Quick answers to common problems

Linux Utilities Cookbook

Over 70 recipes to help you accomplish a wide variety of tasks in Linux quickly and efficiently

James Kent Lewis

[PACKT] PUBLISHING open source community experience distilled

Linux Utilities Cookbook

Over 70 recipes to help you accomplish a wide variety of tasks in Linux quickly and efficiently

James Kent Lewis

[PACKT] PUBLISHING
open source
community experience distilled
BIRMINGHAM - MUMBAI

Linux Utilities Cookbook

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: October 2013

Production Reference: 1211013

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-300-8

www.packtpub.com

Cover Image by Abhishek Pandey (abhishek.pandey1210@gmail.com)

Credits

Author
James Kent Lewis

Reviewers
Samarendra Manohar Hedaoo
Jitesh Marathe
Young-Ho Song
Gene Wilburn

Acquisition Editor
Owen Roberts

Lead Technical Editor
Madhuja Chaudhari

Technical Editors
Aparna Chand
Adrian Raposo
Gaurav Thingalaya

Project Coordinator
Apeksha Chitnis

Proofreader
Ting Baker

Indexer
Hemangini Bari
Tejal R. Soni

Production Coordinators
Aparna Bhagat
Kirtee Shingan

Cover Work
Kirtee Shingan

About the Author

James Kent Lewis has been in the computer industry for over 30 years. He started out writing BASIC programs in high school and used punch cards in college for his Pascal, Fortran, COBOL, and assembly language classes. Jim taught himself the C programming language by writing various utilities, including a fully-functional text editor, which he uses everyday. He started out using DOS and AIX, and then OS/2. Linux is now his operating system of choice.

Jim has worked in the past for several companies, including IBM, Texas Instruments, Tandem, Raytheon, Hewlett-Packard, and others. Most of these positions dealt with low-level device drivers and operating system internals. In his spare time, he likes to create video games in Java.

Jim has written articles for IBM Developer Works and has one patent.

First, I would like to thank Red Hat for creating a great OS. I used Fedora 17 to develop this book and it worked flawlessly. I would also like to thank my brother David for letting me bounce ideas off of him. Last, but certainly not least, I would like to thank my girlfriend, Gabriele. Her patience was greatly appreciated, and she also helped by lending me her Ubuntu laptop from time to time.

About the Reviewers

Samarendra Manohar Hedaoo is currently working towards the automation of the IT infrastructure at SocialTwist (`http://www.socialtwist.com`) using Puppet. He has around two years' experience in IT, most of which he has worked on creating and tweaking existing solutions for automation. He is an alumnus of Symbiosis Institute of Computer Studies & Research, where he spent most of his time on creating and implementing internal solutions for academic and official purposes of Symbiosis, including software that printed his own graduation degree. He also has a bit of experience in writing, which he did mostly at the organizations that he worked for, documenting solutions that he created, and also producing short tutorials for internal use. He has a keen interest in anthropology and would like to work on applying AI to the field of anthropology, someday. He enjoys reading technical books that explain technology using stories and non-technical books that talk about various aspects of daily life.

Jitesh Marathe has more than 14 years' experience in IT. He currently works with the MNC Product company as a Staff Engineer, where he acts as POC for projects and acting architect for a few of them, which involves designing and implementation of project and guiding teams though the project life cycle.

He is a graduate in Computer Applications and has spent a major time frame of his career, acting as a system administrator; he has close links with Linux/Unix and system administrator- related jobs.

He holds a passion for reading books and travelling to new places with his family.

Young-Ho Song has spent over 10 years working as a software engineer for IT companies that specialize in the pay TV industry and networking devices. He currently manages a global team of software engineers who work on a variety of difference subjects, and he has the responsibility to ensure that all Linux-based STBs (Set-Top Boxes) including NDS CA (Conditional Access) system are securely protected by Linux hardening review. He's also very familiar with RTOS (Real Time OS) embedded systems, multimedia delivery systems, and network security. He has a Ph.D. degree in Computer Engineering. You can refer to the following site for further information about him: `https://sites.google.com/site/profileyhsong`.

All the following books were written by him (co-authored) and were published in Korea:

- *Embedded Linux Based Mobile Device, INFO-TECH COREA*, March, 2006
- *Intel PXA255 XScale Based Embedded Linux System, SCITECH Media*, 2005
- *The Design and Implementation of the ARM9 Based SoC, INFO-TECH COREA*, March, 2006

Gene Wilburn is a Canadian writer, photographer, and computer specialist residing in Port Credit, Ontario, near Toronto. He has worked on various Unix and Unix-like operating systems, including AIX, Solaris, Linux, FreeBSD, and OpenBSD. He has been a programmer, database analyst, infrastructure specialist, and technical lead on projects. He has worked at Micromedia Ltd., Ontario Training Corporation, Royal Ontario Museum, and Canada Life. He is currently self employed.

As a writer, his work has appeared in Small Print Magazine, PC Week, Shutterbug, Infoworld, InfoAge, Toronto Star, Computing Canada, Computer Paper, Access, Here's How!, and Photo Life. He is the author of *Northern Journey: A Guide to Canadian Folk Music on CD, Reference Press Recreational Writing, lulu.com* and *Markdown for Writers*. He is a co-author of *Red Hat Linux System Administration Unleashed*, and he wrote the popular *Linux for Newbies, Wilburn Communications Ltd.* and *Linux Inside, Wilburn Communications Ltd., and* columns for The Computer Paper (Canada). His website is `genewilburn.com`.

I would like to thank Ken Thompson and Dennis Ritchie for creating Unix and C, and all the people who have subsequently developed Linux, FreeBSD, Mac OS X, iOS, and Android.

---

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Linux Utilities Cookbook shows how to solve typical problems on a Linux computer. The information is provided in a "recipe format" allowing the user to find desired topics quickly and efficiently. The steps to perform a task are clearly explained and have been tested for accuracy. There is also a section on shell scripting.

What this book covers

Chapter 1, *Using the Terminal / Command Line*, covers how to get the most out of the Linux command line.

Chapter 2, *The Desktop*, introduces some of the desktop environments available for Linux.

Chapter 3, *Files and Directories*, explains files, directories, and how to manage them.

Chapter 4, *Networking and the Internet*, covers connectivity and how to fix it when it goes down.

Chapter 5, *Permissions, Access, and Security*, gives a brief overview of Linux security features.

Chapter 6, *Processes*, explains how to manage processes in Linux.

Chapter 7, *Disks and Partitioning*, gives a brief insight into disk management.

Chapter 8, *Working with Scripts*, covers how to write scripts in Linux.

Chapter 9, *Automating Tasks Using Cron*, explains how to run jobs automatically.

Chapter 10, *The Kernel*, introduces how to make a custom kernel for your system.

Appendix A, *Linux Best Practices*, shows how to set up and run your systems like a pro.

Appendix B, *Finding Help*, covers locating the information you need quickly.

What you need for this book

To follow along with the examples in this book you will need a mainstream Linux distribution running on your computer. The author used Fedora by Red Hat to create this book along with the examples and scripts. However, any distro should work fine. Note that most can be downloaded and installed free of charge from the manufacturer's website.

Who this book is for

This book is intended for somewhat experienced computer users who want to know more about Linux. The recipe format is designed to allow quick access to typical tasks that come up often.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We can include other contexts through the use of the `include` directive."

Any command-line input or output is written as follows:

```
export PS1="screen$WINDOW \h \u \w \$ "
```

New terms and **important words** are shown in **bold**. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Commands that are part of a text section will be indicated like this: run `cd /tmp`

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Using the Terminal / Command Line

In this chapter we will cover:

- Command retrieval and line editing
- Using history
- Filename auto-completion
- The shell prompt
- Other environment variables
- Using aliases
- The `.bashrc` file
- Dealing with blanks and special characters in filenames
- Understanding the `$?` variable
- Redirection and piping
- Sending output from one terminal to another
- Using the Screen program

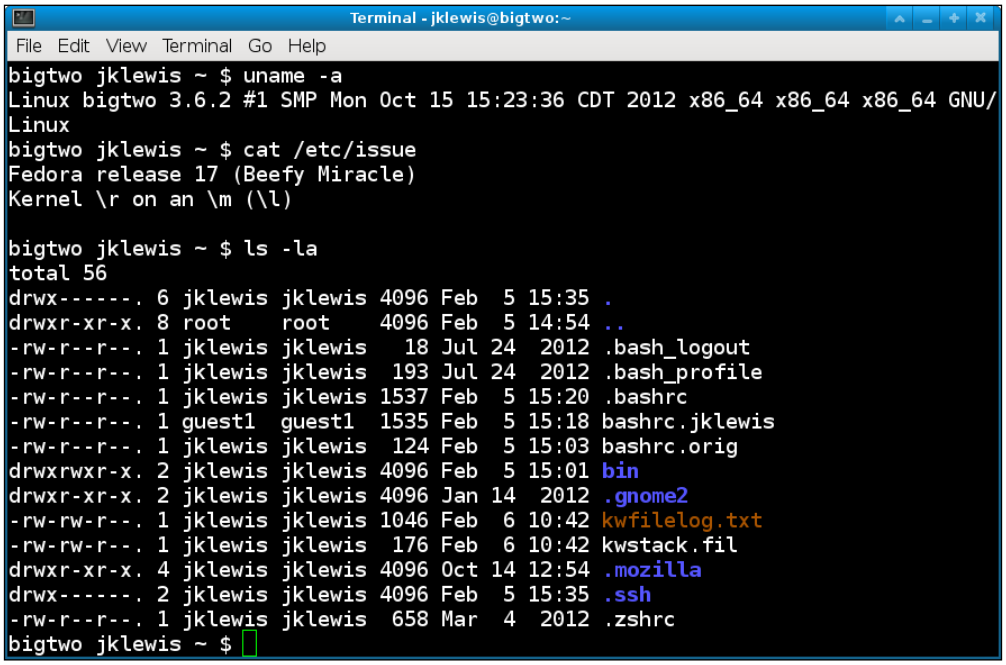
Introduction

Knowing how to use the command line efficiently will really help you get the most out of your computer. There are many ways to save time and effort when typing commands, you just need to know what they are.

There are many different Linux environments available. This chapter focuses on the popular **Bash** shell.

Command retrieval and line editing

A standard Bash terminal is automatically set to insert mode, so you don't have to press the *Insert* key to insert text. Use the up and down arrow keys to recall a previous command, and then other cursor keys to edit that line as needed.



Getting ready

All you need for this example is a terminal running the Bash shell. Other terminals may not have these capabilities.

How to do it...

We will run a few commands as follows:

1. Type in the command `route` and press the *Enter* key.
2. Do the same for `uptime`, `ls`, `date`, and `sync`, pressing *Enter* after each command.
3. Now press the up arrow key one time. You should see the following command:

   `sync`

4. Now press the up arrow two more times. You should see `date` and `ls`.
5. Press *Enter*. The `ls` command will run again. Pressing *Enter* will always run the command shown.

How it works...

The line is stored in a buffer with full editing capabilities. This buffer is sent to the OS when the *Enter* key is pressed.

The summary of the keys used for retrieval and editing is as follows:

- **Up arrow**: It is used to scroll up the history buffer
- **Down arrow**: It is used to scroll down the history buffer
- **Home**: It is used to bring the cursor to the beginning of the line
- **End**: It is used to bring the cursor to the end of the line
- **Delete**: It is used to delete the character to the right of the cursor
- **Backspace**: It is used to delete the character to the left of the cursor and shift the line
- **Left and right arrow**: These are the cursor movement keys

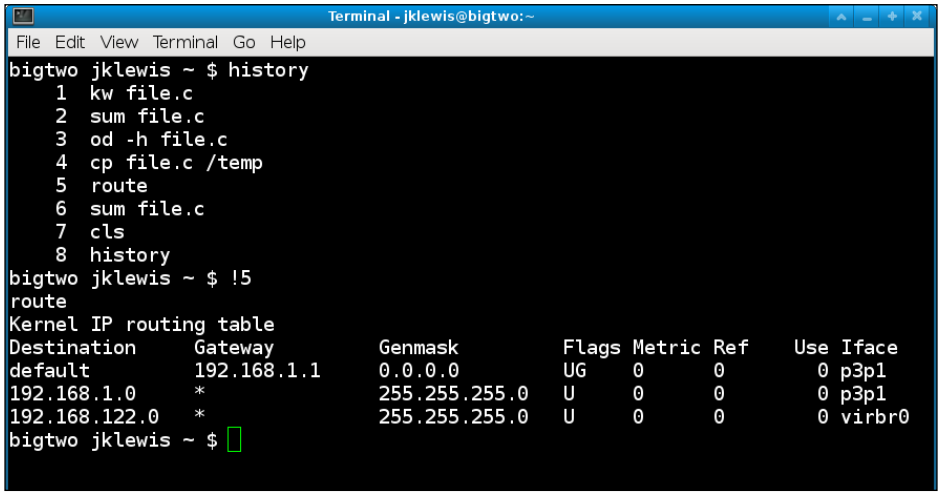
Using history

The standard Bash shell includes a **history** function. It records each command in a database that can be shown by running the `history` command. In this section we have shown how this is done.

Getting ready

All you need is a Bash terminal to follow the given steps.

See the following screenshot:



How to do it...

1. Run a few commands such as `route`, `uptime`, `date`, and `sync`.
2. Run the `history` command.
3. Look for a command you would like to run again, but instead of typing the command, type an exclamation point (`!`) and then the number next to the command as shown in the history listing, and press *Enter*.
4. That command will run again.

How it works...

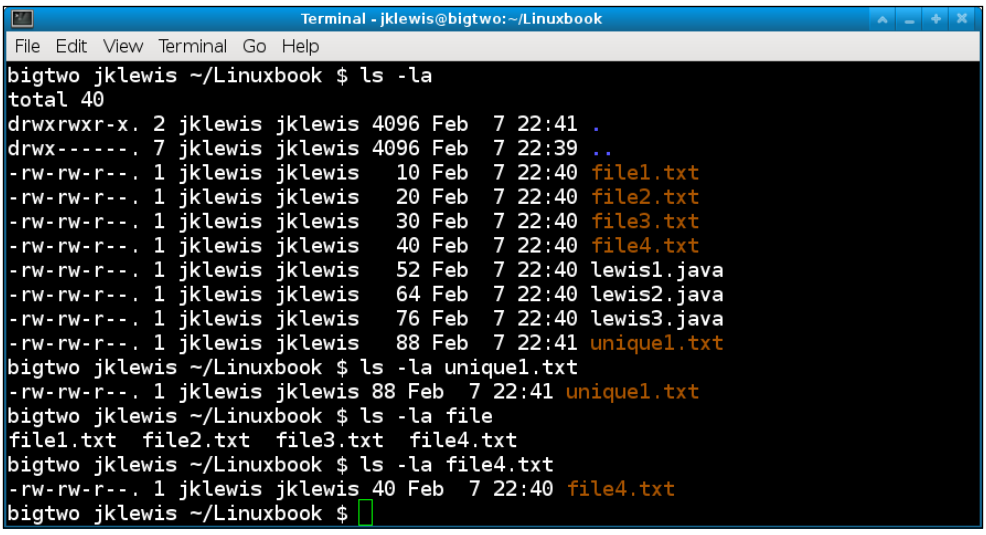
Think of the command line history as a linear database. You can scroll up and down until you see the command you want. This is also helpful to recall something you did a while back. The `HISTSIZE` environment variable controls how many commands will be saved in the buffer.

Be careful with this feature. Make sure you have the correct command before running it.

Filename auto-completion

When running a command, you do not have to type the entire filename. This saves a lot of time and effort, and also helps prevent typos.

The *Tab* key is used to invoke filename auto-completion. See the following screenshot:



You only need to type enough characters to make the filename you want unique, and then press *Tab*. If you didn't type enough characters, you will hear the console beep (in most shells). If you now press *Tab* again, all of the possibilities will be displayed.

Getting ready

All you need for this example is a terminal running the Bash shell.

How to do it...

1. Change to your home directory, in my case it's:

   ```
   cd /home/jklewis.
   ```

2. Create a directory using the following command:

   ```
   mkdir Linuxbook
   ```

3. Change to it `Linuxbook` using the following command:

   ```
   cd Linuxbook
   ls > file2.txt
   ls > file3.txt
   ls > file4.txt
   ls > unique1.txt
   ```

4. Now let's create some dummy files; run using the following command:

   ```
   ls > file1.txt
   ```

5. Now type `ls -la u` and then press *Tab*. The rest of the filename **"unique1.txt"** will appear. Press *Enter*.
6. Now type `ls -al file` and press *Tab*. Nothing will happen, and your console may beep. Press *Tab* again. Now all 4 filenames will appear.
7. Add a `4`, press *Tab* again, and then *Enter*. The command `ls -la file4` will run.

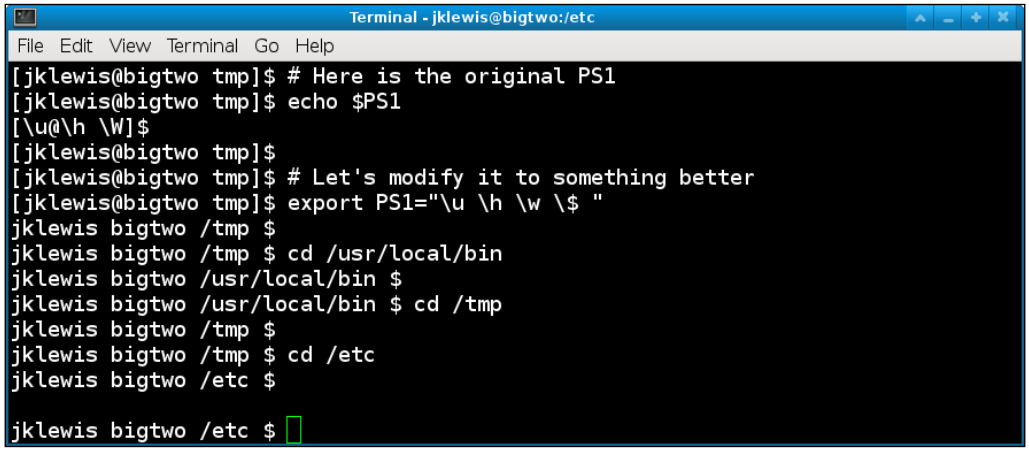
This may look complicated or even clumsy but if you give it a chance you will become an expert in no time at all.

The shell prompt

A standard terminal usually has a rather cryptic command line prompt. This should be changed by modifying the **PS1** environment variable.

How to do it...

An example is shown in the following screenshot:



Refer to the line **export PS1="\u \h \w \$ "**

1. The `\u` command means to show the current user of this shell.
2. The `\h` command shows the hostname of this machine.
3. The `\w` command means to show the full path of the current directory. This change is highly recommended, as the user doesn't have to type **pwd** (**Print Working Directory**) all the time to know what directory is being used.
4. The `\$` means to display a `$` or `#` depending on the effective UID.

There's more...

There are many more options, such as showing the time and date, using colors, and so on. For more information, run `man bash` and search for `PS1`.

Other environment variables

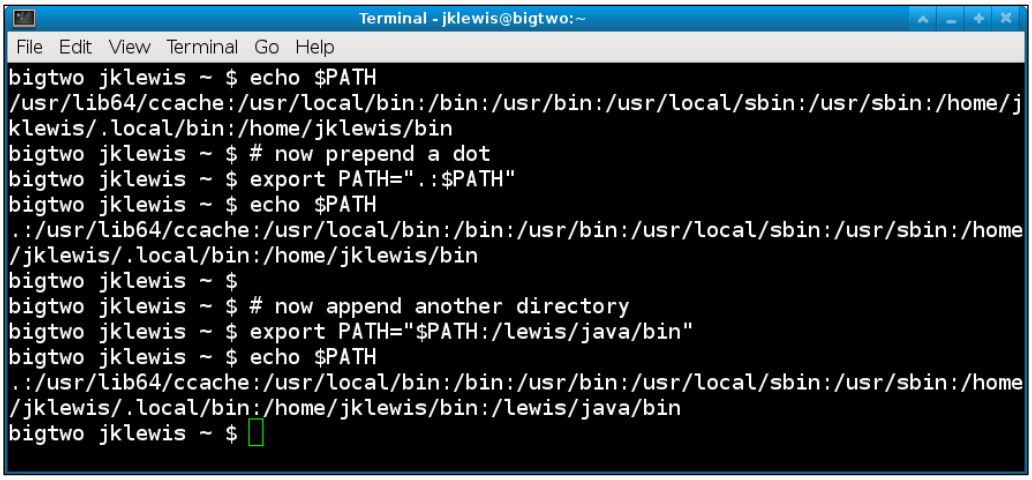
The `PS1` variable is only one of literally hundreds of environment variables. Don't worry, you don't have to know them all! The following are a few very useful ones:

- `PS1`: It shows and sets the command line prompt
- `USER`: It shows the current user
- `HOSTNAME`: It shows the current hostname for this machine
- `HOME`: It shows the home directory of the current user

- `SHELL`: It shows the current shell this terminal is running in
- `TERM`: It shows which terminal type is being used
- `PATH`: It shows and sets the directories where programs are searched for
- `PWD`: It shows the current working directory
- `EDITOR`: It can be set to the full path to your desired text editor for use with certain commands such as `crontab -e`
- `TZ`: It shows and sets the time zone variable
- `HISTSIZE`: It shows and sets the size of the history buffer

Most of these are self-explanatory; however, a few need more discussion. The `PATH` environment variable is where commands are searched for in the filesystem.

The `echo` command is used to display the contents of a variable:



How to do it...

1. Prepending a dot to the `PATH` means the program will be looked for in the current directory first, before searching the rest of the path. This is very useful during the code development for example. Do this by running:

   ```
   export PATH=".:$PATH"
   ```

2. The `EDITOR` variable can be set to your favorite text editor. Most people use `vi` (or `vim`); however, you can point it to the one you want. If you change this, be sure to use the full path. To change the `EDITOR` variable do this:

   ```
   export EDITOR=/lewis/bin64/kw
   ```

3. An `export` can be removed by setting it to nothing:

   ```
   export EDITOR=
   ```

4. By convention, environment variables are usually written in uppercase. View the man pages and/or search Google for more information on these variables.

How it works...

Think of these environment variables just as you would if you were using a programming language. In this case, the type of the variable is determined by the OS. For example, you could type `A=1` or `A="This is a string"`.

The OS knows the difference. Also, there is variable scope. Notice I did not use `export` above. That means this `A` is local to this shell. Only exporting a variable will make it available to other shells (after sourcing the file).

Using aliases

Wouldn't it be nice if you could easily create a simple command without having to make a script out of it? Well, there is a way. This is done using **aliases**.

How to do it...

The following are the steps to create an alias:

1. Type `tput clear` and press *Enter*. Your screen should have cleared.
2. Now enter `alias cls="tput clear"`. Now when you run `cls` it will do the same thing.
3. Let's create some more. To show a long directory listing enter `alias la="ls -la"`. Enter `'la'` to run the alias.
4. To show a long listing with the most current files last enter `'alias lt="ls -latr"'`.

If you create an alias and then decide you no longer want it you can remove it by using the `unalias` command, for example, `unalias cls`.

You can also use aliases to move around the filesystem efficiently. This is very handy and will save you an incredible amount of typing. Here are some examples:

1. `mkdir /home/jklewis/linuxbook`
2. `alias lbook="cd /home/jklewis/linuxbook"`
3. `lbook`

You will now be taken to that directory. Here is something I make frequent use of on my systems:

1. `export LBOOK="/home/jklewis/linuxbook"`
2. `alias lbook="cd $LBOOK"`
3. `lbook`

As you can see, running `lbook` will take you to the directory as shown above. However, you can also use the `LBOOK` variable to copy files to that directory:

1. `cd /tmp`
2. `touch f1.txt`
3. `cp f1.txt $LBOOK`

The file `f1.txt` will now exist in the `/home/jklewis/linuxbook` directory. This becomes even more handy when extremely long filenames are used.

To remove the `lbook` alias run `unalias lbook`

You can list your aliases by just running `alias` without any parameters. Any time you find yourself constantly typing the same commands or filenames consider creating an alias for it.

There's more...

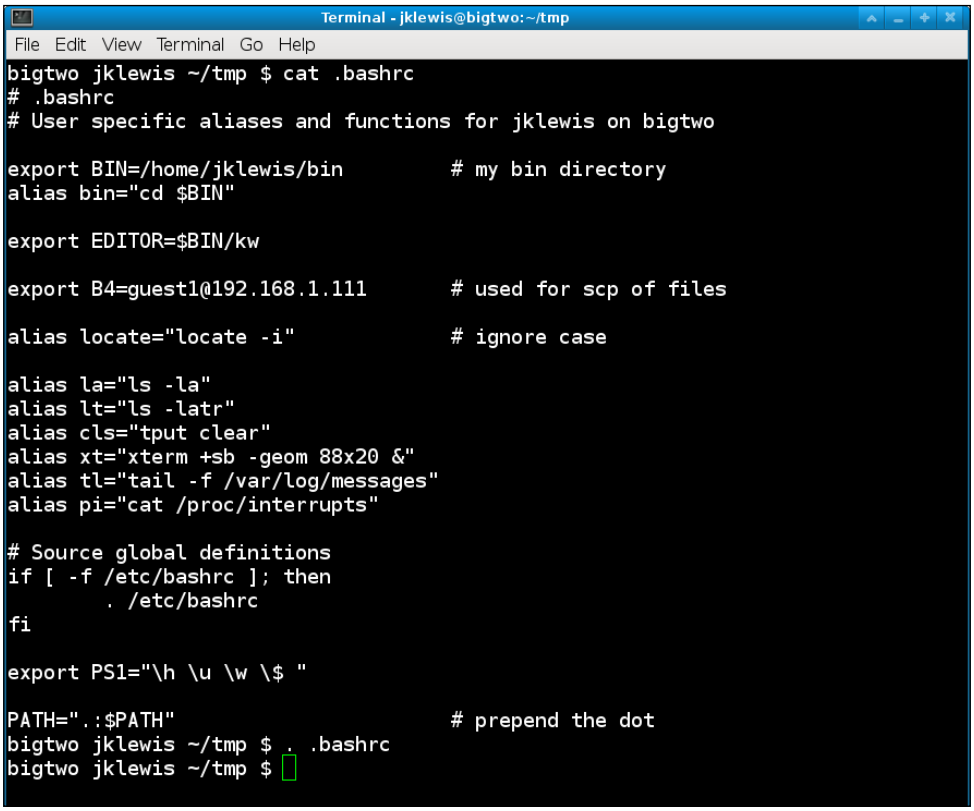
Note that the above examples will only be effective in that terminal and will not persist across a reboot. See the next section on how to make the changes permanent.

Also, in some cases, what you want to do may be too complicated for an alias, for example, to check for the proper number of parameters. This is where you can create a shell script, which will be covered in *Chapter 8*, *Working with Scripts*.

The .bashrc file

There are many environment variables we can look at and change. However, we certainly don't want to enter these every time we start a new shell. There is a special file, named `.bashrc`, which is used to store your settings. It is located in the user's home directory. For example, the `.bashrc` file for the root user is in the `/root` directory.

Here is a `.bashrc` file from one of my systems:



How to do it...

The description of the lines is as follows:

1. To comment a line, precede it with a `#` symbol.
2. To create a variable, use the `export` tag.
3. To create an alias, use the `alias` tag (as shown earlier in this chapter).
4. Control statements are allowed; see the `if` clause in the previous screenshot.

5. After modifying your `.bashrc` file, remember to source it using the dot operator as follows:
   `. .bashrc`

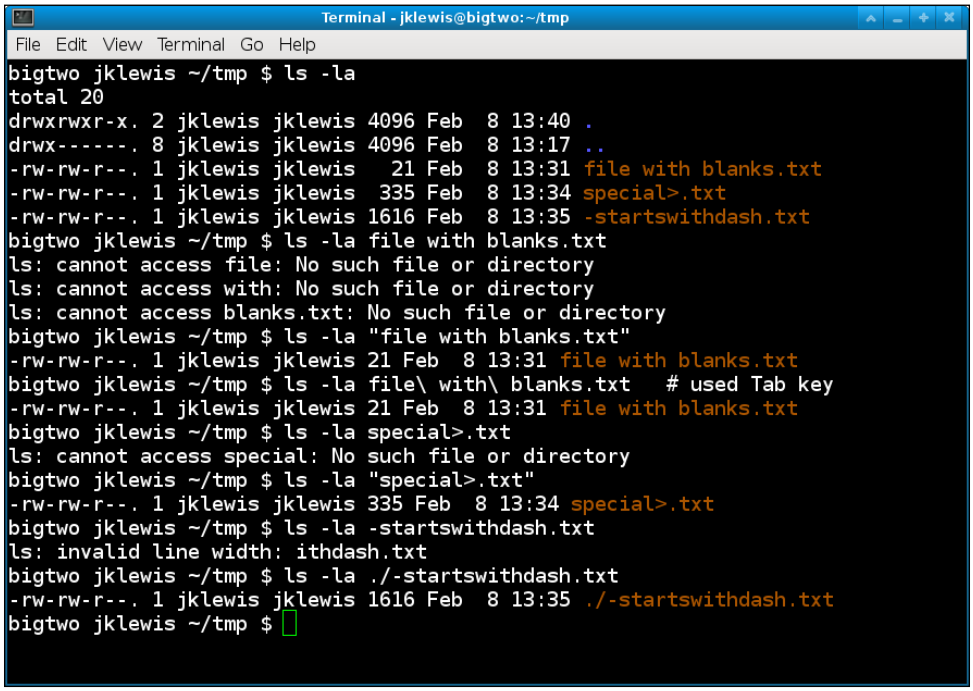
Dealing with blanks and special characters in filenames

Linux (and Unix) filesystems were not originally designed to handle blanks in filenames. This can cause quite a few problems, as the shell treats each item after a blank as another file or parameter. A solution is to use quotes, the backslash, or the *Tab* key.

The following sections assume the user has not modified the Bash **Internal Field** Separator (**IFS**) variable.

How to do it...

See the following screenshot. I purposely created three "bad" filenames:



1. Run `ls -la file with blanks.txt` and notice the errors.

2. Now run it again, but enclose the filename in quotes: `ls -la "file with blanks.txt"`; it will work properly now.
3. Enter `ls -la file` and press *Tab*. It will escape the blanks for you.
4. Run `ls -la special>.txt`. Observe the error.
5. Enclose in quotes as before using the following command:
   `ls -la "special>.txt"`
6. Now try `ls -la -startswithdash.txt` and then try quoting it. Doesn't work, right?
7. Precede the filename with the `./` operator using the following command:
   `ls -la ./-starWtswithdash.txt`

As you can see, this can also be a problem if special characters have been used in the filename. Study this one a bit and it will become clear. Remember the *Tab* key; it works really well for just about every case. If the file starts with a dash, use the `./` operator. It means to refer to the file in the current directory.

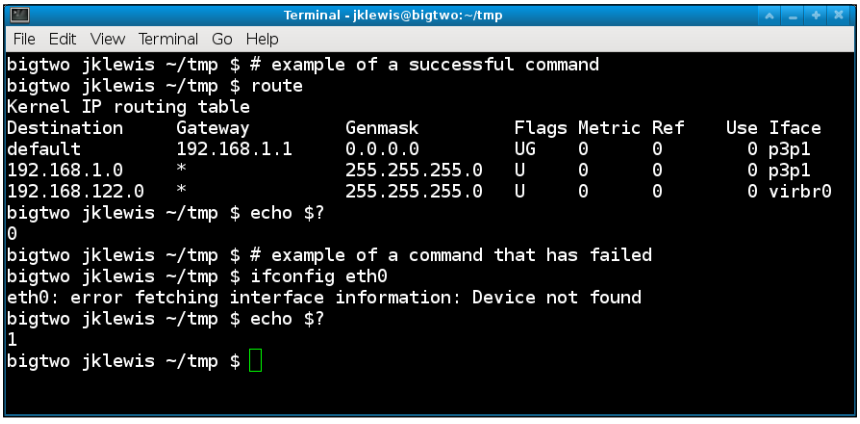
There's more...

The issue of blanks and special characters is even more of a problem in scripts. This will be covered in more detail in *Chapter 8*, *Working with Scripts*.

Understanding the $? variable

Typically, when a command is run in Linux it performs a task; it either reports what it did or indicates an error occurred. An internal return code is also generated, and is displayed by running the `echo $?` command. Note that this must be the very next thing typed after the original command.

The following screenshot shows `echo $?`:



How to do it...

Here is a quick example of **echo $?**:

1. Run the following command:
   ```
   ping -c 1 packt.com
   ```
2. It should succeed. Run the following command:
   ```
   echo $?
   ```
3. You should get a `0` for success.
4. Now run the following command:
   ```
   ping -c 1 phooey
   ```

It may take a second or two to complete.

5. Run `echo $?` again. It should return a non-zero value to indicate failure.

How it works...

In general, a return of zero means success. A non-zero return means an error has occurred, and in many cases the code returned indicates what the error was. Remember this the next time you type a command, hit *Enter*, and get the shell prompt back without anything appearing to happen.

There's more...

The `man` and `info` pages for a command typically contain an entry showing what the errors mean. If the `man` page is lacking, consult the web.

Redirection and piping

Suppose you run a command, say `route`, and want to save the output in a file. The **redirection** (`>`) operator is used to do this instead of sending the output to the screen.

How to do it...

Let's try some redirection:

1. Enter `ifconfig > file1.txt`. You won't see anything, because the output went into the file.
2. Run `cat file1.txt`. You should now see the output.
3. This works the other direction as well, to read from a file run the following command:

   ```
   sort < file1.txt
   ```

4. You can even do both in one step:

   ```
   sort < file1.txt > output-file.txt
   ```

5. You can also send the output to another command using the pipe operator. For example, run `route | grep eth0`. The above command would display only the lines from `route` that contain the phrase `eth0`.

There's more...

Here is something that I use all the time. Say I have written a program in C a long time ago, have several versions, and want to find the latest one. I could run `locate` to find them all:

```
locate crc.c
```

This might return quite a few lines. How can I run `ls` on each file to find the latest one? By piping the output into the `xargs` command and then `ls`:

```
locate crc.c | xargs ls -la
```

This will now show the time and date of each file.

This might seem a bit complicated at first, but if you experiment a little it will become second nature to you.

Sending output from one terminal to another

This is a really handy feature that is unique to Linux/UNIX systems. It's most useful in scripts but can be used on the command line as well. If you have a system available try the given steps.

Getting ready

You will need two open terminals.

How to do it...

We show how to send the output from one terminal to another in the following steps:

1. In one terminal run the `tty` command. The output should be something like `/dev/pts/16`.
2. In the other terminal run the `route` command. You will see the output in that terminal.
3. Now run `route` again, but now using the command:

   ```
   route > /dev/pts/16
   ```

4. The output will go to that other terminal.

How it works...

Terminals on Linux systems are devices that have their own buffer space. By referring to the device by name you can write to it.

There's more...

This feature is even more useful in scripts, which we will see in *Chapter 8*, *Working with Scripts*.

Using the Screen program

Screen is a full-screen window manager that shares a physical terminal with other processes (which are usually other terminals/shells). It is normally used when no other manager or desktop is available, such as on a server. It has a scroll-back history buffer and also allows for copy and paste of text between windows.

Getting ready

The following is a brief list of some of the many key bindings available with Screen:

- *Ctrl* + *A* + *?*: It displays a list of commands and their key bindings
- *Ctrl* + *A* + *C*: It brings up a new window
- *Ctrl* + *A* + *D*: It detaches a window
- *Ctrl* + *A* + *N*: It is used to go to the next window in the sequence
- *Ctrl* + *A* + *P*: It is used to go to the previous window in the sequence
- *Ctrl* + *A* + # (where # is a number): It is used to go directly to that window
- *Ctrl* + *A* + *"*: It shows the list of windows; user can select any one by the number

The following is a list of frequently used commands:

- `screen -list`: It shows all of the windows
- `screen <program>`: It creates a new window and run that program in it

How to do it...

An example of running the Screen utility is as follows:

1. In a terminal run the `screen -L` command.
2. Now press *Ctrl* + *A* and then press *C*. This will create another window.
3. Do this two more times.
4. Try typing *Ctrl* + *A* + *0*.
5. Try *Ctrl* + *A* + *3*.

How it works...

In the previous section, step 1 will create a new window, `window 0`. If you are running inside a window manager you may notice the title change showing which window it is.

Step 2 will create another window. After step 3, you will have 4 windows in total.

When you perform the actions in step 4, you should be in `window 0`. Typing *Ctrl* + *a* + *3* will take you to `window 3`.

There's more...

Here is a helpful hint, if you are running only a command line with no desktop, you may want to change your `PS1` variable to something like the following in your `.bashrc` file:

```
export PS1="screen$WINDOW \h \u \w \$ "
```

Now the prompt will always show which window you are in.

This describes only a small part of what Screen can do. Consult the `man` page for more information.

2
The Desktop

In this chapter we will cover these desktop environments:

- GNOME 2
- KDE desktop
- xfce
- LXDE
- Unity
- Mate

Introduction

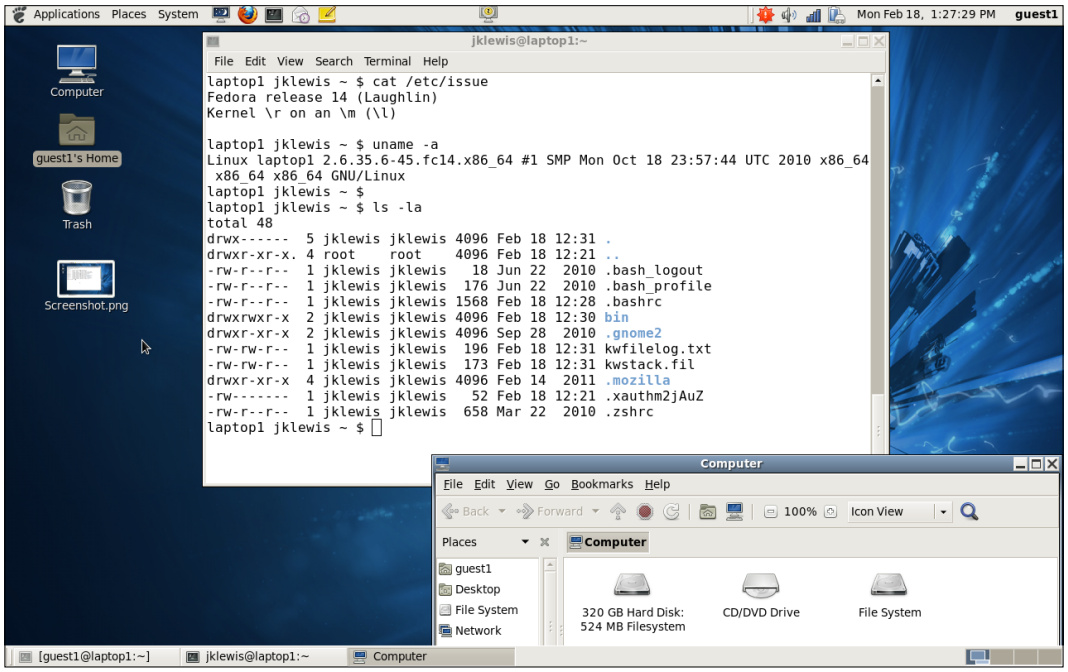
A computer desktop is normally composed of windows, icons, directories/folders, a toolbar, and some artwork. A window manager handles what the user sees and the tasks that are performed. A desktop is also sometimes referred to as a **graphical user interface** (**GUI**).

There are many different desktops available for Linux systems. Here is an overview of some of the more common ones.

GNOME 2

GNOME 2 is a desktop environment and GUI that is developed mainly by Red Hat, Inc. It provides a very powerful and conventional desktop interface. There is a launcher menu for quicker access to applications, and also taskbars (called **panels**). Note that in most cases these can be located on the screen where the user desires.

The screenshot of GNOME 2 running on Fedora 14 is as follows:

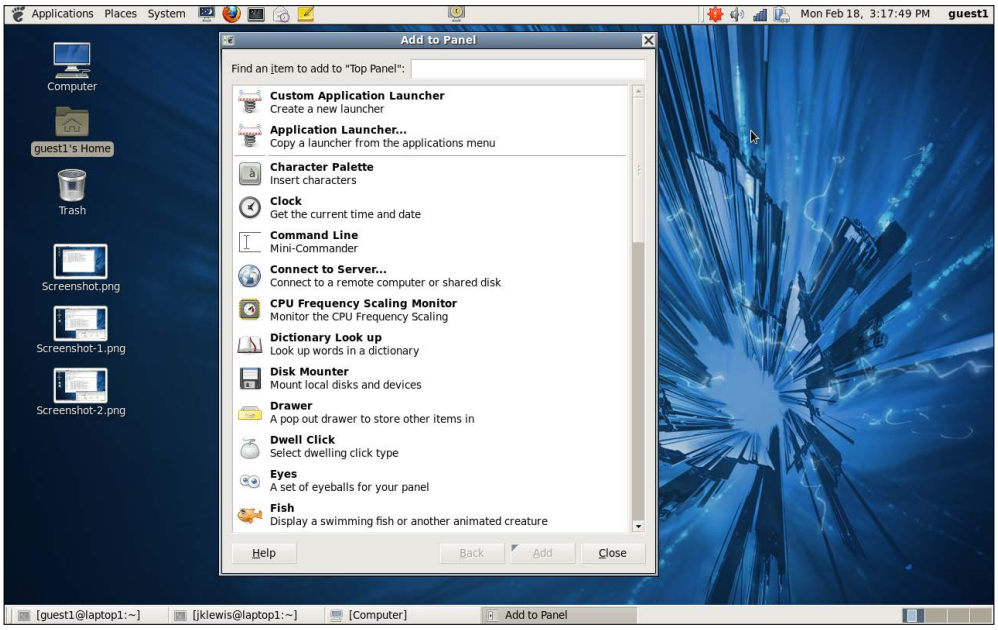


This shows the desktop, a command window, and the **Computer** folder. The top and bottom "rows" are the panels. From the top, starting on the left, are the **Applications**, **Places**, and **System** menus. I then have a screensaver, the Firefox browser, a terminal, **Evolution**, and a Notepad. In the middle is the lock-screen app, and on the far right is a notification about updates, the volume control, Wi-Fi strength, battery level, the date/time, and the current user. Note that I have customized several of these, for example, the clock.

Getting ready

If you have a computer running the GNOME 2 desktop, you may follow along in this section. A good way to do this is by running a Live Image, available from many different Linux distributions.

The screenshot showing the **Add to Panel** window is as follows:



How to do it...

Let's work with this desktop a bit:

1. Bring this dialog up by right-clicking on an empty location on the task bar.
2. Let's add something cool. Scroll down until you see **Weather Report**, click on it and then click on the **Add** button at the bottom.
3. On the panel you should now see something like **0 °F**. Right-click on it.
4. This will bring up a dialog, select **Preferences**.
5. You are now on the **General** tab. Feel free to change anything here you want, then select the **Location** tab, and put in your information.
6. When done, close the dialog. On my system the correct information was displayed instantly.
7. Now let's add something else that is even more cool. Open the **Add to Panel** dialog again and this time add **Workspace Switcher**.
8. The default number of workspaces is two, I would suggest adding two more. When done, close the dialog.
9. You will now see four little boxes on the bottom right of your screen. Clicking on one takes you to that workspace. This is a very handy feature of GNOME 2.

There's more...

I find GNOME 2 very intuitive and easy to use. It is powerful and can be customized extensively. It does have a few drawbacks, however. It tends to be somewhat "heavy" and may not perform well on less powerful machines. It also does not always report errors properly. For example, using Firefox open a local file that does not exist on your system (that is, `file:///tmp/LinuxBook.doc`). A **File Not Found** dialog should appear. Now try opening another local file that does exist, but which you do not have permissions for. It does not report an error, and in fact doesn't seem to do anything. Remember this if it happens to you.

