if they are associated with an established connection. Next you can **look at the nat table**:

```
$ sudo iptables -t nat -L Display current iptables nat
table
Chain PREROUTING (policy ACCEPT)
target prot opt source      destination
DNAT   tcp  --  0.0.0.0/0    11.22.33.44  tcp dpt:8785
to:10.0.0.155:22
DROP   tcp  --  0.0.0.0/0    0.0.0.0/0    tcp dpt:135
DROP   udp  --  0.0.0.0/0    0.0.0.0/0    udp dpt:135
Chain POSTROUTING (policy ACCEPT)
target prot opt source      destination
MASQUERADE all  --  0.0.0.0/0    0.0.0.0/0
Chain OUTPUT (policy ACCEPT)
target prot opt source      destination
```

The nat table just shown applies to a feature called Network Address Translation. The nat table allows you to do such things as use private IP addresses behind your firewall. As the packets from internal LAN machines exit the firewall, the source private address is rewritten with the IP address of the firewall's external interface. The firewall keeps track of these sessions in order to allow the return traffic through to the LAN machines. All this is configured with the MASQUERADE line on the POSTROUTING chain.

In the preceding example, the DNAT line in the PREROUTING chain causes any requests to port 8785, at IP address 11.22.33.44, to be forwarded to the internal LAN IP address 10.0.0.155 on port 22 (a trick to let someone ssh into a computer behind the firewall through a non-standard port).

The following are other examples for **listing information about your firewall**.

```
$ sudo iptables -L --line-numbers      Show line number for each rule
$ sudo iptables -nvL --line-numbers    Add numeric (-n) and verbose (-v)
```

You can **stop or flush all iptables rules** on an Ubuntu system as follows:

```
$ sudo iptables -F                      Flush all iptables rules
```

This command removes all the rules, so be careful when running it. You'll want to immediately add new rules.

Setting Other Firewall Rules

Here are some more examples of how the `iptables` command can be used to **change rules on an active firewall**:

```
$ sudo iptables -A INPUT -p TCP \      Add filter input rule for TCP
-i eth0 \                             on the first Ethernet interface
--destination-port 25 \              destined for mail service port (25)
-j ACCEPT                             accept the packets when
encountered
$ sudo iptables -t nat \                Add nat rule
-A POSTROUTING \                     POSTROUTING chain
-o eth1 \                             packets received on eth1 interface
-j SNAT \                             jump to network address translation
--to-source 11.22.33.1 \             using outgoing address
11.22.33.1
```

Of the two examples shown, the first example creates a rule that allows new incoming requests to your system on port 25. This is presumably because you have configured your computer as a mail server (with sendmail, postfix, or other SMTP service). The second example creates a nat table rule to allow the firewall to do Source Network Address Translation (SNAT). The SNAT feature lets you have private IP addresses behind your firewall that can communicate to the public Internet using the firewall's external IP address.

To use SNAT or any other form of network address translation, you must **also enable IP forwarding** on the machine. This can be done by editing the `/etc/sysctl.conf` file and uncommenting the following variable:

```
net.ipv4.ip_forward=1
```

In cases where you have an Internet-facing service offered on a machine behind your firewall, you can instruct the firewall to **forward requests for that service** to that machine. The following example uses a feature called **port forwarding** to pass requests for a service through the firewall to the destination machine behind the firewall:

```
$ sudo iptables -t nat -A PREROUTING \  Add nat PREROUTING rule
```

<code>-p tcp -d 11.22.33.1 \</code>	tcp requests on 11.22.33.1
<code>--dport 80 \</code>	for port 80 (Web service)
<code>-j DNAT \</code>	jump to the DNAT target
<code>--to-destination 10.0.0.2</code>	forward the packets to 10.0.0.2

There are many other types of rules you can create to change how your firewall behaves. Refer to the iptables man page or the Netfilter website (www.netfilter.org) for further information on using the iptables facility.

After making the preceding changes, you'll see the following rules defined:

```
$ sudo iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target      prot opt source      destination
DNAT        tcp  --  anywhere    11.22.33.1  tcp dpt:www
to:10.0.0.2
Chain POSTROUTING (policy ACCEPT)
target      prot opt source      destination
SNAT        0    --  anywhere    anywhere    to:11.22.33.1
Chain OUTPUT (policy ACCEPT)
target      prot opt source      destination
```

Saving and Reloading Firewall Rules

All of the changes to the iptables rules that you simply type into the shell apply only to the current kernel session. Without somehow saving those rules, when you next reboot Ubuntu all the rules you entered will be erased, which is probably not what you want. To preserve your iptables rules, run `iptables-save`:

```
$ sudo iptables-save > iptables.rules  Save rules to a file in the
current directory
$ sudo cp iptables.rules /etc          Copy the saved rules to /etc
```

The two-step process is required because of the restrictions on the `/etc` directory. (You can change those restrictions, but that probably isn't a good idea.) Thus far, the rules are saved for later usage.

Next, you can configure Ubuntu to load these saved rules each time it enables an Ethernet interface (this is specific to each network card or interfaces on your system). Edit the `/etc/network/interfaces` file. After the `iface` configuration for an Ethernet interface, such as `eth0`, make a call to `iptables-restore`, as shown in this snippet:

```
auto eth0
iface eth0 inet dhcp
    pre-up iptables-restore < /etc/iptables.rules
```

This addition to the interface files calls `iptables-restore` to restore the rules saved previously to the file `/etc/iptables.rules`.

Using Advanced Security Features

A dozen or so pages covering security-related commands are not nearly enough to address the depth of security tools available to you as a Linux system administrator. Beyond the commands covered in this chapter, here are descriptions of some features you may want to look into to further secure your Linux system:

- **Security Enhanced Linux (SELinux)**—The SELinux feature provides a means of securing the files, directories, and applications in your Linux system in such a way that exploitation of one of those areas of your system cannot be used to breach other areas. For example, if intruders were to compromise your web daemon, they wouldn't necessarily be able to compromise the rest of the system.

SELinux was developed by the U.S. National Security Agency (NSA), which hosts a related FAQ at www.nsa.gov/selinux/info/faq.cfm. You need to install SELinux as separate packages. See <https://wiki.ubuntu.com/SELinux> for details.

- **Central logging**—If you're managing more than a couple of Linux servers, it becomes preferable to have all your systems log to a central syslog server. When you implement your syslog server, you may want to explore using syslog-ng. Also, if you outgrow logwatch, you should consider using a log parser such as Splunk.
- **Tripwire**—Using the tripwire package, you can take a snapshot of all the files on your system, and then later use that snapshot to find if any of those files have been changed. This is particularly useful to find out if any applications have been modified that should not have been. First, you take a baseline of your system file. Then at regular intervals, you run a tripwire integrity check to see if any of your applications or configuration files have been modified.
- **APT database**—Another way to check if any of your applications have been modified is by using the `APT` commands to validate the applications and configuration files you have installed on your system. See Chapter 2 for information on using the `APT` and `dpkg` commands to verify the contents of installed packages.
- **chkrootkit**—If you suspect your system has been compromised, download and build `chkrootkit` from www.chkrootkit.org. This will help you detect rootkits that may have been used to take over your machine. We recommend you run `chkrootkit` from a LiveCD or after mounting the suspected drive on a clean system.

Summary

While many tools are available for securing your Linux system, the first line of security

starts with securing the user accounts on your system and the services that run on your system. Commands such as `useradd`, `groupadd`, and `password` are standard tools for setting up user and group accounts.

Because most serious security breaches outside your organization can come from intruders accessing your systems on public networks, setting up secure firewalls is important for any system connected to the Internet. The `iptables` facility provides the firewall features that are built into the Linux kernel.

Chapter 15

Setting Up a Virtualization Host and Virtual Machines

IN THIS CHAPTER

- Checking your computer for virtualization support
- Installing virtualization software
- Creating and managing VMs with `virt-manager`
- Working with VMs from the command line

By using your Ubuntu system as a virtualization host, you can run multiple computer operating systems on a single computer. The systems you create on the host are referred to as virtual machines (VMs). A VM can be running Microsoft Windows, Fedora, another Linux system, or just about any operating system that can run directly on the computer architecture of the host.

Once a VM is installed, you can work with it in much the same way as you would work with operating systems installed directly on computer hardware. However, with VMs, it's easier to duplicate them, migrate them to other virtual hosts to improve performance, or configure them to failover to another host when a host becomes inoperable. With VMs, you can make more efficient use of your computer infrastructure.

There are lots of reasons for setting up a virtual host. For example, you may want to:

- Try out a different operating system without consuming a whole computer.
- Run an application that requires a specific operating system version and configuration that's different from what you normally run.
- Configure a system to test a new application, without interrupting the other work on your computer.
- Create a copy of an installed operating system and quickly spin off a new version from that copy.
- Experiment with an operating system in a way that doesn't disrupt the host system.