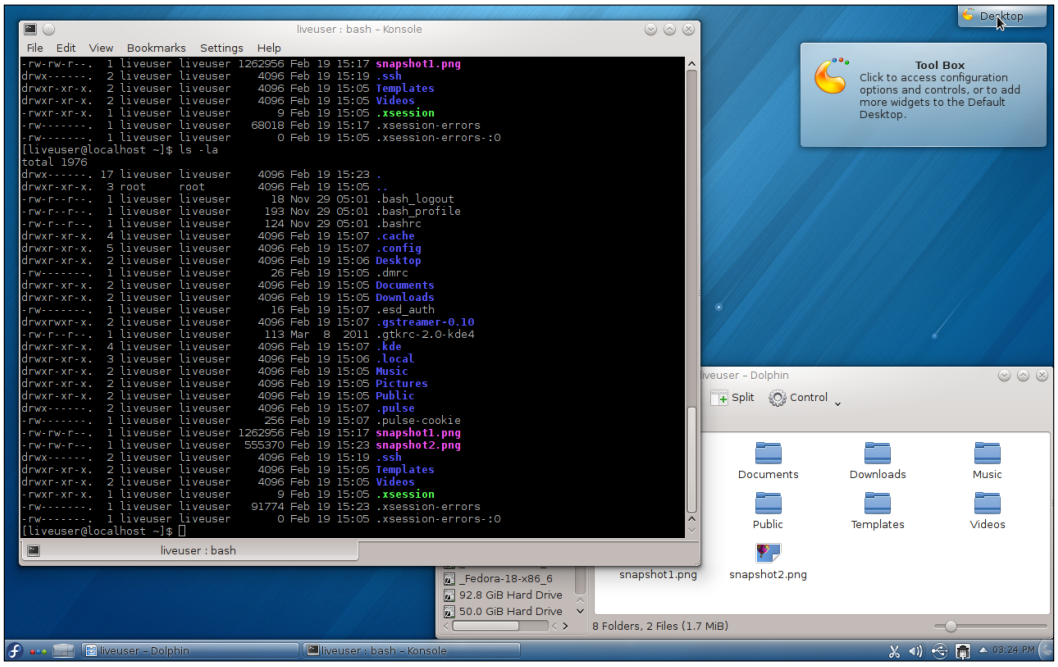
KDE desktop

The **KDE** desktop was designed for desktop PCs and powerful laptops. It allows for extensive customization and is available on many different platforms. The following is a description of some of its features.

Getting ready

If you have a Linux machine running the KDE desktop you can follow along. These screenshots are from KDE running on a Live Media image of Fedora 18.

The desktop icon on the far right allows the user to access **Tool Box**:

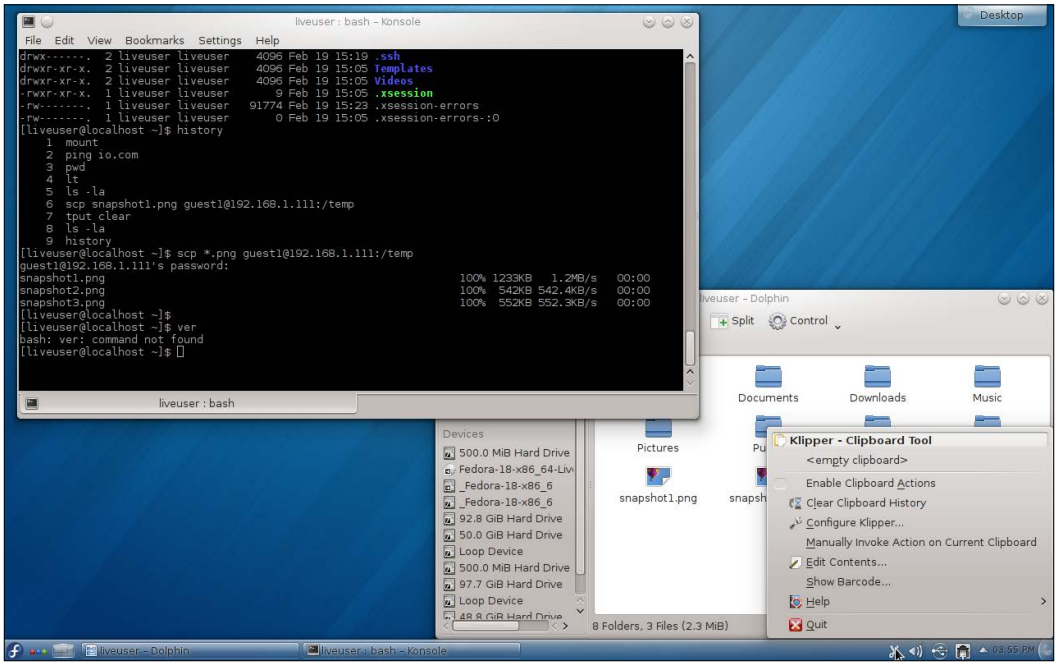


You can add panels, widgets, activities, shortcuts, lock the screen, and add a lot more using this dialog.

The default panel on the bottom begins with a Fedora icon. This icon is called a **Kickoff Application Launcher** and allows the user to access certain items quickly. These include **Favorites**, **Applications**, a **Computer** folder, a **Recently Used** folder, and a **Leave** button.

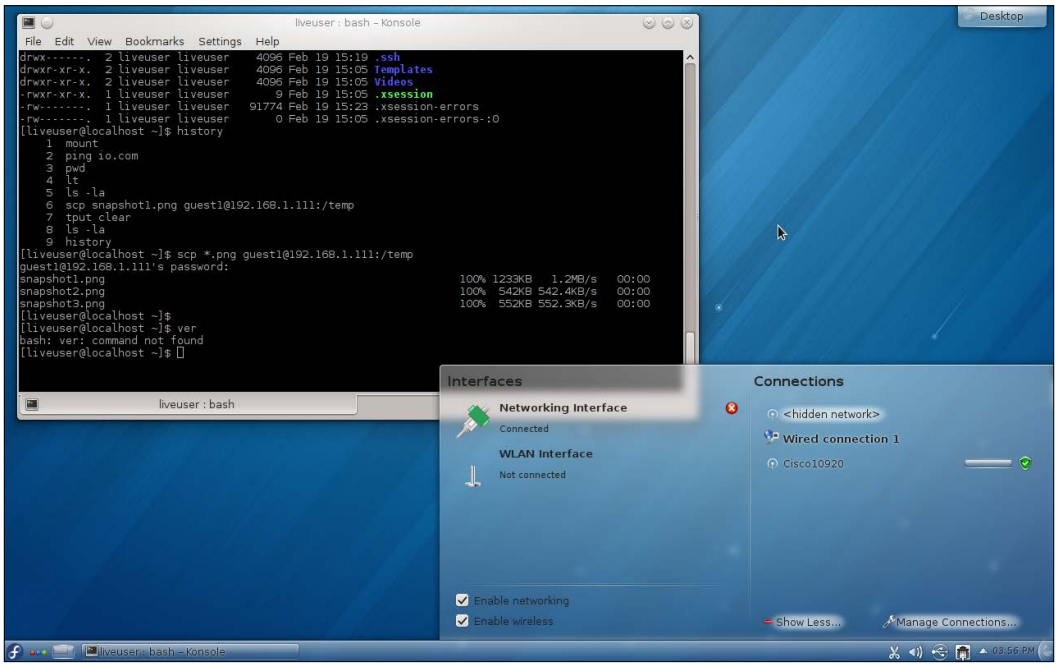
If you click on the next icon it will bring up the **Activity Manager**. Here you can create the activities and monitor them. The next icon allows you to select which desktop is currently in the foreground, and the next items are the windows that are currently open. Over to the far right is the **Clipboard**.

Here is a screenshot of the clipboard menu:



Next is the volume control, device notifier, and networking status.

Here is a screenshot of **Interfaces** and **Connections** dialog:



Lastly, there is a button to show the hidden icons and the time.

How to do it...

Let's add a few things to this desktop:

1. We should add a console. Right-click on an empty space on the desktop. A dialog will come up with several options; select **Konsole**. You should now have a terminal.
2. Close that dialog by clicking on some empty space.
3. Now let's add some more desktops. Right-click on the third icon on the bottom left of the screen. A dialog will appear, click on **Add Virtual Desktop**. I personally like four of these.
4. Now let's add something to the panel. Right-click on some empty space on the panel and hover the mouse over **Panel Options**; click on **AddWidgets**.
5. You will be presented with a few widgets. Note that the list can be scrolled to see a whole lot more. For example, scroll over to **Web Browser** and double-click on it.
6. The web browser icon will appear on the panel near the time.

There's more...

You can obviously do quite a bit of customization using the KDE desktop. I would suggest trying out all of the various options, to see which ones you like the best.

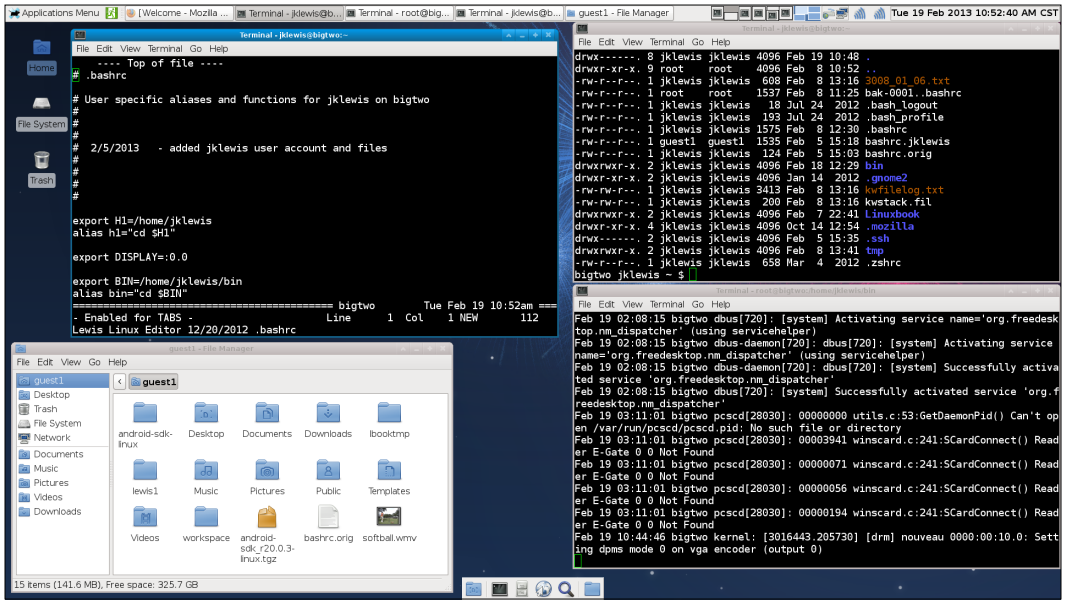
KDE is actually a large community of open source developers, of which KDE Plasma desktop is a part. This desktop is probably the heaviest of the ones reviewed, but also one of the most powerful. I believe this is a good choice for people who need a very elaborate desktop environment.

xfce

xfce is another desktop environment for Linux and UNIX systems. It tends to run very crisply and not use as many system resources. It is very intuitive and user-friendly.

Getting ready

The following is a screenshot of xfce running on the Linux machine I am using to write this book:



If you have a machine running the xfce desktop, you can perform these actions. I recommend a Live Media image from the Fedora web page.

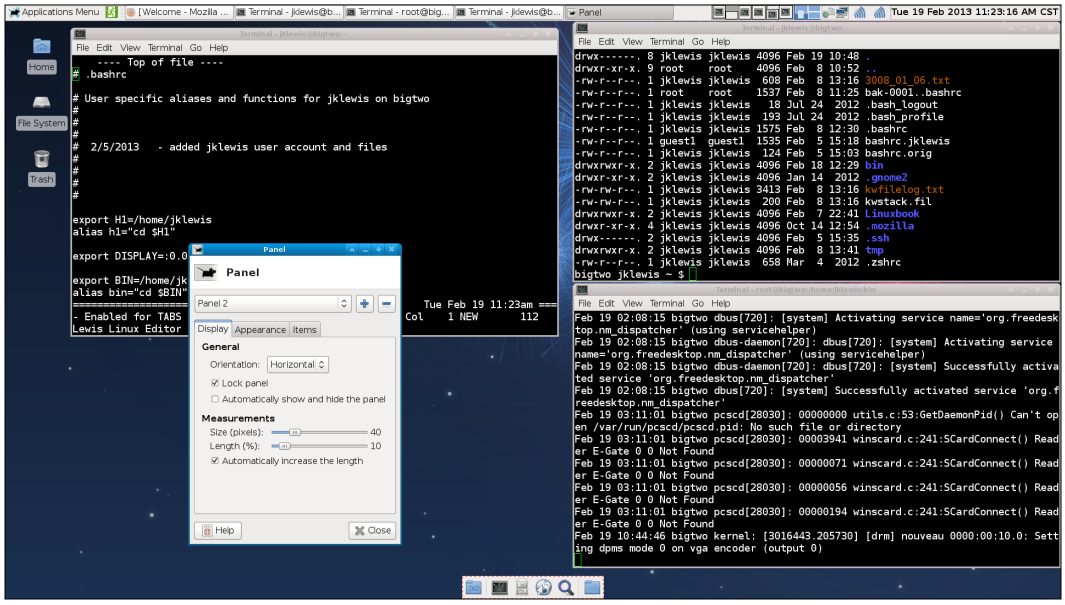
While somewhat similar to GNOME 2, the layout is somewhat different. Starting with the panel on the top (**panel 1**) is the **Applications Menu**, following by a **LogOut** dialog. The currently open windows are next. Clicking on one of these will either bring it up or minimize it depending on its current state. The next item is the **Workspaces** of which I have four, then the Internet status. To complete the list is the volume and mixer apps and the date and time. The screen contents are mostly self-explanatory; I have three terminal windows open and the **File Manager** folder.

The smaller panel on the bottom of the screen is called **panel 2**.

How to do it...

Let's work with the panels a bit:

1. In order to change panel 2 we must unlock it first. Right-click on the top panel, and go to **Panel** | **PanelPreferences**.
2. Use the arrows to change to panel 2. See the screenshot below:



3. You can see it is locked. Click on **Lock panel** to unlock it and then close this dialog.
4. Now go to panel 2 (on the bottom) and right-click on one of the sides. Click on **AddNewItems...**.
5. Add the applications you desire.

There's more...

This is by no means an exhaustive list of what xfce can do. The features are modular and can be added as needed. See `http://www.xfce.org` for more information.

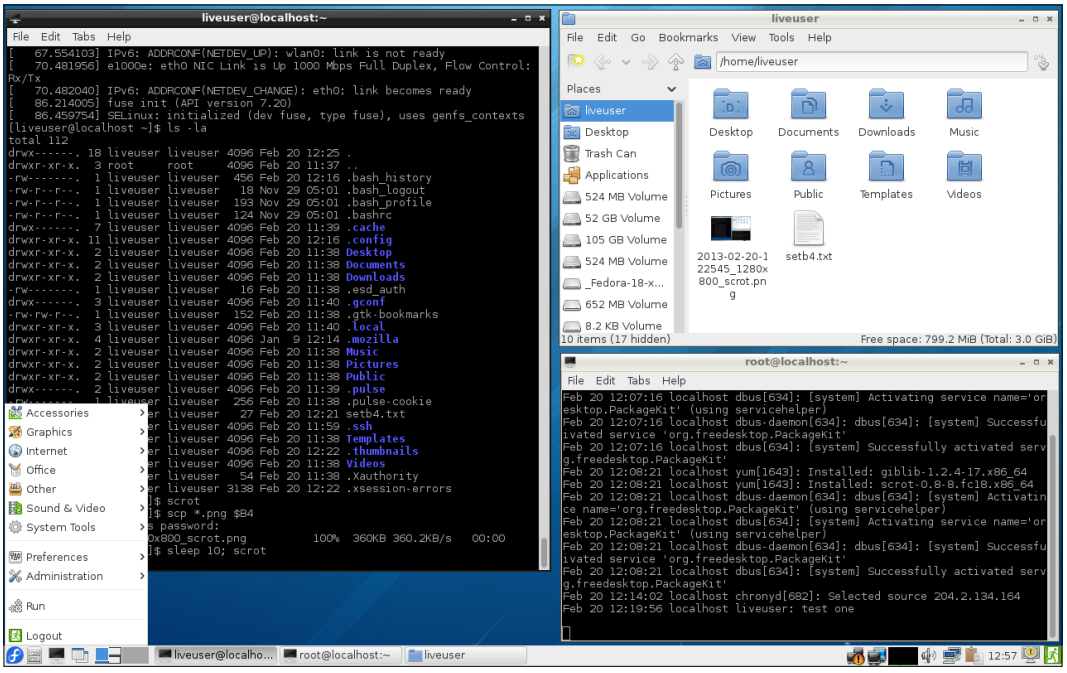
LXDE

LXDE (**Lightweight X11 Desktop Environment**) was designed to work well in low resource conditions and is a relatively new environment. Unlike most of the other desktops, the components of LXDE do not have many dependencies and can run independently.

Getting ready

If you have a machine using this desktop you can follow along with this section.

This is a screenshot of LXDE running on a Live Media image of Fedora 18:



As you can see, there are two terminals open and the file manager. Starting on the left of the panel is the familiar-looking Fedora icon, which has just been clicked on. It brings up the pulldown as shown. The next icon is the file manager and then an LXTerminal.

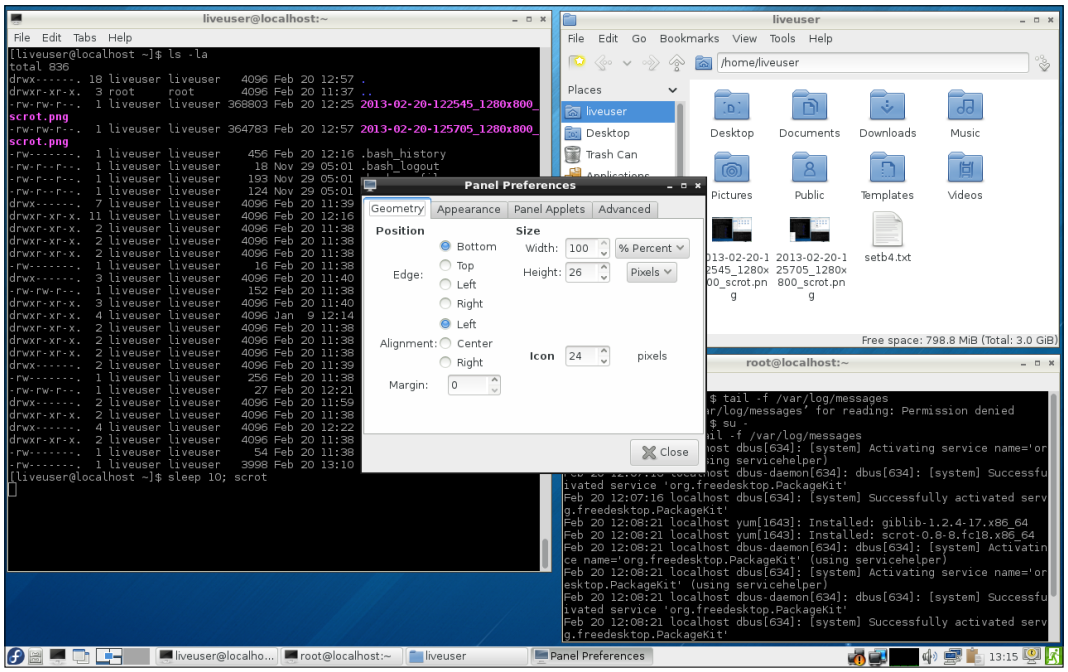
The next icon says "Left click to iconify all windows. Middle click to shade them". I chose to leave this icon as is.

The next are two desktop icons, and then the event list. Farther to the right is a Wi-Fi icon (Wi-Fi not activated), a wired Ethernet status, a system monitor, volume control, and the Network Manager Applet. After that is the clipboard manager, time, a lock-screen icon, and a logout box.

How to do it...

Let's work with this desktop a bit:

1. Right-click on an empty spot of the panel, a pulldown will be displayed.
2. Click on **Panel Settings**. The following screen will pop up:



3. Let's change the font size. Click on **Appearance**, and then **Size** under **Font**.
4. Using the scroll keys change the value to something else. The change will appear instantly. When it looks good, select **Close**.
5. Let's add an app. Bring up the panel settings again and click on **Add / Remove Panel Items**.
6. Click on **Add**, scroll down and click on **Desktop Number / Workspace Name**. The name of the workspace you are currently in shows up on the far right of the panel. I personally like this feature a lot.

There's more...

I found LXDE to be very intuitive and fast. I believe it would work well, particularly on laptops and mobile devices, where power is at a premium.

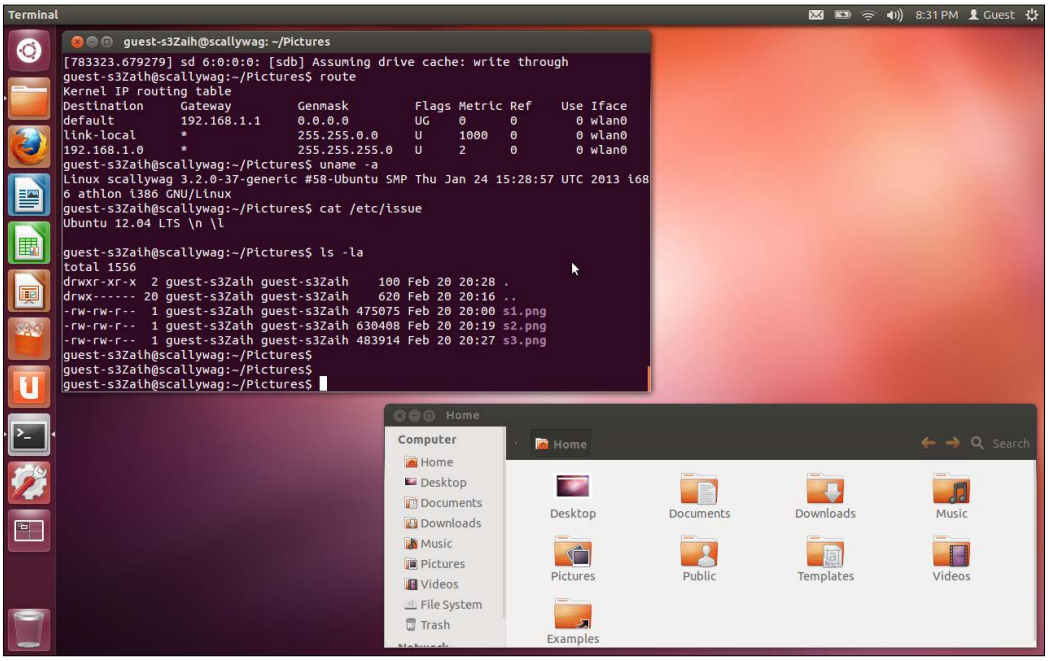
Unity

Unity is a shell interface for the GNOME environment used primarily on Ubuntu systems. It was designed to work well on small screens, for example, it employs a vertical application switcher. Unlike the other desktops/managers, it is not itself a collection of executables but uses existing applications.

Getting ready

If you have a machine running the Unity desktop, you can follow along with this section.

The following is a screenshot of Unity running on Ubuntu 12.04:



On the desktop is a GNOME terminal session and the **Home** folder. Starting with the vertical panel on the left is the Dash Home icon. It allows the user to find things quickly. Under that is the **Home** folder (already opened) and then the Firefox web browser. The next three are Libre Office Writer, Calculator, and Impress. Next is the Ubuntu Software Center, which is used to search for and purchase applications. The next icons are for Ubuntu One, a Terminal, System Settings, the Workspace Switcher, and the Trash folder.

To complete the discussion of the top panel, on the far right is the icon for Evolution. The next is the Battery status icon, network status (both wired and wireless), and the volume control. The remaining icons are the time, a switch user accounts icon, and the log out button.

Interesting enough, the terminal was not available by default on this guest desktop.

How to do it...

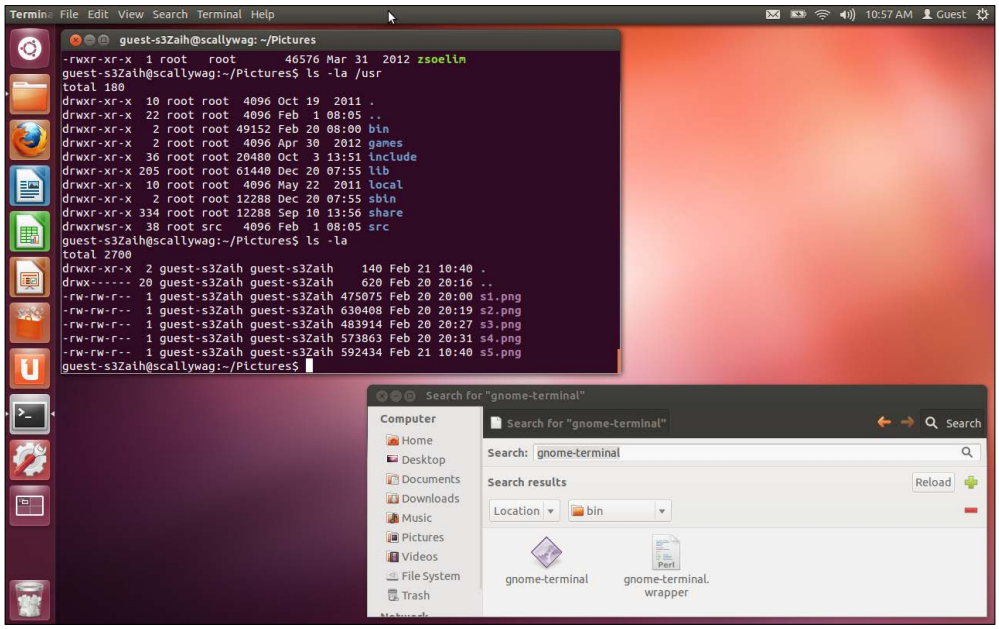
Let's add a terminal to this desktop:

1. Open the **Home** folder and then click on **File System**.
2. Double-click on the **usr** folder and then the **bin** folder.
3. Click on **Search** to open that dialog box.
4. Type in `gnome-terminal` and press *Enter*.
5. Double-click on the **gnome-terminal** icon.
6. It will open up on the screen, and you also see it as an icon along the left side panel.
7. Right-click on this icon and select **Lock to Launcher**. You now have a terminal icon.

The top panel on Unity works a little differently from the other desktops. Try the given steps:

1. Open the **Home** folder.
2. Open a terminal if you haven't already done so.
3. Now, click somewhere on the **Home** folder. The text **Home Folder** will be shown on the panel.

4. Now click on the **Terminal**. The text **Terminal** now appears. The menu items listed on the panel always correspond to the window or app that has the focus.



There's more...

I found Unity to be very different from the other desktops. At first it was a bit difficult, but like everything else it gets better with time. I believe this desktop would be popular with users who do not have much experience with Linux/UNIX systems.

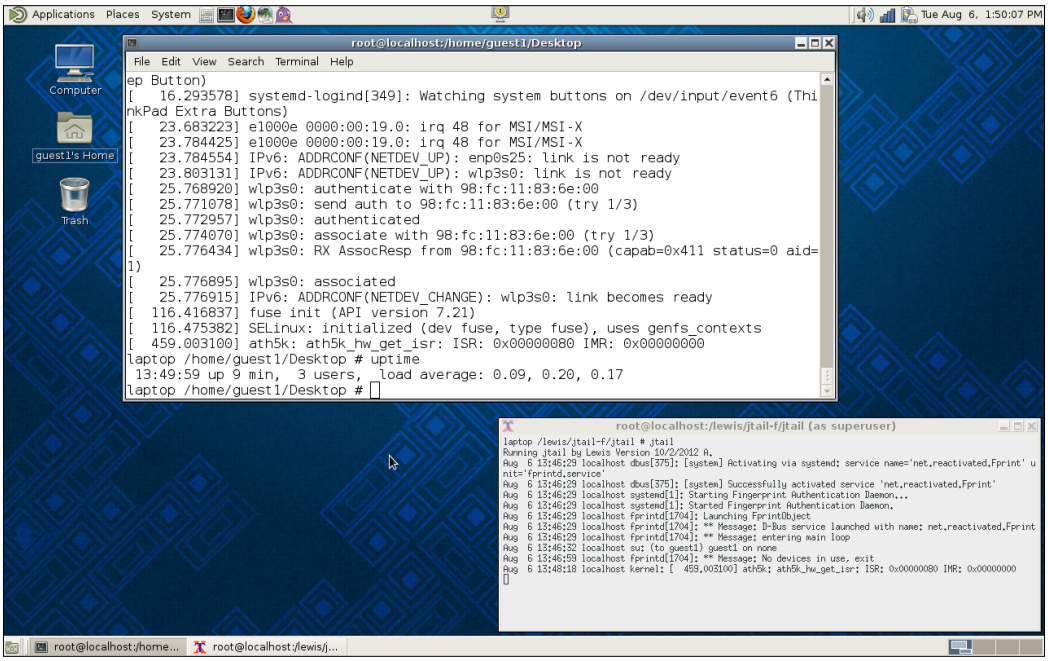
Mate

The **Mate** desktop was created to give users a more productive environment similar to GNOME 2. I am currently running Fedora 19 on my laptop using Mate and it is running fine. Note that I downloaded the F19 installation DVD and chose Mate during the install process.

Getting started

You can use a Live Image or a full install DVD from the Fedora site to follow along with these steps, whichever you prefer.

The following is a screenshot of Mate on Fedora 19:



You can see I already have two terminals open. On the top left is the **Applications** pulldown, which allows you to browse and run installed applications. The next one is **Places**, which allows you to access documents, folders, and network places. Next is **System**, where you can change the desktop appearance and behavior, get help, or log out. The icons are Caja, a file manager, and then a terminal. Yes, the Mate people were smart enough to include one by default. The next icons are Firefox, a mail app, and a messenger app. I added the lock-screen **icon**, which is in the middle. On the right is the volume, then the Wi-Fi bars, the battery status, and the date (which I customized a bit).

On the bottom left is an icon that allows you to hide all windows and show the desktop. And finally, on the far right are four workspaces.

How to do it...

Let's change a few things on this desktop:

1. First let's add the **Lock Screen** app. Right-click in the middle of the top panel.
2. Click on **Add to Panel...**.
3. Click on **Lock Screen** and follow the instructions. Close the dialog.
4. Now let's work with the time and date. Left-click on it and you will notice a calendar is displayed.
5. Left-click on the time and date again to close the calendar and then right-click on it. Click on the **Preferences** tab.
6. The **Clock Preferences** window should be displayed. Here you can change how the time and date are shown. I clicked on **Show seconds** because I like seeing the full time.
7. Close the dialog.

There's more...

As you can see, **Mate** works very much like **GNOME 2**. It is very intuitive and easy to use. The designers did a fine job creating this desktop.

3
Files and Directories

In this chapter we will cover:

- Copying, removing, and updating files and directories
- Finding files using find and locate
- Creating text files – vim, Emacs, and others
- Using the file command
- Using grep to find patterns
- Compressing files using ZIP and TAR
- Other helpful commands such as stat, sum, touch, and more

Introduction

You can think of everything in a Linux filesystem as a stream of bytes. This is simply called a file. A directory is also a file that contains other files. Most of the files are located on your computer's hard disk. However, some are in memory, for example, `/proc` and `/sys` are actually virtual filesystems. Files can also be stored on removable media too such as USB devices, CD/DVDs, and on other machines (that is, NFS mounts).

Understanding inodes and the superblock

Every file under a filesystem has a special number called an **inode**. The inode is where the OS stores the properties of the file and contains the following information:

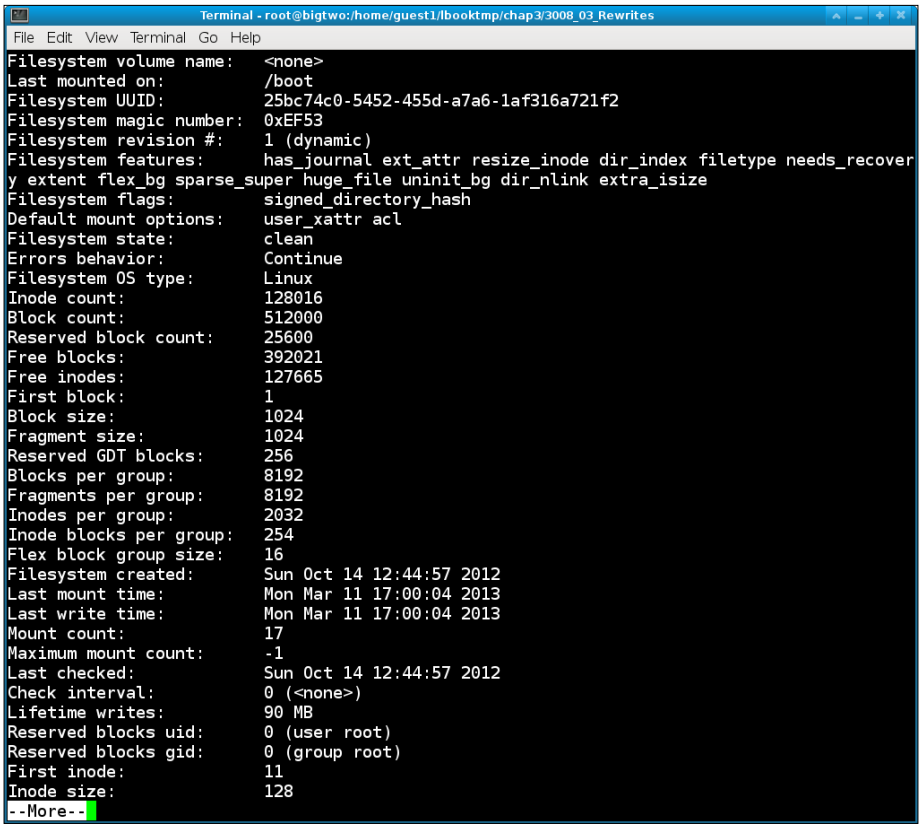
- The file type such as regular, directory, special, link, socket, pipe, or block device
- The owner and group information

- The permissions of the file (more on this in *Chapter 5*, *Permissions, Access, and Security*)
- Date and time on which the file was created and when last changed or read
- The file size
- The inode contains some other information as well

Things that are not available in the inode are the full path and name of the file itself. This is stored in the `/proc` filesystem under the **PID** (**process ID**) of the process that owns the file.

The **superblock** is what ties all of the inodes together on a filesystem. It contains all of the information needed to manage the files. Being very important to the system, most Linux filesystems have a backup copy of the superblock at regular intervals, while also being retained in memory.

The `dumpe2fs` command can be used to show the contents of the superblock. The following is a screenshot of `dumpe2fs` taken on Fedora 17:



This shows the superblock information for the first partition on this hard drive.

Copying, removing, and updating files and directories

In this section we will briefly explore how to copy, remove, and update files.

Getting ready

Several books have been written about filesystem management, and so this will serve as just a brief overview. If you have a Linux machine available you can try out these commands. We will be doing everything in the `/tmp` filesystem, so you don't have to worry about messing anything up on your system.

How to do it...

The following is the method to create some files and directories:

1. Go to the `/tmp` directory:

   ```
   cd /tmp
   ```

2. Make a test directory:

   ```
   mkdir lbooktest3
   ```

3. Go to the directory:

   ```
   cd lbooktest3
   ```

4. Let's create some files:

   ```
   ls > f1; ls > f2; ls > f3
   ```

5. Also, create some directories:

   ```
   mkdir dir1 dir2 dir3
   ```

6. The syntax for copy is `cp source-file destination-file`, so now run `cp f1 f5`. This will create file `f5` which is a copy of file `f1`.

7. You can copy to a directory:

   ```
   cp f1 dir1
   ```

8. The above is a relative path, because it starts from the current directory. To use an absolute path do this:

   ```
   cp f1 /tmp/lbooktest3/dir1
   ```

9. Now let's remove file `f3` from the current directory:

   ```
   rm f3
   ```

10. Let's move file `f2` to `dir2`:

    `mv f2 dir2`

11. The `mv` command is also used to rename a file. To rename `f1` to `f6`:

    `mv f1 f6`

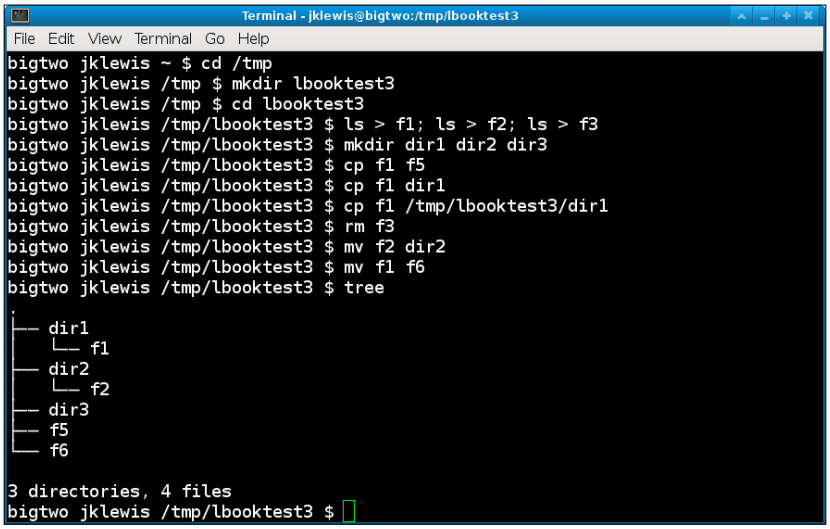
12. To see a text mode graphical representation of this directory run the following command:

    `tree`

13. It is optional to clean up everything we just did, `cd /tmp` and then run:

    `rm -r lbooktest3`

The following is a screenshot of the above commands:



There's more...

A file can be copied from another location on your computer to the current directory. This is performed using the dot operator:

```
cp /tmp/somefile
```

You don't have to `cd` to a directory first. You can do something like the following command:

```
cp /bin/bash /tmp
```

This is very handy, especially when configuring a machine.

So what happens if you copy a file to a file that already exists? Assuming you have the proper permissions, the file being copied overwrites the other file. Also, be careful when using `rm`, as it is very difficult to recover a deleted file on a Linux system.

Finding files using find and locate

The `find` command is normally used to search for files starting from the current directory. The `locate` command uses the `updatedb` database to find files or directories on the entire system (with some exceptions).

How to do it...

Let's use `find` and `locate` to look for some common Linux files:

1. First change the directory to `/usr`:

   ```
   cd /usr
   ```

2. Run the following command:

   ```
   find -name bash
   ```

3. Now try it with a wildcard:

   ```
   find -name bash*
   ```

4. It will also find directories:

   ```
   find -name bin
   ```

Now suppose we want to look for a file, but don't really know where it might be on the system. The `find` command is also slow at times, because it has to search the filesystem from the current point. Here's where `locate` comes in real handy.

1. You can be in any directory for this example. Run the following command:

   ```
   locate gnome-terminal
   ```

2. Now try the command:

   ```
   locate vim
   ```

3. See how fast this is? Now try:

   ```
   locate ifconfig
   ```

4. To ignore case do:

   ```
   locate -i sudo
   ```

There's more...

The `find` command has over 100 parameters; consult the man page for more information.

The `locate` command uses a database(s) to store the location of files. This database is usually automatically recreated by a cron job every night. If you want to refresh the database immediately run the `updatedb` command. Note that it may take a while on a large filesystem and/or slow computer.

Creating text files – vim, Emacs, and others

Most users are probably familiar with a GUI-based word processing program. For example, I am using LibreOffice Writer to compose this book. However, you can edit files using the command line as well. Some of us even prefer it.

Getting ready

It is assumed the reader has access to a Linux machine with a selection of text editors available. We will start with **vim**, a text editor that is available on every Linux/Unix system. If **Emacs** is not on your system, try installing it with `yum install emacs` or `apt-get install emacs`.

How to do it...

The following is the method to create and edit a text file using the `vi` command:

1. Let's go to the `tmp` directory. Run the following command:

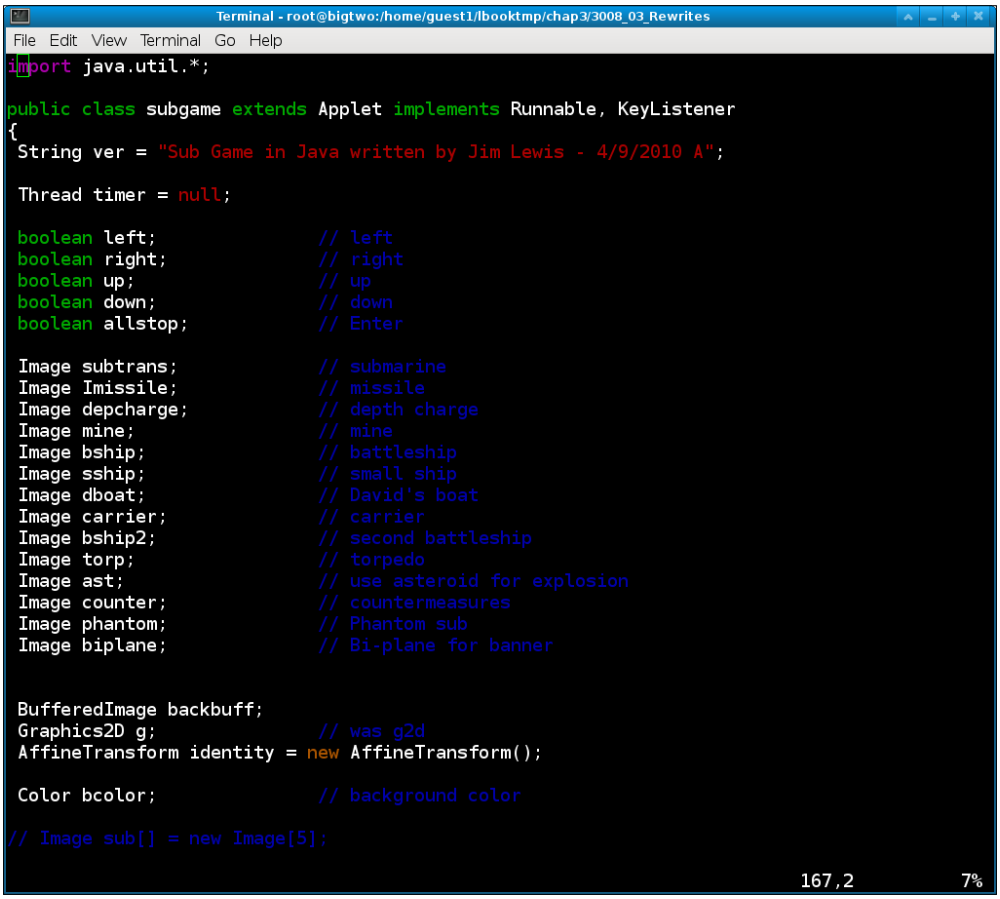
   ```
   cd /tmp
   ```

2. Now run:

   ```
   vim lbookfile1.txt
   ```

3. Your terminal should have cleared with vim running in it. Vim is modeless, and so always has to be told what mode to be in. Press the *A* key.
4. Vim should now be in Insert mode. You may see something like `-- INSERT --` at the bottom of the screen. Now type some characters.
5. The normal cursor control keys should be functional. When done editing, press the *Esc* key to get out of Insert mode.
6. To enter command mode, press the colon key, and then any letter. To save the file for example type `:w`.
7. To save the file and then quit vim, press *Esc* and then type `:wq`.

The following is a screenshot of vim editing a Java file:



Now let's take a look at the Emacs editor:

1. Change to the `/tmp` directory:

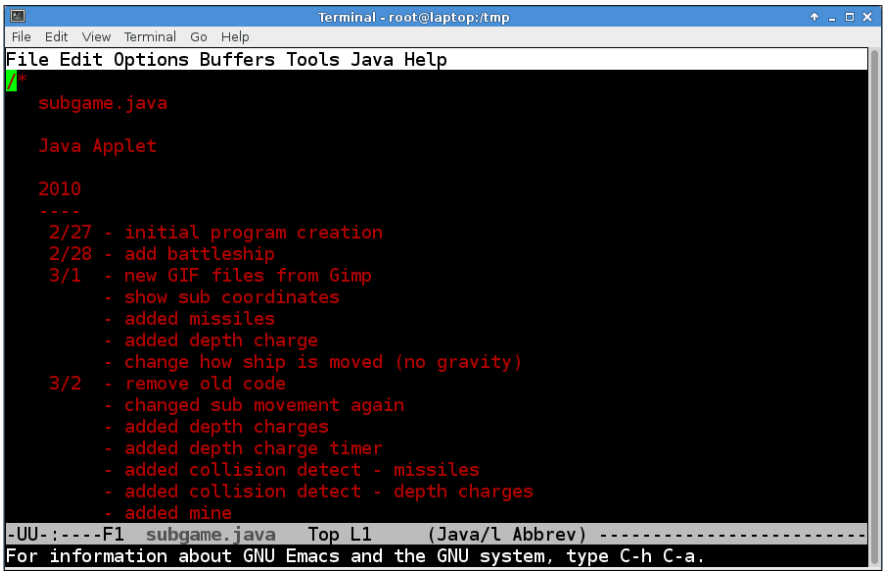
   `cd /tmp`

2. To start Emacs in text mode run:

   `emacs -nw lbooktest3.txt`

3. It will start up in edit mode. Type a few lines
4. To save the file, press *Ctrl* + *C* and then press *Ctrl* + *S*.
5. The standard cursor keys should get you around the screen; try it.
6. Press *Ctrl+H+?* to bring up the **Help** window.
7. To go back to your original screen, press *Ctrl+X+1*.
8. To close the Emacs session, press *Ctrl* + *X* and then press *Ctrl* + *C*.