By using your Ubuntu system as a virtualization host, you can start building a computer infrastructure that can scale up as you need more computing power. By having multiple hosts, you can migrate your virtual machines to get better performance or shut down hosts that you underutilize.

This chapter describes how to configure Ubuntu as a virtualization host using Kernel-based Virtual Machine (KVM), which is a feature built into the Linux kernel. It then describes commands you can use to create and manage virtual machines.

Can Your Computer Support Virtualization?

The most critical requirement for running Ubuntu as a KVM virtualization host is to have a CPU that supports virtualization. Beyond that, you mostly need to make sure that the computer has enough resources to effectively run virtual machines.

Besides the right CPU, the KVM host needs to have enough memory (RAM) and disk space available to provide the appropriate level of performance needed to run your VMs. Likewise, you need to plan for the network bandwidth needed to provide the level of service required.

The following sections describe how to check your computer to make sure that you have the right assets available to run as a virtualization host.

Checking for CPU Virtualization Support

To use KVM, the processors on your system must support either Intel VT technology or AMD-V virtualization support. Ubuntu has a command named `kvm-ok` available from the `cpu-checker` package that you can use to test your CPU for virtualization support. Here's how to **install and use the `kvm-ok` command**:

```
$ sudo apt-get install cpu-checker
$ sudo kvm-ok
INFO: /dev/kvm does not exist
HINT:   sudo modprobe kvm_intel
INFO: Your CPU supports KVM extensions
KVM acceleration can be used
```

The output from `kvm-ok` indicates that the CPU does support the proper CPU extensions needed by KVM. However, the software needed to use KVM is not yet installed and the modules needed are not yet loaded. I'll do that later.

In the meantime, there's a more manual way to **check for virtualization support**. You can check the flags set for the CPU in `/proc/cpuinfo`. Using the `egrep` command, you can search that file for Intel-VT support (`vmx`) or AMD-V support (`svm`) as follows:

```
$ egrep "(svm|vmx)" /proc/cpuinfo
flags      : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
```

```
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe syscall nx lm constant_tsc arch_perfmon pebs bts rep_good nopl
aperfmpperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr
pdcm xsave lahf_lm dtherm tpr_shadow vnmi flexpriority
```

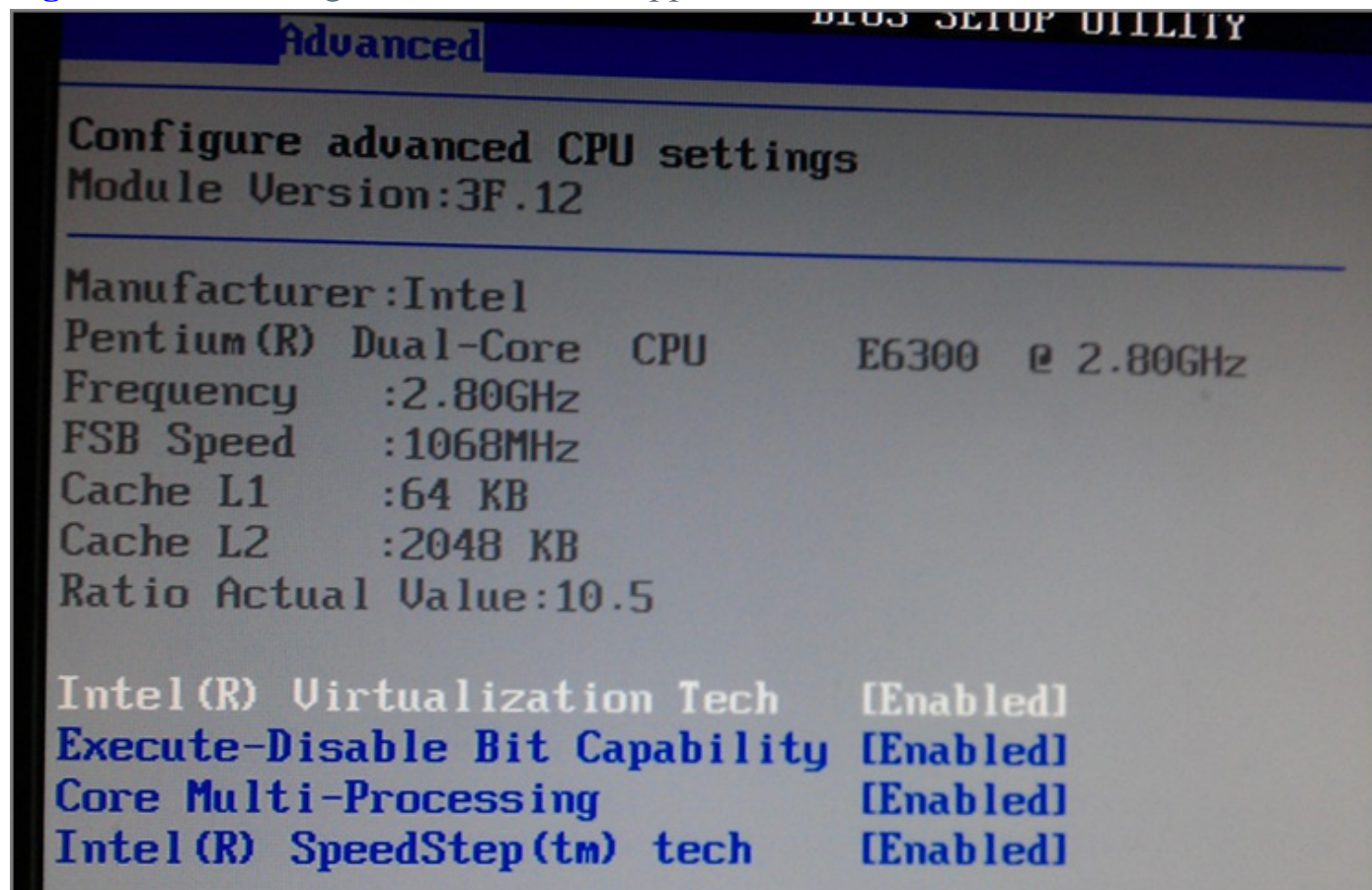
You should see either the `vmx` or `svm` flag (but not both). If you don't get any output from the `egrep` command, it means that the computer's CPU(s) doesn't support KVM virtualization. At that point, the next step is to check the BIOS.

Enabling Virtualization Support in the BIOS

If you believe that your computer can support virtualization, but the appropriate flag is not set, you may need to **turn on virtualization support in the BIOS**. To do that, reboot your computer and interrupt the boot process when you see the first BIOS screen. The screen will probably say something like “Press F12 to go into Setup.” Press the Delete key, Function key, or other key that is noted.

From the BIOS screen that appears, look for something like a CPU or Performance heading and select it. Then look for a virtualization selection such as “Intel Virtualization Tech” and enable it. [Figure 15-1](#) shows an example of such a BIOS screen with virtualization enabled.

Figure 15-1: Turning on virtualization support in the BIOS



After you change the virtualization BIOS setting and save it, you should power down the computer to make sure the BIOS settings take effect. After that, you can check the other features of your computer.

Is the Host Computer 32- or 64-Bit?

If at all possible, use a 64-bit computer as your virtualization host. Some operating systems, such as Red Hat Enterprise Linux, don't even support KVM on 32-bit systems. One of the main drawbacks of a 32-bit KVM host is that each VM is limited to 2GB of memory.

To **check if your computer is a 32- or 64-bit computer**, you can examine the CPU flags (much as you did when you were checking for virtualization support). Here's how:

```
$ sudo egrep lm /proc/cpuinfo
Flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe syscall nx lm constant_tsc arch_perfmon pebs bts rep_good nopl
aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr
pdc_m xsave lahf_lm dtherm tpr_shadow vnmi flexpriority
```

The flag `lm` stands for long mode. If the flag is present, it means that the processor is a 64-bit processor.

Checking Available RAM and Disk Space

Because you are running multiple operating systems on one physical computer, the amount of RAM and disk space needed is likewise multiplied. When thinking about how much RAM and disk space you need on your virtualization host, consider the following:

- For RAM, you need enough memory to service the host plus all the VMs you plan to run at the same time.
- You can over-commit memory, meaning that you can allocate more total memory to the VMs that are running than is actually available. The assumption is that not all VMs will need all the memory they are allocated at the same time.
- Although with KVM, the disk space used for a VM's virtual image can be consumed from the hard disk on the local host, if you have more than one host, consider using some sort of networked storage (such as iSCSI). Having shared storage allows you to do live migration of your VMs to other hosts.

To **check your available memory**, use the `free` command:

```
$ free -m
```

	total	used	free	shared	buffers	cached
Mem:	3920	3442	478	0	165	2616

```
-/+ buffers/cache:      659      3261
Swap:                4059          0      4059
```

The total amount of RAM on this system is 4GB (3920MB). This is not very much RAM for a virtualization host. However, I can run one or two VMs simultaneously on this system, just to confirm that virtualization is working.