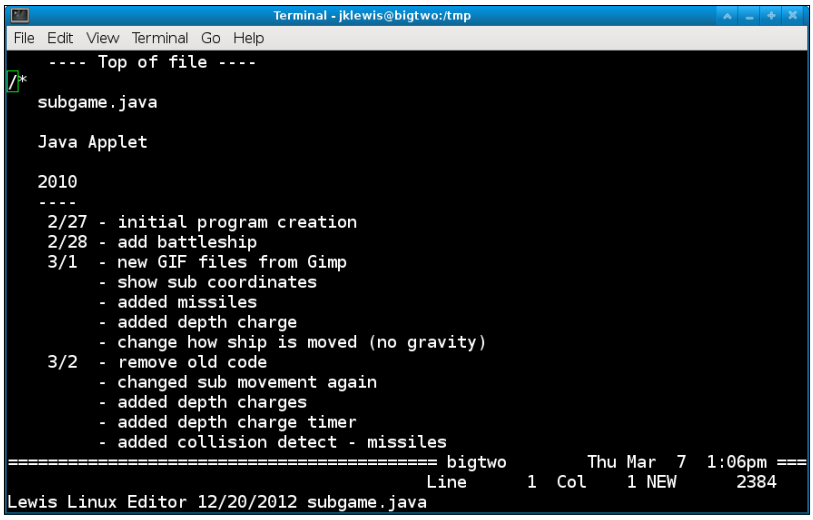
The following is a screenshot of Emacs editing the Java file:



There's more...

Vim and Emacs are very powerful editors. They are most often used by programmers writing code and sys admins maintaining shell scripts. To get more information consult the man and/or info pages, or their websites.

And for a bonus, here's a picture of my personally-written text editor:



This was written in C language under DOS about 20 years ago. I have now compiled and run it on over 20 different platforms.

Using the file command

We talked about text files above. In general, humans can read and edit text files rather easily. A binary file is different, it is (loosely) what the computer "reads". For example, when you run a command such as `vim filename1.txt` vim is a binary file and `filename1.txt` is a text file.

How to do it...

The following is an example of running the Linux `file` command:

1. Change the directory to `/tmp` as usual:

   ```
   cd /tmp
   ```

2. Let's create a text file:

   ```
   ls > temp1.txt
   ```

3. What kind of file is it? Run the following command:

   ```
   file temp1.txt
   ```

4. As you can see, the `file` command can tell us what kind of file something is.
5. Now run the following command:

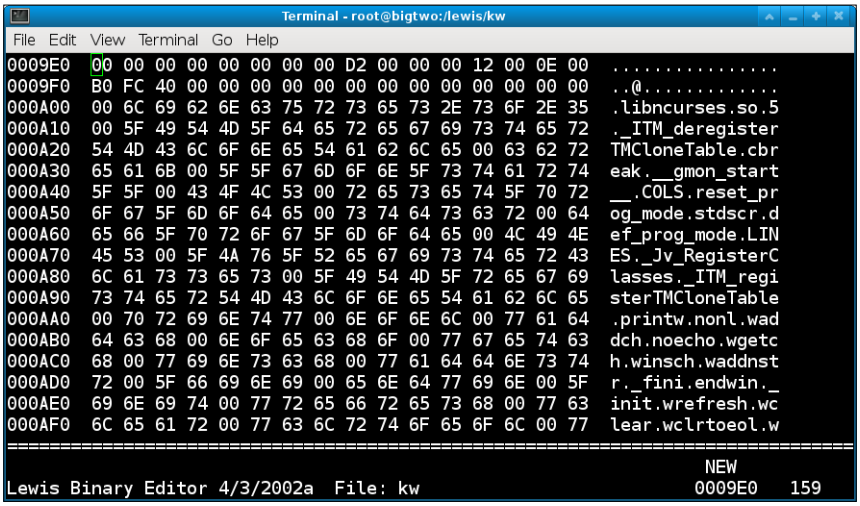
   ```
   file /bin/bash
   ```

6. All of that information means it's a binary file. It also shows what platform bash was written for, and some other info.
7. Try running file on some different files on your system to get an idea of what there is out there.

You must have the proper permissions to run the `file` command as it must perform an open operation on the file.

There's more...

We mentioned editing text files. Binary files can be edited as well, if you really know what you are doing. This is sometimes needed during low-level work, for example, when working on device drivers. A binary editor may look something like the following screenshot:



Oh no, I have edited a file but now can't save it!

From time to time, you may get into a text editing situation that seems hopeless. You have spent some time changing a file but now can't save it. The error is usually something like "Permission denied". There are usually two things that can cause this; you can't write into the directory where the file is located, or the file already exists but not with the proper writeable permissions. The best way to save your work, if the program allows it, is to simply write the file somewhere else, for example, in `/tmp`.

This is a rather simplistic example, but it gets the idea across. I wish I had already known this procedure the first time it happened to me.

How to do it...

The following is the method to save a file when you get a permission error:

1. As a normal user (not root) go to the `/usr` directory:

   ```
   cd /usr
   ```

2. Now run the following command:

   ```
   vim lbookfake1.txt
   ```

3. It should open up an empty file as normal. Press *A* and then then enter some text.
4. Now press *Esc*, and type `:w`.
5. It should show a write error, so save the file to `/tmp`:

   ```
   :w /tmp/lbookfake1.txt
   ```

6. That should succeed. You can then take steps to correct the real problem (that is, don't edit files where you are not supposed to!).

Using grep to find patterns

When dealing with files, it is convenient to be able to search for patterns within the text. This is often used in code development for example. Here we show how to use **grep**.

Getting ready

We will use the `dmesg` program, which shows information about the running kernel, for this example. If it is not available, or if your computer has been running for a long time, the following may not match up quite right on your system.

How to do it...

The following is an example on using `grep`:

1. Run the following command:

   ```
   cd /tmp
   ```

2. Use `dmesg` to create a file, so we can search for some information about your system:

   ```
   dmesg > dmesg1.txt
   ```

3. Let's see if we can determine what network device is being used. Run:

   ```
   grep network dmesg1.txt
   ```

4. The output might not be very informative. But what if case is an issue? Try the following command:

   ```
   grep -i network dmesg1.txt
   ```

5. The `-i` tells `grep` to ignore case. You should now see which network driver your system is using. Now try the following command:

   ```
   grep -i cdrom dmesg1.txt
   ```

6. `grep` returns a code based on the results of the search. Run the above command again (remember the up arrow shortcut?):

   ```
   grep -i cdrom dmesg1.txt
   ```

7. Now run the following command:

```
echo $?
```

8. Assuming the text was found, the return code will be `0`. Now search for a string that should not be there:

```
grep -i jimlewis dmesg1.txt
```

9. Run the following command:

```
echo $?
```

10. The return code should not be `0`.

This might seem a little backwards, that zero means success and non-zero means failure. This is common to many Linux commands, because in many cases the number returned is an error code.

There's more...

The `grep` program can search in an incredible amount of different ways. You can specify very complicated patterns as well. Consult the man page for more information. Also, we will visit the return code from `grep` again in *Chapter 8*, *Working with Scripts*.

Compressing files using ZIP and TAR

It's no secret that a Linux system has a *lot* of files. A typical code development project might have over 1000 files, spread across several directories. And how do we back all of that stuff up?

The answer is file packaging and compression. Here we will show two favorites, **ZIP** and **TAR**.

Getting ready

Most Linux systems have both ZIP and TAR, and so it will be assumed they are already available on your system.

How to do it...

Here we will experiment with the `zip` and `unzip` command::

1. Run the following command:

   ```
   cd /tmp
   ```

2. Let's make a temporary directory:

   ```
   mkdir lbooktemp
   ```

3. Run the following command:

   ```
   cd lbooktemp
   ```

4. Now let's create some files:

   ```
   ls > f1.txt; route > f2.txt; dmesg > f3.txt
   ```

5. Then, create some more files:

   ```
   ifconfig > ifconfig.dat; dmesg > dmesg.dat
   ```

6. Let's package and compress the first ones into a single file:

   ```
   zip lbook1.zip f1.txt f2.txt f3.txt
   ```

7. As you can see, the syntax for ZIP is "zip zipped-file files-to-zip". We could have also used wildcards above:

   ```
   zip lbook1.zip *.txt
   ```

8. Now let's include the others:

   ```
   zip lbook1.zip *.txt *.dat
   ```

9. The unzip program is used to extract files out of a zipped file (also called an archive). Make another directory using the command:

   ```
   mkdir test
   ```

10. Run the following command:

    ```
    cp lbook1.zip test
    ```

11. Run the following command:

    ```
    cd test
    ```

12. Now unzip the file:

    ```
    unzip lbook1.zip
    ```

13. Perform an `ls -la` command. You should see the files as before.

14. You can also view the contents of a ZIP file, without extracting anything, by running the following command:

    ```
    unzip -l zipped-file
    ```

I use ZIP when there are just a few files, or when I am sending them to someone running a non-Linux OS. While ZIP can be used to span directories, it doesn't normally deal well with some Linux files. Tar is a better alternative:

1. We can use the same files as before: cd /tmp/lbooktemp
2. Run `tar cvzf lbook1.tar.gz *.txt`. This will create a gzip compressed archive file.
3. Now run `file lbook1.tar.gz`. It should show something like the following output:

   ```
   lbook1.tar.gz: POSIX tar archive (GNU)
   ```

4. To extract, first copy it to the test directory:

   ```
   cp lbook1.tar.gz test
   ```

5. Run the following command:

   ```
   cd test
   ```

6. Run the following command:

   ```
   tar xvzf lbook1.tar.gz
   ```

7. Perform an `ls -la` command. You should see the files again.
8. To view a tar archive, use the `t` (for tell) option:

   ```
   tar tvzf lbook1.tar.gz
   ```

9. Now let's TAR the the whole directory:

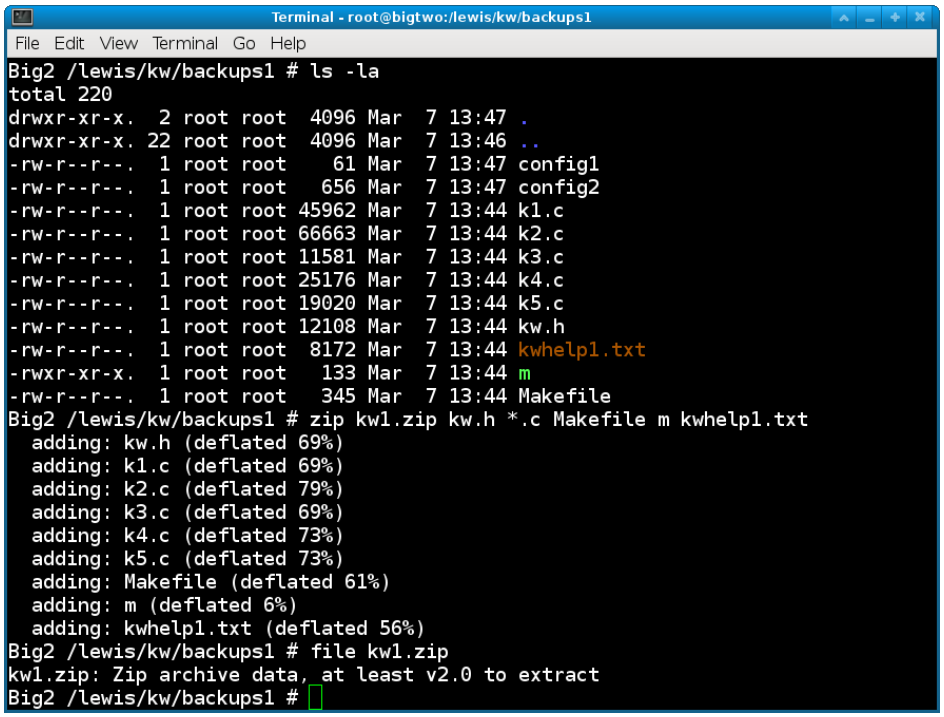
   ```
   cd /tmp
   ```

10. Run the following command:

    ```
    tar cvzf lbooktemp1.tar.gz lbooktemp
    ```

11. This will get the entire directory, even the hidden files if any exist.

The following is a screenshot of the `zip` command:



There's more...

ZIP and TAR are used quite frequently in system administration to back up everything. It should be noted that "tarring up" a system, copying that file to another machine, and "untarring" the file is a great way to clone a box (I use this all the time).

Later, when we discuss crontab in *Chapter 9*, *Automating Tasks Using Cron*, I will show how I use TAR to backup my system every night.

Other helpful commands such as stat, sum, touch, and more

There are many more commands available in Linux that deal with files. In this section I show how to use a few of them.

How to do it...

The following are some of the commands I find myself using all the time:

1. Run the following command:

   ```
   cd /tmp
   ```

2. Create a file:

   ```
   dmesg > file1.txt
   ```

3. Now run `ls -la` and remember the info. We will use this later.
4. Use the `stat` command to see practically everything you would ever want to know about the file:

   ```
   stat file1.txt
   ```

5. Now suppose you have sent that file to someone that is running a Linux system, and want to ensure it did not get corrupted along the way. Run the following command:

   ```
   sum file1.txt
   ```

6. The first number is the checksum and the second is the number of blocks for that file. If the other person runs `sum` on his copy of the file and sees the same info, the files are the same.

   The file names do not have to match.

7. We have created a lot of files by using the redirection operator. You can also use the touch command:

   ```
   touch file2.txt
   ```

8. Since `file2.txt` did not already exist, touch will create it as an empty file. In fact, let's prove that:

   ```
   file file2.txt
   ```

9. So what happens if we run touch on an existing file? Does it empty it? No, it updates the time and date on it. Run the following command:

   ```
   ls -la file1.txt
   ```

10. Now run the following command:

    ```
    touch file1.txt
    ```

11. Run `ls -la` again. You should notice it now shows the current date and time on that file.
12. Suppose you want to just view a text file. Run the following command:

    ```
    less file1.txt
    ```

13. When using the `less` command press the Space bar to scroll down. Press *Q* to exit.
14. Say we want to see just the first few lines in that file:

    ```
    head file1.txt
    ```

15. The `head` command shows the first 10 lines by default. How about the last 10 lines? Run the following command:

    ```
    tail file1.txt
    ```

There's more...

As I have said before, there are many, many more commands that deal with files. And, of the commands I have mentioned, I have only scratched the surface of what they can do.
As always, consult the man pages for more information.

4
Networking and the Internet

In this chapter we will cover:

- Troubleshooting bad connections
- Copying files to another machine–FTP and SCP
- Logging into another machine – Telnet and Secure Shell
- Getting a web page without a browser – wget
- Browsing the web – Firefox
- E-mail – using a web mail program
- Running your own web server – httpd
- What is using that port? The /etc/services file
- IPv4 versus IPv6

Introduction

Being "on the net" is crucial in today's world. Here we explain connectivity and what to do when it doesn't work. But first we will explain the pros and cons of a wired versus a wireless connection.

A typical wired Ethernet connection is fast and reliable. It does not normally suffer from dropouts or lost packets. A good quality wire such as Cat 5E or Cat 6 can run for several feet without any loss of signal.

Wireless connectivity gives you the freedom to, well, not have a wire for one thing. When properly configured it can be effortless to use and be very reliable.

Wired connection has the following pros:

- Wired connection is typically faster than wireless
- In general, it is more reliable
- It doesn't suffer from periodic dropouts or lost packets
- Is easier to configure and troubleshoot

Wired connection has the following cons:

- We need to use wires; it can be very difficult to rearrange or change, especially in a computer lab environment
- We need someone to design and physically connect the network
- It can be confusing when using large servers with multiple Ethernet ports

Wireless connection has the following pros:

- It is convenient; we don't have to deal with wires
- In some cases, it can go pretty far from the base router
- It is easily set up for guest access

Wireless connection has the following cons:

- It is not always reliable. Dropped packets happen much more frequently than with a wired connection.
- It can be hard to set up, especially for the first time.
- We have to deal with passwords and encryption to prevent unwanted access.

Troubleshooting bad connections

This seems to happen occasionally, and at the worst possible time. These are the steps I normally take to diagnose and solve the problem on a wired connection. It is assumed that the connection worked properly at some time in the past.

Getting ready

There is no special setup for this example, unless you happen to have a broken machine somewhere. You can run most of these commands without fear of hurting a good system.

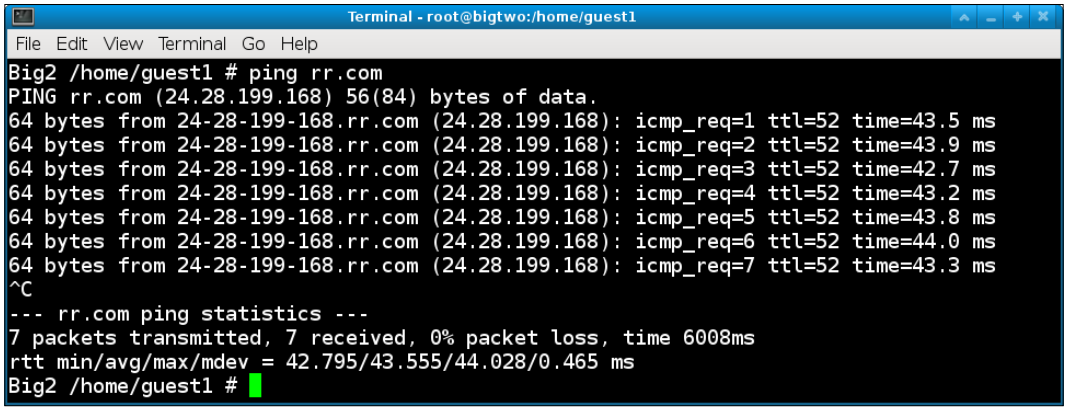
How to do it...

Try these steps when diagnosing a network issue:

1. First let's make sure the connection is really down by pinging a known external address. I use Road Runner as my ISP, so for me the command would be:

   ```
   ping rr.com
   ```

2. Something like this should appear:



3. Press *Ctrl* + *C* to stop the output. If the command line says timed out, or says something about no host or bad route, the connection is indeed down. Let's try to fix it.
4. Find your gateway address. Run the `route` command; your gateway should show up at the top of the output. Note, the `route` command taking a long time to complete is also a sign the connection is down.
5. Ping the gateway provided by route above. If it pings successfully, then the problem is most likely on the router, or at the ISP itself.
6. If it doesn't get a good ping, let's try this. If you already know the interface name, skip to the next step. If not, find it by using the `ifconfig` command. Look for a stanza that has something like `eth0` or `p3p2`.
7. Run the `ethtool` command on the interface. In my case it is `ethtool eth0`. Look at the last line; it should say:

   ```
   Link detected:   yes
   ```

8. If it does not, the problem may be the wire. If your port has LEDs see if they are on. One of them should blink from time to time. Try wiggling the wire. Of course, if you have a known good wire, try replacing it. Note, a "known good wire" does *not* necessarily mean one right out of the package. Got bit by that once.

9. If the wire seems okay, let's look deeper. Make sure, in the steps above, you have the right interface. Note at this point you may want to just read these steps instead of running the commands, as they will take your network down.
10. Try running the `ifdown <interface>` command. You probably will not see any output.
11. Now run the `ifup <interface>` command. It may take a few seconds, and then you should see some output. If you are using DHCP, you should see the connection being made. When you get the prompt back, try the ping again.
12. I'll assume if you're still reading, the `ping` failed and the connection is still down. It is possible your router and/or modem has become jammed up. Try turning those off, wait about a minute, and then turn them back on.
13. After waiting for the network to settle in try the `ping` again. If it is still down, continue with the next steps.
14. Even though I despise this as a solution, at this point, I would suggest a cold shutdown. Close down all except one terminal and run the following command:

    ```
    shutdown -h now
    ```

15. I like to wait a few minutes before rebooting, just to make sure the memory is actually clear. Okay, so turn it back on and wait for it to boot all the way up.
16. Try the `ping` command again. If it now works, I would suspect some kind of transient error or problem. If it continues to work, well that's great. However, if it fails again I would suspect the hardware might be going bad.
17. If the `ping` command stills fails after a reboot, I would try booting up a Live Media image. If this works and gets on the net, something is out of whack in the files on your base system. If it does not work, I would suspect the hardware again.
18. One last thing; make sure no one has made any changes to your DHCP server. A misconfigured server can cause all kinds of bad things to happen, such as not getting a connection.

There's more...

I hope this section has been useful. These steps have worked for me countless times in the past. One more thing I would suggest is to look at your `/var/log/messages` file (and/or `dmesg`). It may shed some light on why the connection is failing.

Working with IP addresses, subnets, domains, and so on

This will serve as a very brief overview of what is meant by an IP address, subnet, and domain. Understanding this information will help you with future sections of this book.

An IP address, in general, is what distinguishes one computer from another on a network. It is a set of numbers such as `66.69.172.30`. Within a home network, since **NAT** (**Network Address Translation**) is most likely used, you may be more familiar with the local addresses, such as `192.168.1.115`.

The following is a quick list of the terminology:

- **IP address**: The Internet Protocol address of this machine, for example, `192.168.1.115`.
- **Subnet**: The third octet (or set of numbers) is the subnet. For example, `192.168.1.115` and `192.168.1.120` are on the same subnet. However, `192.168.2.115` is not.
- **Domain**: A domain is usually referred to by a hostname, such as `rr.com`. It too has a numeric IP. Use the `ping` command to determine what IP a hostname is using.

Copying files to another machine – FTP and SCP

Not counting email, the two most common ways to copy files to another machine are **FTP** (**File Transfer Protocol**)and **SCP** (**Secure copy**). Here we will discuss both.

FTP has been around for many years and is still used quite a lot today. However, standard FTP does have a serious drawback. The data is sent in what is called "clear text". This means a knowledgeable person could obtain the data under the right conditions. We will talk about this a bit later.

Getting ready

We will assume an FTP server has already been set up and is available for this exercise. Normally you would use `ftp` command from one machine (the client) to another (the server). However, for this example I am using just one machine. The command to start an FTP session is `ftp server-name` where server-name can be a host name known on your network, or a numeric IP address. For this example, I have created a file in `/tmp` named `f1.txt` and a file in `/home/jklewis` named `f5.txt`.

Most Linux distributions are already set up to allow the `scp` command, which is very convenient. We will assume your distro allows this. Note that some do not allow root access so we will use a guest account.

The syntax for SCP comprises of the given commands.

To copy to a machine:

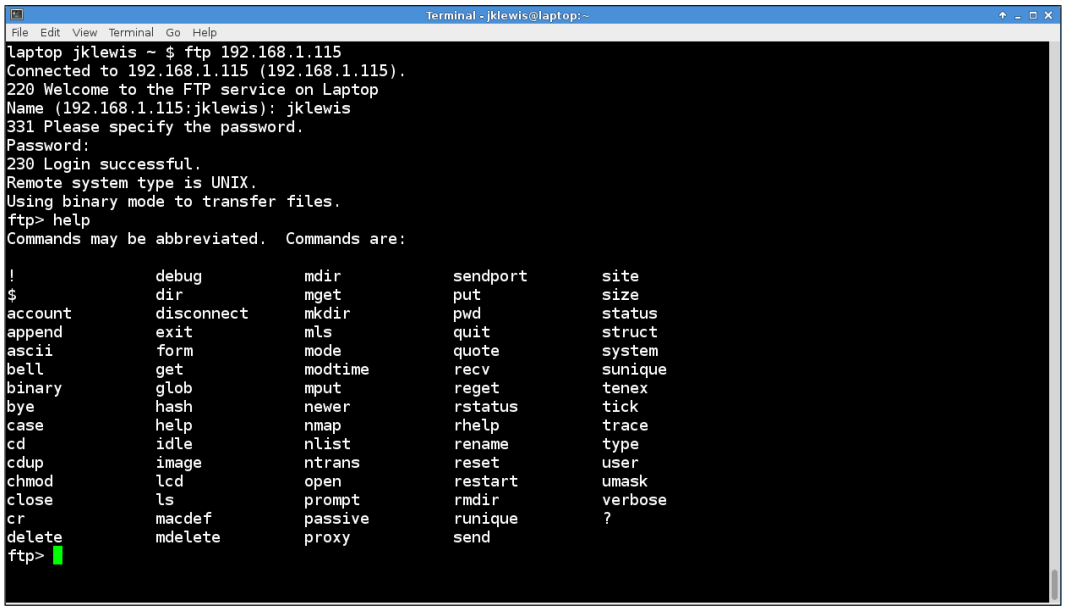
```
scp local-filename username@hostname:/directory
```

To copy from a machine:

```
scp username@hostname:/directory/filename
```

(Don't forget the colon above.)

The following is a screenshot of the FTP help screen on Fedora 18:



How to do it...

Here is how to use the `ftp` command:

1. In a terminal session on the client, run the following command:

   ```
   ftp 192.168.1.115
   ```

2. It should present you with a prompt. Type in the username and press *Enter*.
3. Type in the password and press *Enter*.
4. It should show something like **Login successful**. If it doesn't, then you are not logged in, although there will still be an FTP prompt. Assuming it logged in run the following command:

   ```
   ls -la
   ```

5. You should see a listing of files. Run `help` to show a list of commands. They might not all be available on your system (which I consider a bug).

6. One thing to remember when using `ftp` is to know where you are at all times. It is very easy to get confused. Use `pwd` often. Run it now:

   ```
   pwd
   ```

7. The commands you type are on the server. To run a command locally, prefix it with an exclamation point. Try these commands:

   ```
   !pwd, !ls
   ```

8. So, let's actually do something. Change your local directory to `/tmp` by running :

   ```
   lcd /tmp
   ```

9. Run the `!ls` command. Now let's copy file `f1.txt` to the server:

   ```
   put f1.txt
   ```

10. Now let's retrieve file `f5.txt` from the server:

    ```
    get f5.txt
    ```

11. This may appear a bit confusing at first. That's because it is. However, if you use it a lot it will become much easier. Remember, you put it to the server, and get it from the server.
12. When you are finished with your FTP session run the following command:

    ```
    quit
    ```

13. Now let's run some `scp` commands, to begin:

    ```
    cd /tmp
    ```

14. Create a test file by running:

    ```
    ls > f1.txt
    ```

15. Run the following command:

    ```
    scp f1.txt guest1@192.168.1.115:/temp
    ```

16. Enter the password for user `guest`.
17. Assuming file `f5.txt` is on the remote `/temp`, to get it run:

    ```
    scp guest1@192.168.1.115:/temp/f5.txt
    ```

18. Don't forget the colon above. Run `ls -la` to see how it went.

There's more...

I mentioned that FTP sends its data in clear text. The Secure Shell commands (more on that in the next section) use encryption and so security is not a problem. Also, it is possible to set up a machine so that the password is not required. For more on this, see the `ssh-keygen` command. This topic is also covered in *Appendix A*, *Linux Best Practices.* To configure `sshd`, for example, to change how root logins work, see the `sshd_config` file.

Logging into another machine – Telnet and Secure Shell

Telnet is an older protocol but is still used a lot today. It suffers from the same security problem as FTP; it sends text in a non-encrypted format. However, it is still useful in say a lab environment protected by a good firewall.

Getting ready

I will again use the same machine as both the client and server. For this example, we will assume both the Telnet client and server are already installed and operational.

The command to start a Telnet session is `telnet hostname`. The hostname can be a name reachable on your network or a numeric IP.

Secure Shell (**SSH)**, is a more popular protocol as it provides for strong encryption of both the password and text. It's also much easier to use, in my opinion.

To start a secure shell, the command is as follows:

```
ssh username@hostname
```

How to do it...

Here we will run a few Telnet and SSH commands:

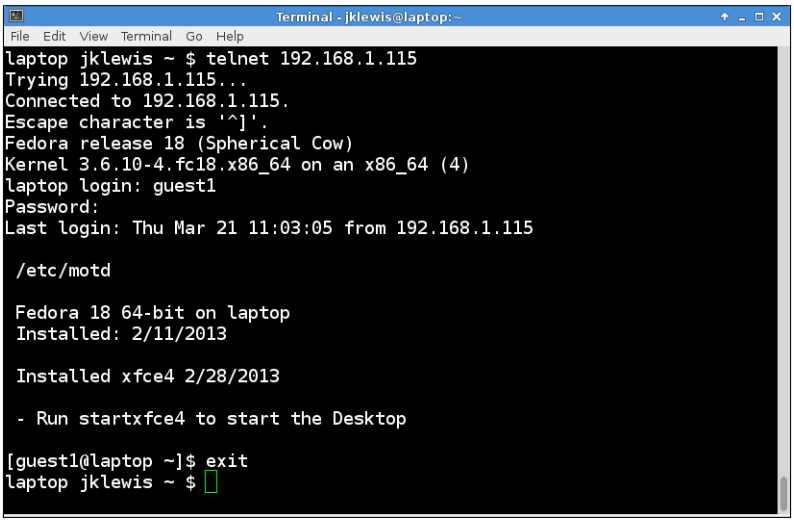
1. Run the following command:

   ```
   telnet 192.168.1.115
   ```

2. A banner and prompt should appear. Enter the username.
3. Enter the password.
4. The message of the day (`/etc/motd`) file will probably be displayed. This is a good place to put messages and other information for anyone who is logging in. In this terminal you can run pretty much any text-mode command you want. Try `pwd`.
5. You can change to another directory, edit files, and pretty much whatever you would normally do if you were actually on the system.
6. When you are done with the session, run:

   ```
   exit
   ```

This is a screenshot of Telnet running on Fedora 18:



1. Now let's try running a Secure Shell session:

   `ssh jklewis@192.168.1.115`

2. Enter your password. The `/etc/motd` file should display as before. Let's try some commands. Run the following command:

   `pwd`

3. Run the following command:

   `uptime`

4. As with Telnet, every text-based command should work just as if you were on the actual machine. When done with the session, run:

   `exit`

There's more...

Secure copy and Secure Shell are the best ways to copy files and to access remote machines. Get to know them well. I mentioned above that the `ssh-keygen` program allows one to copy files without a password. This works for SSH as well. I use this on all my home machines, and also on my website, which is hosted by a service provider. For more information on this very cool feature run `man ssh-keygen`.

One other thing, I mentioned that the Telnet and SSH sessions should work just like you were on the actual machine. This is not always true when the client is running on a non-Linux machine. For example, some of the key strokes may be mapped differently. Take a look at the documentation for the client program you are using as there may be a way to adjust the key mappings.

Getting a web page without a browser – wget

Probably everyone has received an e-mail from a questionable source. You know you shouldn't click on any of the links, but wouldn't it be nice if there was a safe way to determine what was on that site? Well, there is.

The `wget` program allows you to download files from URLs. Although it can do a lot, the simplest form of the command is: `wget <some URL>`. Assuming no errors, it will place that file in the current directory. If you do not specify a filename, by default it will attempt to get the `index.html` file.

How to do it...

The following is the method to run `wget`:

1. Run the following command:

   ```
   cd /tmp
   ```

2. Run the following command:

   ```
   wget www.jklewis.com
   ```

3. The resulting file will be named `index.html`. View it with more `index.html`. Yes, it's my personal web page.
4. You can also refer to specific files. Try:

   ```
   wget www.jklewis.com/shipfire.gif
   ```

5. Try it with some other sites. You can do this on a suspicious site to see if it is safe.
6. If you have a suspect site in mind do a `wget` on the file. View it and look for links to other sites. If you see something like `http:/DoWeCheatThemAndHow.com`. I probably wouldn't click on that.

There's more...

The `wget` command can do a lot more. With the right parameters, it can be used to clone entire websites and a lot of other neat things. Consult the man page for more information.

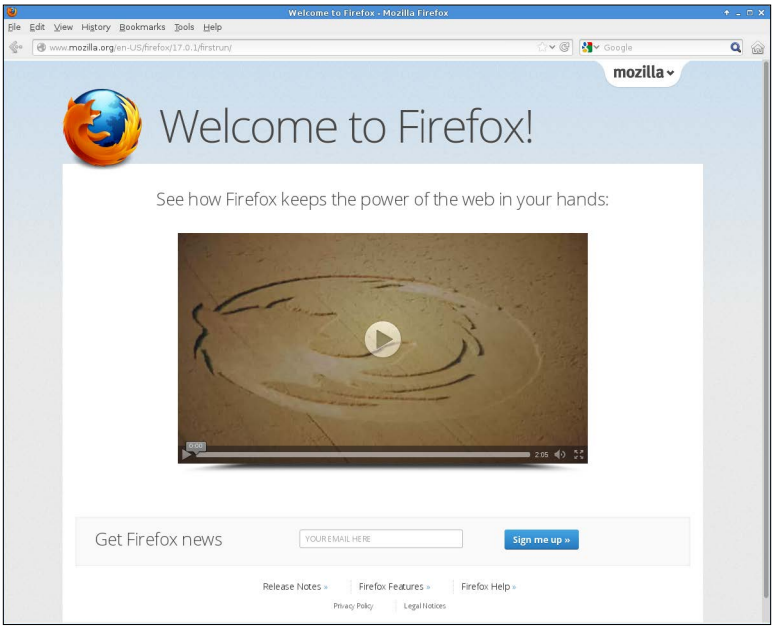
Browsing the web – Firefox

There are several different browsers to choose from for your Linux system. Here we will focus on Firefox by Mozilla.

Getting ready

Firefox is usually available by clicking on the browser icon on your desktop. You can also start it from the command line. For this example we will start Firefox for the first time with user `jklewis`.

The following screenshot shows how Firefox looked like the first time I opened it under user `jklewis`:



How to do it...

The following is the method to run Firefox from a terminal:

1. Open a terminal session for a guest user. For this example, I ran `su - jklewis` and entered my password
2. Run `firefox &` (the ampersand tells it to run the command in the background so you can still access that terminal. Error messages will also be displayed here).
3. Firefox should come up with the default settings. Let's change some of them. **Go to Edit | Preferences**.

   ```
   Click on Edit->Preferences
   ```

4. The **Firefox Preferences** screen should come up, on the **General** section. See where it shows **Home Page**, change it to something more useful. I changed mine to `http://jklewis.com`.

5. The **General** section is also where you can control the behavior of the **Downloads** window. I have chosen to leave that checked for this user.
6. Now click on the **Tabs** section. I personally despise browser tabs and so I have turned them off. To do this remove the check mark from **Open new windows in a new tab instead** and **Always show the tab bar**. Note you do not have to do that for this exercise.
7. The **Content** section allows you to control what gets displayed. Take a good look at this page, however, I usually leave these settings as is.
8. The **Applications** section shows what browser plugins are available. I don't change anything here.
9. Now click on the **Privacy** section. This is where you can clear your history and deal with cookies.
10. Now click on **Security**. I suggest leaving this as it is.
11. The **Sync** section lets you sync up with a mobile device.
12. The **Advanced** section has a few pages to it. The **General** page lets you control how the browser operates. Now click on the **Network** page. This is where you configure the cache.
13. The **Update** page deals with what gets updated automatically, and the **Encryption** page is where the protocols are set and where you can view and modify your certificates. I suggest leaving these as they are.
14. Now close out the **Preferences** dialog.
15. Now let's view a website and bookmark it. Go to the URL field and double-click on it. Press the **Delete** key. Now enter `jklewis.com`.

The "www" is not normally required.

16. You should see my web page. Now click on **Bookmarks**.
17. Click on **Bookmark This Page**. A dialog titled **Page Bookmarked** will appear (actually this is absolutely wrong as the page has *not* been bookmarked yet). Here you can change the title of the bookmark, or the folder to put it in. For now just click on **Done** to finish.
18. The page should now be bookmarked.

There's more...

As you can see, there are quite a few more options. Explore these as you need them. The following are a few more tips when browsing.

When bookmarking a page, take a good look at the title given. For some reason they don't always give a good descriptive name. "Account Login" just doesn't tell me much. However, if I change it to "Hi-Fee Bank and Trust – Login" that tells me exactly which account it is.

I highly recommend your browser be run from a guest account, and not root. Note that if you are running your desktop under a guest user (also highly recommended) starting Firefox from the icon will be fine.

On the right of the URL field is a star. Clicking on this brings up **History**. Next to that is a semi-circle with an arrow. This is the **Refresh** button. If a page ever seems to be loading too slowly, or is just not working right, try refreshing the page by clicking on this icon.

E-mail – Using a web mail program

There are many web mail clients available, such as **Evolution** and **Thunderbird**. I have used both and find them lacking in some areas. For this reason and others I use a browser web mail app instead.

Getting ready

If you already use a web mail program this will probably be old hat to you. However, if you have not used one before this should be very helpful. Note, for this section I do not suggest you try and run these commands, just read them.

How to do it...

The following is the method to access a webmail client using Firefox:

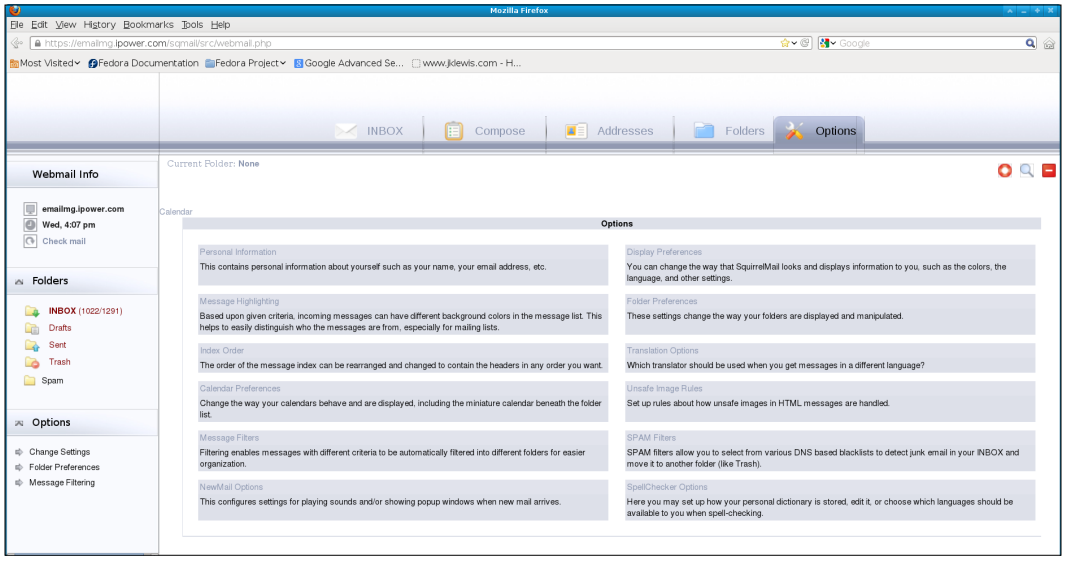
1. Log in to your account on your provider's website. Somewhere on there should be a link to to access your e-mail. Your provider may even give you a choice of which web mail app to use. I chose **Squirrel Mail**.
2. Whichever one you pick, it should open up and ask for your email account password. Remember that this might be different from the password you used to get into your provider's main account.
3. Once you are in the mail app should present you with the contents of your inbox. Here you can respond to mails, forward them, provide attachments, and so on.
4. There should also be an address book. Find it, open it, and get to know how to use it. If the provider has a way to export the address book file to your computer make frequent use of that feature so you can restore it later if it gets corrupted.
5. Keep this in mind; if you are going to compose an email and attach a file to it, attach the file first before doing anything else. Then go fill out the other fields and compose the text. Trust me, your friends, family, potential employers, and so on will all thank you for not forgetting the attachment.

6. Most web mail programs do not auto refresh the screen. So, if you leave it open (which I do all the time) and leave the computer, when you come back be sure to refresh the email session. This is usually done by clicking on the **Inbox** title. You can also click on the browser's Refresh button.
7. One last thing about web mailers; the mail will probably eventually fill up your allocated space, and you might even get some e-mails about it from your provider. What I do is wait until I have used approximately 80 percent of my space in the inbox. I then go directly to the last page (usually page 30 or so) and move these to the trash. I go through and delete these one page at a time until I am back on page 20.
8. Some people delete mail as they go. This is fine if you are good at it. I am not. If I do it that way I'll eventually delete something I wish I could get back.
9. Speaking of getting a file back, you can usually recover a deleted file by looking for it in the **Trash** folder. However, once you have purged the folder, the e-mails will be gone for good. The purge usually occurs automatically about once a week (configurable on some mailers).

There's more...

One of the nice features of a web mailer is you should be able to access your email from any browser on any machine. All of the information is stored on the provider's website.

The following is a screenshot of **Squirrel Mail** | **Options**:



Running your own web server – httpd

There may come a time when you will want to run your own web server. In my career as a software engineer I have been often asked to run the department lab. The easiest way to keep my users informed as to what is going on is to put the notes and files on my own local website that I control. This may sound hard but it really isn't.

Getting ready

This assumes you have a Linux system available to try this out on. If not, you can still get a lot out of this section. This also assumes that **httpd** is not already installed. These steps were performed on my laptop running Fedora 18.

How to do it...

The following are the steps to install your own Apache httpd server on Fedora:

1. First install httpd:

   ```
   yum install httpd
   ```

2. Now change to the configuration directory:

   ```
   cd /etc/httpd/conf
   ```

3. Make a backup copy of the file: `cp httpd.conf /tmp/httpd.conf.orig`. You may want to choose a more suitable backup directory.
4. Edit the file with a text editor:

   ```
   vi httpd.conf
   ```

5. This file contains almost everything you need to know in order to set up your httpd server. Read the first page very closely where it talks about ServerRoot.
6. In brief, if a filename begins with a slash, the server will use that path as indicated. However, if a relative path is used, the value of ServerRoot is prepended. For example, scroll down to this line:

   ```
   Include conf.modules.d/*.conf
   ```

7. Since this is a relative path and ServerRoot is set to `/etc/httpd`, the directory that will be included is really `/etc/httpd/conf.modules.d`.
8. In another terminal change to that directory now:

   ```
   cd /etc/httpd
   ```

9. Be sure to remember how ServerRoot works! It will save you a lot of time. Now let's go through the `httpd.conf` file a bit more. The `Listen` directive tells httpd which port to use. Leave it at `80` for now.

10. Scroll down to ServerName. This is where you can enter the name of your particular server. Leave it blank for now.
11. The last one we will talk about is DocumentRoot. This is the directory where your web pages will be served from. It should be set to `/var/www/html`.
12. Let's actually start the httpd service. Run `systemctl start httpd.service`. This should silently return back to the command line.
13. We have to enable it as well using the command `systemctl enable httpd.service`. This should show a successful link message.
14. Now let's try it:

    ```
    cd /var/www/html
    ```

15. Run `ls -la`. It is probably empty. Create a file by running:

    ```
    dmesg > dmesg1.txt
    ```

16. Now either go to or open a browser session. We will check things locally first, so in the URL field enter `file:///var/www/html/dmesg1.txt`.
17. The file contents should be displayed. Now let's create another file:

    ```
    echo "This is a new file" > newfile.txt
    ```

18. Substituting your IP for mine, try this in the URL field: `192.168.1.115:/newfile.txt`. The contents of `newfile.txt` should be displayed.

There's more...

If you decide to do a lot of "web serving" be sure to read and understand the `httpd.conf` file. There is also quite a lot of information available on the net.

You may find that you cannot actually access your files from another computer on your network. That is most likely due to a firewall (`iptables`) issue; however, your router may need some configuring as well. This will be covered in *Chapter 5*, *Permissions, Access, and Security*.

What is using that port? The /etc/services file

From time to time you may need to know what service is running on a particular port. The `/etc/services` file contains that information and more.

How to do it...

Here we will take a look at some common services on a Linux system:

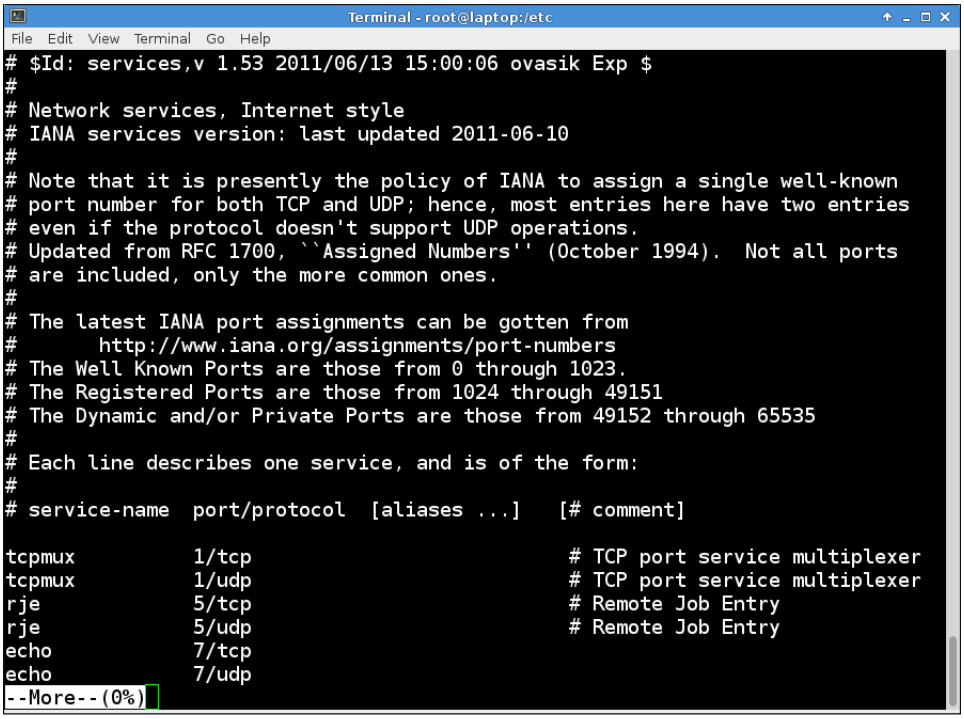
1. View your `/etc/services` file: `more /etc/services`.

2. Press the Space bar once to scroll the page down. See where it shows `"ftp"?`. The `21` in the next column indicates the port that FTP is using. The `/tcp` and `/udp` means that port is available for both protocols.
3. You should also see `telnet` (if not, press *Enter* a few times). It should show port `23`.
4. Now let's search for a service. Press the Backspace key a few times to go back to the top, and enter `/nameserver`.
5. We can see that `nameserver` is using port `42` and is available for both TCP and UDP.
6. The file is quite long (over 11,000 lines on my system) as there are many, many standard ports.

There's more...

By convention, ports numbered `0` to `1023` are considered the "well-known ports". Ports numbered `1024` to `49151` are the "Registered Ports". The Private and Dynamic ports go from `49152` to `65535`. If you are developing an application or are otherwise dealing with ports be sure to follow this port use convention.

The following is a screenshot of the first few lines of `/etc/services` on Fedora 18:



IPv4 versus IPv6

As of the writing of this book, **Internet Protocol Version 4** (**IPv4**)is used to route most of the network traffic on the Internet. It uses 32-bit (4 byte) addresses, which allows for a space of 2^{32} addresses. Under IPv4, there are not enough addresses remaining to accommodate all of the Internet devices now in use, which is why **IPv6** was created. In this section we will first run some commands and then discuss IPv4 and IPv6.

How to do it...

Let's try some commands using both protocols:

1. Let's start with IPv4. Run the following command:

   ```
   ping 192.168.1.115
   ```

2. Then, run the following command:

   ```
   ssh 192.168.1.115
   ```

3. Now let's try IPv6. First let's see if your computer has an IPv6 address. Run `ifconfig` on your interface; on my laptop it's `ifconfig wlan0`.
4. You should see something like the following output:

   ```
   inet 192.168.1.115  netmask 255.255.255.0  broadcast 192.168.1.255
   ```

5. You should also see something like the following output: `inet6 2002:244:b303:0:221:5eff:feff:f15d`. Try pinging it. On mine, it's `ping 2002:244:b303:0:221:5eff:feff:f15d`.
6. Did you get a `ping: unknown host` error? That's because we have to use the IPv6 version of ping. Try again, but like the following command:

   ```
   ping6 2002:244:b303:0:221:5eff:feff:f15d
   ```

7. Now try `ssh`, it seems to already know about IPv6:

   ```
   ssh 2002:244:b303:0:221:5eff:feff:f15d
   ```

There's more...

In general, under IPv4, a numeric IP address is written as `xxx.yyy.zzz.www` which is called dotted decimal format. Each of the 3 numbers separated by the dots are called an octet. The high order octet, the `xxx` in this example, indicates which class of IP this is. The following is a partial list of the classes available in IPv4:

Classes	Ports
Private network	10.0.0.0
Local loopback	127.0.0.1
Link-local	169.254.0.0
Private network	192.168.0.0
IP multicast	224.0.0.0
Broadcast	255.255.255.255

You have already seen some examples of the private addresses I use on my home network. The format is usually something like the following set of commands:

```
ping 192.168.1.115
scp file1.txt guest1@192.168.1.115:/temp
telnet 192.168.1.115
ssh 192.168.1.115
```

The IPv4 address space became officially exhausted in 2011. For this, and other reasons, IPv6 was created.

IPv6 is the most current version of the internet protocol. As opposed to IPv4 which uses a 32-bit address, IPv6 uses a 128-bit address. This is 2^{128} addresses, or enough for every person on Earth to each have well over a million for their own use. We probably won't run out of these anytime soon.

An IPv6 address has 8 groups of 4 hex digits, separated by colons: `2002:4244:b323:0:4687:fcff:fe69:4d0f`.

Yes, this looks formidable but there were several reasons for the change:

- The longer address allows for more efficient routing
- It allows for special addressing techniques
- The size of a subnet is now 2^{64} addresses
- Renumbering an existing network under IPv4 can be very complicated; when configured correctly, this is not an issue with IPv6
- Multicasting is done more efficiently
- IPv6 has **Stateless Address Autoconfiguration** (**SLAAC**)

IPv6 addresses can be abbreviated as follows: leading zeroes can be omitted.

Consecutive groups of zeroes can be omitted and replaced with two colons.

Look at the following example: the IP address is `2002:0244:0000:0000:0000:fcff:fe69:4d0f`.

Omit leading zeroes, we get `2002:244:0:0:0:fcff:fe69:4d0f`.

Now omit the zero groups and we get `2002:244::fcff:fe69:4d0f`.

For more information consult the man pages and Internet. Large usage of IPv6 is still a long way off, but will be here eventually.

5
Permissions, Access, and Security

In this chapter we will cover:

- Creating and managing user accounts – useradd
- Working with passwords
- Working with file permissions
- Working with firewalls and router settings
- Working with Secure Linux – SELinux
- Using sudo to secure a system
- The /tmp directory

Introduction

This chapter will serve as a brief review of Linux file permissions, and how access to the machine is handled by the password system. It will also show how to work with the security features in the firewall and router, and mentions SELinux and sudo.

Creating and managing user accounts – useradd

In this section we will show you how to add a user account using the `useradd` program.

Getting ready

These commands should not be destructive to your system; however, you will need to be the root user.

In most Linux distributions there are two versions of this command, `useradd` and `adduser`. They do not always do the same thing, so consult your `man` page (and/or the `file` command) to make sure you are running the proper one. On Fedora, `adduser` is a symbolic link to `useradd` and so they are equivalent.

How to do it...

Here we will run the `useradd` command to add a user and the `passwd` command to set the password. There is more discussion of `passwd` in the following section:

1. Firstly, we will be changing `/etc/passwd`, so let's make a backup copy of it. Run the following command: `cp /etc/passwd /tmp/passwd.orig`
2. Now let's create a user named `test1`:

   ```
   useradd test1
   ```

3. It should have returned silently back to the command line. Now let's try it:

   ```
   su  -  test1
   ```

4. You should see the prompt change. Run `whoami`, it should say `test1`. Be sure to run this every time you use `su`. Now let's change the password:

   ```
   run passwd
   ```

5. It will say something like **Changing password for user test1**. But then it prompts for the current password. What does this mean? What password does it want?
6. I don't really know the answer to this and the **man** pages are useless. They always skip this step, which is rather odd. There are ways to do this using the crypto function and some other complicated procedures. However, the following is my approach:
7. Press *Ctrl + C* to come out of the `passwd` command and run `exit` to return to the root account. Now `edit /etc/passwd` and go to the bottom line. `1003` below will probably be different on your system, but you should see a line similar to this:

   ```
   test1:x:1003:1003::/home/test1:/bin/bash
   ```

8. Delete the `x`, so that the line now looks like:

   ```
   test1::1003:1003::/home/test1:/bin/bash
   ```

9. Save the file and exit. If you get a permission error remember you have to be root for this procedure.
10. Now run `su` to become the `test1` user again:

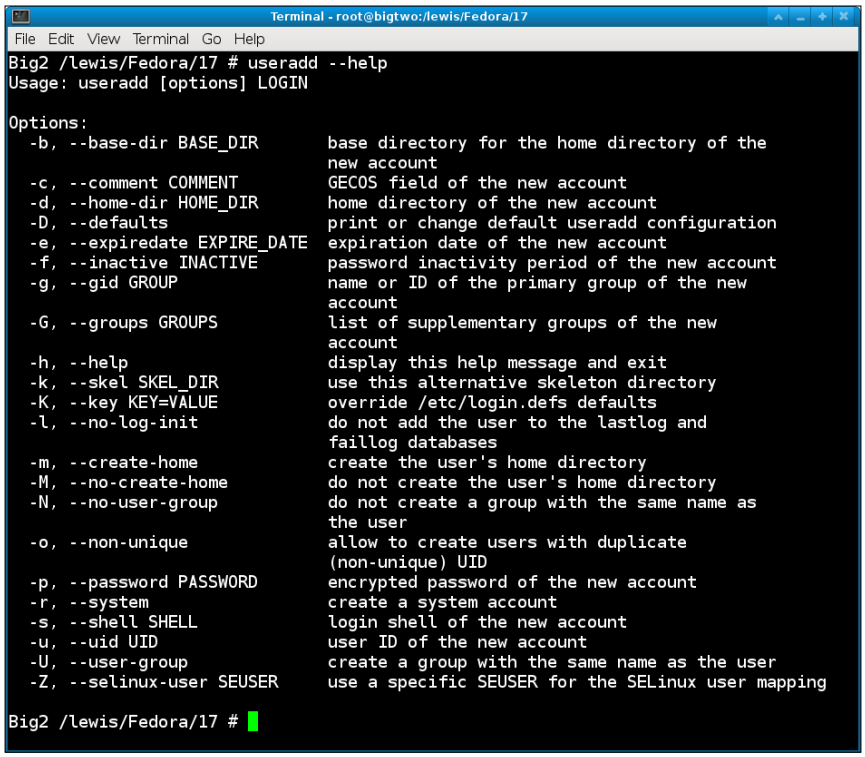
    ```
    su - test1
    ```

11. Run `passwd`.
12. Hey, cool, it's not asking for the current password this time. So go ahead and create one now as we really don't want an open account on the system. If you plan to keep this account I suggest writing this password down or even better putting it into an encrypted file somewhere safe.
13. After entering the same new password twice you should get a message similar to:

    ```
    passwd: all authentication tokens updated successfully
    ```

We now have a new user. Note that in general this user can perform most of the activities on the command line that he has the proper permissions for. However, depending on the Linux distribution, the user may not be able to access all resources (the sound system, for example).

Here is a screenshot of useradd –help taken on my Fedora 17 system:



There's more...

The `useradd` command can do a whole lot more than just create new accounts. You can change how an existing account works, or when it expires. You can even give a user system authority so he has almost has much power as root. Consult the man pages or use the `–help` option for more information.

Working with passwords

I mentioned the `passwd` command in the previous recipe. It is used to update a user's authentication tokens. You will need to be the root user for this example. We will use the `test1` user created in the above section.

How to do it...

Let's work with the `passwd` command a little:

1. From a user account, login to `test1` to make sure that still works as expected:

   ```
   su - test1
   ```

2. Enter the password when prompted. This should work without errors.
3. Now let's lock this account. Exit back to root and run:

   ```
   passwd -l test1
   ```

4. From a user account run `su - test1` and enter the password again. It should fail.
5. Go back to root and unlock the account using the command `passwd -u test1`. Log in again to make sure it works.
6. Now let's expire the account. This will force the user to create a new password. As root run the following command:

   ```
   passwd -e test1
   ```

7. Now as a guest user, log into `test1` using the command `su - test1`. Enter your password.
8. You will be told to create a new password. Be careful here as you have to enter the old (current) password again, and then the new one two times. Yes, it does seem odd that we have to type the old password again, since we just did that.
9. Note that you can delete a password for a user account by running `passwd -d test1`. This is easier than editing the `/etc/passwd` file directly as we did in the above section.

There's more...

You can set a lot of other things on a user account. These include the amount of time for the account to remain active, and when to start warning the user to change their password. See the man pages for more info.

A word about passwords

In the old days, we would pick a relatively simple password and keep it forever. We didn't need to change it all the time, and could use it for everything, so it was not necessary to write it down. Unfortunately, that has now changed. Passwords usually need to be a combination of uppercase letters, numbers, and maybe even special characters. They have to be a lot longer in length as well. You can't always use them for everything because the password rules on one system may differ from the rules on another. For these reasons I suggest using a different password for every account when reasonable, and record it somewhere safe. You will most likely have to change this password at regular intervals.

Working with file permissions

Since Linux was designed to be a multiuser operating system, every file has file permissions and ownership associated with it. This is to prevent one user from overwriting the files of another (either intentionally or unintentionally). The root user can (usually) access every file in the system.

Getting ready

Here is a quick review of basic file system permissions. For this example, it is assumed `umask` is set to `0022`. Run `umask` to make sure.

Observe the following `ls -la` listing of my backup script b:

```
-rwxr-xr-x. 1 guest1 root 559 Mar 28 12:43 b
```

Starting from the left, the first position indicates what kind of file this is. A `-`, as shown, means this is a regular file. A `d` there would mean a directory, and an `l` would indicate it's a link. The next three sets of three letters are the file permissions and can be referred to in either symbolic or numeric mode. We are going to use numeric (octal) mode.

The first three sets, `rwx`, are the settings for the owner (`guest1`) of this file. The next three, `r-x`, are the settings for the group (`root`). The third set is for all others. `r` means the file is readable, `w` means it's writeable, and `x` means it's executable.

The `chmod` command accepts one to four octal digits. If a digit is missing it is assumed to be a leading zero. The first digit sets the user ID, group ID, or the sticky bit. The second digit selects the permissions for this user, and the third selects the permissions for the others.

So, let's now change some permissions on a temporary file and see what happens.

How to do it...

Let's work with some file permissions:

1. Let's change to the `/tmp` directory by using the command:

   ```
   cd  /tmp
   ```

2. If the file `f1` exists remove it using the following command:

   ```
   rmf1
   ```

3. Using a guest account (in my case `jklewis`) create a temporary file by using the command:

   ```
   ls>f1
   ```

4. Now run the following command:

   ```
   ls -al f1
   ```

5. It should show something like the following output:

   ```
   -rw-rw-r--. 1 jklewisjklewis 131 Mar 29 10:35 f1
   ```

6. Those are the default permissions based on the `umask` command. This indicates that owner and group have read and write privileges, and others have only read.

7. So how do we change these? By using the `chmod` command. Suppose this is a script and we wanted to make it executable. Run the following command:

   ```
   chmod 775  f1
   ```

8. Now run `ls -la f1`; it should now look like the following output:

   ```
   -rwxrwxr-x. 1 jklewisjklewis 131 Mar 29 10:35 f1
   ```

9. Those `x`s show that every user can run the file. Let's do one more. Run `chmod 000 f1`and then `ls -la f1`, it will show the following output:

   ```
   ----------. 1 jklewisjklewis 131 Mar 29 10:35 f1
   ```

Wow! No one will be able to do anything with this file now, right? Well, no, the owner of the file can still change the permissions. Speaking of ownership, the `chown` command is used to change that field. It is normally run as the root user.

There's more...

I did not mention the `setuid`, `setgid`, or `sticky` bits. Consult the `chmod` man page for information on these settings. The restricted deletion bit is mentioned in *The /tmp directory* section.

Working with the firewalls and router settings

A firewall is used to prevent unauthorized network access to a machine(s) while still allowing normal (or legal) traffic to pass through. The `iptables` command is used to set up, configure, and view the tables of the IPv4 rules in the kernel. It is somewhat complicated and so this will serve as just a simple overview.

`iptables` uses one or more tables. Each table has a number of pre-made chains and can also contain user-created chains. A chain is a list of rules, and a rule specifies what to do with a packet that matches. This "match" is called a **target**.

When a packet does not match, the next rule in the chain is looked at. If it does match, one of the following can be specified for the packet:

- `ACCEPT`: It allows the packet to pass on
- `DROP`: It rejects the packet
- `QUEUE`: It passes the packet on to the user space
- `RETURN`: It stops the running of this chain and continues at the next rule in the calling chain

How to do it...

Here are a few `iptable` commands. Do not run these commands on your system; this is an example only:

1. To delete all existing rules, use the following command:

   ```
   iptables -F
   ```

2. To block a specific IP address, use the following command:

   ```
   iptables  -A INPUT -s 192.168.1.115  -j DROP
   ```

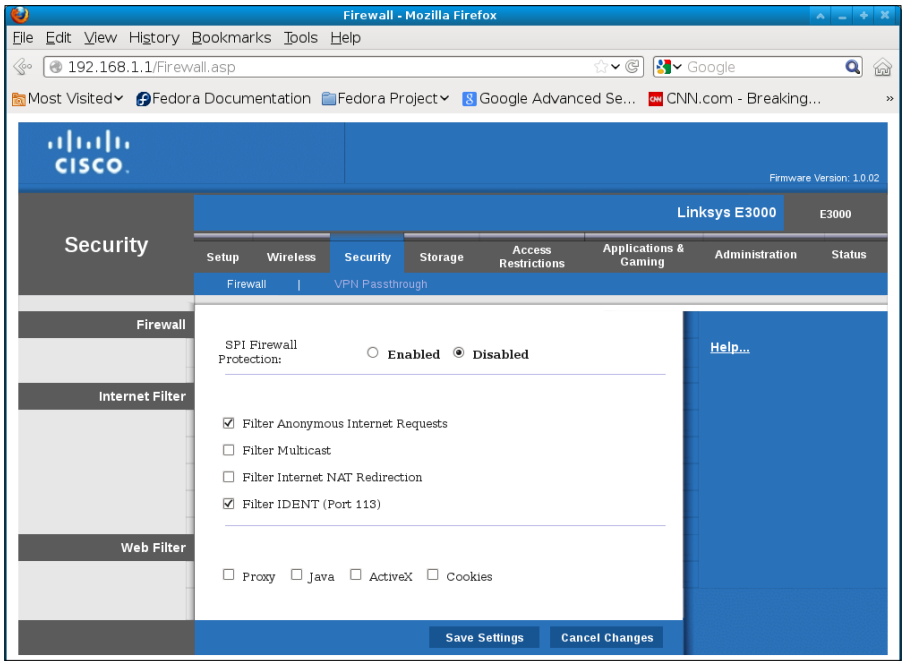
3. To allow loopback access, use the following command:

   ```
   iptables -A INPUT -i lo -j ACCEPT
   iptables -A OUTPUT -o lo -j ACCEPT
   ```

Now let's talk about routers. Most routers have a firewall built-in that can be managed by a web browser. While it does not replace `iptables`, it is usually easier to configure and can work across your entire network.

The web page for a typical home router usually has a `192.168.1.1` address. Try it in your browser now.

Here is a screenshot of my router on the **Security** page:



You may have to enter an ID and password. Consult your router documentation for the defaults if you haven't already changed them. Go to the **Security** (or equivalent) tab to access those features.

There's more...

There is a whole lot more to `iptables`, enough to fill an entire book. For more information refer to the man pages, or a book on firewalls. There are also quite a few good websites on the topic.

Working with Secure Linux – SELinux

This section will serve as an overview of **Security Enhanced Linux** (**SELinux**). In the *Working with file permissions* section, we discussed how standard Linux provides protection for the system. This method is called **Discretionary Access Control** (**DAC**), and has some limitations. For example, a typical user could open his files up, either accidentally or on purpose, for any other user to read or write. This could allow unauthorized access to sensitive information. To provide more security, SELinux uses **MAC** (**Mandatory Access Control**). MAC uses a security policy that covers all processes and files in the system. All files in SELinux have labels that contain security-relevant information.

For example, the following is a normal listing of a file under DAC:

```
ls -la ifcfg-eth0
-rw-r--r--. 1 root root 73 Apr 22  2011 ifcfg-eth0
```

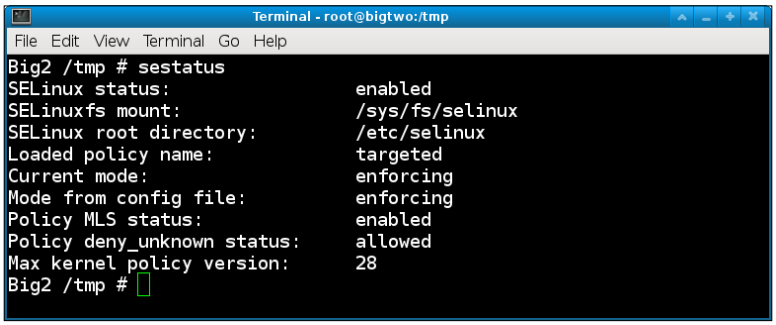
Same file, but with the Z (security context) option to `ls`:

```
ls -Z ifcfg-eth0
-rw-r--r--. root root unconfined_u:object_r:default_t:s0ifcfg-eth0
```

`unconfined_u` is the user, `object_r` is the role, `default_t` is the type, and `s0` is the level. This info is used to make access control decisions. Note that the normal DAC rules are checked first, if they do not allow the action then the SELinux rules are not used.

Getting ready

We are only going to run some commands as root and view some of the settings. We will not be making any changes to the configuration. This example will assume you are already running SELinux in Enforcing mode. To determine if this is so run the `sestatus` command. The output should be similar to the following screenshot



How to do it...

Okay, so let's run some SELinux commands.

1. Run `getenforce`; it should report enforcing.
2. Now let's view the list of mappings:
   ```
   semanage login -l
   ```
3. To view the SELinux contexts for processes run the following command:
   ```
   ps -eZ
   ```
4. To view the context for user run the following command:
   ```
   id -Z
   ```

5. The `sealert` command is used to view the complete SELinux message when an error occurs. Check your `/var/log/messages` file to see if any alerts have been generated, and if so can run `sealert -l` on the number to get a detailed view.
6. To get a detailed list of the SELinux Booleans with descriptions run the following command:

   ```
   semanage boolean –l
   ```

7. To see the list but without the descriptions run the following command:

   ```
   getsebool –a
   ```

8. To check if files and directories have the correct SELinux context run the following command:

   ```
   matchpathcon
   ```

There's more...

As I mentioned SELinux is normally installed by default in most distributions. In some cases you may not even realize it is there. However, at times it will get in your way. If you try to install a service, for example, `vsftpd`, it may fail because it will violate the SELinux policy. It will usually pop up a fairly decent error message. The message may even tell you how to fix the problem, however, I have found in practice that it doesn't work. You perform the action(s) it gives, it seems to run without error, but then the access is still denied. In these cases I use the `setenforce` command to put SELinux into Permissive mode and then carry on with my activity:

```
setenforce 0
```

Note that this only works until the next reboot.

For more information on SELinux there is a rather excellent guide on the Fedora website.

Using sudo to secure a system

There may be times, especially if you are a system administrator, when you would like to give a user more access to the computer, but not actually root authority. This can be done by modifying the `/etc/sudoers` file and having your users invoke the `sudo` command.

Getting ready...

The following steps should not harm your system. We will perform these with a user account made above. You will need to be the root user for this section.

How to do it...

Here we will work with the `/etc/sudoers` file:

1. Make a backup copy of your `/etc/sudoers file`:

   ```
   cp /etc/sudoers /tmp/sudoers.orig
   ```

2. You don't edit this file directly, the `visudo` command is used. This is poorly named as any text editor can be used if you export the `EDITOR` variable accordingly. Set the variable if desired then run the command:

   ```
   visudo
   ```

3. This command makes a temporary copy of the `sudoers` file and edits it. If all goes well it then copies the temporary file over the original when you are done. So, let's take a look at this file.
4. Read the section on aliases. They are divided into groups for things such as **networking, software, services, locate**, and others. For now let's just jump in and see how this works.
5. But first let's try something. Open another session as a guest user. I'll use my `jklewis` account.
6. Under the `jklewis` account type the following command:

   ```
   cd /tmp
   ```

7. Create a file using the command:

   ```
   ls>f1
   ```

8. Now try to copy that file to `/usr/bin`:

   ```
   cp f1 /usr/bin
   ```

9. You should have received an error. This is correct of course, a normal user can't normally write to the `/usr/bin` directory. Now go back to your `visudo` session.
10. You will need the hostname for your machine. For this example we will use the numeric IP. You can obtain that from the `ifconfig` command if needed.
11. Just after the lines that mention the shutdown command we are going to add a line for our guest user. The syntax is username, hostname, commands, and options, so add the line:

    ```
    jklewis  192.168.1.115=(ALL)  ALL
    ```

12. Save the file and close the `visudo` session. Now try this command again. While still in the `/tmp` directory run `cpf1 /usr/bin`. It should still give the error message. Now try it like the following command:

    ```
    sudo cp f1 /usr/bin
    ```

13. Ah, it's asking for a password, right? Yes, and it's asking for the user's password, not the one for root. This can be easy to confuse (well, it was for me anyway). The best way to remember this is you probably aren't supposed to know what the root password is. Enter yours.
14. If this is the first time `sudo` is being used by this user an interesting notice may pop up. Read it and understand it to avoid the wrath of your system admin.
15. After all of this the command should proceed without error this time. Pretty cool, huh? Since we used `ALL` in the line we added to the `sudoers` file it effectively has given this user full access. Note that some things will still not work the same, redirection for one.
16. Okay, so we probably don't really want this `jklewis` jerk messing up our system, so let's fine tune this a bit. Run `visudo` again.
17. Scroll down or search for **Processes**. Uncomment the `# Cmnd_Alias PROCESSES` line by removing the # mark.
18. Scroll back down to the `jklewis` line we added earlier. Change it to read:

    ```
    jklewis  192.168.1.115=(ALL) PROCESSES
    ```

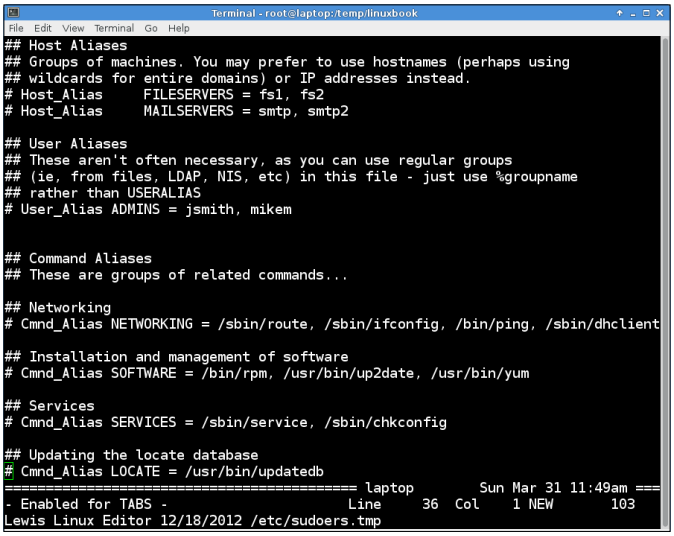
19. Now we need a process to kill. As root start up a `vi` session. Something like `vi mybook` will work.
20. In your user session run `psauxw | grep"vi mybook"` and remember the process number (**PID**).
21. Also in your user session run `kill -9` on the above PID. It will give an error. Now run it again but with like the following command:

    ```
    sudo kill -9 <pid>
    ```

22. The process with `vi` should be killed.

If the screen stays blue or some other color just run `the ls command.` That should clear it up.

The following is a screenshot of my `/etc/sudoers` file:



There's more...

You can see from the `sudoers` file that a lot of fine tuning can be performed on it. Users can be given very little extra authority, or a whole lot. For more information run `man sudoers`.

Here's my two cents on `sudo`. There are many computer users who can be trusted to run as root without harming the system. However, there are exceptions. If you set up `sudo` you might spend a whole lot of time trying to get it just right, only to find that you are always having to add more things. Your users will get upset because they can't do their work until you make the changes. Then, when you think you finally have it just right, someone, using `sudo`, will still mess up and mess up bad. It has been my experience that if a user makes this mistake once, he will do it again and again.

The /tmp directory

The `/tmp` directory is somewhat special as by default it allows all users to write files to it. Here is what the listing for `/tmp` looks like on my system:

```
drwxrwxrwt.  10 root    root      4096 Mar 31 03:48 tmp
```

You can see this is open for everyone. The `t` in the permissions indicates that the restricted deletion bit is set on the directory. So what does this actually mean? For directories, it prevents normal users from removing or renaming a file in the directory that they don't have the proper permissions for.

As a normal user you still need to be careful when writing to `/tmp`, as there are some restrictions.

How to do it...

Let's try a few things to get an idea of how `/tmp` works:

1. Run the following command:

   ```
   cd /tmp
   ```

2. If there are any temporary `.txt` files lying around from earlier sections clean them up; an `rm *.txt` file should do it.
3. Now run the following command:

   ```
   ls>root1.txt
   ```

4. In another session, as a guest user (I'll use `jklewis`), run the following command:

   ```
   cd /tmp
   ```

5. Run the following command:

   ```
   ls>jklewis.txt
   ```

6. This should work without error. Now try:

   ```
   ls>root1.txt
   ```

7. You should have received the **Permission denied** error. Why? Because even though normal users can all write to the `/tmp` directory, the normal file system (DAC) permissions must still be followed.

There's more...

For the reasons outlined above I would suggest not using the `/tmp` directory for anything except for truly temporary files. In addition, most distributions routinely clean out `/tmp` and so anything not owned by root is going to get deleted. Remember all of this when generating temporary files especially when writing scripts (more on that in *Chapter 8*, *Working with Scripts*).

6
Processes

In this chapter we will cover the following topics:

- Understanding processes
- Examining processes with ps
- Examining processes using top
- Changing priorities with nice
- Observing a process using the /proc filesystem

Introduction

All the programs running in Linux are processes. In this chapter, you'll learn how to view their status by using ps and top, how to set the priority at which they run, and how to view the internals of a process by using the /proc filesystem.

Understanding processes

Every process has a unique identifier called a **Process Identifier** (**PID**). Also, every process has a **Parent Process Identifier** (**PPID**). There is an exception, `init` (or `systemd`). The `init` process starts all other processes and has a PID of 1. This process is special because it cannot be killed (and any attempt to do so is often fatal).

Similar to files, a process also has access permissions. These are referred to as the Real user and Group IDs. This provides a level of protection by not allowing unprivileged users to access critical operating system tasks. Memory, open files, and other resources are owned by the process and are kept separate from other processes (in most cases).

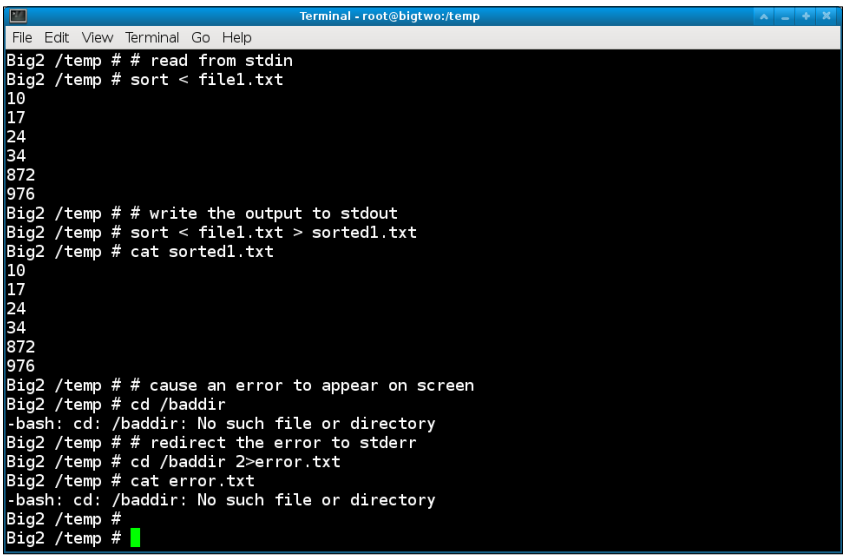
From the user's perspective, a process is typically started from the command line or desktop. Take editing a file with `vi`, for example. The user starts the session and works interactively with the editor. He can save the file or edit another one. As long as the session is active, there is a PID associated with it. When the user terminates `vi`, the PID and all the associated resources are terminated as well. That PID will be recycled and used by the OS again later. A program run is this way is called a foreground process.

A process can also be run in the background. For example, in *Chapter 4*, *Networking and the Internet*, we ran the browser with the `firefox &` command. The ampersand puts the process into the background, freeing up that terminal for more input/output. Note that messages coming from the application (Firefox in this example) will still be output to that terminal. This can really be helpful when debugging. Also note that a process run in the background tends to run at a lower priority than a foreground task (more on priorities later). In the old days, background processes were also commonly referred to as jobs.

There are some special file handles associated with each process:

- **Standard input** (`stdin`): A process takes its normal input from here (Handle 0)
- **Standard output** (`stdout`): A process writes its normal output to here (Handle 1)
- **Standard error** (`stderr`): A process writes its error output to here (Handle 2)

The following is a screenshot showing the standard handles:



Other files opened by a process start at handle 3. There are some processes that have been given special names. For example, many of the processes on a Linux system are run in the background and, in most cases, are never meant to be run from the command line. These are called services or daemons. The **Hypertext Transfer Protocol Daemon** (**HTTPD)** web process is a good example of a service.

Typically, a service waits for an event or events to occur, performs an action or some actions, and then goes back to waiting again. If a service logs any activity, it will generally do so to the `/var/log/<service-name>` directory.

If the terminal a background job was started from is terminated, the background job will end as well. Some Linux distributions will attempt to warn the user of this condition.

How to do it...

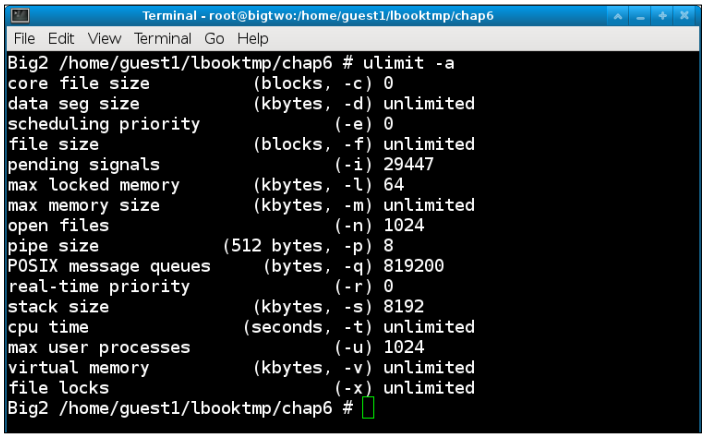
The following is a brief list of commands one can use to look at processes:

- To see a snapshot of the processes currently running on the system:
  `ps auxw`
- To see the processes in real-time run:
  `top`
- To see all the types of process directories:
  `ls /proc`

There's more...

A process can spin off other processes. It can also spin off threads. A thread inherits all the handles and resources of the parent. It is generally used in programming to perform a small task concurrently, while the main task is running, and return quickly.

Are there limits on the resources? Yes. The `ulimit` command is used to view and set the hard and soft limits on a process. It is not normally needed by the user; however, if you're curious, run `ulimit -a` on your system. You should see an output similar to the following screenshot:



For more information on `ulimit`, consult the man page.

Examining processes with ps

The `ps` program allows the user to see a snapshot of the processes running on the system. By using the appropriate parameters, the output can be changed to include more or less information. For this section we will run as root, and use the BSD style `ps`. The options may be grouped, and no dash is used.

How to do it...

Carry out the following steps to run `ps`:

1. Running just `ps` with no parameters will give something like the following output:

```
Big4 /temp/linuxbook/chap6 # ps
  PID TTY          TIME CMD
 5197 pts/25   00:00:00 su
 5218 pts/25   00:00:00 bash
17789 pts/25   00:00:00 ps
```

2. Since this is not very informative, let's show every process that has a TTY:

```
ps a
```

3. Now, include the processes that don't have a TTY:

```
ps ax
```

4. Display the output in a more user-oriented format: `ps aux`. Note the change in the header.
5. If the lines are cut off at the end on your system, add the wide option using the following command:

```
ps auxw
```

6. There sure is a lot of output. Here are some ideas on how to deal with it:

```
ps auxw | more
```

7. You can redirect the output to a file as well: `ps auxw > ps-output.txt` and then view it with `more` or `vi`.
8. You can also use `grep` to find a particular process ID. In another terminal, run the following command:

```
vi file1.txt
```

9. Now back in your original terminal, run the following command:

```
ps auxw | grep file1.txt
```

10. You should see a line containing the text `vi file1.txt`. This is the PID you are looking for and one of the most common uses of `ps`.
11. You can also show a tree view of the processes:
    `pstree`

How it works...

The `ps` command gets its information from the `/proc` file system. Every running process has an associated entry here. We will discuss `/proc` in more detail later in this chapter.

There's more...

The BSD style header will look something like the following:

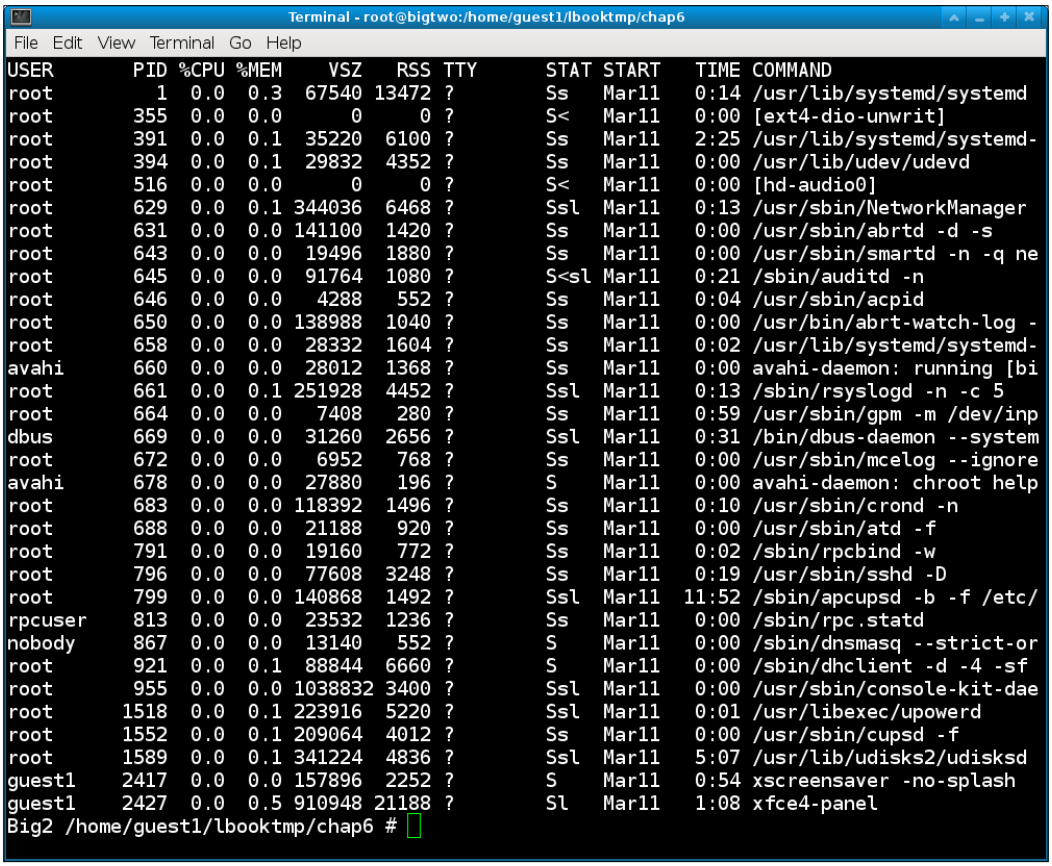
```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
```

The definitions of each of the terms are given as follows:

- `USER`: It is the owner of the process
- `PID`: It is the process identifier
- `%CPU`: It gives the CPU time divided by the time the process has been running for
- `%MEM`: It gives the ratio of the process memory to the physical memory
- `VSZ`: It contains the virtual memory size of the process
- `RSS`: It contains real memory resident set size
- `TTY`: It represents the terminal associated with this process
- `STAT`: It represents the process state
- `START`: It gives the time the process started
- `TIME`: It gives the total CPU time
- `COMMAND`: It is the name of the command

The `ps` command has quite a few other options to it. You can view threads, get security (`SELinux`) info, tighten what is displayed by the username, and change the output format. You can even modify some environment variables to change how `ps` works. See the man page for more information.

The following screenshot gives an idea of what `ps - auxw` looks like on my Fedora 17 system:



Examining processes using top

The `top` program is similar to `ps` except that it shows the state of the system in real time. You can control how it operates using command line switches and/or its interactive interface. It can also use a configuration file. You can do quite a bit with `top`. The following are some examples of what is possible.

Getting ready

No special setup is needed. These commands will be run as root.

To get help for `top`, you can run the following command line:

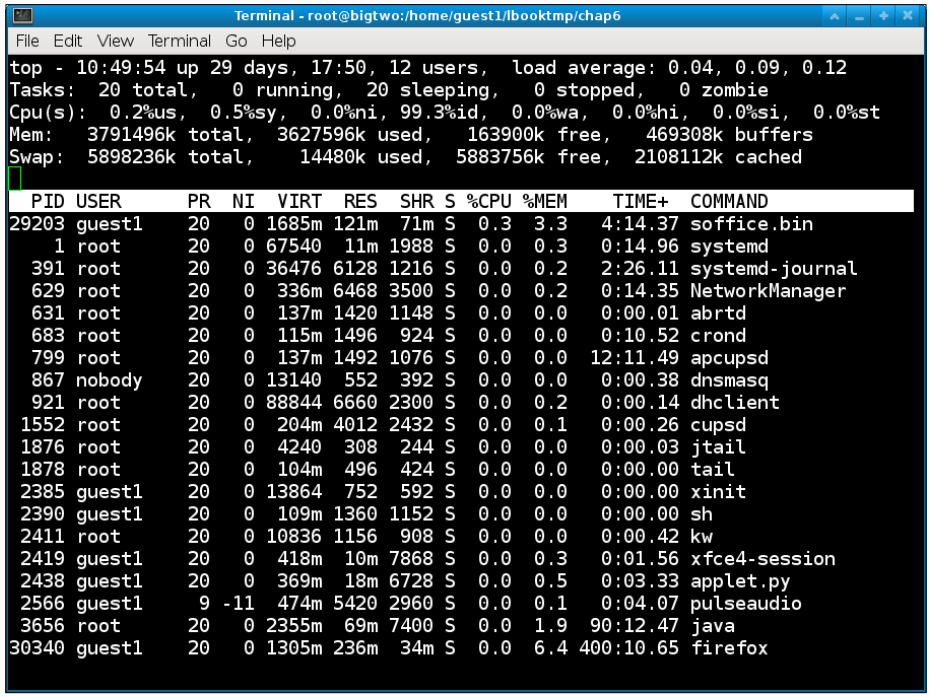
```
top -h or -v
```

These are equivalent and show the library version and usage.

The general syntax for `top` is as follows:

```
top -bcHisS -d delay -n iterations -p pid [,pid...]
```

The following is a screenshot of `top` running on Fedora 17:



How to do it...

Following are some examples using the command line switches:

1. To update the screen every 2 seconds:

   ```
   top -d 2
   ```

2. To update the screen every half second:

   ```
   top -d 0.5
   ```

3. To update 10 times and then quit:

   ```
   top -n 10
   ```

4. To do both:

   ```
   top -d 0.5 -n 10
   ```

5. To update the screen immediately, press Spacebar or *Enter*.
6. To monitor a specific PID, use the `-p` option. This will show the `init` (or `systemd`) process:

   ```
   top -p 1
   ```

7. Let's monitor a few processes. First, run `ps auxw` and remember the PIDs for four processes.
8. Then run `top`, substituting the PIDs obtained previously:

   ```
   top -p pid1,pid2,pid3,pid4
   ```

9. To omit showing idle processes, run the following command:

   ```
   top -i
   ```

10. To also show threads, run `top -H`. To monitor only the processes for a certain user, the syntax is `top -u <username>`. Try it with root:

    ```
    top -u root
    ```

11. You can run `top` in the batch mode. For example, to save the output to a file:

    ```
    top -b -n 10 > top1.txt
    ```

The following are some examples using the interactive commands. Start `top` and follow along:

1. To show just the processes owned by a particular user (root in this example), press `U` and then enter `root`.
2. To change the delay time, press `D` and then enter a time by pressing *D* followed by *1*.
3. To show all the CPUs on a multi-core machine, press *1* (press *1* again to toggle back).
4. To toggle the showing of the command line versus the program name, press *C*.
5. To change the nice setting on a process, press *R* and then enter the priority desired.
6. To activate Secure Mode, press *S* (see Secure Mode explained afterwards).
7. To send a signal to a process, press *K* and then enter the signal to send. Be sure you have the correct PID and signal before performing this action.
8. To write the configuration file, press `W`. This will create the file `.toprc` in the user's home directory and will use the settings in it on the next startup of `top`.
9. To display the Help screen, press *H*.
10. To quit top, press *Q*.