Keep in mind that as you evaluate how much memory you need, you can use a command such as `top` to get a sense of how much memory you require. When adding up actual memory used in `top`, total the resident memory column and not the virtual memory (which indicates how much each process allocated, but is usually more than the amount actually used).

For disk space, you want to make sure that there is plenty available in the directory that will store the disk images used by the VMs. By default, the location is: `/var/lib/libvirt/images`. So try the `df` command to **check the disk space available** in that location:

Note The `/var/lib/libvirt/images` directory won't be created until the `libvirt-bin` package is installed in the next section, so you can just use the `/var/lib` directory if it's not installed yet.

```
$ df -h /var/lib/libvirt/images
Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/ubuntutb-root 88G   12G   72G   14% /
```

In this case, the selected directory is part of the root (`/`) filesystem, which has 72GB of available space. What some people do is mount a filesystem from some form of networked storage or create an LVM logical volume on the local system that can be grown as needed. In any case, 72GB is enough to try out a couple of VMs.

Adding Virtualization Software

The software components you want to add to perform KVM virtualization include:

- **libvirt**—Provides the interface to the virtualization hardware
- **qemu**—Emulates PC hardware to the virtual machines
- **bridge-utils**—Offers a way to bridge your networking from the virtual machines through the host

Run the following `apt-get` command to **install the basic software needed for KVM** virtualization:

```
$ sudo apt-get install libvirt-bin kvm bridge-utils qemu-common \
    qemu-kvm qemu-utils
```

In addition to the basic KVM software, you can add a graphical interface for

managing your virtual machines. Here's how to install the `virt-manager` graphical software for managing VMs:

```
$ sudo apt-get install virt-manager
```

With `virt-manager` installed, you now have the choice of managing your virtual machines from a graphical interface or from the command line. Next, you want to make sure that the user account you want to manage virtualization is configured to do so.

Adding Your User Account to `libvirt`

If the user account you are going to use to manage KVM is not a member of the `libvirt` group, that user needs to be added to the group. For example, if you want the user account `chris` to manage virtualization, here's how to **add the user to the `libvirt` group**:

```
$ sudo adduser chris libvirt  
Adding user `chris' to group `libvirt' ...  
Adding user chris to group libvirt  
Done.
```

At this point, your system should be ready to start using virtualization. However, you may want to reboot your system to make sure that all the necessary services are up and running and that the user account you just added to the `libvirt` group is logged in and ready.

Managing Virtual Machines with `virt-manager`

The Virtual Machine Manager (`virt-manager`) graphical interface is a popular tool for managing your KVM virtual machines. Before using commands, such as `virt-install` (to install a new VM) or `virsh` (to manage VMs), you might want to try out `virt-manager`, which can lead you through the creation of your first VMs in an intuitive way.

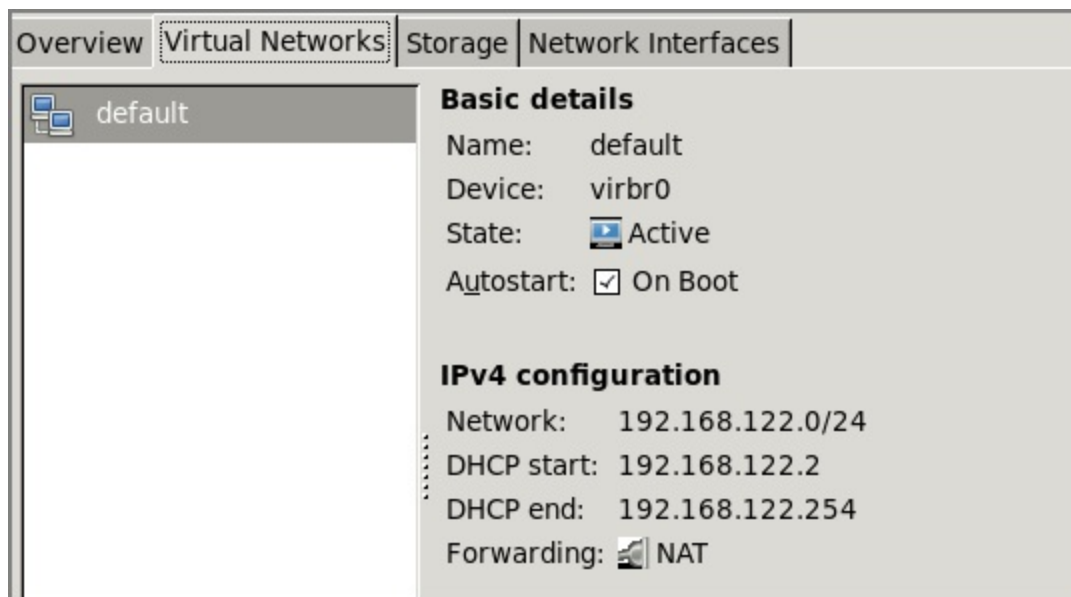
Here are a few things you need to do to get started with `virt-manager`:

- **Get ISO images.** Download ISO images of the operating systems you either want to install or run live as a VM. For the examples in this chapter, I use Ubuntu and Fedora live CDs as installation and live media.
- **Start `virt-manager`.** From the Ubuntu Dashboard, choose the Virtual Machine Manager icon (or run `virt-manager` from the command line) as the user you added to the `kvm` group. The Virtual Machine Manager window will appear.

Before you create your first VM, explore your Virtual Machine Manager window for information about your virtualization settings. Here are some examples:

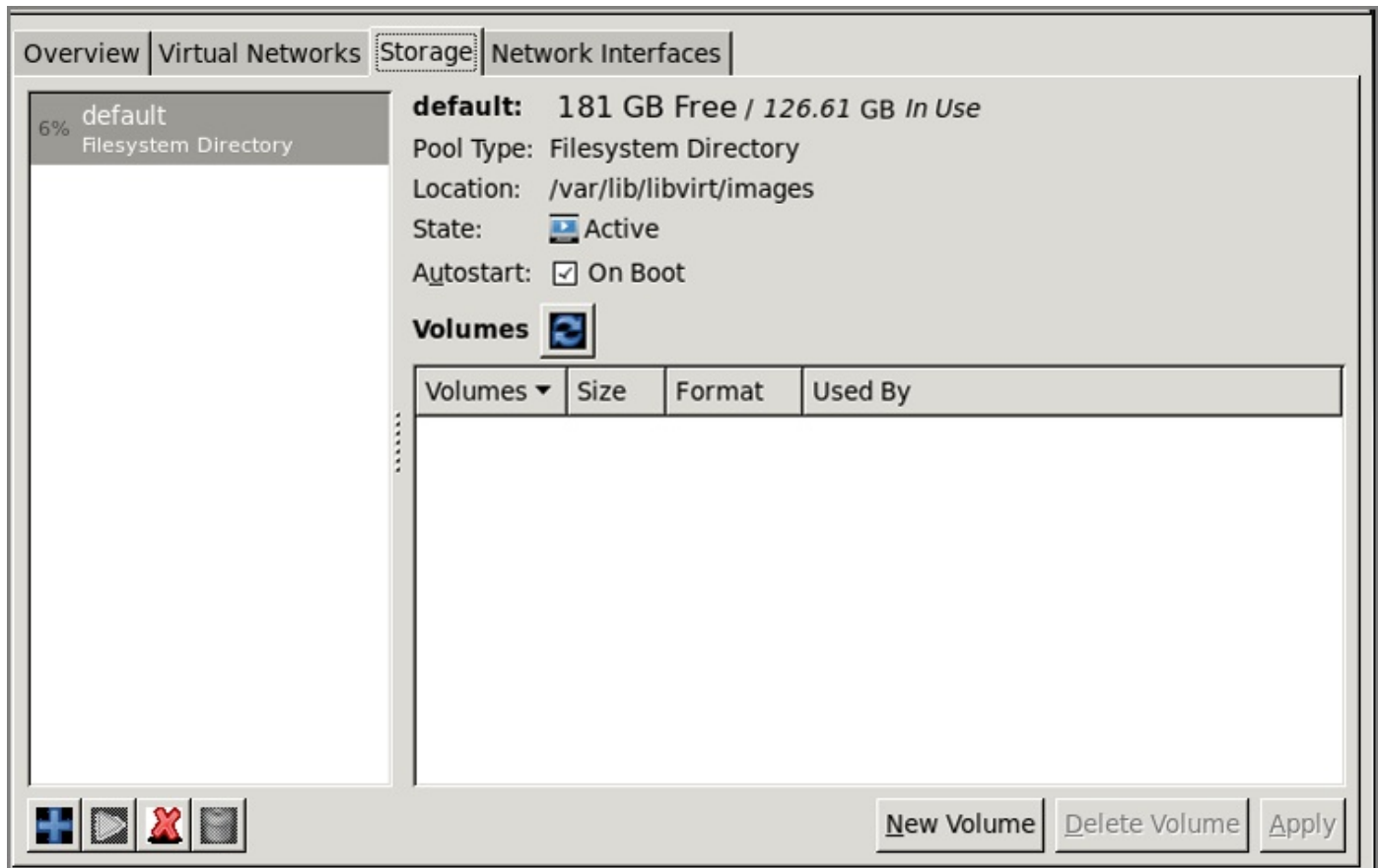
- **Virtual networks**—Select Edit and then Connection Details. From the Connection Details window that appears, choose the Virtual Networks tab. You can see that the default network managed by your virtual host is 192.168.122.0/24. The virtual host will hand out IP addresses to VMs from the range 192.168.122.2 to 192.168.122.254. To make this work, the host does network address translation (NAT) on device virbr0. [Figure 15-2](#) shows an example of this window.