How it works...

The following is a description of the first five lines shown in the previous screenshot:

- `top`: It contains the time of day, machine uptime, number of users, and load average
- `Tasks`: It gives the number of total tasks, number currently running, sleeping, stopped, and zombie
- `Cpu(s)`: The different types of CPU states are as follows:
  - **us**: It represents the user time
  - **sy**: It represents the system time
  - **ni**: It represents the nice time
  - **id**: It represents the idle time
  - **wa**: It represents the I/O wait time
  - **hi**: It gives the time spent servicing hardware interrupts
  - **si**: It gives the time spent servicing software interrupts
  - **st**: It represents the stolen CPU time
- `Mem`: It gives the total memory in machine; used, free, and buffers in KBs
- `Swap`: It gives the total swap space; used, free, and cached in KBs

The following are the definitions for the standard header. Note that these can change based on command line options or interactive commands:

- `PID`: It defines the process identifier
- `USER`: It holds the username of the owner of this task
- `PR`: It holds the priority of this task
- `NI`: It consists of the nice value (more on priorities in the next section)
- `VIRT`: It exhibits the total amount of virtual memory used by this task
- `RES`: It gives the physical memory used by this task
- `SHR`: It gives the shared memory used by this task
- `S`: It stands for the process status, which will be one of the following:
  - **S**: It stands for sleeping
  - **D**: It means uninterruptible sleep
  - **R**: It shows the process is running (or ready to run)
  - **T**: It shows that the process has been traced or stopped
  - **Z**: It signifies the zombie status

- `%CPU`: It gives the share of the elapsed CPU time since the last update, expressed as a percentage
- `%MEM`: It holds the currently used share of available physical memory, expressed as a percentage
- `TIME+`: It gives the total CPU time used by this task since it began running
- `COMMAND`: It is the command used to start this task

There's more...

In addition to the user configuration file, there can also be a global file. It is named `/etc/toprc` and consists of only two lines given as follows:

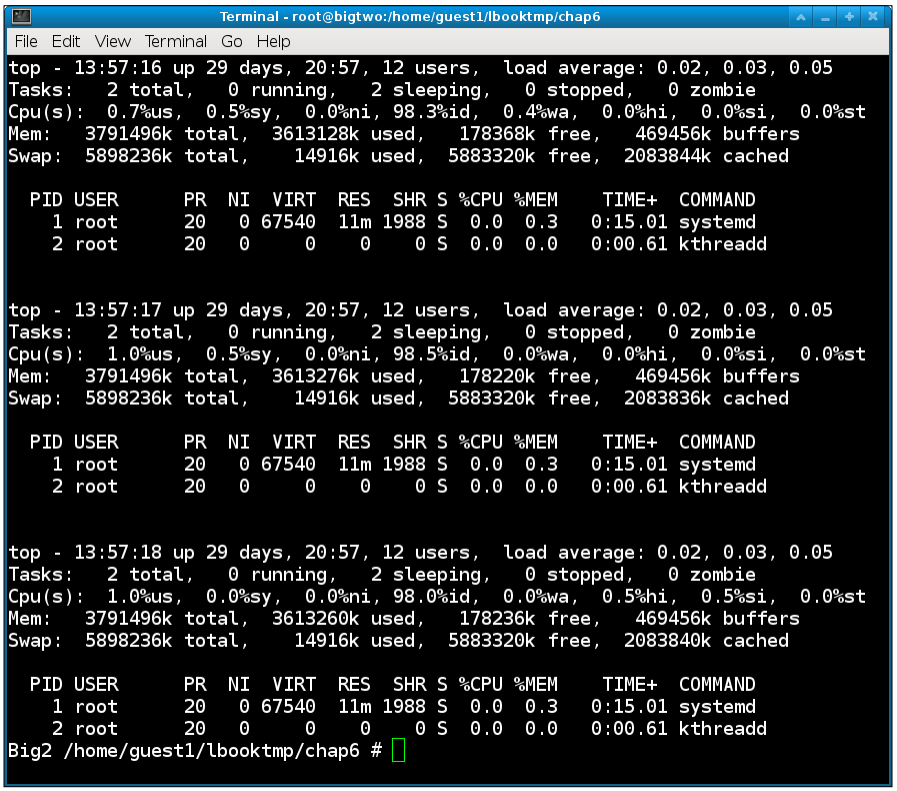
```
S # Line 1: secure mode switch
2.0 # Line 2: delay in seconds
```

This file is created manually by the root user. If this file exists, it activates the Secure Mode and changes how `top` operates:

- A different version of the Help screen is shown
- A user cannot kill a task
- A user cannot renice a task
- A user cannot change the delay interval of `top`

If `top` doesn't appear to be working as you expected, check for the existence of the configuration files both for the users and root. There are plenty of other things you can do with `top`. You can change how the fields are arranged and sorted. You can change the color and highlighting. There is a multiple window option too. For more information, see the man page for `top`.

The following is a screenshot of `top` on my Fedora 17 system:



Changing priorities with nice

The `nice` command allows you to adjust the priority a process runs at. Modern-day operating systems usually do a great job of task scheduling. However, a particular process may require some tweaking on occasions. Here, we will show how to use `nice`.

The priority is referred to as the niceness level. The range of niceness a process can have goes from 19, which is the least favorable, down to a maximum of -20, which is the most favorable (if this seems backwards to you, it does to me as well).

Most favorable_____Least favorable

20 _____19

Highest priority_____Lowest priority

You may recall the NI column from the previous section on `top`. This is the niceness setting and on Fedora, several services run at the most favorable setting of -20.

Note that changing the niceness setting of a process is not a guarantee that the OS will honor it. It is simply a suggestion to the scheduler.

Also note that the command given to nice must not be a built-in command.

The syntax for `nice` is as follows:

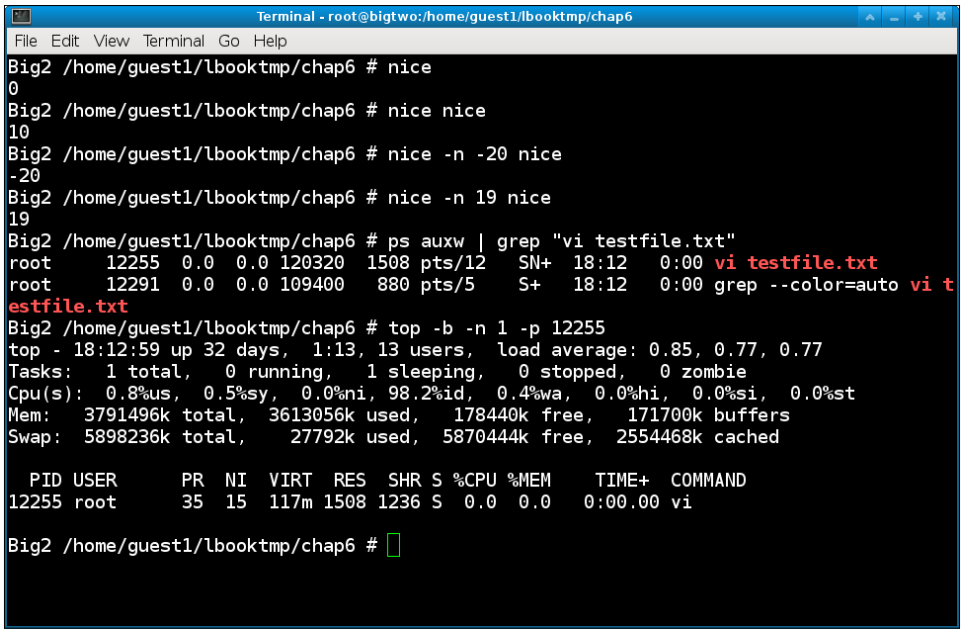
```
nice [Option]... [Command [Arg] ...]
```

How to do it...

Now, let's run a few `nice` commands:

1. To see the current nice value, run nice by itself:
   ```
   nice
   ```
2. Now run nice on itself:
   ```
   nice nice
   ```
3. See the output of 10? That's the default adjustment if none is given.
4. Now set it to the maximum value:
   ```
   nice -n -20 nice
   ```
5. For the minimum value:
   ```
   nice -n 19 nice
   ```
6. Now let's try something else. In another terminal, run the following command:
   ```
   nice -n 15 vi testfile.txt
   ```
7. Then find its PID:
   ```
   ps auxw | grep testfile.txt
   ```
8. Now run `top` on the previous PID:
   ```
   top -p <pid>
   ```
9. The nice column (NI) should show a niceness of 15.

The following is a screenshot showing `nice` running on Fedora 17:



There's more...

Unlike most other Linux programs, there isn't anything more to nice. If you try to set it above the maximum of 20, it will still use 20 and if you try to set it lower than the minimum of -19, it will use -19. Also, the `top` program allows you to adjust the niceness level of a process dynamically.

Observing a process using the /proc filesystem

The `/proc` filesystem is where Linux stores the data related to the currently running processes. In most cases, a casual user would probably never (hopefully never) need to know this information. However, it can be really helpful during debugging, or if you just want to know about some of the internals of Linux.

How to do it...

The following steps explain how to see the information contained in `/proc`:

1. Let's try an experiment in `/proc` as root run:

   ```
   file uptime
   ```

2. It shows `uptime: empty`, right? So if we cat it, there should be no output ideally, right? Try it:

   ```
   cat uptime
   ```

3. Wow, so how did that happen? The files in `/proc` are special because the information is read out of them in real time. Let's try some more and run the following command:

   ```
   cat interrupts
   ```

4. Now run `cat version` and `cat meminfo`.
5. In another session, run the following command:

   ```
   vi test5.txt
   ```

6. Let's find its PID:

   ```
   ps auxw | grep "vi test5.txt"
   ```

7. Change to that directory in `/proc`: `cd /proc/<pid-from-above>`.
8. Now change the directory to the **File Descriptor** (**FD**) directory:

   ```
   cd fd
   ```

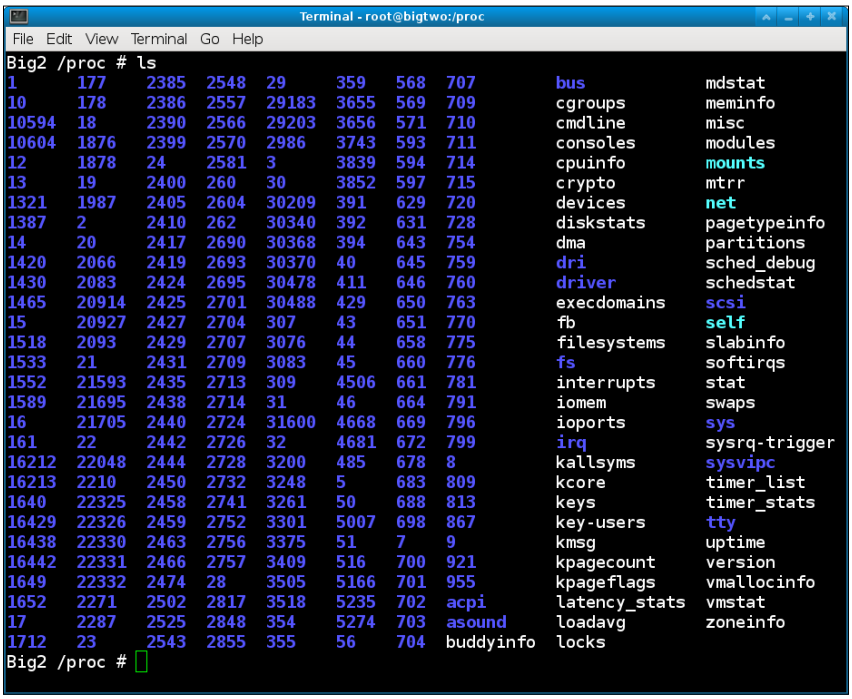
9. Run the `ls -la` command. You should see something like the following output:

   ```
   Big2 /proc/20879/fd # ls -la.
       total 0
       dr-x------. 2 root root  0 Apr 11 16:27 .
       dr-xr-xr-x. 8 root root  0 Apr 11 16:27 ..
       lrwx------. 1 root root 64 Apr 11 16:27 0 -> /dev/pts/10
       lrwx------. 1 root root 64 Apr 11 16:27 1 -> /dev/pts/10
       lrwx------. 1 root root 64 Apr 11 16:27 2 -> /dev/pts/10
       lrwx------. 1 root root 64 Apr 11 16:27 4 -> /tmp/.test5.txt.
   swp
   ```

You can see this is indeed our session of `vi` editing the `test5.txt` file. Note that the file shown is a temporary file created by `vi` and left open during execution.

There's more...

The following screenshot shows a listing of the `/proc` directory on the Fedora 17 system:



So what does all of that mean? The numbers are, as you may have guessed, process IDs. Every process will have a number here, which are actually directories that contain practically everything you would ever want to know about that process.

The following are what some of those files are for. I didn't list each one but covered the ones that I think are the most interesting:

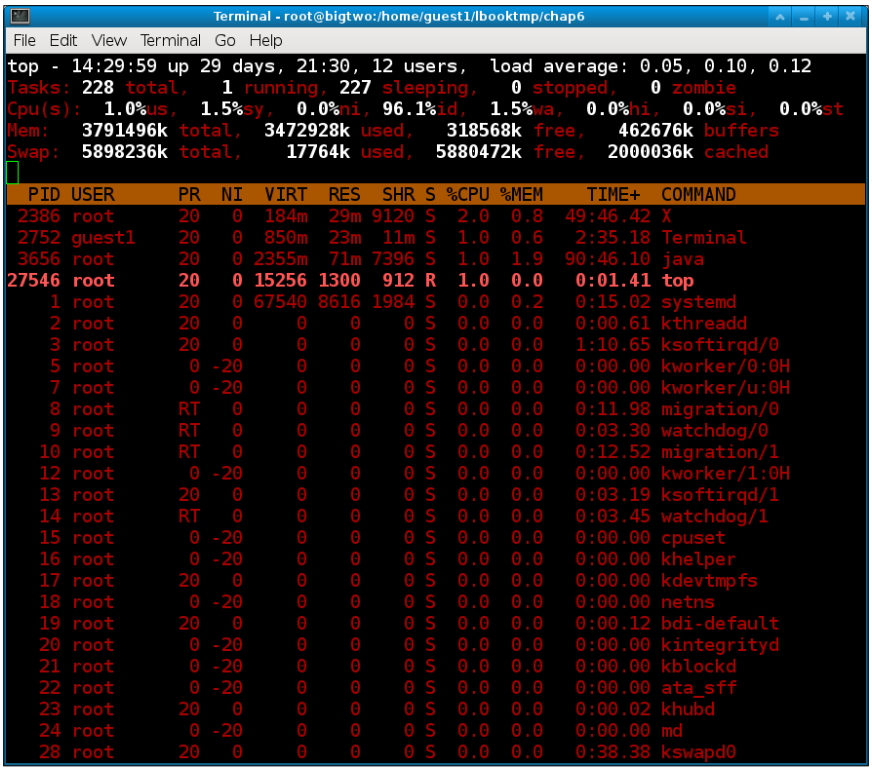
- `buddyinfo`: It contains data about nodes and memory
- `cgroups`: It contains data about CPU groups
- `cmdline`: It is the command line given to start the process
- `consoles`: It gives information about consoles
- `cpuinfo`: It has a very informative listing of the CPUs in your system
- `crypto`: It contains information about the crytographic routines available in the system
- `devices`: It has a list of the devices

- `diskstats`: It has a list of the disk statistics
- `dma`: It has a list of DMA
- `filesystems`: It gives a list of the filesystems available
- `interrupts`: It contains a very detailed listing of the interrupts being used by the system
- `iomem`: It gives the I/O memory information
- `ioports`: It gives the I/O port information
- `kallsyms`: It consists of a list of the OS symbols
- `kcore`: It represents the memory image of this machine
- `meminfo`: It contains a very detailed list of how memory is being used by the system
- `modules`: It contains a list of the modules used by the system
- `mounts`: It contains a list of the mounted filesystems (real and virtual) in the system
- `partitions`: It contains a list of the partitions
- `slabinfo`: It consists of a list of the slab memory objects
- `softirqs`: It is another IRQ listing
- `uptime`: It gives the amount of time the machine has been up (see the `uptime` command)
- `version`: It is the kernel version (see the `uname` command)
- `vmstat`: It gives the virtual memory statistics
- `zoneinfo`: It is another rather detailed memory listing

With a few exceptions, you can cat most of these files to get at important internal data. Note, do not cat (or do anything else with) the `kcore` file. Do not try to edit these files. Also, if you cat a file and nothing seems to happen, pressing *Ctrl* and *C* should make you back out.

Programs written in a language such as C can take advantage of the information in `/proc` to do some pretty cool things. For example, I have developed code that can allow a program to determine if it was run normally to the screen, or redirected to a file. The program can then take appropriate action, for example, clear the screen if run normally, and not clear it if redirected to a file. This way, control codes don't get embedded in the file. I have another C code that can determine the full path and filename of any file currently opened by the program.

The following is another screenshot of `top`:



This was created by running `top` and then pressing *Z*. You can customize `top` quite a bit. See the man page for more information.

7
Disks and Partitioning

In this chapter we will cover the following topics:

- Using `fdisk`
- Using `mkfs` to format a drive
- Using `fsck` to check the filesystem
- Logical Volume Management (LVM)

Introduction

When installing a machine, you can take the defaults or set up your disk(s) practically any way you want. Here is a brief review of partitions and filesystems. A typical Linux system usually has at least three separate partitions. They are labelled `/`, `/boot`, and `swap`. The `/` (pronounced *root*) is the parent of the directory structure. `/boot` is where the system kernel and map files reside. The `swap` partition is used when parts of memory need to be moved to the hard drive because of over-commitment. This is called being *swapped out* to disk.

The following are usually on separate partitions:

- `/`: It is the parent directory
- `/boot`: The boot and map files are located in here
- `(swap)`: It signifies the swap space

The following are traditionally on a separate partition, but do not have to be:

- `/home`
- `/tmp`
- `/etc`
- `/var`

Here is a partial list of the filesystems and directories and their general use, that are normally present on a Linux system. This is taken from Fedora 17 64-bit:

- `/`: It is the parent directory.
- `/boot`: The boot, map files, and system kernel are present in it.
- `(swap)`: It stands for the swap space.
- `/root`: It is the home directory for the superuser.
- `/home`: The user directories go here (for example, `/home/guest1`, which has been used throughout this book).
- `/tmp`: It is a directory that has the file deletion bit set. Temporary files can be placed here by users and various other programs. A `cron` job run by root cleans `/tmp` at periodic intervals (usually once a week).
- `/usr`: It consists of the operating system's (OS)'s parent user directory.
- `/usr/bin`: It has the OS's executable programs.
- `/usr/etc`: It contains the OS's configuration files.
- `/usr/games`: It consists of the games provided by the distribution.
- `/usr/include`: It includes files for programming languages such as C.
- `/usr/lib`: It contains the OS's library files.
- `/usr/lib64`: It consists of the OS's library files of 64 bit versions.
- `/usr/local`: The user programs can be placed here by the system admin for their use.
- `/usr/sbin`: It has the OS's executable programs used by the system.
- `/usr/share`: It includes the OS's shared programs and files.
- `/usr/src`: The OS's kernel source, include, and make files are placed here
- `/dev`: It represents the device directory. For example, the `/dev/sda` device points to the first SCSI hard drive.
- `/lib`: It is a symbolic link to `/usr/lib`.
- `/lib64`: It is a symbolic link to `/usr/lib64`.
- `/mnt`: It is used as a mount point.
- `/opt`: It represents the optional files.
- `/var/logs`: It consists of the OS's logs.
- `/var/spool`: It contains the printer files.
- `/var/run`: The OS keeps data on running programs here.
- `/run`: It is the symbolic link to `/var/run`.
- `lost+found`: It is where the OS keeps track of filesystem data.

- `/etc`: It is pronounced *etcetera* and stands for *everything else*. Configuration files tend to be located here.

The different types of virtual filesystems are as follows:

- `/proc`: The OS keeps track of processes here. See the previous chapter for more information.
- `/sys`: The OS keeps track of other processes in this directory.

Some other useful terms have been defined in the following list:

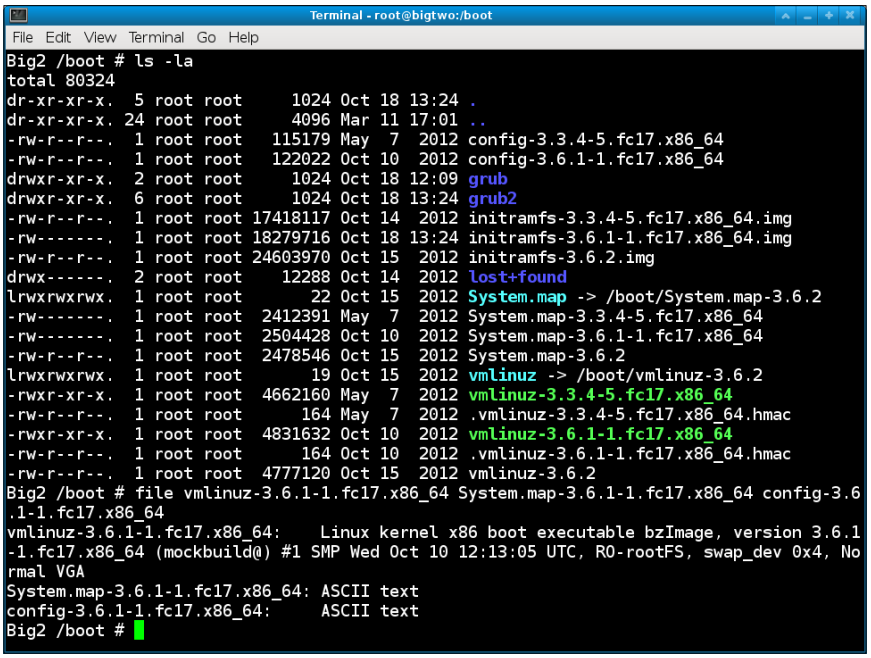
- **device**: It refers to the entire disk. For example, the first SCSI disk is normally named `/dev/sda`.
- **partition**: It is the device name followed by a number. The first SCSI partition would be `/dev/sda1`.
- **filesystem**: It defines the type of filesystem being used. Some examples are `ext2`, `ext3`, `ext4`, `vfat`, and `xfs`.
- **mount point**: It is the directory that points to the partition. The `/etc/fstab` file contains a table showing mount points and the partitions they are associated with. This file, which is created by a system installer such an Anaconda, can be manually edited by the superuser to add or delete devices and mounts.

The boot partition is where the operating system kernel and other startup files are located. Here is a description of the files found in `/boot`:

- `vmlinuz`: It represents a symbolic link to the kernel. For example, it points to `vmlinuz-2.6.35.6-45.fc14.x86_64` on Fedora 14.
- `initramfs`: It represents the initial RAM disk. On Fedora 14, it is named as `initramfs-2.6.35.6-45.fc14.x86_64.img`.
- `config`: This file is used to configure the kernel. We will see more on that in *Chapter 10, The Kernel*. On Fedora 14, it is named as `config-2.6.35.6-45.fc14.x86_64`.
- `map`: This is the system map file, which contains entry points into the various kernel routines. On Fedora 14, it is named as `System.map-2.6.35.6-45.fc14.x86_64`.

The initial RAM disk needs a bit more explanation. This file contains all the device drivers needed to get the kernel loaded and running. For example, the kernel needs to access the hard drive in order to boot up. If it needs a special driver for this (that is, if it can't be accessed by the BIOS alone), it must be located in the `initramfs` file or else it will not be able to complete the process. The `initramfs` file is created during system install, and can be modified by an experienced person. This is normally done when testing and/or using a new hardware. This is a compressed `GZIP` file that is uncompressed and placed on a ram (in memory) disk during boot up.

The following is a snapshot of `/boot` and the file command on my Fedora 17 system:



The `df` program is used to report the filesystem disk space usage statistics. The following output is a `df -h` listing from my Fedora 14 system. The `-h` parameter puts the output into a more human readable form (useful on large disks):

```
Big4 ~ # df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda3              97G   48G   45G  52% /
tmpfs                 5.9G  780K  5.9G   1% /dev/shm
/dev/sda1             485M   51M  409M  12% /boot
/dev/sda2             385G  124G  242G  34% /data
/dev/sda5              49G  8.3G   38G  19% /lewis
/dev/sdf1             932G  243G  690G  27% /megadrive
/dev/sdg1             7.3G  3.1G  4.3G  42% /usb
```

The first column shows the partition of the filesystem. The second shows the size of the partition. The third is the amount used followed by the amount still available. The `Use%` is the percentage used, and the last column is the mount point.

This output tells you a lot about how I set up this system. I did not make separate partitions for `/home`, `/tmp`, `/etc`, or `/var`. They are all in the same partition under `/`. I have the required `/boot` partition which I made larger than the default since I frequently build new kernels. There is a separate `/data` and `/lewis` partition. All the mentioned partitions are on the `/dev/sda` device. So what device and partitions are `/dev/sdf1` and `/dev/sdg1`? The first is an Iomega external USB drive and the second is an 8 GB **Universal Serial Bus** (**USB**) stick. These are used for backups, just in case the primary drive goes out.

The `fsck` program is used to check and optionally repair damaged filesystems. It can check multiple filesystems in parallel to speed up processing. If no parameters are given, `fsck` defaults to checking the filesystems in the `/etc/fstab` file.

When using partitions, you must keep in mind where a directory is mounted on. For example, suppose the space on `/` is getting tight and so you decide to delete some big files that are located in the `/tmp` directory. You need to first make sure that `/tmp` is indeed mounted on `/`, and not on its own partition. It is easy to make this mistake, so keep it in mind. This is even more of an issue if you are the system administrator for a lot of machines that are not all set up the same way.

Using fdisk

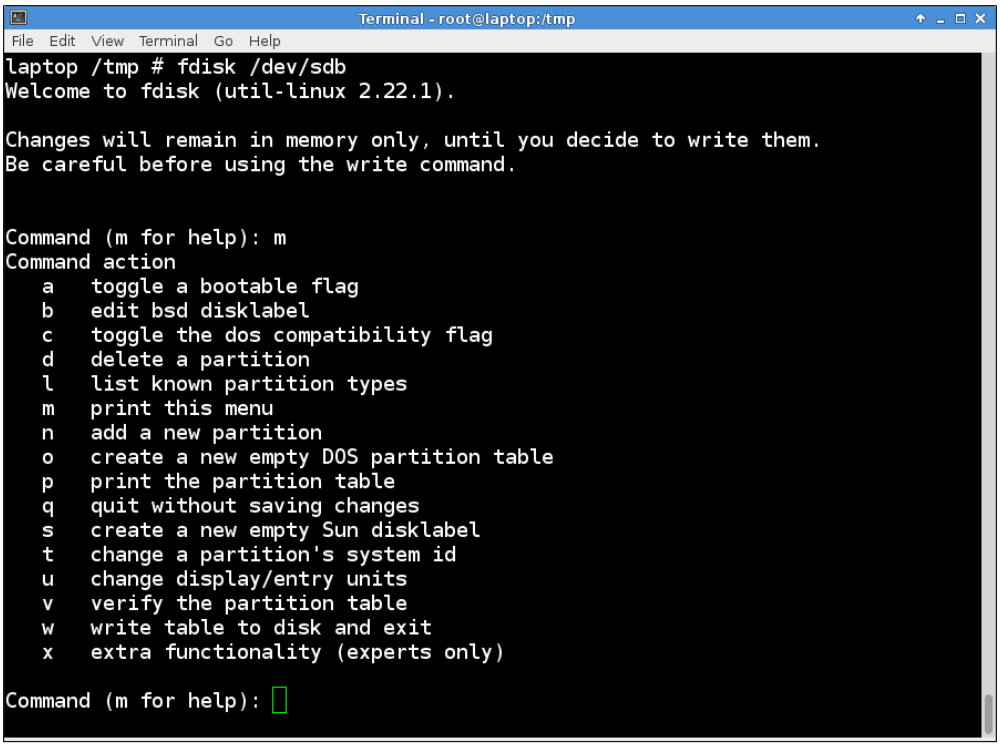
The `fdisk` program is used to manipulate the disk partition table. You can create, modify, and remove partitions with this utility.

Getting ready

You may follow along with these commands as long as you do not use the write table to disk action. However, just to be safe you may want to use a USB stick, or some other drive or system you don't care about.

The syntax for the interactive version of `fdisk` is `fdisk <device-name>`. I will be using a 4GB USB stick, `/dev/sdb` for this example. Remember that `fdisk` works with the entire device, so you do not use a partition number when starting the command. We will perform these actions as root.

The following is a screenshot showing `fdisk` on Fedora 18:



How to do it...

Carry out the following steps for running `fdisk`:

1. Start the command by running `fdisk` on the device:

   `fdisk /dev/sdb`

2. To display the Help menu: `enter an m`. It brings up the list of actions that can be performed on this device.
3. To display the partition table: `enter a p`. A list of the partitions on this device will be displayed.
4. In the case of my USB stick, there is only one partition. So let's delete it:

   `enter a d`

5. Since there was only one partition, it was deleted by default. Now let's add one:
   `enter an n`
6. Since this is the first partition, `press a p` for Primary.

7. Now run `enter a 1` and take the default starting sector.
8. We now have to enter a size. This can be done by the sector number or size. Let's use size and make it 1 GB:

   enter +1G

9. Run `enter a p` again to see what we have. You should see something like the following output:

```
Command (m for help): p
Disk /dev/sdb: 4009 MB, 4009754624 bytes, 7831552 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xc3072e18
   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048     2099199     1048576   83  Linux
Command (m for help):
```

10. This looks good. Now let's add another one and then take the defaults:

    enter n, then p, then 2, and the default first sector

11. For size, let's use 2 GB this time:

    enter +2G

12. To see what it looks like:

    enter a p

13. The table should now show two partitions. At this point, you could continue to make more partitions, or save the table. You can always make more partitions later if you have the required disk space.
14. For this example, we will quit without saving the changes:

    enter a q

15. To make sure nothing went wrong, run `fdisk /dev/sdb -l` to get a listing. It should show the same as when we started.

There's more...

You may have noticed some other actions available on the `fdisk` help screen. You can change the way the units are displayed, list the known partition types, and a few other things. For more information, consult the `fdisk man` page.

Using mkfs to format a drive

The `mkfs` command builds a filesystem on the device (similar to formatting a drive). It determines the filesystem type and calls the appropriate `mkfs.<type>` program. For example, if you were to run `mkfs -t ext4 /dev/sdb5`, it would actually run the `mkfs.ext4` program. The `mkfs` options are as follows:

- `-t`: It specifies the type of filesystem desired
- `-V`: It produces verbose output
- `-V`: It displays version information if used as the only parameter
- `-h`: It displays a help screen

Note that no other parameters are passed to the filesystem's specific program. Also note that in some cases, a size value needs to be provided.

Getting ready

The `mkfs` program will destroy any data currently residing in the partition. So, make sure you have taken the desired backups before proceeding with this command. Once this action has been taken, it cannot be reversed. If you would like to follow these steps, you should first practice on a USB stick or some other device you don't care about.

Be sure to unmount the partition before proceeding.

How to do it...

Carry out the following steps for formatting a partition in Linux:

1. For this example, I have inserted a USB stick (one I don't care about). To make sure we have the correct device, run `dmesg` and look at the last line:

   ```
   dmesg | tail -20
   ```

2. On my system, it shows the device is `/dev/sdh1` (substitute your device accordingly). It did not mount it, so I am ready to proceed. However, if yours is auto-mounted, you should be able to unmount it by running:

   ```
   umount /dev/sdh1
   ```

3. Note that the following command will destroy any data on this device. So, be sure you have backed up anything that want to keep before proceeding!

4. Run the command `mkfs -V -t vfat /dev/sdh1`. The following output is shown on my system:

   ```
   Big2 /temp # mkfs -V -t vfat /dev/sdh1
   mkfs (util-linux 2.21.2)
   mkfs.vfat /dev/sdh1
   mkfs.vfat 3.0.12 (29 Oct 2011)
   ```

5. Now mount the device: `mount /dev/sdh1 /usb2` (substitute your device and mount point as you feel appropriate).
6. Run `ls -la /usb2`. It should look something like the following output:

```
Big2 /temp # ls -la /usb2
total 8
drwxr-xr-x.  2 root root 4096 Dec 31  1969 .
dr-xr-xr-x. 25 root root 4096 Jun  7 10:04 ..
```

That's pretty much it for `mkfs`. Just be very careful when using this command and double check everything first before proceeding.

Using fsck to check the filesystem

In order to check and repair a Linux filesystem, the `fsck` program is used. In reality, `fsck` is just a frontend for the filesystem specific checker (similar to `mkfs`).

The `fsck` program can be run on different filesystems in parallel to speed up processing. Note that this feature can be disabled on low resource machines.

In general, the syntax for `fsck` is as follows:

```
  fsck [-lsAVRTMNP] [-C [fd]] [-t fstype] [filesys...]  [--] [fs-
specific-options]
```

Getting ready

The `fsck` program if used incorrectly, can corrupt the data currently residing on the partition. So make sure you have taken all the desired backups before proceeding with this command. Once this action has been taken, it cannot be reversed. If you would like to follow these steps, you may first want to practice on a USB stick you don't care about.

The device must be unmounted before running the command.

How to do it...

Carry out the following steps to check and run a partition:

1. For this example, I have inserted a USB stick (one I don't really care about). To make sure we have the correct device, run `dmesg` and look at the last line:

```
dmesg | tail -20
```

2. On my system, it shows the device is `/dev/sdh1` (substitute your device accordingly). It did not mount it, so I am ready to proceed. However, if yours is auto-mounted, you should be able to unmount it by running `umount /dev/sdh1`.
3. We can now run the `fsck` program. I first want to see what actions it will take and so, will use the `-N` option. Run `fsck -N /dev/sdh1`. The following output is shown on my system:

```
Big2 /home/guest1 # fsck -N /dev/sdh1
fsck from util-linux 2.21.2
[/sbin/fsck.vfat (1) -- /dev/sdh1] fsck.vfat /dev/sdh1
```

4. This looks good, so let's include a progress bar and run it for real: `fsck -C /dev/sdh1`. The following result is obtained on my system:

```
Big2 /home/guest1 # fsck -C /dev/sdh1
fsck from util-linux 2.21.2
dosfsck 3.0.12, 29 Oct 2011, FAT32, LFN
/dev/sdh1: 1743 files, 234381/977025 clusters
```

Since there were no problems on this USB stick, the output is not very exciting. I have seen some really bad errors in the past, and if this happens to you, I suggest getting the data off the drive as soon as possible, and then either reformatting the drive with `mkfs` or getting a new one just to be safe.

There's more...

The following is a brief list of the options for `fdisk` along with their functions:

- `-l`: It will lock the device before checking and can be used on only one device at a time.
- `-s`: It will serialize the file checking operations. It is useful when checking more than one filesystem at a time in the interactive mode.
- `-A`: It uses the entries in the `/etc/fstab` file to go through and check all the filesystems in one run. Typically, the root filesystem is checked first and then the others based on the values of the `passno` value. See the `man` page of `fstab` for more information on the available options.
- `-C`: It will display a progress bar.
- `-M`: It specifies not to check the mounted filesystems.
- `-N`: It will show what actions would be performed on this device but does not actually make them
- `-P`: It will check the root filesystem in parallel with the other systems. Don't ever use this option.

- `-R`: It will skip the root filesystem while checking the others.
- `-V`: It will enable verbose output and so its usage is recommended.

The following is a list of options usually supported by the filesystem specific program:

- `-a`: It will automatically repair the filesystem without any questions. Be very careful with this option as I have seen it go horribly wrong on more than one occasion.
- `-n`: This option is supposed to tell the specific checker to not perform any repairs. It is unreliable and so not recommended.
- `-r`: It will repair the filesystem interactively. Do not use this option if running `fsck` in parallel.
- `-y`: It tells some specific checkers to make repairs automatically.

In theory, a journaling filesystem, such as `ext3` or `ext4`, should not need a lot of checking or repair. If this is not the case on your system, I would suspect the hardware or maybe the hardware CMOS levels. Make sure everything in your system is flashed to the proper version.

In order to repair a filesystem, it must not be mounted. Also, checking a mounted filesystem may give bogus error messages sometimes.

Logical Volume Management (LVM)

LVM for Linux allows one to manage disks or arrays of disks as one large pool of storage. An LVM consists of one or more physical volumes along with one or more logical volumes.

Directories such as `/root` and `/home` for example, are located in a logical volume. Several commands exist to manage the LVM. Some operate on physical volumes, some operate on logical volumes, and some on both.

The following list can be used as a quick reference guide for the LVM commands:

- `pvcreate`: It initializes a disk or partition
- `pvchange`: It changes the allocation permissions of one or more physical volumes
- `pvck`: It checks the physical volume metadata
- `pvdisplay`: It displays the attributes of a physical volume
- `pvmove`: It moves physical extents
- `pvremove`: It removes a physical volume
- `pvresize`: It resizes a disk or partition
- `pvs`: It reports information about physical volumes
- `pvscan`: It scans all the disks for physical volumes

- `vgcfgbackup`: It backs up the volume group descriptor area
- `vgcfgrestore`: It restores the volume group descriptor area
- `vgchange`: It changes attributes of a volume group
- `vgck`: It checks the volume group metadata
- `vgconvert`: It converts the volume group metadata format
- `vgcreate`: It creates a volume group
- `vgdisplay`: It displays the attributes of volume groups
- `vgexport`: It makes volume groups unknown to the system
- `vgextend`: It adds physical volumes to a volume group
- `vgimport`: It makes exported volume groups known to the system
- `vgimportclone`: It imports and renames a duplicated volume group
- `vgmerge`: It merges two volume groups
- `vgmknodes`: It recreates a volume group directory and logical volume special files
- `vgreduce`: It removes the unused physical volumes from a volume group
- `vgremove`: It removes a volume group
- `vgrename`: It renames a volume group
- `vgs`: It reports information about volume groups
- `vgscan`: It scans all the disks for volume groups and rebuilds caches
- `vgsplit`: It splits a volume group into two
- `lvchange`: It changes the attributes of a logical volume
- `lvconvert`: It converts a logical volume from linear to mirror or snapshot
- `lvcreate`: It creates a logical volume in an existing volume group
- `lvdisplay`: It displays the attributes of a logical volume
- `lvextend`: It extends the size of a logical volume
- `lvmdiskscan`: It scans for all the devices visible to lvm2
- `lvmdump`: It creates lvm2 information dumps for diagnostic purposes
- `lvreduce`: It reduces the size of a logical volume
- `lvremove`: It removes a logical volume
- `lvrename`: It renames a logical volume
- `lvresize`: It resizes a logical volume
- `lvs`: It reports information about logical volumes
- `lvscan`: It scans all the disks for logical volumes

Getting ready

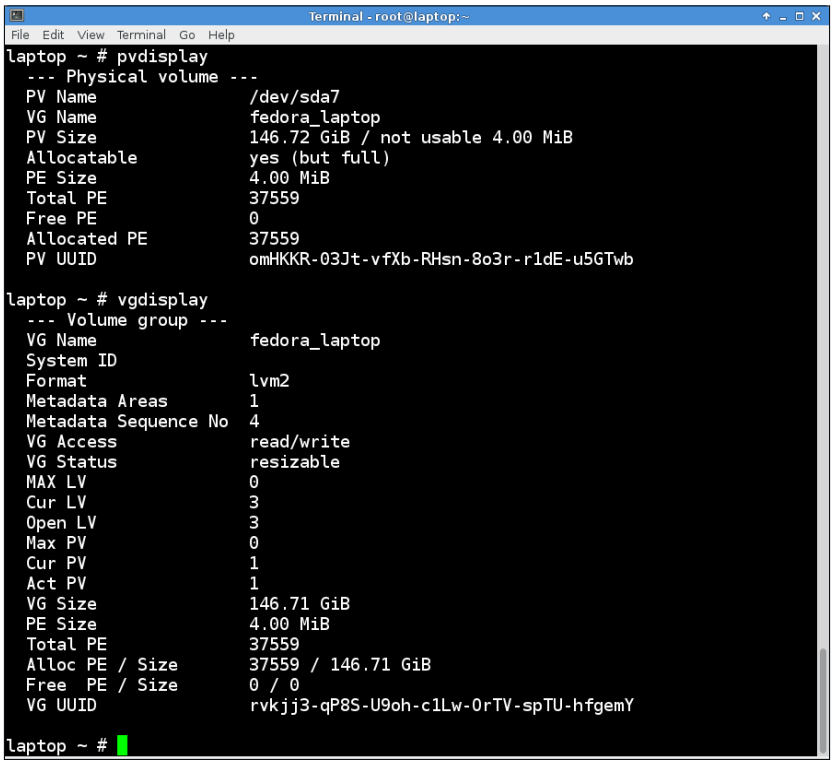
The following commands assume that you are running an LVM-aware system. Note that you do not need to actually have your drive in an LVM format in order to try the commands.

How to do it...

The following are some non-destructive commands you can try on your system (you will need to be in root to run these):

1. Let's see all the physical volumes on the system:
   ```
   pvdisplay
   ```
2. Now let's scan the disks:
   ```
   pvscan
   ```
3. Then, display the attributes of the volume groups:
   ```
   vgdisplay
   ```
4. We report information about volume groups:
   ```
   vgs
   ```
5. The disks for all the volume groups are then scanned:
   ```
   vgscan
   ```
6. Then the attributes for the logical volume (this one gets used a lot) need to be displayed:
   ```
   lvdisplay
   ```
7. Scan for all the devices visible to LVM:
   ```
   lvmdiskscan
   ```
8. Report information about the logical volumes:
   ```
   lvs
   ```
9. Scan all the disks for logical volumes:
   ```
   lvscan
   ```

The following is a screenshot of `pvdisplay` and `vgdisplay` on my Fedora 14 system:



There's more...

If you are accustomed to the traditional method of configuring disk drives, LVM can take a while to get used to. I have found the man pages to be pretty good in this area, and the Internet has some good articles on it as well.

Understanding different filesystems

At the time of writing this, there are over 60 different filesystems available for Linux. The following is a brief overview of the common ones you may frequently encounter:

- `Btrfs`: B-tree filesystem is a copy-on-write filesystem developed by Oracle in 2007. It is still in the developmental stage and considered experimental. The intention of `Btrfs` is to allow the filesystem to scale as drives get larger and larger. Because of its benefits, it is being considered as the filesystem to replace `ext4`.
- `ext2`: It was introduced in 1993. This filesystem does not have journaling and hence, has fewer writes per cycle, making it a good choice for flash drives.

- `ext3`: It is very similar to `ext2` with the addition of journaling, which improves reliability, especially after an unclean shutdown.
- `ext4`: Being released in 2008, it is considered the successor of `ext3`. It can handle volumes up to 1 exbibyte and files up to 16 tebibytes. This filesystem uses extents to replace the traditional block mapping scheme used by earlier versions of ext. It has enhancements allowing for faster filesystem checks (`fsck`).
- `FAT`: This is the abbreviation of File Allocation Table and the format was originally used by DOS, OS/2, and Windows. It is available in the kernel mainly to provide support for external devices such as USB flash drives.
- `ReiserFS` (or `Reiser3`): It is a general purpose filesystem with journaling, originally designed and written by Hans Reiser. When it was created, it contained many features that were not yet available at the time including journaling, online growth, and a scheme to limit internal fragmentation.
- `Reiser4`: It is the successor of `ReiserFS` (due to the conviction of the designer Hans Reiser for murder, future development is uncertain).

8
Working with Scripts

In this chapter we will cover the following topics:

- Removing text from a file
- Using script parameters
- Coding a loop in a script
- Backing up your system
- Locking a file for only one use at a time
- Getting introduced to Perl

Introduction

Knowing how to write scripts will allow you to run your system(s) much more efficiently. Script writing is easy; you don't need a degree in Computer Science or anything like that. Even better, everything you need to create and run scripts is already available on your Linux system.

The main thing to remember while programming is to think like the computer does. Try to understand what each step does. If you run into a problem, look at the line in question carefully. The computer will do exactly what you tell it to do, every time.

We will be writing `Bash` shell scripts in these examples. It is good programming practice to begin each script with a line indicating which shell is being used. This is done with a line that starts in the first column such as the following:

```
#!/bin/sh
```

Use a text editor to create scripts. I have seen people attempting to use a word processor for script writing but I don't recommend it. Word processors are cumbersome and always seem to insert bad characters into the file (even though they claim not to). If you don't already know by now, learn how to use `EMACS`, `vi`, or some other text editor (see *Chapter 3*, *Files and Directories*). You will be glad you did.

In order to execute the script as a command, you need to change the permissions on the file. For example, if you have created and saved a text file named `mycmd`, you can make it executable by running the following command:

```
chmod 755 mycmd
```

If you have not done much script writing, to be on the safe side, I would suggest using a user account when creating and running scripts. When you get comfortable (or if you already are), you can run as root. You will probably have to at some point, especially if you are a systems administrator.

Several script examples have been given in the following paragraphs. These should give a good idea of what can scripts be used for and how to write them.

Removing text from a file

A script can be written to do almost anything you can imagine. The following is one such example.

How to do it...

The following is the listing for a script that can be used to cut the line numbers off the examples. You can then run them as is on your system.

Script 1 – removing line numbers

```
  1  #!/bin/sh
  2  # removelines 5/1/2013
  3  if [ "$1" = "" ] ; then
  4   echo "Usage: removenumbers filename > newfile"
  5   echo " This script will remove the line numbers from the beginning
of the lines"
  6   echo " in filename. It assumes the number field is 5 characters
wide."
  7   echo " Use the redirection operator to save the output to a new
file."
  8   exit
  9  fi
  10  cat $1 | cut -c 1-5 --complement
```

How it works...

Let's discuss what the previous given lines do.

- The first line tells the OS what shell this script is for.
- Line 2 has a `#` in the first column. This is called a comment line. Here I have the date the script was written.
- The `$1` variable is the first parameter you gave after the name of the command. If no parameter is given, it will be equal to the empty string (`""`). The `Usage` text will be displayed and the script will exit at line 8.
- If a parameter was indeed given, this script assumes it's a filename and so, processing will continue with line 10.
- Line 10 uses the `cat` command to stream the contents of the file. In this case, the stream is being piped to the `cut` command.
- The `cut` command would normally keep the first five characters of each line and discard the rest. However, since I used the `--complement` flag, it does just the opposite. Many Linux commands have this option or something similar to it.

There's more...

If you would like to try this on your system, carry out the following steps:

1. Open a text editing session, for example `vi removelines.txt`.
2. Put `vi` into Insert mode by pressing *I*.
3. Copy the text of the script from the book and paste it into `vi`. Save the file and exit: `wq`.
4. Now run the following command:

   ```
   cat removelines.txt | cut -c 1-5 --complement > removelines
   ```

5. Then make it executable:

   ```
   chmod 755 removelines
   ```

6. Run the following command:

   ```
   removelines
   ```

If all went well, the usage screen should appear. You can now use this script on the rest of the examples in this chapter.

Using script parameters

Here we show how to count and show the number of parameters given to a script.

How to do it...

The following is the script listing:

Script 2 – parameters

```
1  #!/bin/sh
2  # 5/1/2013
3  echo Script showing how to count and show the parameters
4  N=$#
5  echo Number of parameters: $N
6  if [ $N -eq 0 ] ; then
7    echo "Usage: $0 parameters and/or a mask"
8    exit
9  fi
10  for i in $* ; do
11   echo Parm: $i
12  done
```

How it works...

- You already know what lines 1 to 3 do. The `$#` built-in variable contains the number of parameters that were given to the script. In line 4, we set the variable `N` to that value.
- Line 5 displays the value of `N`. Note that to access a variable, you must precede it with a `$` character.
- In line 6, we test `N` to see if any parameters were given. If not, we display the `Usage` message and exit at line 8.
- Line 10 is an example of a `for` loop. Note the syntax, and don't forget the `done` command at the end of the loop. Note that `$*` is a built-in variable that contains a list of all the parameters given to the script.
- In line 11, we display the value of the `i` variable.

There's more...

1. Make this an executable script on your system by following the example in the previous section with the only change in its naming as `parameters`.
2. Run it: `parameters` (you may need to run this as `./parameters` on your system).
3. You should see the `Usage` message. Now run it with this command:

```
parameters 1 34 56
```

4. It should display the number 3 and then each parameter on a line.
5. Try it with some other numbers. Also try some wildcard characters.

Variables require a bit more explanation. For example, the following are numbers:

```
i=1
RC=0
```

You test numbers in the following manner:

```
if [ $i -eq 0 ] ; then            # test for equal
if [ $RC -ne 0 ] ; then       # test for not equal
```

The following are strings:

```
FNAME=Jim
LNAME=Lewis
if [ "$FNAME" = "Jim" ] ; then     # test string equality
if [ "$LNAME" = "Lewis" ] ; then   # test string equality
```

This is another case where it helps to remember that it's backwards. Use letters such as `-eq` to test numbers, and `=` and other numeric operators to test strings.

Coding a loop in a script

Our previous scripts were commands that ran quickly to completion. Here is an example of a script that runs until you decide to terminate it. Note that if no parameters are required, a `Usage` section is probably not needed (but be sure to state what the script does in the comment section).

This script monitors the state of the network connection by pinging the provider once a minute. Failures are logged to a file.

How to do it...

The following is the program listing:

Script 3 - loops

```
1  #!/bin/sh
2  #
3  #  Check network once a minute and log failures to a file
4  PROVIDER=192.168.1.102
5  tput clear
6  while [ 1 ]
7  do
8    echo Written by Jim Lewis 2/21/2007
9    echo Pinging $PROVIDER
10    ping -c 1 $PROVIDER
11    rc=$?
12    if [ $rc -ne 0 ] ; then
13      echo Cannot ping $PROVIDER
14      date >> log1.txt
15      echo Cannot ping $PROVIDER >> log1.txt
16    fi
17    sleep 60
18  done
```

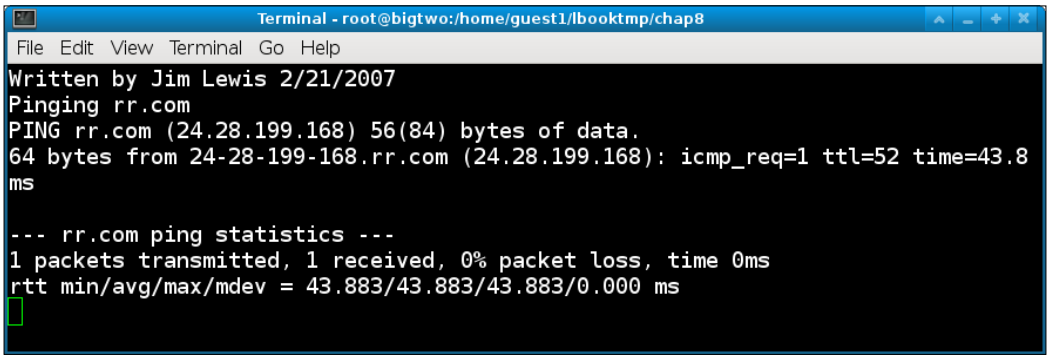
How it works...

- Line 4 is your provider's IP address or domain name. For this example, I used my `bigtwo` machine. Note that this variable could instead be placed in your `.bashrc` file (see *Chapter 1*, *Using the Terminal / Command Line*). That way you would only have to change it in one place if you get a new provider in the future.
- Line 5 clears the screen.
- Line 6 might look a little strange. Yes, that is what is known as an infinite loop. However, the script can still be terminated by pressing *Ctrl* and *C*, by issuing a kill command to it, or by having it watch for a file (more on that later).
- In line 7, remember that a `while` statement needs a statement after it.

- Line 10 uses the `ping` command. The `-c 1` variable tells it to ping only once.
- Line 11 saves the error code from the previous command.
- The `ping` command returns `0` if there are no errors. In case errors are present, line 14 appends the system date to the `log1.txt` file.
- Line 15 then appends the text to the log file.
- Line 17 waits for 60 seconds and then the loop starts the back up again at line 6.

I have been running this script since January 2007. I have used the `log1.txt` file on more than one occasion to convince my provider to improve their service.

The following is a screenshot of the script running on my Fedora 17 system:



There's more...

Carry out the following steps to run this script:

1. Create the file as in the previous sections. You may name it whatever you want, as long as it does not conflict with another script or program. I named mine `loop1`.
2. Open a terminal on your system. I suggest making it rather small as in the previous screenshot.
3. You do not need to be root to run this script. So, simply type the name you gave it to start it up.
4. Remember that this script is in a loop; it will run until you terminate it.
5. From time to time you should look at the `log1.txt` file to see how reliable your Internet connection is. Note that if your connection goes down a lot, this file may get very large.

Scripts such as this one are used quite frequently, especially when performing system administration duties to monitor the resources of the network.

One word of caution, use services such as `ping` with care. For example, do not ping your provider an excessive number of times. Once a minute is fine. When developing a script like this, use a local ping address for testing until you get it just right.

Backing up your system

Automating tasks is what makes scripts really powerful. You can spend some time getting a script just right and then let it do its thing. I easily have at least ten scripts running on each of my Fedora systems right now.

Here's what I call my *poor man's* backup utility. In the past, I have tried some *store-bought* programs only to be disappointed with the quality of the software. There's nothing worse than needing your backup files and then discovering there aren't any! This script is called by a `cron` job every night at 3 a.m. We cover cron in the next chapter.

If you want to try and use this script, be sure you understand what is going on. The backup directories must be created manually first, along with the `filenum1` file.

This script, as written, must run as root.

How to do it...

I use the following script to make a backup copy of my files:

Script 4 – making backups

```
1  #!/bin/sh
2  #   Jim's backup program
3  # Change to an appropriate directory on your system
4  cd /lewis/backup
5  VER="File backup by Jim Lewis 4/25/2011 A"
6  echo $VER
7  date >> datelog.txt
8  T=`cat filenum1`
9  T=`expr $T + 1`
10  if [ $T -gt 7 ] ; then
11   T=1
12  fi
13  echo $T > filenum1
14  TDIR=/temp/linuxbook/chap8/backups$T
```

```
 15  echo "Removing files from $TDIR"
 16  cd $TDIR
 17  pwd
 18  rm *.gz
 19  echo "Backing up files to $TDIR"
 20  X=`date "+%m%d-%H"`              # create a timestamp
 21  echo $X
 22  cd /
 23  tar -cvzf "$TDIR/lewis$X.gz" lewis
 24  tar -cvzf "$TDIR/temp$X.gz" temp
 25  tar -cvzf "$TDIR/root$X.gz" root
 26  cd /home
 27  tar -cvzf "$TDIR/guest$X.gz" --exclude=Cache --exclude=.cache
--exclude=.evolution --exclude=vmware --exclude=.thumbnails  --exclude=.
gconf --exclude=.kde --exclude=.adobe   guest1
 28  echo $VER
 29  cd $TDIR
 30  pwd
 31  ls -lah
```

How it works...

- Line 4 changes to my backup directory. You would want to change this on your system.
- Line 7 appends the current date to the `datelog.txt` file. I check this from time to time to make sure everything looks okay.
- Line 8 looks really weird; what are those backticks for? The `filenum1` file contains a number which is the next directory to copy the backup files to (7 in all). As you know, if you were to `cat` this file, it would display its contents on the screen. Well, the backticks mean to run this command, but place the resulting expression into a variable. This sounds complicated, but you will get used to it very quickly.
- The variable `T` now has the number of the next directory to use. Line 9 uses the `expr` command to evaluate the expression and add `1` to `T`.
- Line 10 checks to see if `T` is greater than `7`, and if so, sets it back to `1`. Yes, I make seven backups, one for each day of the week. Call me paranoid!
- Line 13 copies the new value of `T` back to `filenum1` for the next use.

- Line 14 sets up the `TDIR` variable, which contains the backup directory we are going to copy the files to.
- Line 16 changes to `TDIR`.
- Line 17 displays the current working directory. I had put this in when writing this script and got used to it being there, so left it in. This is completely optional.
- Line 18 removes the previous compressed backed up files. I strongly suggest you comment this line when first trying out this script. Then, after being sure that everything is okay, you can put it back in.
- Line 20 sets up a date timestamp. Again, the backtick operators are being used to put the value of date into a variable (`X` in this case).
- Line 22 changes to `/`.
- Lines 23, 24, and 25 back up the directories `/lewis`, `/temp`, and `/root`.
- Line 26 changes to `/home`.
- Line 27 tars up my `/home/guest1` directory. The parameters on `tar` tell it which directories to exclude from the archive. See the `tar man` page for more information.
- Line 28 displays the script version, line 29 goes back to the backup directory, line 30 displays the directory name, and line 31 shows the file listing.

This script is more of an example than a true backup program, but it's a good place to start from. The real one I use is similar to this, but also copies the files to an external USB drive and to another computer off-site (using unattended `scp`). Whenever I want to create a new Linux machine, I just copy these files to it and extract them, configure and source my `.bashrc`, and I am ready to run.

Locking a file for only one use at a time

This comes up often enough for me to mention it. There may be times when you want to edit a file in a terminal by running a simple script. However, if the file has already been opened in another terminal, there is the possibility that updates made in one session will get overwritten in the other. The following script should help prevent that from occurring.

How to do it...

Here is an easy way to prevent the same file from being edited by more than one terminal at the same time:

Script 5 – file locking

```
1  #!/bin/sh
2  # todo script 5/1/2013
3  FN1=/tmp/file.lock
```

```
4  if [ -f $FN1  ] ; then
5   echo -n "File is being used in tty: "
6   cat $FN1
7   exit
8  fi
9  echo `tty` > $FN1
10  # perform your actions(s) here
11  kw /lewis/todo.txt
12  rm $FN1
```

How it works...

- Line 3 sets the variable `FN1` to the name of the lock file.
- Line 4 checks to see if the lock file exists. If it does, it displays the contents of the lock file and exits the script.
- We get to line 9 if the lock file does not exist. The lock file is now created by redirecting the output of the `tty` command to the file.
- In line 11, you can perform whatever action(s) you desire. In my case, I edit `/tmp/todo.txt` here using `kw`, my text editor. Even after the editor session is closed, the processing of the script continues.
- Line 12 removes the lock file.

This is simple and works really well. I use a version of this script to edit several of my important files. The `todo` script can be run from anywhere on my system and the `todo.txt` file is displayed in my text editor. If I go off to another session or window later while the file is still being edited, and try to run `todo` again, it will not allow the edit. It also tells me if `tty` is in the original session. This can help me find it again more quickly, which is very useful as I always have a lot of workspaces and terminals open.

Getting introduced to Perl

Perl is a programming language that can be used for text manipulation, web development, network programming, system administration, development of GUIs, and a whole lot more. It was designed to be easy to use and efficient, and you can use either a traditional procedural or object oriented approach in your scripts. Perl also has a rather large list of third-party add-on modules that give it even more functionality.

Getting ready

In this section, we will cover just the very basics of Perl. Most typical Linux systems come with it and the documentation already installed. To see a brief introduction, run `perldoc perlintro` on your system. The Perl introduction should come right up.

In order to run a Perl script, you can use the following Perl command:

```
perl filename.pl
```

The `.pl` parameter is the usual extension given to Perl scripts. You can also place the path to Perl in the script similar to how we did with `bash`. First run `which perl` to see where Perl is located and then put that in as the first line of your script:

```
#!/usr/bin/perl
```

That's the correct path on Fedora. As with `bash` scripts, use a text editor to create them and use `chmod 755 filename.pl` to make it executable. Now, you can just type the name of the command to run it.

How to do it...

The following is a small Perl script I wrote when I was first starting out learning the language:

```
1  #!/usr/bin/perl
2  #  t1.pl - Perl practice script 1
3  use strict;
4  use warnings;
5  sub displaymessage
6  {
7   my $message = shift;
8   print "message: $message\n";
9  }
10  system("tput clear");
11  print "Practice script 1 5/4/2013\n";
12  my $name = "Jim";
13  print "name: $name\n";
14  my @numbers = (23, 42, 69, 71);
15  print "numbers 0: $numbers[0]\n";
16  print "Last element is: $numbers[$#numbers]\n";
17  if($numbers[0]==23)
```

```
 18  {
 19   print "numbers 0 is equal to 23\n";
 20  }
 21  my $count = 0 ;
 22  foreach (@numbers)
 23  {
 24   print "Element $count is  $_\n";
 25   $count++;
 26  }
 27  print "Opening input.txt ...\n";
 28  open(my $in,  "<",  "input.txt")  or die "Cannot open input.txt for
reading: $!";
 29  print "Opening output.txt ...\n";
 30  open(my $out, ">",  "output.txt") or die "Cannot open output.txt
for writing: $!";
 31  while (<$in>)
 32  {
 33   print "line $_";
 34   print $out $_;
 35  }
 36  close $in or die "$in: $!";
 37  close $out or die "$out: $!";
 38  displaymessage("Type some keys. Press 'c' to clear the screen and
'q' to quit:");
 39  my $key = 1;
 40  $count = 1;
 41  while($count < 500)
 42  {
 43  # read a key from the keyboard
 44   open(TTY, "+</dev/tty") or die "no tty: $!";
 45   system "stty  cbreak </dev/tty >/dev/tty 2>&1";
 46   $key = getc(TTY);
 47   if($key eq 'c')
 48   {
 49    system("tput clear");
 50   }
```

```
51   if($key eq 'q')
52   {
53    print "\nEnding the script\n";
54    $count = 10000;
55   }
56   $count++;
57  }
```

How it works...

- Line 1 tells the shell which interpreter to use.
- Line 3 tells the compiler to be rather severe with the syntax. Always use this line.
- Line 4 says to show all warnings. Always use this too.
- Lines 5 to 9 are a subroutine. Line 7 puts the first parameter to the subroutine into the variable message. Line 8 displays it.
- Line 10 clears the screen.
- Line 11 tells us what this script is supposed to be. Note the `\n` parameter at the end of the line. This stands for newline, and signifies continuing the next print output on to the next line.
- Line 12 sets the variable name to `Jim`. Note the `$` symbol in front of name. Don't forget it.
- Line 13 displays the variable.
- Line 14 creates an array of numbers. Note the `@` character.
- Line 15 displays the first element in the array.
- Line 16 displays the last element by using the special Perl syntax of `$#numbers`.
- Line 17 checks to see if element `0` really is equal to `23` or not. Note the double equal sign.
- Line 21 creates the variable `count` and sets it to `0`.
- Line 22 is another special Perl convention. The `foreach` command means to traverse through the array one element at a time. This is a very handy feature.
- Line 24 shows how the special `$_` construct works to display each element of an array.
- Line 25 increments the count by `1`.

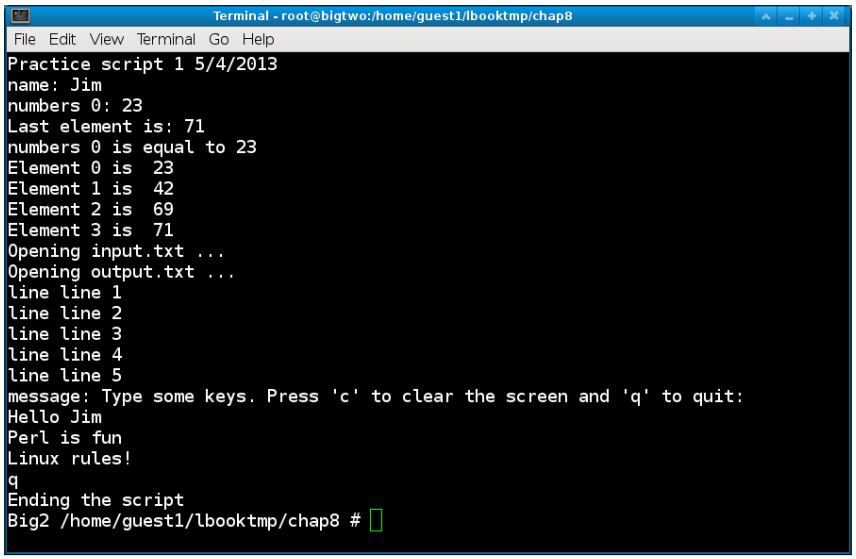
- The next lines are a way to handle files in Perl. It attempts to open the `input.txt` file for reading and `output.txt` for writing. The `input.txt` file must exist or else the program will error out and stop right here. The contents of `input.txt` are as follows:

```
line 1
line 2
line 3
line 4
line 5
```

- Line 30 opens the output file.
- Lines 31 through 35 read each line from `input.txt` and write it to `output.txt`.
- Lines 36 and 37 close the files. Don't forget this step in your scripts.
- Line 38 uses the subroutine we created earlier to display a message.
- Line 39 creates and sets the `key` variable.
- Line 40 resets the `count` variable back to `1` again.
- Line 41 starts a `while` loop.
- Lines 43 through 46 are a way to read a key from the keyboard in Perl. If this looks convoluted to you, I couldn't agree more.

The remaining lines are self-explanatory. When the user presses `q` to quit, the `count` variable is increased to `10000`. This causes the `while` loop at line 41 to finish early, effectively ending the script. You could have instead used a loop control variable here.

The following is a screenshot of the previous script running on my system:



There's more...

Although this is an extremely trivial script, you can do an incredible amount of things with Perl. Many years ago, when I was first starting out with home computers, there was a text-mode game we could play based on Star Trek. I obtained a copy of this game in BASIC and eventually rewrote it in C. A few months ago, I decided to rewrite it again, this time in Perl. It's not done yet, probably because I am too busy writing this book, but it is getting there.

Good books on Perl should be easy to locate. I have bought a few and find them to be very valuable as I attempt to learn it in more detail.

There are several hundred standard Linux utilities that can be used standalone or in scripts. The following is a short list of some of the most common ones. Whenever you need to perform any task, check here to see if you can incorporate some of the given commands into your script.

The following utilities are located in `/bin`:

- `awk`, `gawk`: These are used for pattern scanning and processing language
- `basename`: It is used to strip directory and suffix from filenames
- `bash`: It is a GNU bourne-again shell
- `cat`: It is used to concatenate files and print on the standard output
- `chmod`: It is used to change file mode bits
- `chown`: It is used to change the file owner and group
- `cp`: It is used to copy files and directories
- `cut`: It is used to remove sections from each line of files
- `date`: It is used to print or set the system date and time
- `dmesg`: It is used to print or control the kernel ring buffer
- `echo`: It is used to display a line of text
- `find`: It is used to search for files in a directory hierarchy
- `grep`, `egrep`, `fgrep`: These are used to print lines matching a pattern
- `hostname`: It is used to show or set the system's host name
- `ls`: It is used to list the directory contents
- `mkdir`: It is used to make directories
- `mktemp`: It is used to create a temporary file or directory
- `mv`: It is used to move (rename) files
- `ping`, `ping6`: These are used to send `ICMP ECHO_REQUEST` to network hosts
- `ps`: It is used to report a snapshot of the current processes

- `pwd`: It is used to print the name of the current/working directory
- `rm`: It is used to remove files or directories
- `sed`: It is used to invoke the stream editor for filtering and transforming text
- `sleep`: It represents the delay for a specified amount of time
- `sort`: It is used to sort the lines of text files
- `tar`: It is used to combine and optionally compress files together into a single archive
- `touch`: It is used to change file timestamps

The following utilities are located in `/usr/bin`:

- `diff`: It is used to compare files line by line
- `dirname`: It is used to strip the last component from the filename
- `expr`: It is used to evaluate expressions
- `file`: It is used to determine the file type
- `flock`: It is used to manage locks from shell scripts
- `stat`: It is used to display file or filesystem status
- `tee`: It is used to read from the standard input and write to standard output and files
- `time`: It is used to time a simple command or give the resource usage
- `tty`: It is used to print the filename of the terminal connected to standard input
- `uniq`: It is used to report or omit repeated lines
- `unzip`: It is used to list, test, and extract compressed files in a `ZIP` archive
- `who`: It is used to show who is logged on
- `xargs`: It is used to build and execute command lines from the standard input

The following commands are built into the shell:

- `cd`: It is used to change the directory
- `echo`: It is used to display a line of text
- `exit`: It causes the shell to exit (with an optional return code)
- `export`: It is used to set an environment variable
- `kill`: It is used to send a signal to or terminate a process
- `read`: It is used to get a string from the keyboard and place it into a variable

9
Automating Tasks Using Cron

In this chapter we will cover:

- Creating and running a crontab file
- Running a command every other week
- Reporting the errors from a crontab file

Introduction

The cron daemon, which is usually started automatically by the OS, looks at all of the crontab files once every minute. If the criterion has been met, the command is run. In this chapter we show how to create and maintain your crontab files using the crontab program.

Depending on how your system is set up, cron jobs are permitted (allowed or not allowed) based on the user. The files that control this are in `/etc` and are named `cron.allow` and `cron.deny`. These are explained in the following section:

- `cron.allow`: If this file exists, then the user must be listed in it, in order to use crontab
- `cron.deny`: If `cron.allow` does not exist but `cron.deny` does exist then the user must not be listed in the `cron.deny` file

If neither of the file exists, then only the root user can use the command. In most Linux systems only `cron.deny` exists and it is empty. Check this on your system before running the following commands.

We will use a user account to experiment with crontab. The `crontab` command is used to make changes to the crontab file, as it should not be edited directly. To view the crontab for the current user, run `crontab -l`.

The command to edit the crontab file for the current user is `crontab -e`. By default, this will bring up the crontab file in the vi editor. However, you can change the `EDITOR` environment variable to use whatever text editor you wish.

The following is the format of a typical user crontab file:

```
#
#  crontab for jklewis
# Field 1   2    3            4              5             6
#      Min Hour Day of month Month of year  Day of week
#     0-59 0-23 1-31         1-12           0-6   0=Sun /path/command
#
# Days of the week: 0=Sun 1=Mon 2=Tues 3=Wed 4=Thu 5=Fri 6=Sat
```

I always add this template to the top of the crontab file so I can easily remember what the fields are. The following is an example crontab entry:

```
        10     0     *     *     0     /path/mycommand
```

These values can be integers, a range, an element or list of elements, or an asterisk. An asterisk means to match all valid values.

- `#`: This indicates a commented line. It must be the first thing on the line (don't put it at the end of a line).
- **Field 1**: This is the minute. It starts at 0 meaning 00:00.
- **Field 2**: This is the hour. It starts at 0 which means 12:00 a.m.
- **Field 3**: This is the day of the month.
- **Field 4**: This is the month of the year. Names can also be used.
- **Field 5**: This is the day of the week. It starts at 0, which is Sunday. Names can also be used.
- **Field 6**: This is the path/command to run.

This example entry would run `mycommand` at 12:10 a.m. on every Sunday.

Use white space only to separate the fields. Do not use tabs.

Creating and running a crontab file

Here we will show an example of creating and running a user crontab file.

Getting ready

Be sure your system is set up as outlined in the *Introduction*. We will be using two terminal sessions, to see our results more easily.

How to do it...

The following is an example of how to create and run a crontab file:

1. In a terminal session run `tty` and remember the output. This will be used in step 10.
2. Open or use another terminal under a user account. I'll be using `jklewis` as in earlier chapters.
3. Let's view the crontab file by running the following command:

   ```
   crontab -l
   ```

4. It may say something like `no crontab for jklewis` which is fine.
5. Now let's make one by running the following command:

   ```
   crontab -e
   ```

6. It may say something like `no crontab for jklewis - using an empty one` which is good.
7. Crontab should bring up a temporary file in vi (unless you have changed the `EDITOR` variable as I have, on my system). The file will not be used until it has been saved and the session has been ended.
8. I suggest cutting and pasting the template I created above to your crontab file. It will make it much easier to remember the fields.
9. Now let's add an entry that we can see takes effect quickly. Insert the following lines into the file (cut and paste should work):

   ```
   TTY=/dev/pts/17
   # crontab example 1
   *   *   *    *    *     date > $TTY; echo "Yes it works" > $TTY
   ```

10. Change the `TTY` line to what you found in step 1.
11. Now save the file and quit the session. You should see a message such as the following:

    ```
    crontab: installing new crontab
    ```

12. In the next minute you should see something like the following output in your other session:

```
Tue May  7 19:52:01 CDT 2013
Yes it works
```

13. The entry we just made with all of those asterisks means to run the command every minute. Edit the crontab again and change the line to the following code:

```
*/5    *   *   *   *   date > $TTY; echo "Every 5 minutes"  > $TTY
```

14. That strange looking syntax is a way to skip, or increment, a value. Now try the following command:

```
35     9     *     *     1-5   date > $TTY; echo "9:35 every week
day"  > $TTY
```

15. That runs every weekday (Mon - Fri) at `9:35` a.m. The `1-5` in the day field is a range.
16. Names can also be used for the day and month fields, as in the following command:

```
17   13    8     May   Wed   date > $TTY; echo "May 8 at  13:17"
> $TTY
```

17. And also the following command:

```
0   0    *     *      Fri   date > $TTY; echo "Run every  Friday
at 12:00 am"  > $TTY
```

With names, the standard three letter abbreviations are used, and the case does not matter. Using names might be easier, however, you cannot use ranges or steps with names.

Running a command every other week

Now that we have looked at the basics of cron, how would you set up an entry to run a command every other week? You may be tempted to try something like the following code:

```
*  *  *  *  0/2   /path/command
```

That means to start on Sunday, and then run every other Sunday, right? No, this is wrong, but you often see this given as a solution on websites. Cron doesn't actually have a built-in way to do this, but there is a work-around.

How to do it...

The following are the steps to run a command every other week:

1. Create the following script in your home directory and name it `cron-weekly1` (feel free to cut and paste):

```
#!/bin/sh
# cron-weekly1
# Use this script to run a cron job every other week
FN=$HOME/cron-weekly.txt
if [ -f $FN  ] ; then
  rm $FN
  exit
fi
touch $FN
echo Run the command here
```

2. Make the script executable by running the following command:

```
chmod 755 cron-weekly1
```

3. Under your user account run by running the following command:

```
crontab -e
```

4. Add the following line:

```
0  0  *  *  0    $HOME/cron-weekly1
```

5. End your editing session. This will install the modified crontab file.

How it works...

Take a look at this script. The first time it is run, the `cron-weekly.txt` file does not exist, and so it is created, and the command (the one you want to run every other week) is executed. The next week, when this script is run again, it sees that the `cron-weekly.txt` file is there, deletes it, then exits the script (the command is not run). This alternates every week, effectively running the command every other week. Pretty cool, huh?

In the preceding `cron-weekly1` script, the command is executed the first time the script is run. You can change this to run the command starting with the next week by moving the line to run the command inside the `if` statement.

Although it might be very tempting to do so, do NOT put a comment # at the end of a line. Cron cannot tell if it's a comment or part of a command. If cron ever reports some errors you don't understand, check for comments that are in the wrong place. Yes, I admit I still do this from time to time.

If you do something crontab doesn't like (such as the `* * * * 0/2 command` line shown in the preceding section), it will usually report an error when you close the session. It will then give you the option to re-edit the file. By all means do so, and either fix the problem, or at least comment the line. You can go back and edit it again later.

You can remove a crontab file completely by running `crontab -r`. I would suggest making a backup copy before doing this, just in case. You should be able to save the file to a new name through whatever text editor you have chosen to use.

There's more...

Crontab files can make use of environment variables. The following are a few common ones:

- **SHELL**: This tells the OS to use a particular shell, overriding what is in the `/etc/passwd` file. For example, `SHELL=/bin/sh`.
- **MAILTO**: This tells cron to mail errors to this user. The syntax is `MAILTO=<user>` that is `MAILTO=jklewis`.
- **CRON_TZ**: This is used to set a particular timezone variable, that is `CRON_TZ=Japan`.

Cron has some shortcuts you can use. These are used in place of the 5 time and date fields, and are as follows:

Shortcuts for command	Command	Output
`@reboot`		Run once after reboot
`@yearly or @annually`	`0 0 1 1 *`	1st day of the month on the 1st day of the year
`@monthly`	`0 0 1 * *`	1st day of the month
`@weekly`	`0 0 * * 0`	Run on Sunday at 12:00 am
`@daily`	`0 0 * * *`	Run every day at 12:00 am
`@hourly`	`0 * * * *`	Run every hour on the hour

So in the preceding example we could have put `@weekly $HOME/cron-weekly1`.

Don't use cron for any time sensitive tasks. There is usually a short delay, just a few seconds, before the command is run. If you need better granularity than that, use a script and the sleep routine.

You can also set up a crontab for root. To see that and more, use `man -a crontab`.

Reporting the errors from a crontab file

You may be wondering, if there is an error in a crontab file how does the computer report it? It does this by using the sendmail system, to mail the crontab user.

How to do it...

The following is an example of how errors are reported by cron:

1. Open a terminal with a user account.
2. Edit your crontab file by running the following command:

   ```
   crontab -e
   ```

3. Now let's deliberately cause an error. Scroll to the bottom and add the following line:

   ```
   * * * *    date > /baddirectory/date.txt
   ```

4. Save the file. Wait until cron runs in the next minute, and then press *Enter* in your user terminal.
5. You should see a message saying you have mail. Run the following command:

   ```
   mail
   ```

6. There should be a mail indicating the error (in this case, file not found). You may delete the message by pressing *D* and then *Q* to leave the mail.
7. To finish, be sure to re-edit your crontab file and remove the bad line we just added.

There's more...

You can also monitor the `/var/log/cron` file to see what cron is doing on the system throughout the day. This is very helpful when first creating a crontab file and trying to get it just right.

10
The Kernel

In this chapter we will cover the following topics:

- A brief look at module commands
- Building a kernel from kernel.org
- Using xconfig to modify the configuration
- Working with GRUB
- Understanding GRUB 2

Introduction

The kernel is the main component, or the heart of an operating system. It controls all of the resources, timings, interrupts, memory allocation, process separation, error handling, and logging in the system. In a typical Linux computer, the kernel is modular, in that it has a core file (or files) and then loads the other device drivers as needed. In some cases, say an embedded device, the kernel may consist of one big image with all of the drivers it needs contained inside a file. This is known as a monolithic kernel.

Before deciding whether you need to build a custom kernel, you should first make sure you really do need one. Here are some of the pros and cons to running a custom kernel.

The following are the pros to running a custom kernel:

- If you know what you are doing and have the time to research it, you can end up with a kernel that gets the most out of your hardware
- You can take advantage of the features or devices that the stock kernel might not have
- By going through all of the kernel settings you gain a better understanding of Linux
- Building and running your own kernels is just plain fun

The following are the cons of running a custom kernel:

- Your own custom kernel might not contain features needed by your distribution
- VMware, and other virtual environments will probably not work without additional effort
- Be aware that if you run your own kernel, you will most likely no longer be supported by your distribution support channel

Most of these cons of running your own kernel can be worked around and/or solved. It just depends on how much time you have to spend on getting it right.

A brief look at module commands

There are several commands that are used to manipulate the modules on your system. Note that depending on your distribution, these commands may only be run as root.

How to do it...

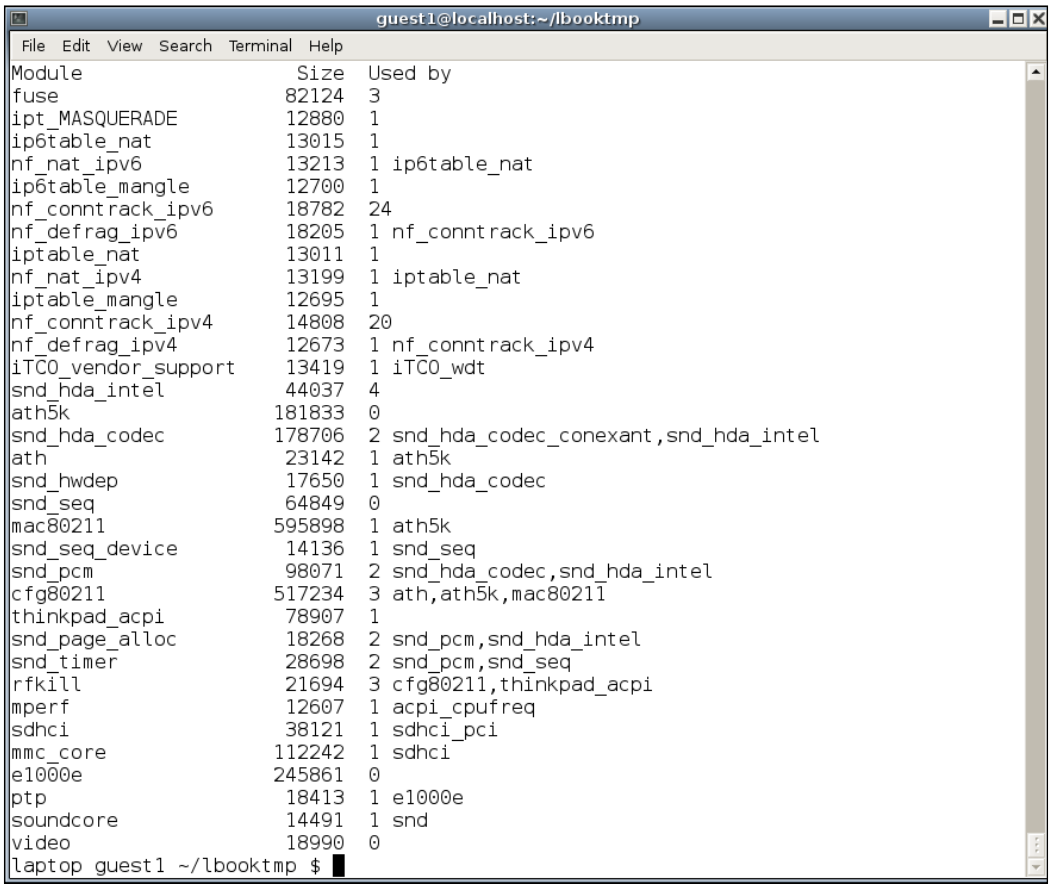
In the following steps, we will run the `lsmod`, `modprobe`, `insmod`, and `modinfo` commands:

1. To see the status of the modules currently loaded on the system, run `lsmod`.
2. To load a module from the current `/lib/modules/<kernel name>` directory, you would use the `modprobe` command. An example would be `modprobe pcnet32`.
3. To load a module directly, use the `insmod` command. An example would be `insmod /temp/pcnet32.ko`.
4. To display the information about a module, use the `modinfo` command. Try this on your system by first running `lsmod` to find a module and then `modinfo` on one of the names `modinfo video`.

How it works...

The `lsmod` command takes the contents of the `/proc/modules` file and displays it in a nice, easy to read format. Use it to determine what modules are loaded in the system.

The following screenshot shows a partial listing of `lsmod` from my Fedora 19 system:



The `modprobe` command is used to add and remove modules from the Linux kernel. It loads the module from the current `/lib/modules/<kernel name>` directory. The `modprobe` command does a lot more than `insmod`, such as load more than one module at a time to resolve dependencies, and is generally preferred over `insmod`. Since `modprobe` can load multiple modules, the files in `/etc/modprobe.d` and the `/etc/modules.conf` file are used to resolve any issues.

The `insmod` command can be used to insert a module into the system. It is normally used to load a module directly. For example, if you wanted to load a newly created version of the `pcnet32` module, you would first change the directory to the proper directory and then run `insmod pcnet32.ko`.

The `modinfo` command shows information about a Linux kernel module. It is a very useful command that allows you to see the details about a particular module, such as what parameters it will accept. The following is what the output from `modinfo` looks like on my Fedora 17 system:

```
BIG2 /temp/linux-3.9.1 # modinfo nouveau
filename:       /lib/modules/3.6.1-1.fc17.x86_64/kernel/drivers/gpu/drm/
nouveau/nouveau.ko
license:        GPL and additional rights
description:    nVidia Riva/TNT/GeForce
author:         Stephane Marchesin
alias:          pci:v000012D2d*sv*sd*bc03sc*i*
alias:          pci:v000010DEd*sv*sd*bc03sc*i*
depends:        drm,drm_kms_helper,ttm,mxm-wmi,i2c-core,wmi,video,i2c-
algo-bit
intree:         Y
vermagic:       3.6.1-1.fc17.x86_64 SMP mod_unload
parm:           agpmode:AGP mode (0 to disable AGP) (int)
parm:           modeset:Enable kernel modesetting (int)
parm:           vbios:Override default VBIOS location (charp)
parm:           vram_pushbuf:Force DMA push buffers to be in VRAM (int)
parm:           vram_notify:Force DMA notifiers to be in VRAM (int)
parm:           vram_type:Override detected VRAM type (charp)
parm:           duallink:Allow dual-link TMDS (>=GeForce 8) (int)
parm:           uscript_lvds:LVDS output script table ID (>=GeForce 8)
(int)
parm:           uscript_tmds:TMDS output script table ID (>=GeForce 8)
(int)
parm:           ignorelid:Ignore ACPI lid status (int)
parm:           noaccel:Disable all acceleration (int)
parm:           nofbaccel:Disable fbcon acceleration (int)
parm:           force_post:Force POST (int)
parm:           override_conntype:Ignore DCB connector type (int)
parm:           tv_disable:Disable TV-out detection (int)
parm:           tv_norm:Default TV norm.
                Supported: PAL, PAL-M, PAL-N, PAL-Nc, NTSC-M, NTSC-J,
hd480i, hd480p, hd576i, hd576p, hd720p, hd1080i.
                Default: PAL
```

```
                *NOTE* Ignored for cards with external TV encoders.
(charp)
parm:           reg_debug:Register access debug bitmask:
                0x1 mc, 0x2 video, 0x4 fb, 0x8 extdev, 0x10 crtc, 0x20
ramdac, 0x40 vgacrtc, 0x80 rmvio, 0x100 vgaattr, 0x200 EVO (G80+) (int)
parm:           perflvl:Performance level (default: boot) (charp)
parm:           perflvl_wr:Allow perflvl changes (warning: dangerous!)
(int)
parm:           msi:Enable MSI (default: off) (int)
parm:           ctxfw:Use external HUB/GPC ucode (fermi) (int)
parm:           mxmdcb:Santise DCB table according to MXM-SIS (int)
```

The `rmmod` command allows you to remove a loaded module from the Linux kernel. The usual syntax is `rmmod modulename`. The extension is not used. You can also use the `modprobe -r` command.

The `depmod` program generates the `modules.dep` and `.map` files. It does not normally need to be executed manually by the user as it is run during the kernel build. It creates a list of module dependencies by examining the modules in `/lib/modules/<kernelname>` and determining what symbols they need and which symbols they export.

Some of these commands have a force option available. It will attempt to perform the desired function, bypassing any checks. I have never seen this work reliably and so do not recommend its use. If you do decide to try it, make sure you have a complete OS backup available.

When running device driver commands, in many cases more information is available by looking at the `/var/log/messages` file. I suggest opening a terminal and running `tail -f /var/log/messages` in it. Keep this terminal where you can see it at all times. Also note that the file will eventually be recycled, so the command will have to be stopped and started again (about once a week on my systems). A simple test is to run `logger hellojim`. If you don't see that show up, then it's time to restart the tail session again.