[Figure 15-2](#): The default virtual network uses NAT and a private pool of network addresses.



- **Storage**—Again, select Edit and then Connection Details. From the Connection Details window, choose the Storage tab. The `/var/lib/libvirt/images` directory is the default location for storing volumes. In the example shown in [Figure 15-3](#), you can see the amount of free disk space and a list of any existing volumes. You can create new volumes here or do that when you create your VMs.

[Figure 15-3](#): Check available free space for volumes.



With everything in place, you can begin configuring and installing your first VM.

Creating a Virtual Machine in virt-manager

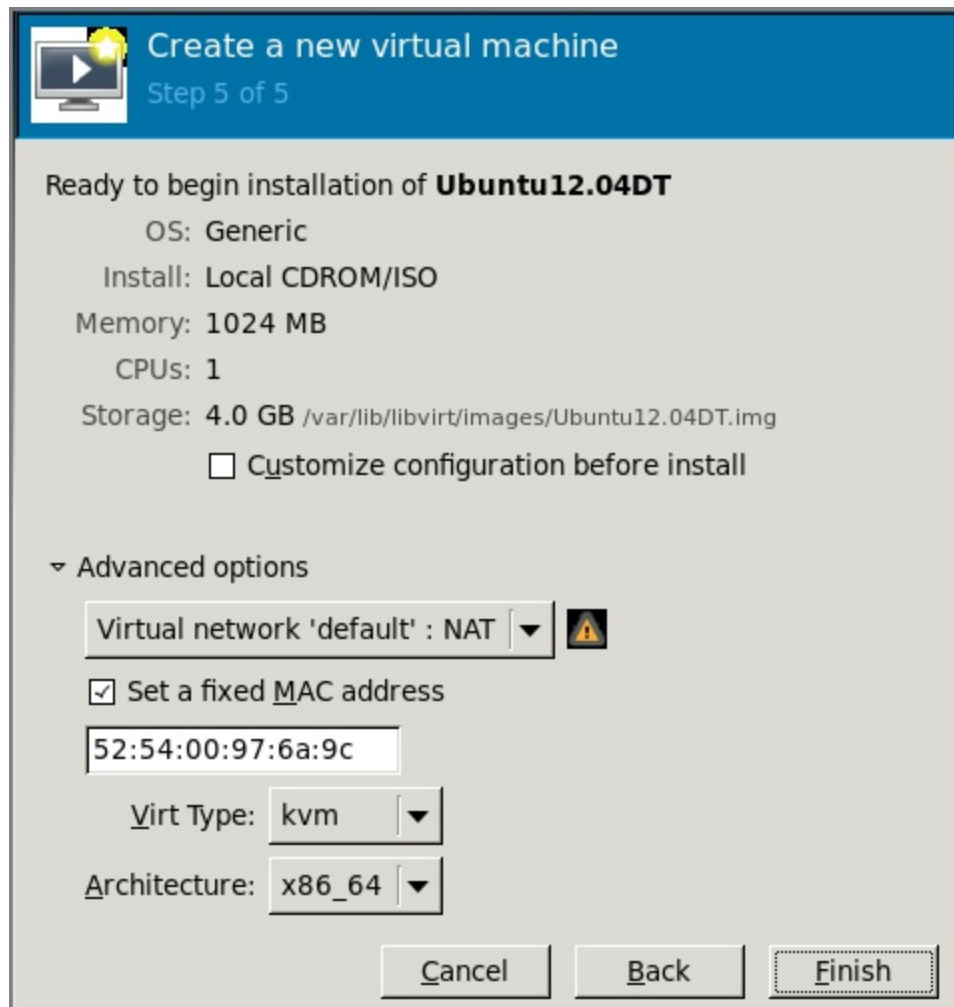
To begin creating a new VM, select the “Create a new virtual machine” icon from the Virtual Machine Manager window. The following is a list of steps you go through to create a new virtual machine:

- 1. Name and install the type.** Type a name for the virtual machine and select how you would like to install it. I used an ISO image that I downloaded to the local host. If you have installation media available from a PXE server or network server (HTTP, FTP, or NFS), choose either of those selections. Then click Forward.
- 2. Install the media and operating system type.** Choose the exact location of your installation medium. I chose “Use ISO image” and clicked the Browse button to browse to select my ISO image from the local filesystem. I left the operating system as generic. Then click Forward.
- 3. Memory and CPU.** Choose the amount of RAM the VM will consume and the number of CPUs. Then click Forward.
- 4. Storage.** To install a new system, make sure the “Enable storage” check box is on. Then select the amount of space you want to devote to the VM. Click Forward

to continue.

5. Review VM configuration. Before you launch the VM, you get a chance to review the settings, as shown in [Figure 15-4](#). If they look all right, click Finish.

Figure 15-4: Review settings for your VM before launching the installer.



At this point, the ISO image boots up in a new window. What happens next depends on whether it is a live CD, installer, or other type of bootable image. Assuming it is an installer, run through the installation process as you would if you were installing directly on the computer hardware.

Once the operating system is installed on the VM, you can use `virt-manager` to manage that VM, as described in the next section.

Starting and Stopping Virtual Machines with `virt-manager`

Once you install a VM using `virt-manager`, an entry for that VM appears in the Virtual Machine Manager window. To open a console window to use the VM, just double-click

the VM's entry. [Figure 15-5](#) shows a VM that was just launched from the Virtual Machine Manager window.

Figure 15-5: Open a console to your VM from the Virtual Machine Manager window.



When you are done using the VM for the moment, just close the window. The VM will keep running and be accessible through any network interface you configured for the VM. Here are other actions you can do with the VM by right-clicking the VM's entry:

- **Run**—If the VM is not currently running, this selection boots it up.
- **Pause**—This selection pauses a running VM.
- **Shutdown**—After making this selection, you can choose to Reboot, Shut Down, Force Off, or Save. A Force Off is similar to just pulling out the plug on a physical machine. A Save saves everything in your VM's memory to a file that you can explore later.
- **Clone**—This selection lets you make a clone of the current VM, so you can have multiple instances of the VM.
- **Migrate**—If you have another compatible KVM host available, you can migrate the VM to that other host.

Now that you have seen what you can do with KVM virtual machines using graphical

tools, the next section will help you use commands to accomplish many of the same things.

Managing Virtual Machines with Commands

If you can install and manage VMs with a nice graphical tool like `virt-manager`, why bother managing your VMs with commands? There are lots of reasons. Maybe your KVM host doesn't have a desktop installed. Or you may want to run commands to work with VMs from a shell script.

To get started, you can use the `virt-install` command to install a virtual machine. With `virt-clone` you can clone an existing virtual image. For managing VMs, you can use the `virsh` command to list information about VMs, as well as start, stop, and reboot them.

Creating a Virtual Machine with `virt-install`

Instead of clicking through the Virtual Machine Manager window to create a VM, you can pass options required to create a VM using the `virt-install` command. Before using `virt-install`, you need to create a storage image. One way to do that is with the `qemu-img` command.

Create Storage for the VM with `qemu-img`

You can use the `qemu-img` command to create image files that VMs can utilize as their storage media. To the installer, the images look like regular hard disks or other block storage devices. Here are descriptions of supported image formats:

- **raw**—This is the default, raw image type for `qemu-img`. This image type is simple. It is the format you should use if you expect to export the image to other virtual environments.
- **qcow2**—If you expect the image to continue to be used by `qemu`, use `qcow2`. The `qcow2` format does not immediately consume all space that is allocated, but instead grows as space is needed. It also supports encryption.
- **Other formats**—Most other image formats supported by `qemu-img` are supported mostly for compatibility with older versions. These include `qcow`, `cow`, `vdi`, `vmdk`, `vpc`, and `cloop`. Type **`man qemu-img`** to see information about those image types.

The following example of `qemu-img` creates a `qcow2` image at

/var/stuff/mine.qcow2. There is 8GB of disk space allocated for the image (although it will actually consume only the amount that the VM uses). The `preallocation=metadata` option can improve the performance of the image as it grows.

```
$ sudo qemu-img create -f qcow2 -o preallocation=metadata \  
/var/stuff/mine.qcow2 8G
```

Once the qcow2 image is created, you can do a consistency check on it using `qemu-img` with the `check` option:

```
$ sudo qemu-img check /var/stuff/mine.qcow2 Check that image is OK
```

At any point in the process, you might want to check the amount of space being consumed by your VM. Do that with the `info` option to `qemu-img`:

```
$ sudo qemu-img info mine.qcow2  
See allocated/actual image sizes  
image: mine.qcow2  
file format: qcow2  
virtual size: 8.0G (8589934592 bytes)  
disk size: 2.6G  
cluster_size: 65536
```

The `qemu-img info` example was run after the VM was installed. You can see that although 8GB of space is allocated, only 2.6GB is currently being consumed.

Installing a VM with `virt-install`

The `virt-install` command is a versatile tool for creating new virtual machines. On the command line, you can identify the attributes of the VM's environment.

Here's an example of a `virt-install` command line that creates an Ubuntu virtual machine. This command incorporates many of the options you would have to click on or fill in on the `virt-manager` window. Note that this command incorporates the image that I created earlier in this chapter using the `qemu-img` command.

```
$ virt-install --connect qemu:///system --name cn_ubuntu12.04 \  
--ram 1024 --disk path=/var/stuff/mine.qcow2,format=qcow2 \  
--network=bridge:virbr0,model=virtio --vnc --os-type=linux \  
--cdrom /var/stuff/ubuntu-12.04.2-desktop-amd64.iso \  
--noautoconsole --keymap=en-us
```

Here is what the different options to `virt-install` mean:

- **--connect** Identifies the location of the virtualization service on the hypervisor. The `qemu:///system` argument that is identified here is the default location used by KVM.
- **--name** Identifies the name to represent the VM. You can call it anything you like,

but typically the name will indicate the type of operating system and possibly a release number.

- **--ram** Lets you set how much RAM the VM can consume.
- **--disk_path** Tells the location of the disk image and its format (in this case, qcow2).
- **network=bridge:virbr0** Identifies the network interface the VM should use from the host to communicate to other hosts.
- **--vnc** Says to use virtual network computing (VNC) to provide access to the VM's console.
- **--os-type** Identifies the VM as a Linux system.
- **--cdrom** Indicates the location of the ISO image that the installation is run from.
- **--noautoconsole** Prevents a console to the VM from automatically opening. This allows you to use other viewers to watch the VM, such as `virt-manager` or `virt-viewer`.
- **--keymap=en-us** Sets the keyboard to US English.

Check the `virt-install` man page (type **man virt-install**) to see other options you can use with the `virt-install` command.

Once the `virt-install` command starts, you can open an application from the desktop to see the progress of your installation. The `virt-manager` and `virt-viewer` commands are among those you can use to view your VM's console. In the case of the Ubuntu live CD used in this example, you would have to click through the install process and enter some information to complete the installation.

After the VM is installed, you can manage your VMs using the `virsh` command.

Starting and Stopping Virtual Machines with `virsh`

The `virsh` command provides a good way to manage your VMs after they are created. You can use `virsh` to see what VMs are running; then you can start, stop, pause, and otherwise manage them.

Here are some examples of the `virsh` command:

```
$ virsh help           View the list of subcommands to virsh
$ virsh list          Show currently running VMs
  Id Name                State
-----
   5 Ubuntu12.04DT       running
   6 Fedora17DT          running
$ virsh shutdown Fedora17DT  Shutdown the Fedora VM
Domain Fedora17DT is being shutdown
```

```
$ virsh destroy Ubuntu12.04DT      Immediately stop the Ubuntu VM
Domain Fedora17DT destroyed
$ virsh undefine Fedora17DT        Totally remove VM's definition
Domain Fedora17DT has been undefined
$ virsh version                    Show current version information
Compiled against library: libvir 0.9.8
Using library: libvir 0.9.8
Using API: QEMU 0.9.8
Running hypervisor: QEMU 1.0.0
$ virsh hostname                  Show hostname of the hypervisor
$ virsh autostart Ubuntu12.04DT    Set VM to start at boot time
```

There are many more options to the `virsh` command that you can use to manage your VMs. Refer to the `virsh` man page (type **man virsh**) for details.

Summary

To make efficient use of computing infrastructures, more and more computers are being used as virtualization hosts (also sometimes called hypervisors). By using a computer as a virtual host, you can run multiple, different operating systems at once on the same physical computer.

The KVM features of Linux provide a means of running Ubuntu or other Linux systems as a virtual host. If you want to use your computer as a virtual host, the chapter takes you through some steps you can take to make sure your computer hardware includes virtualization support.

The Virtual Machine Manager window (`virt-manager` command) provides a graphical way of installing and managing VMs. Because it is fairly intuitive to use, it's a good way to start using virtualization in Ubuntu.

If you prefer command-line tools for virtualization, the `qemu-img` command can be used to create image files needed for storage by a VM. Then the `virt-install` command can be used to actually create the virtual machine.

Once created, your VMs can be managed using the `virsh` command. With `virsh`, you can start, stop, pause and perform other actions on your VMs.

Appendix A

Using vi or Vim Editors

IN THIS APPENDIX

- Using the vi editor
- Starting/quitting the vi editor
- Moving around in vi
- Changing and deleting text
- Using Ex commands
- Using visual mode

Although easy-to-use graphical text editors (such as `gedit` and `kedit`) are readily available with Linux, most power users still use `vi`, `nano`, or `Emacs` to edit text files. In addition to the fact that those text-based editors will work from any shell (no GUI required), they offer other advantages. For example, your hands never have to leave the keyboard and they offer integration with useful utilities. And unlike GUI editors, text-based editors will work if no graphical interface is installed (as is true with many Linux servers and specialty devices).

This appendix focuses on features of the `vi` editor that can not only help you with basic editing, but also help you do some advanced text manipulation. I chose to cover `vi` rather than `Emacs` because `vi` is more universal and leaner, and also because `vi` keyboard shortcuts require fewer contortions.

Because many Linux systems use the Vim (Vi IMproved) editor by default, in place of the older `vi` editor, the descriptions in this appendix are extended to cover Vim as well. Some features in Vim that are not in `vi` include multiple undo levels, syntax highlighting, and online help.

Note If you have never used `vi` or Vim before, try out the tutor that comes with the `vim-enhanced` package. Run the `vimtutor` command and follow the instructions to step through many of the key features of `vi` and Vim.

Starting and Quitting the vi Editor

If you want to experiment with using `vi`, you should copy a text file to practice on. For example, type:

```
$ cp /etc/services /tmp
```

Then open that file using the `vi` command as follows:

```
$ vi /tmp/services
```

To benefit from all the improvements of Vim, make sure you have the `vim-athena` package installed (which gets installed by default on Ubuntu). On many systems, `vi` is a symbolic link to the `vim` command. On Ubuntu, both commands launch `vim`.

Here are a few other ways you can **start vi**:

```
$ vi +25 /tmp/services      Begin on line 25
$ vi + /tmp/services        Begin editing file on the last line
$ vi +/tty /tmp/services    Begin on first line with word "tty"
$ vi -r /tmp/services       Recover file from crashed edit session
$ view /tmp/services        Edit file in read-only mode
```

When you are done with your `vi` session, there are several different ways to save and quit:

- To **save the file before you are ready to quit**, type `:w`
- To **quit and save changes**, type either `ZZ` or `:wq`
- To **quit without saving changes**, type `:q!`

If you find that you can't write to the file you are editing, it may have been opened in read-only mode. If that's the case, you can try forcing a write by typing `:w!` or you can **save the contents of the file to a different name**. For example, type the following to save the contents of the current file to a file named `myfile.txt`:

```
:w /tmp/myfile.txt
```

The `vi` editor also enables you to **line up several files at a time to edit**. For example, type:

```
$ cd /tmp
$ touch a.txt b.txt c.txt
$ vi a.txt b.txt c.txt
```

In this example, `vi` will open the `a.txt` file first. You can **move to the next file** by typing `:n`. You may want to **save changes before moving to the next file** (`:w`) or **save changes as you move to the next file** (`:wn`). To **abandon changes while moving to the next file**, type `:n!`.

You will probably find it easier to open multiple files by splitting your `vi` screen. When you're in `vi` and have a file open, you can **split your screen multiple times** either horizontally or vertically:

```
:split /etc/motd.tail
:vsplit /etc/motd.tail
```

Use the `Tab` key to complete the path to the files, just like you would in a bash shell.

To **navigate between split windows**, press Ctrl+w, followed by the w key. To close the current window, use the usual vi exit command (:q).

Moving Around in vi

The first thing to get used to with vi is that you can't just start typing. Vi has multiple modes that enable you to perform a different set of tasks. You start a vi session in Normalmode, where vi is waiting for you to type a command to get started.

While you are in Normal mode, you can move around the file, to position where you want to be in the file. To enter or modify text, you need to go into Insert or Replace modes.

Assuming vi is open with a file that contains several pages of text, the following list shows some keys and combinations you can type to **move around the file while in Normal mode**.