You can also run the `dmesg` command. The following is an abbreviated example of the output of `dmesg` on Fedora 17:

```
Linux version 3.6.1-1.fc17.x86_64 (mockbuild@) (gcc version 4.7.2
20120921 (Red Hat 4.7.2-2) (GCC) ) #1 SMP Wed Oct 10 12:13:05 UTC 2012
Command line: BOOT_IMAGE=/vmlinuz-3.6.1-1.fc17.x86_64 root=/dev/mapper/
vg_bigtwo-lv_root ro rd.md=0 rd.dm=0 SYSFONT=True rd.lvm.lv=vg_bigtwo/lv_
swap KEYTABLE=us rd.lvm.lv=vg_bigtwo/lv_root LANG=en_US.UTF-8 rd.luks=0
rhgb quiet
smpboot: Allowing 4 CPUs, 2 hotplug CPUs
Booting paravirtualized kernel on bare hardware
```

```
Kernel command line: BOOT_IMAGE=/vmlinuz-3.6.1-1.fc17.x86_64 root=/dev/
mapper/vg_bigtwo-lv_root ro rd.md=0 rd.dm=0 SYSFONT=True rd.lvm.lv=vg_
bigtwo/lv_swap KEYTABLE=us rd.lvm.lv=vg_bigtwo/lv_root LANG=en_US.UTF-8
rd.luks=0 rhgb quiet

Memory: 3769300k/5242880k available (6297k kernel code, 1311564k absent,
162016k reserved, 6905k data, 1032k init)

Console: colour dummy device 80x25

tsc: Fast TSC calibration using PIT

tsc: Detected 2699.987 MHz processor

CPU: Processor Core ID: 0

CPU0: Thermal monitoring enabled (TM2)

smpboot: CPU0: Intel Pentium(R) Dual-Core  CPU      E5400  @ 2.70GHz
stepping 0a

NMI watchdog: enabled on all CPUs, permanently consumes one hw-PMU
counter.

smpboot: Booting Node   0, Processors  #1

smpboot: Total of 2 processors activated (10799.94 BogoMIPS)

atomic64 test passed for x86-64 platform with CX8 and with SSE

NET: Registered protocol family 38

Block layer SCSI generic (bsg) driver version 0.4 loaded (major 252)

Console: switching to colour frame buffer device 80x30

fb0: VESA VGA frame buffer device

input: Power Button as /devices/LNXSYSTM:00/device:00/PNP0C0C:00/input/
input0

ACPI: Power Button [PWRB]

Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled

Non-volatile memory driver v1.3

Linux agpgart interface v0.103

ACPI: PCI Interrupt Link [LSA0] enabled at IRQ 21

ata1: SATA max UDMA/133 abar m8192@0xfe9fc000 port 0xfe9fc100 irq 21

usb usb1: New USB device found, idVendor=1d6b, idProduct=0002

usb usb1: Manufacturer: Linux 3.6.1-1.fc17.x86_64 ehci_hcd

usb usb1: SerialNumber: 0000:00:04.1

hub 1-0:1.0: USB hub found

hub 1-0:1.0: 8 ports detected

usb usb2: Manufacturer: Linux 3.6.1-1.fc17.x86_64 ohci_hcd

usb usb2: SerialNumber: 0000:00:04.0

usbcore: registered new interface driver usbserial
```

```
usbcore: registered new interface driver usbserial_generic
USB Serial support registered for generic
usbserial: USB Serial Driver core
serio: i8042 KBD port at 0x60,0x64 irq 1
serio: i8042 AUX port at 0x60,0x64 irq 12
mousedev: PS/2 mouse device common for all mice
rtc0: alarms up to one year, y3k, 114 bytes nvram, hpet irqs
device-mapper: uevent: version 1.0.3
device-mapper: ioctl: 4.23.0-ioctl (2012-07-25) initialised: dm-devel@
redhat.com
cpuidle: using governor ladder
cpuidle: using governor menu
drop_monitor: Initializing network drop monitor service
ip_tables: (C) 2000-2006 Netfilter Core Team
input: AT Translated Set 2 keyboard as /devices/platform/i8042/serio0/
input/input2
ata1: SATA link up 3.0 Gbps (SStatus 123 SControl 300)
ata1.00: ATA-8: WDC WD5000AAVS-00N7B0, 01.00A01, max UDMA/133
ata1.00: 976773168 sectors, multi 0: LBA48 NCQ (depth 31/32)
ata1.00: configured for UDMA/133
scsi 0:0:0:0: Direct-Access     ATA      WDC WD5000AAVS-0 01.0 PQ: 0
ANSI: 5
Freeing unused kernel memory: 1032k freed
Write protecting the kernel read-only data: 12288k
nouveau 0000:00:10.0: setting latency timer to 64
[drm] nouveau 0000:00:10.0: Detected an NV40 generation card (0x063000a2)
Console: switching to colour dummy device 80x25
usb 1-6: New USB device found, idVendor=0bda, idProduct=0181
Initializing USB Mass Storage driver...
scsi4 : usb-storage 1-6:1.0
tsc: Refined TSC clocksource calibration: 2699.931 MHz
usb 2-3: Manufacturer: American Power Conversion
hid-generic 0003:051D:0002.0001: hiddev0,hidraw0: USB HID v1.00 Device
[American Power Conversion Back-UPS RS 700G FW:856.L3 .D USB FW:L3  ] on
usb-0000:00:04.0-3/input0
EXT4-fs (dm-1): mounted filesystem with ordered data mode. Opts: (null)
e1000: Intel(R) PRO/1000 Network Driver - version 7.3.21-k8-NAPI
```

```
e1000: Copyright (c) 1999-2006 Intel Corporation.
r8169 0000:04:00.0: irq 43 for MSI/MSI-X
r8169 0000:04:00.0: eth0: RTL8102e at 0xffffc90010fae000,
44:87:fc:69:4d:0f, XID 04c00000 IRQ 43
microcode: CPU0 updated to revision 0xa0b, date = 2010-09-28
ALSA sound/pci/hda/hda_intel.c:1593 Enable delay in RIRB handling
```

There's more...

Consult the man or info pages for any of these commands for more information. In particular, look at `man modprobe.conf` for details on how to use the configuration options available with `modprobe`.

You can use the `uname -r` command to see the current kernel version. You will often see that the expression `uname -r` in scripts and aliases works great.

Building a kernel from kernel.org

For this example, we will be using kernel files from the `http://kernel.org` website.

Getting ready

You should be able to perform all but the last step without any possible harm to your system. The `make install` command will modify your GRUB file(s), and so at a minimum, I would back those up. To be very safe, and since we already know I am paranoid, if you are going to install the new kernel, I would run all of these steps on a test machine.

This example assumes your computer has been installed as a full development system. You will need up-to-date versions of `GCC`, `make`, the QT development package, and others. If you have a current distribution elected to install the Software Development package (or equivalent), you are probably good to go. I suggest having at least 10 GB of file space available on the partition you plan to do the builds in; more if you are going to be creating a lot of kernel trees (the files in kernel 3.9.1 are using 6.5 GB).

The `vmlinuz`, `initramfs`, and `map` files will be copied to `/boot`, so make sure it is large enough to handle the number of extra kernels you want (about 500 MB is typical).

You will need to be root to run the `make modules_install` and `make install` commands. I suggest becoming root for all of this procedure to avoid any file permission problems.

How to do it...

The following are the steps to get and build a kernel:

1. On your browser, navigate to `http://kernel.org`.
2. Click inside the yellow square where it says **Latest Stable Kernel** and save the file. On Fedora, the `Downloads` directory is `/home/<user>/Downloads`.
3. Where you want to build is pretty much up to you. I personally dislike long directory paths and so I have put mine in the `/temp` directory. You may choose another location if you wish.
4. Copy or move the `.xz` file from the `Downloads` directory to `/temp`. For this example, the filename is `linux-3.9.1.tar.xz`.
5. Change the directory to `/temp` and extract the file `tar xvf linux-3.9.1.tar.xz`. This used to take a long time, but it's not too bad these days.
6. When it's finished, change directory to the `cd /temp/linux-3.9.1` directory.
7. The next step is to obtain a kernel configuration file. Unless you have a specific one already in mind, I usually take the latest one out of the `/boot` directory. On my system, I ran the following command:

   ```
   cp /boot/config-3.6.1-1.fc17.x86_64
   ```

8. You could have copied the file directly to `.config`, however, I like seeing what I started with. So do it now:

   ```
   cp config-3.6.1-1.fc17.x86_64  .config
   ```

9. We now need to run a kernel build program to get everything in sync. We will use the `xconfig` program, which will be discussed in more detail in the next section. For now, just run the following command:

   ```
   make xconfig
   ```

10. This will bring up a cool-looking screen with about a million things on it. Click on **File** | **Save** and then **File** | **Quit**.
11. You should now be back at the text screen, with it showing something like the following:

    ```
    Big4 /temp/linux-3.9.1 # make xconfig
      HOSTCC  scripts/kconfig/conf.o
      HOSTCC  scripts/kconfig/zconf.tab.o
      HOSTCXX scripts/kconfig/qconf.o
      HOSTLD  scripts/kconfig/qconf
    scripts/kconfig/qconf Kconfig
    #
    # configuration written to .config
    #
    ```

12. Now run the `make` command. Depending on the speed of your computer, this may take a long while. If you're a coffee or tea drinker, this might be a good time to go get some.
13. Check to make sure there were no errors, and then run the following command:

    ```
    make modules_install
    ```

14. This next step will modify your GRUB configuration. I always make sure I have backups for those just in case. When done, to install the kernel, run the following command:

    ```
    make install
    ```

15. In most cases, the `make install` command will set the new kernel as the default. You can check this by viewing your GRUB configuration files (we will see more on GRUB later in this chapter).
16. To actually try out the new kernel, you must reboot the system (more on that later). When the screen comes up, make sure the proper kernel is selected on the menu.
17. Since we made no real changes, the kernel should boot up without any issues. Check this by running the `uname -a` command to make sure it booted the right one. You should not see or notice any differences in this kernel. However, depending on several factors it may not work as expected, or it may not even boot at all. If this is the case, you should be able to reboot back into the previous good kernel.

When rebooting, I strongly suggest a cold start. Perform an orderly shutdown (`shutdown -h now`) and let the machine sit for at least a few seconds; minutes wouldn't hurt either. I have seen some very strange things happen with warm boots that any sane person would say was impossible.

Using xconfig to modify the configuration

As was mentioned in the previous section, the `.config` file is what controls everything that goes into the kernel files. This includes both the `vmlinuz` and `initramfs` files, and the device driver modules. The `.config` is a text file but is not meant to be edited directly, instead one of several different programs can be used. In this chapter we show you how to use the `xconfig` program to make changes to the `.config` file.

Getting ready

Please see the previous section on what is needed to prepare before performing these steps.

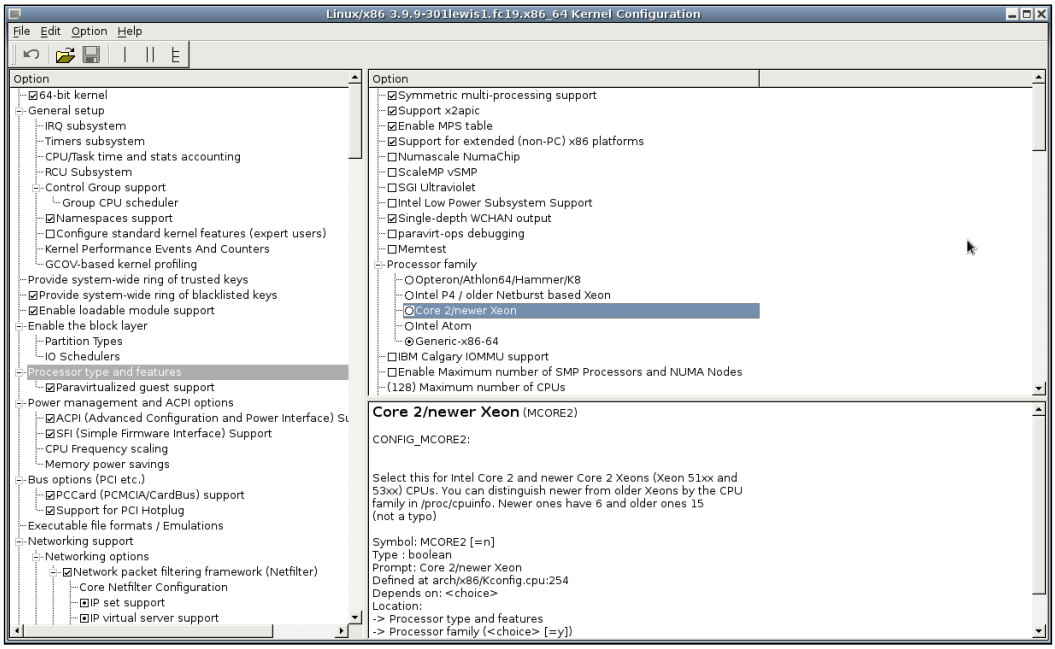
How to do it...

Here we will use `xconfig` to modify the configuration:

1. Change to the kernel build directory and run the following command:

   ```
   make xconfig
   ```

2. That cool-looking screen should come up again. This program takes a few minutes to get used to, so we will go through it step-by-step.
3. First, locate the string **Processor type and features** and click on it. You will see the screen change.
4. Now in the panel on the right-hand side under **Processor family**, click on **Core 2/ newer Xeon**. Click on the text, *not* the radio button.
5. You should now be seeing something like the following screenshot (from my Fedora 19 system using a 3.9.9 kernel):



6. You must use caution when using this program. It is quite easy to accidentally click on a radio button, changing something you did not intend to. For this reason I suggest making frequent backups of your `.config` file. Since it is a text file, you can use the `diff` program to look for changes between files. If you are unfamiliar with `diff`, run `man diff` for more information.

7. So, let's go ahead and change something. If you are running modern hardware, it probably has a Core 2 or Xeon processor in it. Run `cat /proc/cpuinfo` to see what you have. If it looks appropriate, click the radio button on the **Core 2/newer Xeon** line.
8. Those are the basics for configuring a new kernel. When first getting started, I suggest making as few changes as possible between builds. This way it will be much easier to track down what change caused a problem, if one should occur.
9. To complete this discussion of `xconfig`, let's try another field. On the left-hand side, click on the text of **General setup**.
10. You should see the text on the right change. In general, when using `xconfig` you click on the text to change the display, to expand or compress an entry click on the appropriate button, and to actually change a value click on the radio button. A small black dot in a box means a module will be built.

There's more...

You can use the `diff` command to look at the differences between the `.config` files you have saved. This will save a lot of time when debugging.

The program can be somewhat confusing. In places, the text on the right will indicate **Say Y here**. That means to make sure there is a check in the checkbox. Likewise, **No** means no checkmark. In some cases the program will say to indicate Y or N in a field that doesn't even have a checkbox. I suppose those are bugs, if so, they have been there for a long time.

You may also click on **Help** | **Introduction** for some brief text on how `xconfig` is used.

Extra care must be taken when building kernels. It is easy to make a mistake when using `xconfig` to modify your `.config` file resulting in an unbootable kernel. The following are a few pointers:

- Make a backup of the current `.config` every single time you make a change.
- Make as few changes as possible at one time. It's tempting to try and save time by making a lot of changes, if this works for you that is great. It does not work for me though.
- If your latest kernel will not boot, try using `diff` to compare your latest `.config` file with the last good one. You might be able to spot the problem right away.
- If all fails, go back to a known working configuration file and start again from there. You have been making backups of your `.config` files all this time, right?

Working with GRUB

When working with kernels, you may need to change your GRUB configuration file from time to time. You can modify which kernel comes up by default, the timeout value for the kernel selection menu, the parameters passed to the kernel, boot other operating systems, and many other things.

The `grub.conf` file is normally located in `/boot/grub`, or you can use the `/etc/grub.conf` file, which is a symbolic link.

The following is what `grub.conf` looks like on my Fedora 14 system:

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this
file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/sda3
#          initrd /initrd-[generic-]version.img
default=2
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Fedora (3.9.1)
      root (hd0,0)
      kernel /vmlinuz-3.9.1 ro  root=UUID  rhgb quiet
      initrd /initramfs-3.9.1.img
title Fedora (2.6.38.4)
      root (hd0,0)
      kernel /vmlinuz-2.6.38.4 ro root=UUID rhgb quiet
      initrd /initramfs-2.6.38.4.img
title Fedora (2.6.35.6-45.fc14.x86_64)
      root (hd0,0)
      kernel /vmlinuz-2.6.35.6-45.fc14.x86_64 ro root=UUID rhgb quiet
      initrd /initramfs-2.6.35.6-45.fc14.x86_64.img
```

How to do it...

Here we will show you how to change some of the items in the `grub.conf` file. Note that a mistake here can render your system unbootable, so either just follow along or be very careful.

1. Change to the proper directory:

   ```
   cd /etc
   ```

2. Make a backup copy: `cp grub.conf /temp` (or some other suitable location).
3. Edit it with vi or equivalent:

   ```
   vi grub.conf
   ```

4. Referring to my file above let's boot the first stanza by default. Change the `default=2` line to `default=0`. Note that they count starting at 0.
5. Now let's increase the amount of time it will wait for you to make a selection; change the `timeout` value to `10`.
6. Suppose you want to boot up in text mode, to do this comment out (that is, put a `#` in front of) the `splashimage` and `hiddenmenu` lines.
7. And also remove the `rhgb quiet` from the stanza (or all of them).
8. If you have any parameters you want passed to the kernel, you can just add them to the end of the `kernel` line.

How it works...

Let us see the break-up of the above mentioned steps in the following section:

- The commented section says `You have a /boot partition. This means that all kernel and initrd paths are relative to /boot`. What this is trying to say is when you come across a line later containing something like `/vmlinuz-3.9.1`, it really means `/boot/vmlinuz-3.9.1`. Don't forget that it works this way, and you will save yourself a lot of grief later.
- The `default=2` means to use the third title or stanza (yes, this is another place where they start counting at 0 instead of 1).
- The `timeout=5` means to display the kernel boot menu for 5 seconds before booting into the default one.
- The `splashimage` line shows a graphical image on the screen while booting. I dislike this immensely and so I comment it out.
- The `hiddenmenu` line means to hide the kernel boot menu. You comment this line out to show the menu. Yes, it's backwards again, but not quite as awkward as counting things starting at 0.

- The first title line begins a kernel stanza. Everything between that line and the next title line (or end of file) is associated with that kernel. In this case, the first listed kernel is the latest one I have created (3.9.1).
- The `root (hd0,0)` line means that my `/boot` directory is located on the first hard drive on the first partition.
- The next line is the actual kernel file and parameters.
- The last line of this stanza is the initial RAM disk image file.
- As you can see, there are two more stanzas (kernels) available on this machine. I am running `2.6.35-6-45.fc14.x86_64`, the default kernel for Fedora 14 64-bit.

Understanding GRUB 2

GRUB 2 is now being used in many Linux distributions. It is a complete rewrite and was created to fix some of the perceived issues in GRUB Legacy. It is still being developed, and so the information here may not be complete or up-to-date.

The boot configuration when using GRUB 2 is in the `/boot/grub2/grub.cfg` file. You can also refer to it by the `/etc/grub2.cfg` file which is a symbolic link.

The following is what the first few lines look like on my Fedora 17 system:

```
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by grub2-mkconfig using templates
# from /etc/grub.d and settings from /etc/default/grub
#
### BEGIN /etc/grub.d/00_header ###
if [ -s $prefix/grubenv ]; then
  load_env
fi
set default="1"
if [ x"${feature_menuentry_id}" = xy ]; then
  menuentry_id_option="--id"
else
  menuentry_id_option=""
fi
```

As the commented line says, this file is not intended to be edited directly. Instead, the `/etc/default/grub` file is used in combination with the set of files in the `/etc/grub.d` directory.

```
Big2 /etc/grub.d # ls -la
total 76
drwx------.   2 root root  4096 Oct 18  2012 .
drwxr-xr-x. 167 root root 12288 May 15 03:34 ..
-rwxr-xr-x.   1 root root  7528 Aug  2  2012 00_header
-rwxr-xr-x.   1 root root  9265 Aug  2  2012 10_linux
-rwxr-xr-x.   1 root root  9948 Aug  2  2012 20_linux_xen
-rwxr-xr-x.   1 root root  2564 Aug  2  2012 20_ppc_terminfo
-rwxr-xr-x.   1 root root  9339 Aug  2  2012 30_os-prober
-rwxr-xr-x.   1 root root   214 Aug  2  2012 40_custom
-rwxr-xr-x.   1 root root   216 Aug  2  2012 41_custom
-rw-r--r--.   1 root root   483 Aug  2  2012 README
```

How to do it...

The following are the steps to make changes to the boot configuration when using GRUB 2. Remember, the `grub.cfg` file is not edited directly; changes are instead made to the files in the `/etc/grub.d` directory.

1. Let's change the `timeout` and `rhgb` values. Edit the `/etc/default/grub` file.
2. Change `GRUB_TIMEOUT` to `10`.
3. In `GRUB_CMDLINE_LINUX`, remove the `rhgb quiet`. Save the file.
4. Create the new file by running the following command:

   ```
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

5. The modified `grub.cfg` file should be ready for booting.

How it works...

The following is a description of what the scripts in `/etc/grub.d` are for:

- `00_header`: This generates the header for `grub2.cfg` and gets the info from the `/etc/default/grub` file
- `10_linux`: This loads the menu entries
- `20_linux_xen`: This looks for zen kernels to add them to the menu
- `20_ppc_terminfo`: This checks for a correct size terminal on PPC systems

- `30_os-prober`: This searches the hard drives for other operating systems so it can add them to the boot menu
- `40_custom`: This is a template that can be used to add extra entries to the boot menu
- `41_custom`: This reads information from `/boot/grub/custom.cfg` if it exists
- `README`: This is a file that contains some other useful information

There's more...

The following is a partial list of the GRUB 2 commands available in the operating system:

- `grub2-editenv`: This edits the GRUB environment block
- `grub2-fstest`: This is a debug tool for GRUB filesystem drivers
- `grub2-kbdcomp`: This generates a GRUB keyboard layout file
- `grub2-menulst2cfg`: This transforms legacy `menu.lst` into `grub.cfg`
- `grub2-mkfont`: This makes GRUB font files
- `grub2-mkimage`: This makes a bootable image of GRUB
- `grub2-mklayout`: This generates a GRUB keyboard layout file
- `grub2-mkpasswd-pbkdf2`: This generates a hashed password for GRUB
- `grub2-mkrelpath`: This makes a system path relative to its root
- `grub2-mkrescue`: This makes a GRUB rescue image
- `grub2-mkstandalone`: This makes a memdisk-based GRUB image
- `grub2-script-check`: This checks `grub.cfg` for syntax errors
- `grub2-bios-setup`: This sets up a device to boot using GRUB
- `grub2-install`: This installs GRUB to a device
- `grub2-mkconfig`: This generates a GRUB configuration file
- `grub2-mknetdir`: This prepares a GRUB netboot directory
- `grub2-ofpathname`: This finds an OpenBOOT path for a device
- `grub2-probe`: This probes device information for GRUB
- `grub2-reboot`: This sets the default boot entry for GRUB, for the next boot only
- `grub2-set-default`: This sets the saved default boot entry for GRUB
- `grub2-sparc64-setup`: This sets up a device to boot using GRUB

To learn more about GRUB 2, the official web page is `http://www.gnu.org/software/grub/grub.html`.

A

Linux Best Practices

In this appendix we will cover the following topics:

- Root user versus normal user
- Running the GUI
- Creating, verifying, and storing backups
- Permissions and who you are
- Making backups in real time
- Environmental variables and shells
- The best environment
- Using and monitoring a UPS
- Being careful when copying files
- Verifying archive files and using checksums
- Firewalls, router settings, and security
- What to do if you find an intrusion
- Spaces in filenames
- Using scripts and aliases to save time and effort
- Using scp and ssh with automatic authentication
- Saving history and taking screenshots
- Space on drives
- Being open to new ideas

Introduction

There are many things you can do to get the most out of your Linux system(s). Conventional wisdom states that there are many ways to perform a particular task on a computer. This is true, however, in reality there is usually only one good way to accomplish something. The trick is to be open-minded and see the good when it happens.

Root user versus normal user

Running as the root user versus a normal user mostly depends on the environment you are in. If each person has their own workstation and is in charge of how it is set up, then running as root might be very natural for you (especially if you don't make mistakes). However, if you're working in a bank or other situation where a typing error might wipe out a million dollar account, running as root is certainly not advisable. In these cases, assuming you have the authority, you would switch to root only when necessary and only to perform the needed task. Using sudo, if it has been configured correctly, is also an option. See *Chapter 5*, *Permissions, Access, and Security*, for more information on sudo.

Another thing to keep in mind is how comfortable you are running as root. If you are error prone or nervous, and/or have caused serious damage in the past running as root, you obviously need to take great care when doing so. On the other hand, if you consistently run as root and don't ever make a mistake then that is great. It is certainly more efficient.

A special note to system administrators: I have seen this on more than one occasion and so will mention it here. This applies to both novice and seasoned system administrators. It is customary for you (and maybe your manager) to be the only ones with root authority on the systems. This sounds like a good idea, right? It might avoid having someone make a mistake that could take out an entire project. And, it just feels good being the person in charge. People come to you when they need something changed and you happily do it. Then they come again, and again, and again. At some point you realize you can't get any of your work done because of all the requests, and that they can't have their work done if you are not around. So you try and set up sudo. Now it's even worse; every time you think you have it set up to handle anything, someone may come to you if it fails again. So what do you do?

You may be able to give selected users root access. Use your instincts. For example, watch how the individual users type. Are they comfortable when using the command line? Do they type with authority, or seem scared of the machine? If a particular user consistently uses the GUI to perform tasks that are done much more efficiently on the command line, then I would see that as a strong warning sign.

In time you will get a feel of who can be trusted with root access and be able to grant it to them. If, of course, someone does make a mistake, then it should not be the end of the world. They can't really hurt the entire project, because you have been creating and verifying daily backups, right? You can restore the damage and take root back from them. Note, one mistake is all it takes. I would not trust that user with root again.

Running the GUI

While I am somewhat quick to trust my users with root access, and use it myself most of the time, I absolutely do not recommend running the GUI this way. Some distributions won't even allow it. By running the GUI as root, you are in effect running a lot of other things as root, such as your browser and mail programs. Not a good idea at all.

Here is my preferred environment on a Linux or UNIX system. I use Fedora, but these ideas should work on most other distributions. After installing a system, one of the first things I do is change it so that the machine comes up in command line mode and not in a GUI. This way if a graphics problem has occurred, it is much easier to diagnose and correct. I also have a choice in which GUI to bring up by running the appropriate `startx` type command. At the command prompt, I log in as a normal or guest user. On my Fedora 14 system I then run `startx`, which brings up Gnome 2.

After the GUI has come all the way up, I open a terminal session and run `su` to root. I check to make sure the machine can ping, and usually do a few more sanity checks. If all is well, I then run my `jset` script. It performs some desktop customization such as opening terminal windows into their proper directories, and reminding me of what command to run (I have written a lot of programs and so really need this). It also mounts my USB devices, or warns me if there is a problem. I then position the terminal sessions right where I want them to be. I am now set to get some work done.

The following is a script similar to the one that I use to set up my desktop after booting up:

```
#!/bin/sh
# last update 6/9/2013 (rebooted)

echo percentused - run go
cd /lewis/java/percentused
xterm +sb -title Xterm    -geom 80x20 &

echo apcupsd - run go
cd /lewis/java/apc
xterm +sb -title Xterm    -geom 80x20 &

echo jtail - run jtail
cd /lewis/jtail-f/jtail
xterm +sb -title jtail -geom 137x30   &

echo jsecure - run jsecure
cd /lewis/jtail-f/jsecure
```

```
xterm +sb -title jtail -geom 125x33   &

echo ping - run loop1
cd /lewis/ping
xterm +sb -title ping  -geom 86x8 &

echo runbackup1 - run runbackup1
cd /lewis/backup
xterm +sb -title runbackup1 -geom 65x21 &

echo jwho - run jwho
cd /lewis/jwho
xterm +sb -title jwho  -geom 65x8  &

# mount usb stick
mount /dev/sdg1 /usb
# mount Iomega external drive
mount /dev/sdf1 /megadrive
```

Creating, verifying, and storing backups

I cannot stress how important it is to create backups of your system(s). At a minimum, copy your personal and business data, and configuration files to a safe place. Some people backup everything, even the operating system itself. Whatever you decide to do, make a plan and stick to it. As mentioned in *Chapter 8*, *Working with Scripts*, this is a great time to design and use a script. Use `crontab` if desired, to automate the periodic taking of backups.

The `tar` command is great for backing up entire directories. Note that it will get any hidden files as well. You can exclude specific directories if desired, and do quite a few other things with `tar`. The following are the commands similar to the ones I use to backup my `/home/guest1` directory.

`tsback1` is a text file containing the number to start with.

```
cat tsback1
0
```

The following is the start of the script:

```
    cd /home
    NUM=`cat tsback1`        # get the next number to use
    tar -cvzf /megadrive/backups/backup$NUM.gz --exclude=Cache
--exclude=.cache --exclude=.thumbnails  guest1
```

Remember to include the directory to be backed up as the last thing on the line. This first changes the `/home` directory, because for `tar`, you want to be in the parent directory of the sub-directory to be backed up. The next line sets the `NUM` variable to the next one to use. The last line creates the `tar` file directly onto my USB external drive in the appropriate directory.

I attempt to be very careful while creating backups. The script I actually use to back things up does quite a few other things. For example, it checks to make sure my USB external drive is really there, and can be written to (it should also check if there is enough free space on the drive, that's a TODO of mine). If the code determines the drive is not there or some other error occurs, a really loud and obnoxious alarm goes off. And, if I have not responded to this alarm in 5 minutes, an email is sent to my cell phone. How's that for paranoid?

Making backups is great. However, if the backup is unusable, it doesn't do you much good. So, it is wise to verify your backups from time to time. How often do this is up to you and your comfort level. My script routinely copies the backup files to another machine, which then extracts them and runs a few tests. If anything doesn't look quite right, another alarm goes off. This is all done automatically in scripts.

Okay, we are now making backups and verifying them. What about storing them? Suppose you have everything just right, all your files are copied and verified, and they are all located in the same place such as your home or office. And now something unspeakable occurs, such as a fire or theft. I agree, there is a very low chance of this happening, but it still could. I, for one, do not want to try and reproduce the million lines of code I have written since 1982 and so have backups all over the place, including off-site. In some of the companies I have worked at, the files were copied to tape, CDs, and/or hard drives, and stored in a walk-in fireproof safe. Pretty good idea.

Permissions and who you are

This mostly pertains to system administrators. As a system administrator, you probably do most of your work as the root user. You set up guest accounts and quotas, and maybe even create scripts, and so on. It is sometimes easy to forget that your users don't have root authority.

With this in mind, be sure to check out your additions and changes from the user's perspective. Become that user with `su` and make sure you can access everything normally. This will save you a lot of time, and maybe even embarrassment, if you find a problem before your users do.

Making backups in real time

When editing scripts and other files, it is a good idea to make numbered backups. Nothing is more frustrating than having something work, then break after some changes, and then not be able to get it working again quickly. With numbered backups you can always go back to a previous version that worked, and then use `diff` to find the mistake. I sure learned this one the hard way.

The following is a backup script I wrote for the users of this book (the one I normally use is written in C). It is named `mkbak`:

```
#!/bin/sh
# mkbak script to create backup files
if [ "$1" = "" ] ; then
 echo "Usage: mkbak filename(s)"
 echo "Creates numbered backup file(s) in the current directory."
 exit
fi
for i in $* ; do
 if [ ! -f $i ] ; then
   echo File $i not found.
   continue
 fi

 num=1
 while [ 1 ]
 do
   ibak=bak-$num.$i
   if [ -f $ibak ] ; then
     num=`expr $num + 1`
   else
     break
   fi
 done
 cp $i $ibak
 rc=$?
 if [ $rc -eq 0 ] ; then
  echo File $i copied to $ibak
```

```
 else
  echo "An error has occurred in the cp command, rc: $rc"
 fi
done
```

This script comes free of charge, but with some limitations. It will not handle filenames with blanks, and only works on files in the current directory. Note that you can `cd` to the directory you want first and then run it.

The following is the script I use to backup the current book file I am working on:

```
#!/bin/sh
# b1 script to copy book file
# Date 1/22/2013
FN=startA1.txt                    # name of file to back up
STARTDIR=`pwd`                    # remember the starting directory
cp $FN /usb/book                  # copy to USB stick
cd /usb/book                      # cd to it
mkbak $FN                         # make the numbered backup

cd $STARTDIR                      # go back to the starting directory
cp $FN /megadrive/book            # copy to USB external drive
cd /megadrive/book                # cd to it
mkbak $FN                         # make the numbered backup

cd $STARTDIR                      # go back to the starting directory
sum $FN /usb/book/$FN /megadrive/book/$FN     # use sum to check
scp $FN $B2:/temp                 # copy to my other machine
ssh $B2 /usr/bin/sum /temp/$FN       # check the copy
```

While editing the file (the `FN` variable), I will manually run this from time-to-time, usually after a lot of changes, and definitely just before I get up to take a break or whatever.

Environment variables and shells

One thing that comes up a lot during system administration is the monitoring of several machines. It's not uncommon to have 5 or 6 `ssh` sessions open at a time, more if you have multiple monitors. It's crucial to know which session is running on which machine, as typing the right command on the wrong machine can be a disaster. For this reason and others, I recommend using a custom `PS1` variable when logging into a remote machine.

This was mentioned in *Chapter 1*, *Using the Terminal / Command Line*, during the discussion of environment variables. The following is what my `PS1` variable looks like on my machine running Fedora 17:

```
Big2 /temp/linuxbook/chapA # echo $PS1
Big2 \w #
Big2 /temp/linuxbook/chapA #
```

Simple, and not too cluttered. The following is what `PS1` looks like on my other machine when I log into it:

```
BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 /temp # echo $PS1
BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 \w #
BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 BIG4 /temp #
```

It should be pretty hard to mix those up.

While we are on the subject of environment variables, there is something else to keep in mind. When you make a change to your `.bashrc` file and source it, the changes are only visible in that session (as well as any newly opened sessions). In order to see the change in other existing sessions you must source it in them as well. It would be rather cool if there was a way to make the changes visible in every session with just one command, however, I do not believe that is possible. Of course, one could argue that `shutdown -r` now will do it.

The best environment

What works best for one person may not work the best for another. However, I know that I am most productive when using a fast desktop system with plenty of memory and storage, and two big displays. The following is my typical set up:

On my left-hand side display, I put the scripts, and the following programs that I use to monitor the system:

- A disk space monitoring program written in Java and C
- A program that monitors my **Uninterruptible Power Supply** (**UPS**), also written in Java and C

- A script that pings the network once a minute and logs any failures
- A program that uses `tail -f /var/log/messages` to monitor kernel messages
- My backup script that runs every night at 3 a.m.
- A "poor man's" intrusion detection script (more on that later)
- A script that e-mails the system status to my cell phone twice a day
- I have the `Computer` and `Guest` folder icons visible and easy to access
- Any `ssh` sessions to remote machines
- And a few others that are too boring to mention

All of these are set to remain visible in all workspaces. Speaking of workspaces, I usually have four of them. I always place the same programs and terminal sessions in the same workspace, and in about the same place on the screen. This way, I can get to where I want to be very quickly. Have you ever been in a situation where your team is about to miss an important deadline, and you are being forced to watch and wait as someone else wastes a lot of time trying to find or do something on their system? You do not ever want to be that person.

On the right-hand side display I do most of my actual work. The following is how my workspaces tend to be laid out:

- In Workspace 1 are a couple of terminals. Those are there and ready in case I need to do something right away
- Workspace 2 is normally used for program development. I do C, Java, and script development here
- Workspace 3 is where I am currently typing this book in my custom written text editor (which will eventually be imported into `LibreOffice`)
- Workspace 4 is where I have my webmail client

Speaking of browsing, I tend to open those on the left-hand side display, and in whatever workspace that goes with what I am currently working on. This is very fast and efficient, and is also easier to cut and paste when needed.

Not all of us have the luxury of fast machines or dual monitors, particularly at our jobs, where it sometimes seems to be more important to management to save money, instead of giving the employees what they need to be productive. All I can say to that, is try your best to get what you need to do your job as efficiently as possible.

Using and monitoring a UPS

In my opinion it is imperative that a UPS be used on at least your primary workstation. All kinds of bad things can happen to the hardware if the power suddenly goes off (or worse, browns out), not to mention what might happen to your data. With modern journaling filesystems, I realize data loss is somewhat rare, but why take the chance? Also, I really just don't like to reboot. Ever.

Depending on your situation, try to get the best UPS you can afford. You want one that will run your system for a long time and also power your display, modem, router, and external drives if you have them. This way, if the power goes out for just a short time you won't lose anything, and won't have to wait for everything to come back up.

There are many different UPS brands available today. I am somewhat partial to the **American Power Conversion** (**APC**) devices. I have several of them, and they work well with Linux. Be sure to get one with a phone-connector to USB port, as the old-style serial port units do not work properly.

The `apcupsd` daemon can be used to monitor the UPS. If your distribution does not already have it, the package can be installed.

- If using Fedora, run `yum -y install apcupsd` (substitute your package installer as appropriate)
- Comment out the `WALL` statement in the `/etc/apcupsd/apccontrol` file to keep annoying messages from being broadcasted to every terminal
- Run `apcaccess status` to query the UPS

There's quite a bit more you can do with `apcupsd`, for more information check its website at `http://www.apcupsd.com`. This also lists some UPS units that might not be as compatible with Linux as the ones I have.

One more thing, you will probably want to use the auto-shutdown feature of the UPS. It can be set up to automatically shutdown your machine if the power has been out for too long. Most units allow you to set the amount of time to stay running, before shutting down. Remember that the longer the UPS runs on the batteries, the shorter their life span will be.

Being careful when copying files

When copying files to a directory, make sure it really is a directory. This happens enough for me to mention it, and I have to admit I still almost do it from time-to-time. It is quite easy to copy a lot of files to what you believe is a directory, but isn't. The result is that just the last file that was copied will be there, and if you don't still have the source files, they might now be lost. Use the `file` command to verify whether the target really is a directory before you copy the files.

Verifying archive files and using checksums

One thing that comes up a lot is mistakes that go unnoticed in the creation of `tar` or `zip` archives that are going to be sent to another person or site.

The following are the steps that should be followed:

1. Copy the files to an appropriate directory (make sure it really is a directory first).
2. Use `zip` or `tar` to compress and create the archive.
3. Use the `tell` or `list` option to be sure it looks correct. For TAR it's `tar -tvzf filename.gz` and for ZIP it's `unzip -l filename.zip`.
4. Run the `sum` command against your file, and then send the file to where it needs to go.
5. If using `scp`, use `ssh` to run the `sum` command on the file on the remote system like the following:

   ```
   ssh <user@remote-host> /usr/bin/sum filename.gz
   ```

6. The two `sum` values should match.
7. If using e-mail, run `sum` on your end, and send the result along with the e-mail.

A piece of advise for developers; suppose you are creating an archive of a programming project. To make absolutely sure you have copied every file it needs, create the archive and then copy it to another machine. Un-compress and build it as you normally would. An error will occur if a needed file is missing.

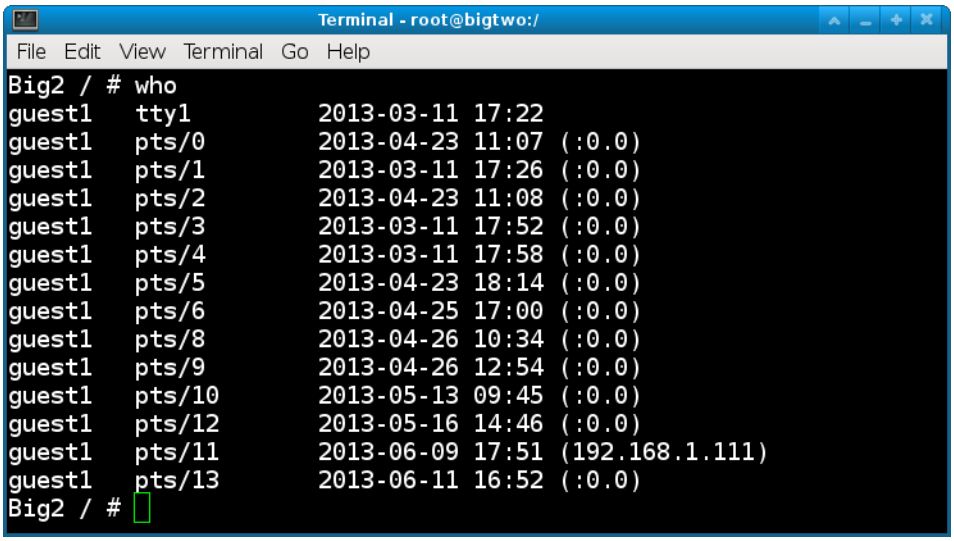
Firewalls, router settings, and security

Firewalls were covered in *Chapter 5*, *Permissions, Access, and Security*, and so this will just be a brief recap. If you are running a home system and using a good router, the default settings for `iptables` is probably all you need. It may require some tweaking, for example, to use a scanner, but for the most part you are probably safe from hackers. On the other hand, if you are the system administrator for a large company, `iptables` is probably not enough. I would investigate using a hardware intrusion appliance or some other method, to insure data and system security.

It is highly suggested that a router with a built-in firewall be used at all times. By no means would I connect a system directly to the Internet. While a typical Linux system may survive this, I have seen Windows boxes get infected with a virus in less than 30 minutes.

The default router settings are probably already strong enough to keep out the typical hacker. To be sure, and just to get an idea of what is going on inside your router, it's a good idea to login and check everything from time-to-time. On most routers, pointing your browser to `192.168.1.1` will bring up the login screen. In most cases, an ID and password are required.

The `who` command can be used in Linux to show the username, tty, date, time and IP address of each user on the system, as shown in the following screenshot:



Here is another thing you can do to help prevent an intrusion. It is a good idea to deny root access by `ssh/scp` because hackers will usually attempt to break in as root. This can be accomplished by editing the `/etc/ssh/sshd_config` file. Locate the line that says `#PermitRootLogin yes` and change it to `PermitRootLogin no`. Don't forget to remove the # (pound sign). You will also need to restart `sshd`. Now, any attempt to login as root will fail. I have all of my machines set up this way as an added precaution.

One last thing, any time someone logs (or attempts to log) into your system, a record is made of it. On Fedora this is put into the `/var/log/secure` file. You can check this file from time-to-time, or monitor it by using the `tail -f /var/log/secure` command.

And now for a bonus. The following is a simple script I use to watch for unauthorized access to my machine:

```
#!/bin/sh
tput clear
echo "jwho by Lewis 10/23/2011"
numusers=`who | wc -l`
while [ 1 ]
do
  rc=`who | wc -l`       # get number of users
  if [ $rc -gt $numusers ] ; then
    echo "Someone new has logged on!!!!!!!!!!!"
```

```
    date
    who
    jalert5 &            # see below
    numusers=$rc
  elif [ $rc -lt $numusers ] ; then
    echo "Someone logged off."
    date
    numusers=$rc
  fi
  sleep 5
done
```

Basically what this does is check every 5 seconds to see if the number of users have changed. If it has increased, the `jalert5` script is run in the background. It plays a really obnoxious WAV file every 5 seconds until I turn it off. This will also fire every time you open a new session, so you will probably want to run it last after a boot up.

What to do if you find an intrusion

Suppose you have found that an intrusion has occurred. What should you do?

These instructions are for your machine or for one you have total responsibility for. If this occurs on a machine where you work, immediately follow whatever procedures your company has in place for a security event.

Quick action is needed if you suspect a break-in. Run the `who` command or `cat /var/log/secure` and check the output. If you see a suspicious IP address, take the following actions:

- If this were a very important machine with critical data on it, I would pull out the Ethernet wire(s) and shut it down now. I would then boot up from rescue media and try to determine if anything bad had occurred. Checking the date and time they got in (from the `who` command) could give you an idea of how much damage they may have caused.
- If this were my home system, I would first pull the Ethernet wire. I would then run `ps auxw` to a file to capture what is going on in the system right now. I would copy this file to some other machine or device and then shutdown the system.

By examining the `ps` output and looking at the `tty` value I could probably determine what programs they were running, if any. This might point to what they were trying to accomplish by getting into the system.

Obviously, if someone does get into your system, they most likely did it by guessing or somehow determining a password. I would probably reset all the passwords to something much harder to crack, and then inform my users to pick better ones. Or probably assign them myself.

Okay, so at least one person reading this is thinking why pull out the Ethernet wire? Why not just bring down the interface? Well, because a shrewd attacker is going to think of that, and as soon as he has access, he is going to put code on the system to automatically bring the interface back up if it goes down. He may even put a timer on it, or hide it in some other way.

It is possible that an attacker had time to do all kinds of things. He may have even been able to modify the `who`, `ps`, and other commands to make it almost impossible to track what he did (or is still doing) from the running system. With this in mind, you still need to shutdown asap and then boot up with a rescue disk or equivalent. Some of the things to look at are the commands such as `ps` and `who`. Run the `file` command, it should show them as being a binary executable and not a shell script. If they are shell scripts, you may discover the attacker has renamed the executable files with a `.` to hide them, and then wrapped them around a script to help cover up his presence. There are many other ways to hide as well.

Spaces in filenames

When generating files for yourself or other people, do not include blanks in the filename. This can cause a lot of problems on Linux and UNIX machines. If necessary, use capital letters and/or underscores. Do not use parentheses or other special characters either. I was really amazed the first time I downloaded a file using Firefox, as it inserted parentheses to differentiate it from another file of the same name. I appreciate the fact that it didn't just over-write the original file, but using parentheses was and is a really bad idea.