PageDown or Ctrl+f—Move down one page.

Ctrl+d—Move down half page.

Shift+g—Go to last line of file.

Shift+h—Move cursor to screen top.

Shift+m—Move cursor to middle of screen.

Enter—Move cursor to beginning of the next line.

Home or \$—Move cursor to end of line.

(—Move cursor to beginning of previous sentence.

{—Move cursor to beginning of previous paragraph.

w—Move cursor to next word (space, new line, or punctuation).

b—Move cursor to previous word (space, new line, or punctuation).

e—Move cursor to end of next word (space, new line, or punctuation).

Left arrow or Backspace—Move cursor left one letter.

k or up arrow—Move cursor up one line.

/string—Find next occurrence of string.

n—Find same string again (forward).

PageUp or Ctrl+b—Move up one page.

Ctrl+u—Move up half page.

:1—Go to first line of file (use any number to go to that line).

Shift+l—Move cursor to screen bottom.

Ctrl+l—Redraw screen (if garbled).

- —Move cursor to beginning of the previous line.
- End or ^ or 0**—Move cursor to line beginning.
-)—Move cursor to beginning of next sentence.
- }—Move cursor to beginning of next paragraph.
- Shift+w**—Move cursor to next word (space or new line).
- Shift+b**—Move cursor to previous word (space or new line).
- Shift+e**—Move cursor to end of next word (space or new line).
- Right arrow or l**—Move cursor right one letter.
- j or down arrow**—Move cursor down one line.
- ?string**—Find previous occurrence of string.
- Shift+n**—Find same string again (backwards).

Changing and Deleting Text in vi

To begin changing or adding to text with vi, you can enter Insert or Replace modes, as shown in the following list. When you enter Insert or Replace mode, the characters you type will appear in the text document (as opposed to being interpreted as commands).

Press the Esc key to exit to Normal mode after you are done inserting or replacing text.

- i**—Typed text appears before current character.
- a**—Typed text appears after current character.
- o**—Open a new line below current line to begin typing.
- s**—Erase current character and replace with new text.
- c?**—Replace ? with l, w, \$, or c to change the current letter, word, end of line, or line.
- r**—Replace current character with the next one you type.
- Shift+i**—Typed text appears at the beginning of current line.
- Shift+a**—Typed text appears at the end of current line.
- Shift+o**—Open a new line above current line to begin typing.
- Shift+s**—Erase current line and enter new text.
- Shift+c**—Erase from cursor to end of line and enter new text.
- Shift+r**—Overwrite as you type from current character going forward.

The following list contains keys you type to delete or paste text.

- x**—Delete text under cursor.

d?—Replace ? with l, w, \$, or d to cut the current letter, word, or end of line from cursor or entire line.

y?—Replace ? with l, w, or \$ to copy (yank) the current letter, word, or end of line from cursor.

p—Pastes cut or yanked text after cursor.

Shift+x—Delete text to left of cursor.

Shift+d—Cut from cursor to end of line.

Shift+y—Yank current line .

Shift+p—Pastes cut or yanked text before cursor.

Using Miscellaneous Commands

The following list shows a few miscellaneous, but important, commands you should know.

u—Type **u** to undo the previous change. Multiple **u** commands will step back to undo multiple changes.

.—Typing a period (.) will repeat the previous command. So, if you deleted a line, replaced a word, changed four letters, and so on, the same command will be done wherever the cursor is currently located. (Entering input mode again resets it.)

Shift+j—Join the current line with the next line.

Esc—If you didn't catch this earlier, the Esc key returns you from an input mode back to command mode. This is one of the keys you will use most often.

Modifying Commands with Numbers

Nearly every command described so far can be modified with a number. In other words, instead of deleting a word, replacing a letter, or changing a line, you can delete six words, replace twelve letters, and change nine lines. The following list shows some examples.

7cw—Erase the next seven words and replace them with text you type.

5 Shift+d—Cut the next five lines (including the current line).

3p—Paste the previously deleted text three times after the current cursor.

9db—Cut the nine words before the current cursor.

10j—Move the cursor down ten lines.

y2)—Copy (yank) text from the cursor to the end of next two sentences.

5 Ctrl+f—Move forward five pages.

6 Shift+j—Join the next six lines.

From these examples, you can see that most vi keystrokes for changing text, deleting text, or moving around in the file can be modified using numbers.

Using ex Commands

The vi editor was originally built on an editor called ex. Some of the vi commands you've seen so far start with a semicolon and are known as ex commands. To enter ex commands, start from normal mode and type a colon (:). This switches you to command line mode.

In command line mode, you can use the Tab key to complete your command or filename, and the arrow keys to navigate your command history, as you would in a bash shell. When you press Enter at the end of your command, you are returned to Normal mode.

The following list shows some examples of ex commands.

!:bash—Escape to a bash shell. When you are done, type **exit** to return to vi.

!:date—Run `date` (or any command you choose). Press Enter to return.

!!—Rerun the command previously run.

:20—Go to line 20 in the file.

:5,10w abc.txt—Write lines 5 through 10 to the file `abc.txt`.

:e abc.txt—Leave the current file and begin editing the file `abc.txt`.

:.r def.txt—Read the contents of `def.txt` into the file below the current line.

:s/RH/Red Hat—Substitute `Red Hat` for the first occurrence of `RH` on the current line.

:s/RH/Red Hat/g—Substitute `Red Hat` for all occurrences of `RH` on the current line.

:%s/RH/Red Hat/g—Substitute `Red Hat` for all occurrences of `RH` in the entire file.

:g/Red Hat/p—List every line in the file that contains the string `Red Hat`.

:g/gaim/s//pidgin/gp—Find every instance of `gaim` and change it to `pidgin`.

From the `ex` prompt you can also see and change settings related to your vi session using the `set` command. The following list shows some examples.

:set all—List all settings.

:set—List only those settings that have changed from the default.

:set number—Have line numbers appear to the left of each line. (Use `set nonu` to unset.)

- :set ai**—Sets autoindent, so opening a new line follows the previous indent.
- :set ic**—Sets ignore case, so text searches will match regardless of case.
- :set list**—Show \$ for end of lines and ^I for tabs.
- :set wm**—Causes vi to add line breaks between words near the end of a line.

Working in Visual Mode

The Vim editor provides a more intuitive means of selecting text called **visual mode**. To begin visual mode, move the cursor to the first character of the text you want to select and press the `o` or `v` key. With `o`, you highlight character-by-character; with `v` it's line-by-line. You will see that you are in visual mode because the following text appears at the bottom of the screen:

```
-- VISUAL --
```

At this point, you can use any of your cursor movement keys (arrow keys, Page Down, End, and so on) to move the cursor to the end of the text you want to select. As the page and cursor move, you will see text being highlighted.

When all the text you want to select is highlighted, you can press keys to act on that text. For example, `d` deletes the text, `c` lets you change the selected text, `:w /tmp/test.txt` saves selected text to a file, and so on.

Appendix B

Shell Special Characters and Variables

IN THIS APPENDIX

- Using special shell characters
- Using shell variables

Ubuntu provides bash as the default shell. Chapter 3 helps you become comfortable working in the shell. This appendix provides a reference to the numerous characters and variables that have special meaning to the bash shell. Many of those elements are referenced in [Table B-1](#) (“Shell Special Characters”) and [Table B-2](#) (“Shell Variables”).

Using Special Shell Characters

You can use special characters from the shell to match multiple files, save some keystrokes, or perform special operations. [Table B-1](#) shows some shell special characters you may find useful.

Table B-1: Shell Special Characters

Character	Description
*	Match any string of characters.
?	Match any one character.
' ... '	Remove special meaning of characters between quotes. Variables are not expanded.
" ... "	Same as simple quotes except for the escape characters (\$, `, and \) that preserve their special meaning. Variables are expanded.
\	Escape character to remove the special meaning of the character that follows.
~	Refers to the \$HOME directory.
~+	Value of the shell variable PWD (working directory).
~-	Refers to the previous working directory.
.	Refers to the current working directory.
..	Refers to the directory above the current directory. Can be used repeatedly to reference several directories up.
\$param	Used to expand a shell variable parameter.
cmd1 `cmd2` or cmd1 \$(cmd2)	cmd2 is executed first. Then the output of cmd2 is used as input to cmd1.

cmd1> file	Redirects standard output from a command to a file.
cmd1< file	Redirects standard input from a file to a command.
cmd1>> file	Appends standard output to a file from a command, without erasing its current contents.
cmd1 cmd2	Pipes the standard output of one command to the input of the next.
cmd&	Runs the command in the background.
cmd1&&cmd2	Runs the first command; then if it returns a zero exit status (success), runs the second command.
cmd1 cmd2	Runs the first command; then, if it returns a non-zero exit status (not success), runs the second command.
cmd1 ; cmd2	Runs the first command and when it completes, runs the second command.

Using Shell Variables

You identify a string of characters as a variable by placing a `$` in front of it (as in `$HOME`). Shell environment variables can hold information that is used by the shell itself, as well as by commands you run from the shell.

Many commands check for particular variables to be set. Not all of those variables will be populated by default. Some of these variables you can change (such as the default printer in `$PRINTER` or your command prompt in `$PS1`). Others are managed by the shell (such as `$OLDPWD`). [Table B-2](#) contains a list of many useful shell variables.

Table B-2: Shell Variables

Shell Variable	Description
BASH	Shows path name of the <code>bash</code> command (<code>/bin/bash</code>).
BASH_COMMAND	The command that is being executed at the moment.
BASH_VERSION	The version number of the <code>bash</code> command.
COLORS	Path to the configuration file for <code>ls</code> colors.
COLUMNS	The width of the terminal line (in characters).
DISPLAY	Identifies the X display where commands launched from the current shell will be displayed (such as <code>:0.0</code>).
EUID	Effective user ID number of the current user. It is based on the user entry in <code>/etc/passwd</code> for the user that is logged in.
FCEDIT	Determines the text editor used by the <code>fc</code> command to edit <code>history</code> commands. The <code>vi</code> command is used by default.
GROUPS	Lists groups (by group ID) of which the current user is a member.
HISTCMD	Shows the current command's history number.
HISTFILE	Shows the location of your history file (usually located at <code>\$HOME/.bash_history</code>).
HISTFILESIZE	Total number of history entries that will be stored (default, 1000). Older commands are discarded after this number is reached.

HISTCMD	The number of the current command in the history list.
HOME	Location of the current user's home directory. Typing the <code>cd</code> command with no options returns the shell to the home directory.
HOSTNAME	The current machine's hostname.
HOSTTYPE	Contains the computer architecture on which the Linux system is running (i386, i486, i586, i686, x86_64, ppc, or ppc64).
LESSOPEN	Set to a command that converts content other than plain text (images, RPMs, zip files, and so on) so it can be piped through the <code>less</code> command.
LINES	Sets the number of lines in the current terminal.
LOGNAME	Holds the name of the current user.
LS_COLORS	Maps colors to file extensions to indicate the colors the <code>ls</code> command displays when encountering those file types.
MACHTYPE	Displays information about the machine architecture, company, and operating system (such as x86_64-pc-linux-gnu).
MAIL	Indicates the location of your mailbox file (typically the username in the <code>/var/spool/mail</code> directory).
MAILCHECK	Checks for mail in the number of seconds specified (default is 60).
OLDPWD	Directory that was the working directory before changing to the current working directory.
OSTYPE	Name identifying the current operating system (such as linux or linux-gnu).
PATH	Colon-separated list of directories used to locate commands that you type (<code>/bin</code> , <code>/usr/bin</code> , and <code>\$HOME/bin</code> are usually in the <code>PATH</code>). Directories are searched from left to right.
PPID	Process ID of the command that started the current shell.
PRINTER	Sets the default printer, which is used by printing commands such as <code>lpr</code> and <code>lpq</code> .
PROMPT_COMMAND	Set to a command name to run that command each time before your shell prompt is displayed. (For example, <code>PROMPT_COMMAND=ls</code> lists commands in the current directory before showing the prompt).
PS1	Sets the shell prompt. Items in the prompt can include date, time, username, hostname, and others. Additional prompts can be set with <code>PS2</code> , <code>PS3</code> , and so on.
PWD	The directory assigned as your current directory.
RANDOM	Accessing this variable generates a random number between 0 and 32767.
SECONDS	The number of seconds since the shell was started.
SHELL	Contains the full path to the current shell.
SHELLOPTS	Lists enabled shell options (those set to <code>on</code>).

Appendix C

Getting Information from /proc

IN THIS APPENDIX

- Viewing /proc information
- Changing /proc information variables

Originally intended to be a location for storing information used by running processes, the `/proc` filesystem eventually became the primary location for storing all kinds of information used by the Linux kernel. Despite the emergence of `/sys` to provide a more orderly framework for kernel information, many Linux utilities still gather and present data about your running system from `/proc`.

If you are someone who prefers to cut out the middleman, you can bypass utilities that read `/proc` files and read (and sometimes even write to) `/proc` files directly. By checking `/proc`, you can find out the state of processes, hardware devices, kernel subsystems, and other attributes of Linux.

Viewing /proc Information

Checking out information in files from the `/proc` directory can be done by using a simple `cat` command. In `/proc`, there is a separate directory for each running process (named by its process ID) that contains information about the process. There are also `/proc` files that contain data for all kinds of other things, such as your computer's CPU, memory usage, software versions, disk partitions, and so on.

The following examples illustrate some of the information you can get from your Linux system's `/proc` directory:

```
$ cat /proc/cmdline           Shows options passed to the boot prompt
BOOT_IMAGE=/vmlinuz-3.2.0-37-generic root=/dev/mapper/ubuntutb-root
ro
```

```
$ cat /proc/cpuinfo           Shows information about your processor
```

```
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 23
model name    : Pentium(R) Dual-Core CPU           E6300   @ 2.80GHz
stepping      : 10
```

```
microcode      : 0xa07
cpu MHz        : 1603.000
cache size     : 2048 KB
...
```

In the preceding example, the MHz speed may be well below your actual system speed if a CPU governor such as `cpuspeed` is running. The next example lists character and block devices:

```
$ cat /proc/devices          Shows existing character and block
devices
```

```
Character devices:
```

```
1 mem
4 /dev/vc/0
4 tty
4 ttyS
5 /dev/tty
...
```

```
Block devices:
```

```
1 ramdisk
259 blkext
7 loop
8 sd
```

```
$ cat /proc/diskstats       Display disks, partitions, and statistics
```

```
1      0 ram0 0 0 0 0 0 0 0 0 0 0 0
1      1 ram1 0 0 0 0 0 0 0 0 0 0 0
...
8          0 sda 21883 11293 874900 194544 7813 9966 932416 1073344
          0 120276 1267864
8          1 sda1 278 25 2310 11636 8 0 16 476 0 11960 12112
8          2 sda2 2 0 12 144 0 0 0 0 0 144 144
...
7      0 loop0 0 0 0 0 0 0 0 0 0 0 0
```

In the `diskstats` output just shown, you can see ramdisk (`ram0`, `ram1`, and so on) and loopback (`loop0`, `loop1`, and so on) devices. For hard disk partitions, the example shows statistics for the whole hard disk (`sda`) and each partition (`sda1`, `sda2`, and so on).

The 11 fields for the entire hard disk show (from left to right): total number of reads, number of reads merged, number of sectors read, number of milliseconds spent by all reads, number of writes completed, number of writes merged, number of sectors written, number of milliseconds spent writing, number of input/output requests currently in progress, number of milliseconds spent doing input/output, and weighted number of milliseconds spent doing input/output. Fields for a particular partition show (from left to right): number of reads issued, number of sectors read, number of writes issued, and

number of sectors written.

```
$ cat /proc/filesystems      List filesystem types in current kernel
nodev      sysfs             nodev means type is not used by any
device
nodev      rootfs
...
      ext4                   ext4 is used on a mounted block device
      iso9660                iso9660 is used on a mounted block
device
```

```
$ cat /proc/interrupts      View IRQ channel assignments

      CPU0      CPU1      CPU2      CPU3
0:      126        0        0        0      IO-APIC-edge
timer
1:      3815      2223      4786      2596      IO-APIC-edge
i8042
8:        1        0        0        0      IO-APIC-edge      rtc0
9:     15351     14722     5296     4894      IO-APIC-fastestoi  acpi
12:    268497    419436    322826    145849      IO-APIC-edge
i8042
```

```
...
$ cat /proc/iomem           Show physical memory addresses
00000000-0000ffff : reserved
00010000-0009ffff : System RAM
0009f000-0009ffff : reserved
000a0000-000bffff : PCI Bus 0000:00
000c0000-000c7fff : Video ROM
000d0000-000dffff : PCI Bus 0000:00
000e0000-000fffff : reserved
      000f0000-000fffff : System ROM
00100000-bdd9ffff : System RAM
      01000000-0166a78b : Kernel code
```

```
...
$ cat /proc/ioports         Show virtual memory addresses
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-0060 : keyboard
0064-0064 : keyboard
0070-0071 : rtc0
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
```

```
...
$ cat /proc/loadavg         Shows 1, 5, and 15 minute load
```


averages,
1.77 0.56 0.19 2/247 1869 running processes/total and highest
PID

\$ cat /proc/meminfo Shows available RAM and swap

MemTotal: 4014504 kB
MemFree: 3202120 kB
Buffers: 90048 kB
Cached: 492060 kB
SwapCached: 0 kB
Active: 377604 kB
Inactive: 274020 kB
Active(anon): 70136 kB
Inactive(anon): 25448 kB
Active(file): 307468 kB

...

\$ cat /proc/misc Shows name/minor number of devices
229 fuse registered with misc major device
(10)

236 device-mapper
173 agpgart

...

\$ cat /proc/modules Shows loaded modules, memory size,
instances loaded, dependencies
load state, and kernel memory
bnep 18281 2 - Live 0x0000000000000000
rfcomm 47604 0 - Live 0x0000000000000000
bluetooth 180153 10 bnep,rfcomm, Live 0x0000000000000000
parport_pc 32866 0 - Live 0x0000000000000000
ppdev 17113 0 - Live 0x0000000000000000
ext2 73795 1 - Live 0x0000000000000000
snd_hda_codec_realtek 224173 1 - Live 0x0000000000000000
snd_hda_intel 33773 3 - Live 0x0000000000000000

...

\$ cat /proc/mounts Show mounted local/remote file system
info

rootfs / rootfs rw 0 0
sysfs /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
proc /proc proc rw,nosuid,nodev,noexec,relatime 0 0
udev /dev devtmpfs
rw,relatime,size=19981k,nr_inodes=499543,mode=755 0 0
/dev/sda1 /boot ext2 rw,relatime,errors=continue 0 0

...

\$ cat /proc/partitions Show mounted local disk partitions

major	minor	#blocks	name
8	0	156290904	sda
8	1	248832	sda1
8	2	1	sda2

```

      8          5  156039168 sda5
    252          0   93495296 dm-0
    252          1   4157440  dm-1
      11          0   1048575 sr0

```

...

```

$ cat /proc/mdstat          If using software RAID, show RAID status
Personalities : [raid1]
read_ahead 1024 sectors
Event: 1
md0 : active raid1 sdb1[1] sda2[0]
      69738048 blocks [2/2] [UU]

```

```

unused devices: <none>

```

The `/proc/mdstat` file contains detailed status information on your software RAID devices if you have set up such a software RAID device. In this example, `md0` is a RAID1 (mirror) composed of the `/dev/sdb1` and `/dev/sda1` partitions. On the following line, there is one `U` for each healthy RAID member. If you lose a drive, the output would appear as `[U_]`.

```

$ cat /proc/stat           Shows kernel stats since system
boot
cpu  58394 4008 18635 50620848 9216 0 151 0 0 0
cpu0 29693 2416 9551 25311091 4459 0 87 0 0 0
cpu1 28701 1592 9084 25309756 4757 0 64 0 0 0
intr 7720709 889 11 0 0    ...
ctxt 11180402
btime 1361187396
processes 5107
procs_running 1
procs_blocked 0

```

The `/proc/stat` file contains statistics related to CPU and process activities. The `cpu` line shows totals for all CPUs, while separate lines for each processor (`cpu0`, `cpu1`, and so on) show stats for each CPU on the computer. There are seven fields (from left to right) of CPU information: number of normal processes executed in user mode, niced processes executed in user mode, kernel mode processes, idle processes, `iowait` processes (waiting for input/output to finish), servicing interrupts (IRQ), and servicing soft IRQs.

```

$ cat /proc/swaps          List information about swap space
Filename      Type      Size      Used      Priority
/dev/sda2     partition 4157436   0         -1
$ cat /proc/uptime         Seconds since system booted/total seconds
idle
2300251.03 2261855.31
$ cat /proc/version        List kernel version and related compiler

```

```
Linux version 3.2.0-37-generic (buildd@allspice) (gcc version 4.6.3
(Ubuntu/Linaro 4.6.3-1ubuntu5) ) #58-Ubuntu SMP
Thu Jan 24 15:28:10 UTC 2013
```

Changing /proc Information

On some versions of Linux, some values in the `/proc/sys` directory can actually be changed on the fly. On Linux systems that allow it, you could simply echo a value to any file you want to change and that change immediately takes effect.

The preferred method of changing `/proc/sys` information on the fly is using the `sysctl` command. To change those settings on a more permanent basis, you should add entries to the `/etc/sysctl.conf` file. Here are some examples of the `sysctl` command:

```
$ sudo sysctl -A | less           Display all kernel
parameters
$ sudo sysctl -w net.ipv4.ip_forward=1 Turn on IPV4 packet
forwarding
```

See Chapter 10 as well as the `sysctl` and `sysctl.conf` man pages for further information.

Ubuntu® Linux® TOOLBOX

1000+ Commands for Ubuntu
and Debian® Power Users,
Second Edition

Christopher Negus

WILEY

Ubuntu® Linux® Toolbox: 1000+ Commands for Ubuntu and Debian® Power Users, Second Edition

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2013 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-118-18352-6

ISBN: 978-1-118-22799-2 (ebk)

ISBN: 978-1-118-24052-6 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author

or the publisher endorses the information the organization or website may provide or recommendations it may make. Further, readers should be aware that Internet websites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2013939157

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Ubuntu is a registered trademark of Canonical, Ltd. Linux is a registered trademark of Linus Torvalds. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

As always, I dedicate my work on this book to my wife, Sheree.

— Christopher Negus

About the Author

Christopher Negus has more than 25 years of experience teaching and writing about Linux and UNIX. He has authored dozens of books on Linux, including the bestselling Red Hat Linux Bible, Linux Bible, the Linux Toys series, and the Linux Toolbox series.

As an employee of Red Hat Inc., Christopher has achieved certifications that include Red Hat Certified Engineer, Instructor, and Examiner (RHCE, RHCI, and RHCX). Currently, he creates technical content for the Red Hat Customer Portal, which covers topics related to Red Hat Enterprise Linux, virtualization, and cloud computing.

Awards for Christopher's writing include "Best Linux book of the year" for his Red Hat Linux 8 Bible, as voted by readers of Linux World magazine. For the Linux Journal 2009 Readers' Choice Awards, his Linux Bible was voted one of the top five "Favorite Linux Books of All Time."

About the Technical Editors

Richard Blum has worked in the IT industry for more than 25 years both as a systems and network administrator. He has published numerous Linux and Open Source books, and is an online instructor for web programming and Linux courses that are used by colleges and universities across the U.S. When he's not busy being a computer nerd, he enjoys playing piano and guitar, and spending time with his wife, Barbara, and his two daughters, Katie Jane and Jessica.

David Duffey lives in Austin, TX with his wife and two children and is employed by Canonical, the commercial sponsor of Ubuntu. David is a bit of a geek and holds numerous Linux certifications from Canonical, Red Hat, Novell/SuSe, and LPI. David earned his undergraduate degrees in Mathematics and Computer Science at Kansas State University and his EMBA from the University of Texas McCombs School of Business.

Credits

Acquisitions Editor

Mary James

Project Editor

Maureen Spears

Technical Editors

Richard Blum

David Duffey

Production Editor

Daniel Scribner

Copy Editor

Nancy Rapoport

Editorial Manager

Mary Beth Wakefield

Freelancer Editorial Manager

Rosemarie Graham

Associate Director of Marketing

David Mayhew

Marketing Manager

Ashley Zurcher

Business Manager

Amy Knies

Production Manager

Tim Tate

Vice President and Executive Group Publisher

Richard Swadley

Vice President and Executive Publisher

Neil Edde

Associate Publisher

Jim Minatel

Project Coordinator, Cover

Katie Crocker

Compositor

Jeff Lytle, Happenstance Type-O-Rama

Proofreader

Nancy Carrasco

Indexer

Ron Strauss

Cover Image

Wiley

Cover Designer

©Stockbyte/Getty Images, Inc.

Acknowledgments

I would like to acknowledge Canonical Ltd. and the Ubuntu community for their ongoing excellent work producing the Linux-based Ubuntu operating system.

For this edition, I enlisted the excellent skills of David Duffey, Server Partner Programme Manager at Canonical. His help was invaluable when it came to sizing up the new content for this edition. Also on this edition, thanks to Richard Blum (a fine Linux author in his own right) for providing a thorough technical edit on this edition.

Special thanks to François Caen for his work as coauthor on the first edition of this book. Also on the first edition, Thomas Blader went far beyond his technical editor title, providing excellent insights and meticulous testing throughout the book. Eric Foster-Johnson came in near the end of the process on the first edition and provided Ubuntu feature enhancements throughout the book.

Thanks to my friends and associates at Red Hat who make it a joy to go to work every day. In particular, thanks to Henry Hutton for his constant encouragement and great ideas promoting my Linux books. And on the home front, thanks to my wonderful wife Sheree and cool son Seth for making it fun to come home too.

At Wiley, I'd like to thank Mary James for helping to establish the vision for this book, as well as Maureen Spears for her gentle encouragement to keep me on schedule and for her steady hand developing and editing the book. Thanks to Daniel Scribner for steering it through the production phases. Nancy Rapoport's detailed copy editing has added a layer of polish to the book that I couldn't hope to get on my own.

—Christopher Negus

CHRISTOPHER NEGUS

UBUNTU[®] LINUX[®] **TOOLBOX**

**2nd
Edition**

1000+
Commands
for Ubuntu and
Debian[®] Power Users

WILEY