Using scripts and aliases to save time and effort

One thing I see a lot of in the field is people wasting time and effort typing the same things over and over again. Don't do this. Use aliases and scripts. Don't think about how much time you might spend writing the script, think about how much time you will save by being able to use it all the time. You might also be able to incorporate it into another script later on (especially if it was written well, to begin with). Also, having these available should help with meeting deadlines.

Using scp and ssh with automatic authentication

Follow these steps to allow the use of `ssh/scp` without having to enter a password. You will need to be root.

1. First, make sure the client has used `ssh` at least once. This will create the proper directory that is needed.
2. On the master machine run the `ssh-keygen -t rsa` command. This will create some necessary files.
3. If the `/root/.ssh/authorized_keys` file does not already exist on the client, you can run `scp /root/.ssh/id_rsa.pub  <hostname>:/root/.ssh/authorized_keys`.
4. Otherwise, copy the `id_rsa.pub` file over to the client and then add it to the `authorized_keys` file (I usually put it at the bottom).
5. You should now be able to `scp` and `ssh` to the client without having to enter a password. This is really handy, especially in scripts.

You can also add this entry to another user account. For example, I added it to my `/home/guest1/.ssh/authorized_keys` file. This way I can copy files as root from one machine, and it will still be accepted by the other.

Saving history and taking screenshots

We all have to learn new things when dealing with computers. Sometimes the steps involved are pretty complicated, and I have found that in practically every situation whatever document or site I am using to perform the steps, has errors. It's not complete, the author skipped an important step, and so on. For these reasons and others, after I have (finally) gotten something to work, I run the `history` command in that session and output it to a file. I then save this under a suitable name so I can find it again later.

Depending on how much effort was required, and if appropriate, I may take screenshots of each step as well. This can be valuable later as a reference, and if you ever have to help someone else accomplish the same task. Or, if someone talks you into writing a book about it someday.

Space on drives

In the old days there was never enough hard drive space. We were always running low or out, and trying to find ways to increase our storage. Now, in modern times, this might not be as much of an issue. However, it is still a good idea to monitor your available space at all times.

There are quite a few ways to do this. On my systems I use a program I wrote in C and Java. It's called `Percent Space Used` and just uses `df -h` under the covers. You can put `df` in a script, or just check the space manually from time to time. Just don't run out! Filling up a partition is a good way to have a disaster on your hands, especially if it is a system partition.

Being open to new ideas

Here is my last bit of advice for people wanting to know Linux better. I consistently see people in the field doing their everyday jobs, and doing it the same way. Always be on the lookout on how to improve the way you perform your daily tasks. If you see a co-worker doing something that seems odd to you, don't just assume his way is wrong and yours is right. His process may be a whole lot better than yours. Learn from it. On the other hand, he may *not* have a better way, yours may be better. At this point, it's up to you to decide whether to attempt to share your ideas. I have found most people very resistant to this.

Do not let yourself get caught in the "Your way isn't better than mine, it's just different" argument. As I mentioned before, there is usually only one right way to perform a task, but most people just don't see this. Try to find it when you can, and share your ideas only if the person you are trying to help is receptive.

B
Finding Help

In this appendix we will cover the following topics:

- Using `man` pages
- Using the `info` command
- Commands and the `Usage` section
- Local documentation directories
- Browsing the web to find help
- Distribution release notes
- Linux users' groups
- Internet Relay Chat (IRC)

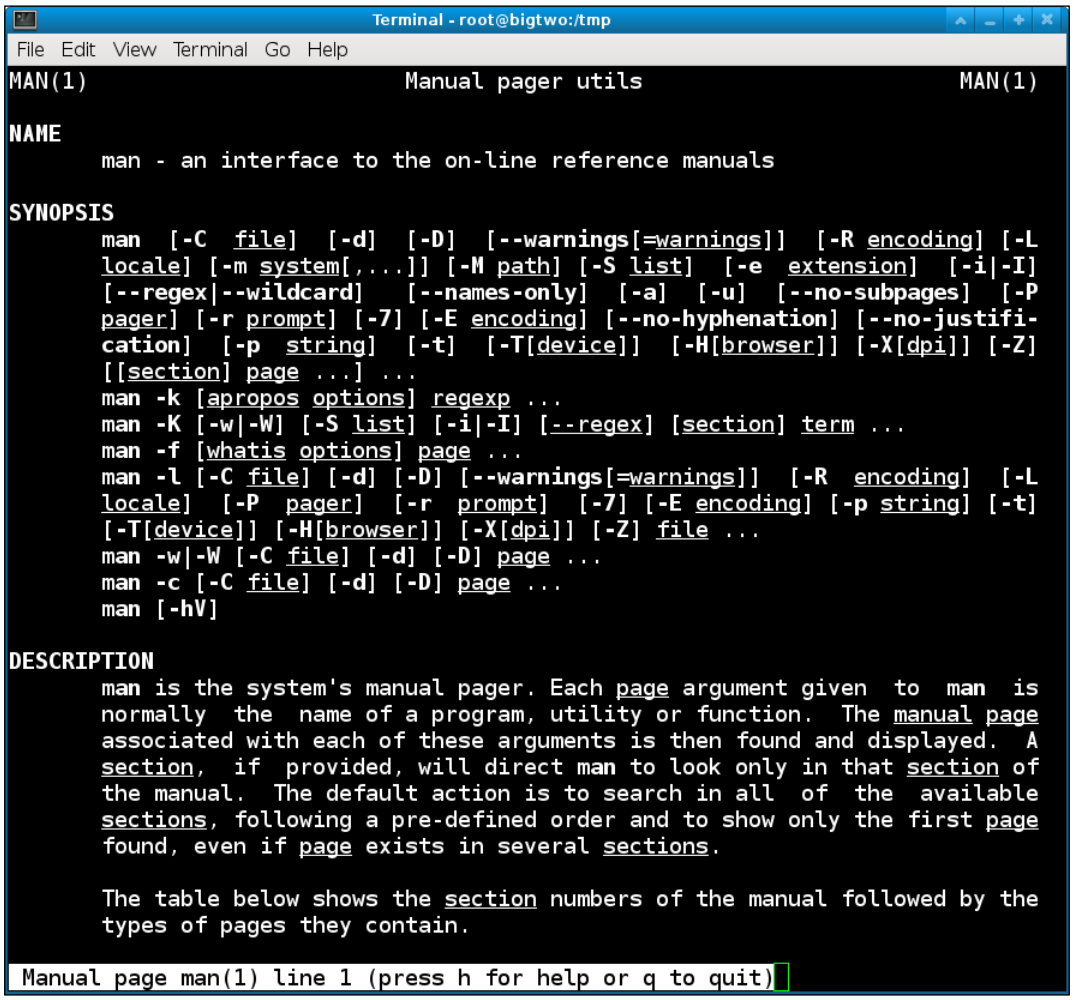
Introduction

There are many different places to find help on Linux. There is also quite a lot of information available; in fact, it is too much in some cases. It can be difficult to filter out the noise from the good stuff. Here we try to show you how to get what you need quickly and efficiently.

Using man pages

The `man` utility is an interface to the local reference manuals. It is used to quickly find information on programs, utilities, functions, and other topics. The `man` utility will accept several options; however, the usual invocation is simply `man page`, where page actually refers to a topic. You can even run `man` by itself to learn how to use it.

The following is a screenshot of the command `man man`:



Running `man` on a page displays that topic of interest. The spacebar is used to page down, and *Q* is used to quit. Pages (topics) are presented in a more or less standard order with these possible section names: `NAME`, `SYNOPSIS`, `CONFIGURATION`, `DESCRIPTION`, `EXAMPLES`, `OVERVIEW`, `DEFAULTS`, `OPTIONS`, `EXIT STATUS`, `RETURN VALUE`, `ENVIRONMENT`, `FILES`, `VERSIONS`, `CONFORMING TO`, `NOTES`, `BUGS`, `AUTHORS`, `HISTORY`, and `SEE ALSO`.

`man` shows the first page found, even if there are more pages in other sections. For example, suppose you are looking for information on how to code the `readlink` function in your C program. You might try the following command:

```
man readlink
```

It brings up a page, but of the command `readlink` and not the C function. Why? Because it shows the first page, unless you specify the section number before the page. Well, how do you know what that is? You can run `man` with the `-a` option in the following manner:

```
man -a readlink
```

This brings up the command `readlink` as before. Now press *Q* to quit. The page goes away, but instead of terminating, the `man` session shows something as follows:

```
Big4 /lewis/Fedora/17 # man -a readlink
--Man-- next: readlink(2) [ view (return) | skip (Ctrl-D) | quit (Ctrl-C)
]
```

This is giving you a choice: pressing *Enter* will show the next page (topic), *Ctrl + D* will skip to the next topic, and *Ctrl + C* will end the `man` session. Pressing *Q* when you are on the last topic will cause `man` to terminate normally as before.

So what if you already know that you want the page from section 3 and load it directly? You can specify it in the following manner:

```
man 3 readlink
```

This will skip the first two and go right into the page for `readlink` out of the POSIX Programmer's Manual.

Here is a list of the section numbers and their names:

- 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 (for example, `/etc/passwd`)
- 6: Games
- 7: Miscellaneous (including macro packages and conventions), for example, man(7) and groff(7)
- 8: System administration commands
- 9: Kernel routines

The local reference manuals can be a great source of information. They contain a wealth of data on just about everything in a Linux system. Unfortunately, they do have some drawbacks. Most are well-written and make sense. Some are, well, rather horrible. When this happens, there are other places to find help.

Using the info command

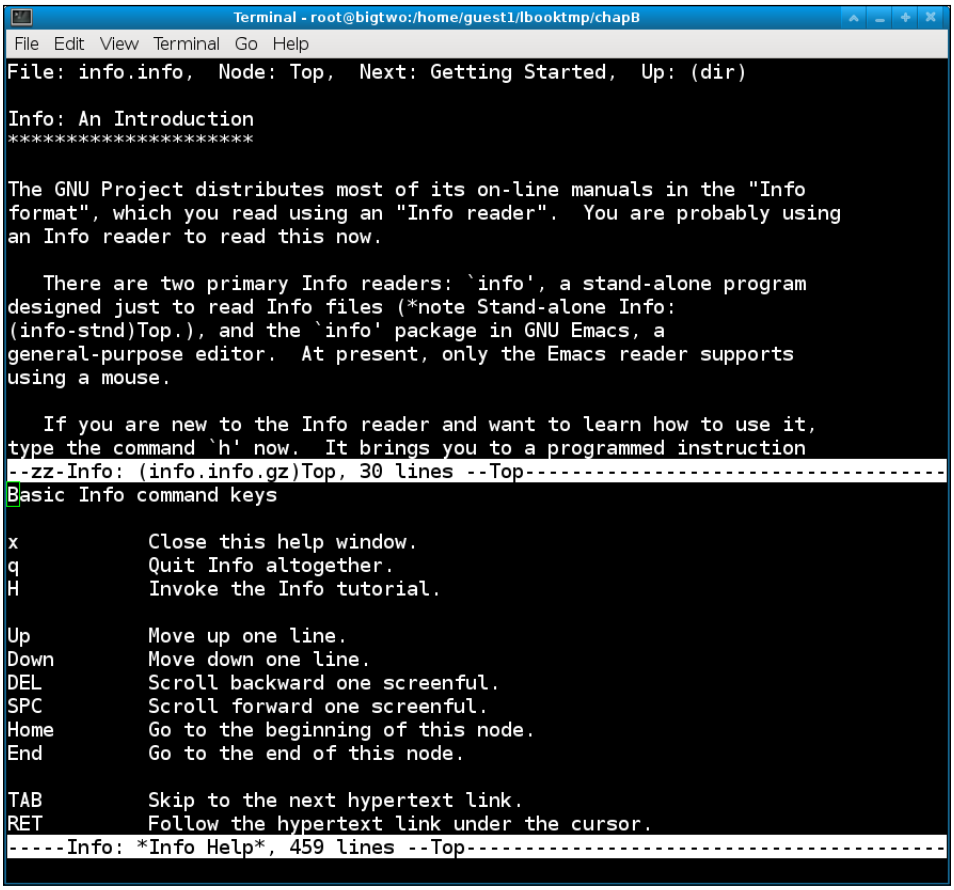
In addition to the man pages, most Linux systems also have `Info` documents. These are accessed by using the `info` program. In general, the data provided by `Info` documents tend to go into more detail and are more informative than a typical `man` page.

Like `man`, you can run info on its own:

```
info info
```

This presents an introduction on how to use `info`. The last paragraph says **If new to info, type 'h' now. This brings you to a programmed instruction sequence**. If you are interested in learning how to use `info` to its fullest, I suggest pressing *H* here to run the tutorial.

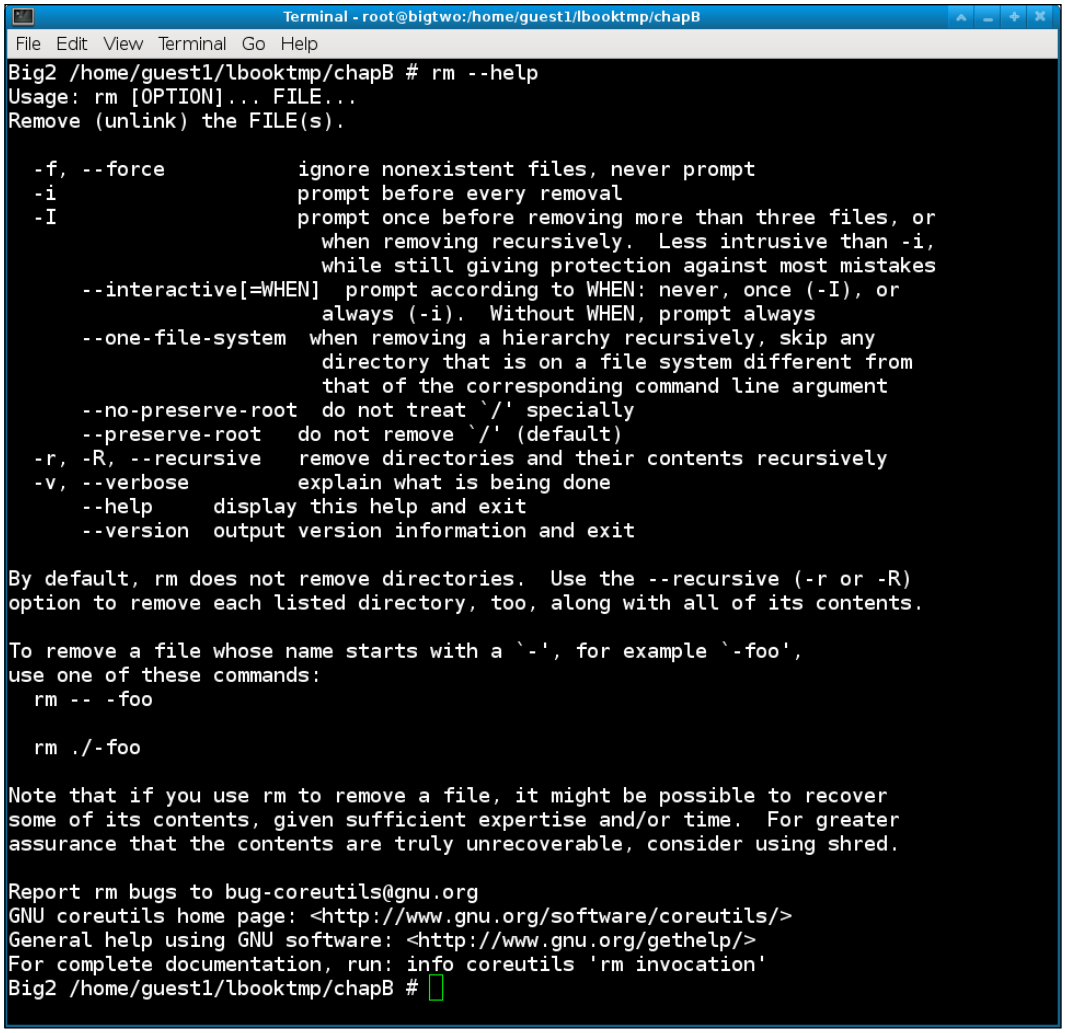
The following is a screenshot of running `info info` and then pressing *H*:



Commands and the Usage section

Most commands in Linux have a `Usage` section, which can be displayed by running it with the `--help` option. Typical examples of this are `cat`, `cut`, `ifconfig`, `bash`, `rm`, and many others.

The following is a screenshot of `rm --help`:



Note that, in general, the `Usage` section is not really intended to teach someone a whole lot about a command. It's really used to remind the user what the parameters are, and the general format of the command.

Note that some commands, especially those that require parameters in order to accomplish something, will display their usage information just by being invoked with no parameters given.

The following is a screenshot of running the `awk` command with no parameters:

Local documentation directories

Most full Linux distributions have directories that contain documentation on various topics. Depending on which distribution is being used, the layout may differ slightly but in most cases, the files are located in the `/usr/share/doc` directory. The following is a partial listing of the `/usr/share/doc` directory taken from Fedora 14:

- `/usr/share/doc/BackupPC-3.1.0`
- `/usr/share/doc/ConsoleKit-0.4.2`
- `/usr/share/doc/Django-1.2.3`
- `/usr/share/doc/GConf2-2.31.91`

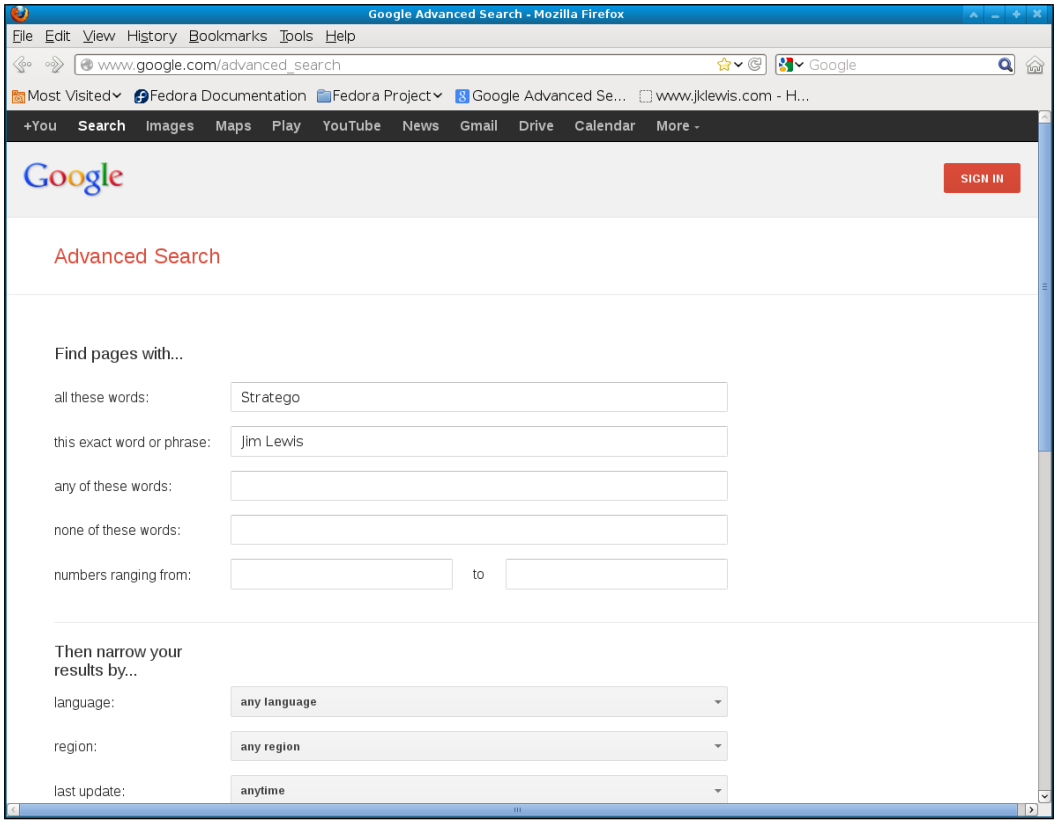
- /usr/share/doc/GeoIP-1.4.7
- /usr/share/doc/GitPython-0.2.0
- /usr/share/doc/HTML
- /usr/share/doc/ImageMagick-6.6.4.1
- /usr/share/doc/ModemManager-0.4
- /usr/share/doc/MySQL-python-1.2.3
- /usr/share/doc/NetworkManager-0.8.1
- /usr/share/doc/abrt-1.1.13
- /usr/share/doc/ant-1.7.1
- /usr/share/doc/apcupsd-3.14.8
- /usr/share/doc/doxygen-1.7.1
- /usr/share/doc/ethtool-2.6.38
- /usr/share/doc/fedora-release-14
- /usr/share/doc/gcc-4.5.1
- /usr/share/doc/gcc-c++-4.5.1
- /usr/share/doc/gimp-2.6.11
- /usr/share/doc/git-1.7.3.1
- /usr/share/doc/gnome-desktop-2.32.0
- /usr/share/doc/gnuchess-5.07
- /usr/share/doc/httpd-2.2.16
- /usr/share/doc/httpd-tools-2.2.16
- /usr/share/doc/java-1.6.0-openjdk-1.6.0.0
- /usr/share/doc/java-1.6.0-openjdk-devel-1.6.0.0
- /usr/share/doc/kaffeine-1.1
- /usr/share/doc/mailx-12.4
- /usr/share/doc/make-3.82
- /usr/share/doc/man-db-2.5.7
- /usr/share/doc/man-pages-3.25

There is also a documentation viewer/browser, which is normally accessed through the file folder dialog. For example, if you open your file manager and go to one of the directories under `/usr/share/doc`, you will see many files. Clicking on the `README` file will bring up more information about that particular program on your system. There may be other readable files as well, such as `CONTENT`, `AUTHOR`, `MAINTAINERS`, `INSTALLATION`, and so on.

Browsing the web to find help

Using the Internet is certainly a great way to find help on Linux tasks. In many cases, it may even be better than relying on local sources as updates may have occurred since the documentation was last put on your system. When I need to look up something using the web, I go directly to Google Advanced Search.

The following is a screenshot of `http://www.google.com/advanced_search` with some of the fields already filled in:



Using this search method is fast and you can use those fields to narrow down what you are looking for.

Keep in mind that there is a lot of information out on the Internet. Some of it is accurate and is exactly what you are looking for. However, in many cases the information is not correct. The person giving the answer may act as though he or she is an expert on the subject when in fact, he or she is not. Many people also don't always check their solution before presenting it as a definitive answer to your question/problem. Be aware of this as you try solutions given on the Internet.

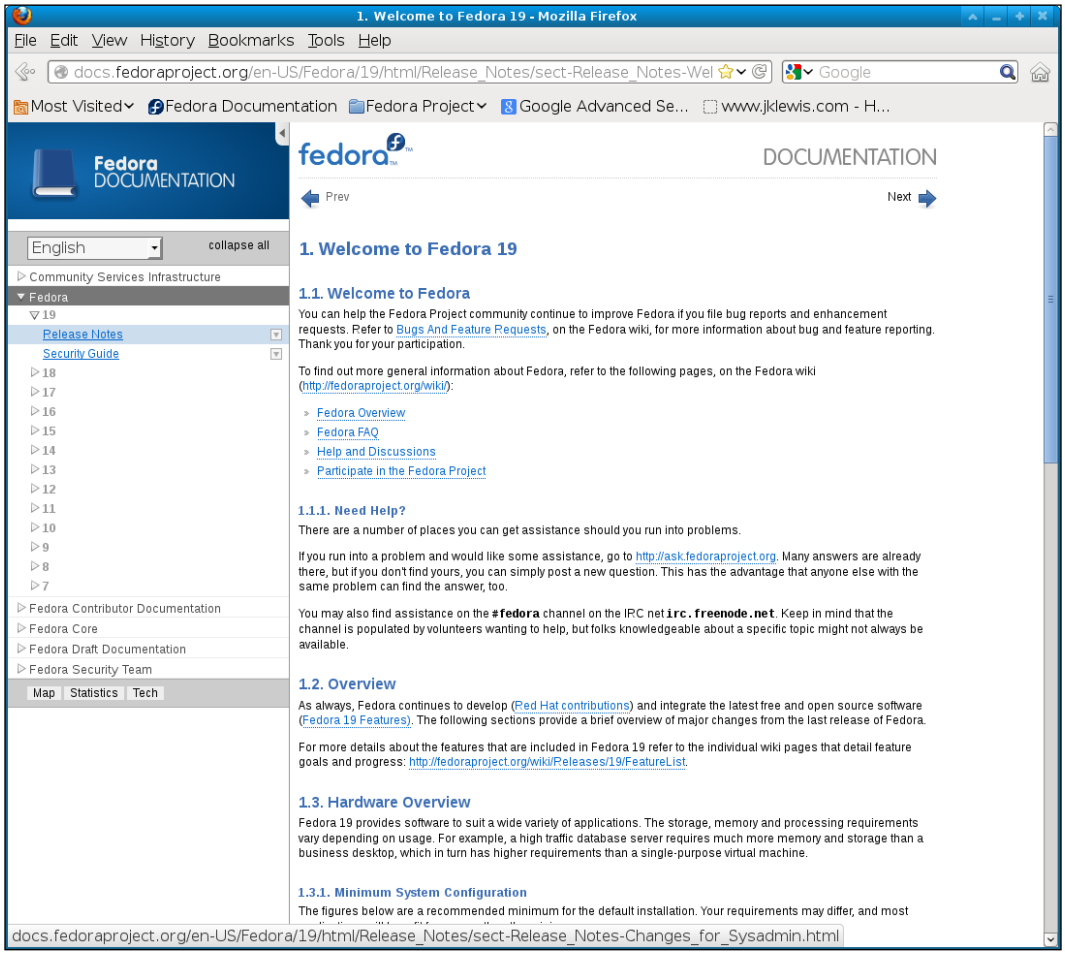
The reverse of this situation is also true. If you want to help people, then that is absolutely great. However, be considerate and test any solution you wish to provide for accuracy before sending it as a reply to their problem.

Distribution release notes

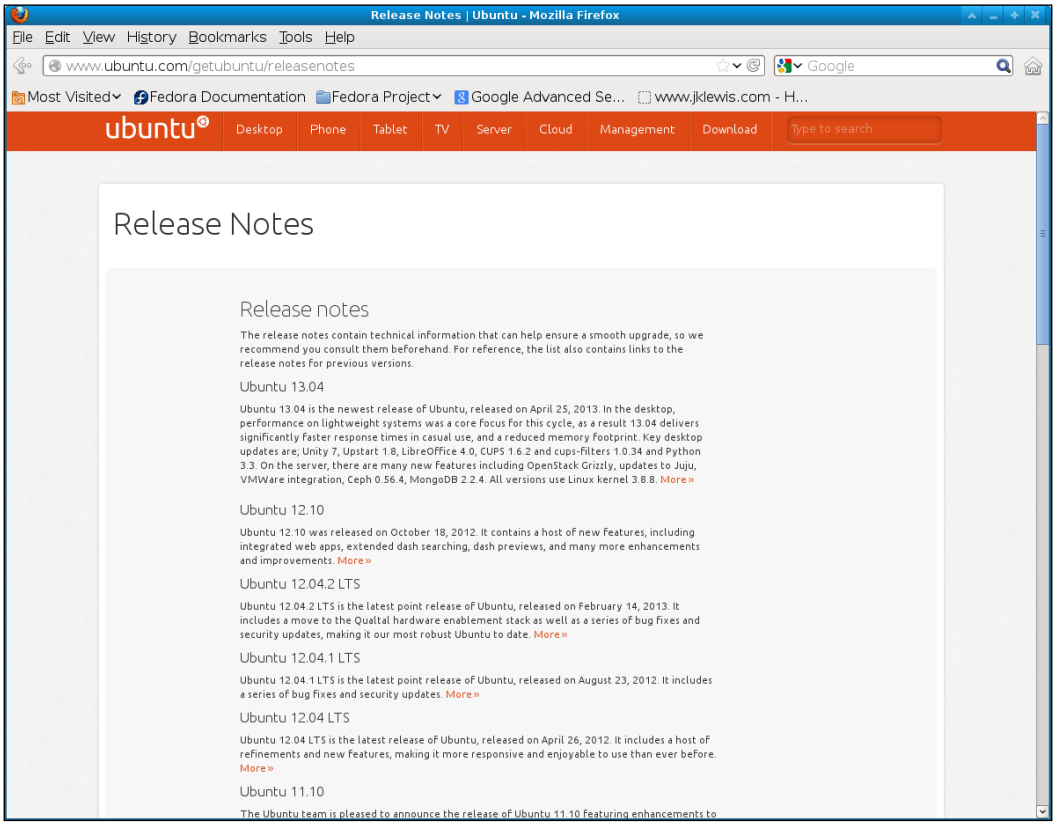
A great way to learn more about your Linux distribution is to look at the release notes for it. These usually contain information such as the following:

- They keep a record of the changes that have been made since the last release. This is usually separated into sections for specific users such as system administrators, desktop users, developers, and others. Note that in some distributions, more information is available in the `Technical Notes` document.
- They spell out the minimum hardware requirements/architecture needed to run the `distro` effectively. Special attention is given to memory, graphics, and video concerns.
- They give installation instructions with emphasis on booting, and special or unusual setups.
- They provide a list of the possible desktop environments that can be installed, often accompanied by the steps to do so. This is a very important section because using a poorly designed and/or buggy desktop will hinder your productivity.
- They have an explanation of the new features, functions, and programs that have been added into the release. This is sometimes followed by the reasoning behind the addition, and what program(s) it replaces.
- They consist of a list of the depreciated (removed) programs and features.
- They have pointers on where to get additional help such as websites and chat rooms.
- They contain a list of the known bugs and problems that still exist in the `distro`, along with information about possible workarounds. Always consult this list before filing a bug report.
- They give instructions on how to provide feedback on both the distribution and release notes, as well as any features you would like to see added/changed.

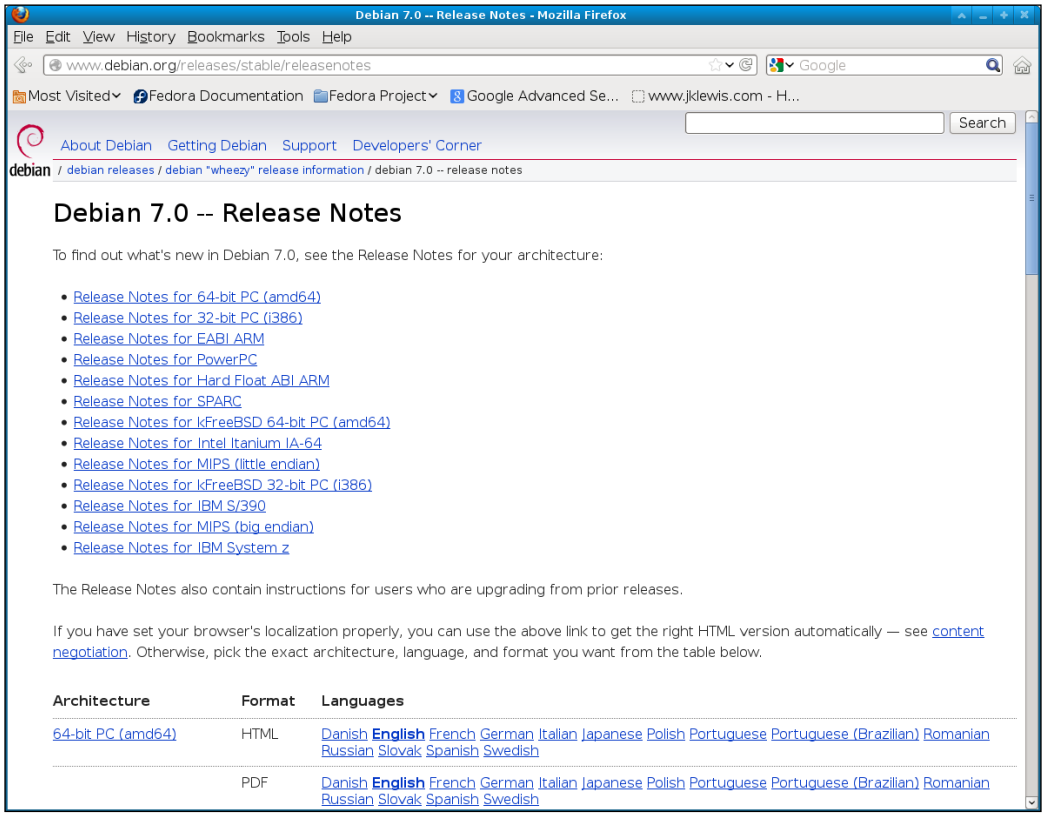
The following is a screenshot of the release notes for Fedora 19 from `http://docs.fedoraproject.org/en-US/Fedora/19/html/Release_Notes/index.html`:



The following is the link to the release notes for Ubuntu 13.04 from `https://wiki.ubuntu.com/RaringRingtail/ReleaseNotes`:



The following screenshot is of Debian 7.0 (Wheezy) taken from `http://www.debian.org/releases/stable/amd64/release-notes`:



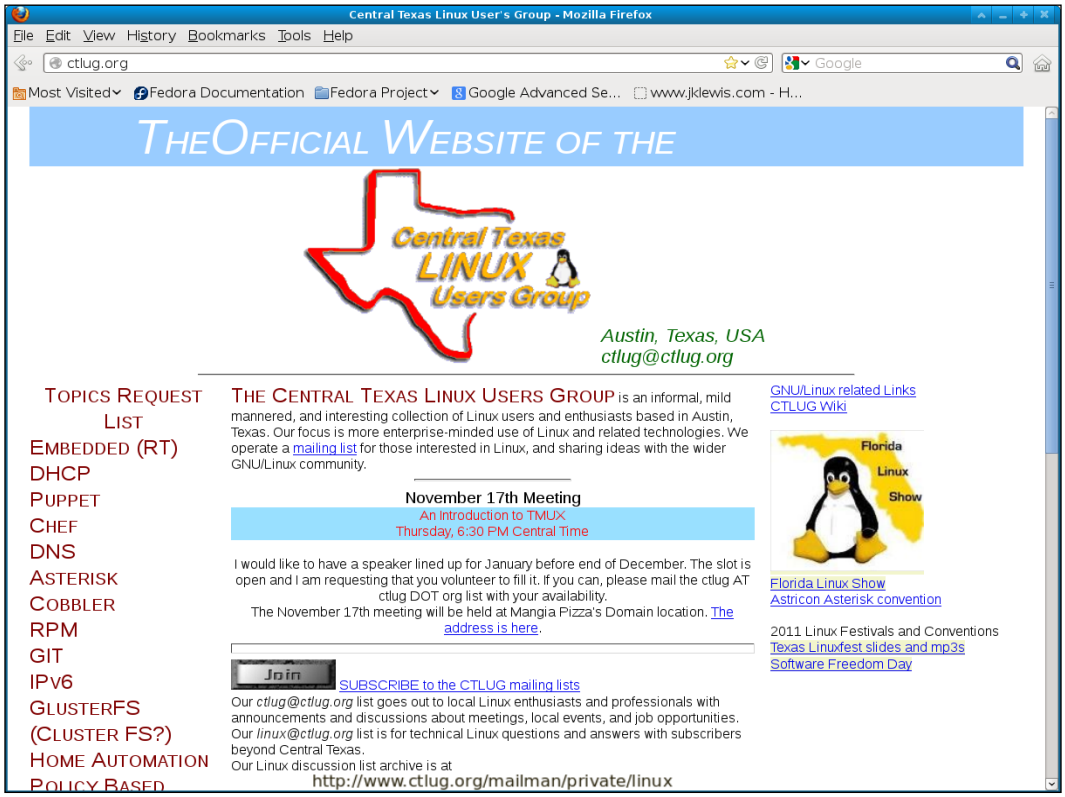
The release notes make for very good reading. I browse them before, during, and after installing a new distribution. This ensures I get the most out of my distribution, informs me of new features, and helps me avoid spending too much time and effort trying to solve a known bug or problem.

Linux users' groups

Another really good place to ask for help is on your local users' group. To find one near you, try an advanced Google search on Linux User Group, and then put in your city (and state if necessary). You should be presented with a few options. Note that most just require a valid e-mail address to get subscribed to the group. In general, to ask a question you simply compose it like a normal one and then e-mail it to the group's e-mail address. People who are knowledgeable in that area are usually quick to provide help and will e-mail back to the group with a possible answer. In most cases, you can also search through the group's archives to find things.

I have found answers to many difficult issues by asking questions on the **Central Texas Linux Users Group** (**CTLUG**).

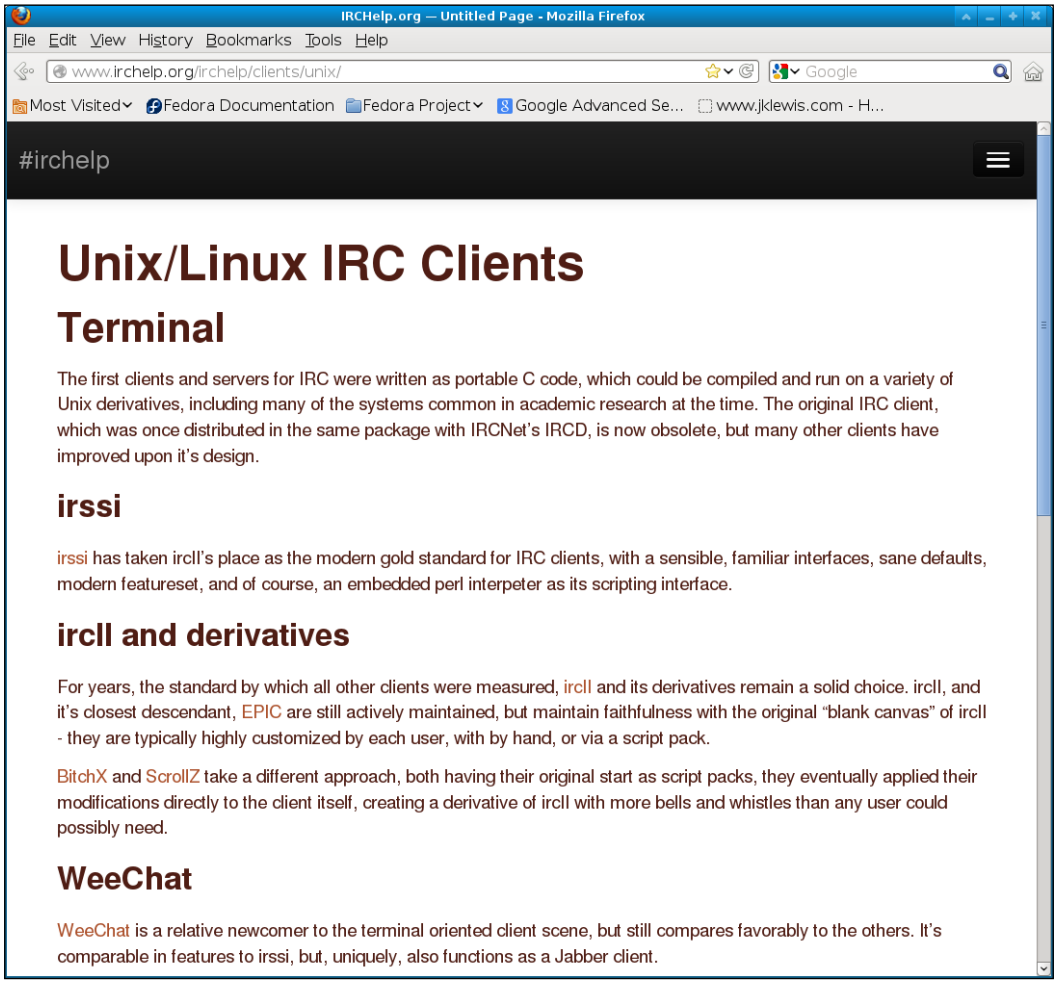
The following is a screenshot of the CTLUG site at `http://ctlug.org/`:



Internet Relay Chat (IRC)

Using IRC is a great way to stay informed about various topics that are of interest to you. It is also a very good place to ask for help. The people who frequent these chat rooms do so by joining channels that pertain to the subject they are interested in and are knowledgeable of. This is all done in real time as well with no need to wait for an e-mail reply to come back. You just need an IRC client, a server, and a group (channel) to join and (in most cases) are ready to go. There are quite a few different IRC clients. Some of these are text- mode (command line) and some are GUI-based.

The following is a screenshot of `http://www.irchelp.org/irchelp/clients/unix/`, a site that shows different IRC clients for Linux and Unix:

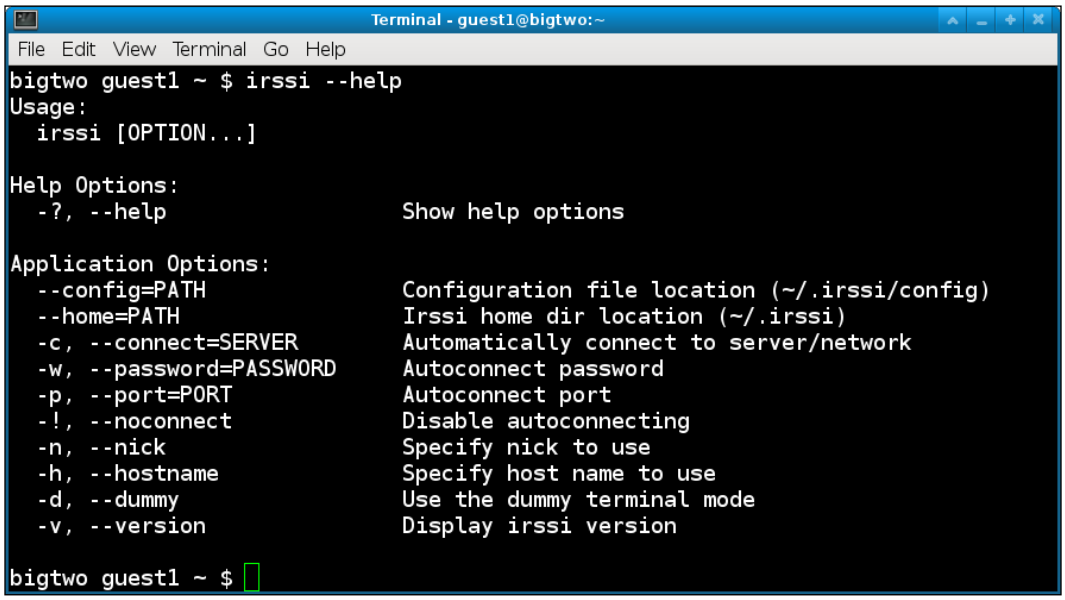


If you are new to IRC, here are a few pointers to get you started. I did not already have one and so began by getting a client. I prefer command line programs and so installed `irssi` on my Fedora 17 machine by running the `yum` command as root:

```
yum -y install irssi
```

This worked just fine.

The following is a screenshot of `irssi -help` on Fedora 17:



This program allows for quite a bit of customization. The default configuration on my system is contained in the `/home/<user>/.irssi/config` file. You can override this using the previous settings. For now, let's just run it plain to see what it looks like.

1. Start by running `irssi`. It should bring up a text mode screen and present you with a welcome message since this is your first time in.
2. Connect to a server. For this example, we will use freenode. Run:

   ```
   /connect irc.freenode.net
   ```

3. You should see another welcome-type message. Now we need a channel. For this example, run the `/join #chat` command (don't forget the # symbol).
4. You should now be connected to freenode through the channel `#chat` and be able to chat with the other users.

Note that `irssi` does take a bit of getting used to. At the bottom is a status screen. You may see something such as **[Act: 2]** or equivalent. This indicates that there is new text present in another window, which you can access by pressing the *Alt* key followed by the number. So, *Alt + 2* will get you to the next screen.

Anything you type in that is not preceded by a `/` symbol, will go to everyone currently in the group. Remember that this is a public forum; be careful with what you say and follow the directions. Also take care to not put personal information in a chat session.

There are quite a few websites that contain information on IRC. The following are a few I found:

- `http://www.irchelp.org/`
- `http://www.linux.org/article/view/irssi-for-beginners-2012`
- `http://www.tldp.org/LDP/sag/html/irc.html`
- `https://wiki.archlinux.org/index.php/IRC_Channel`

There are so many Linux channels available that it is difficult to put them in a list. Some require authentication, while some let you start chatting right away. The best way to find these is to search the Internet for the subject you are looking for help on, and include the phrase IRC. Connect to the appropriate server, join the channel, follow any special directions there might be, and have fun chatting!

Index

Symbols

A

B

C

D

E

F

L

M

N

O

P

R

S

T

U

V

W

X

Z

Thank you for buying
Linux Utilities Cookbook

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

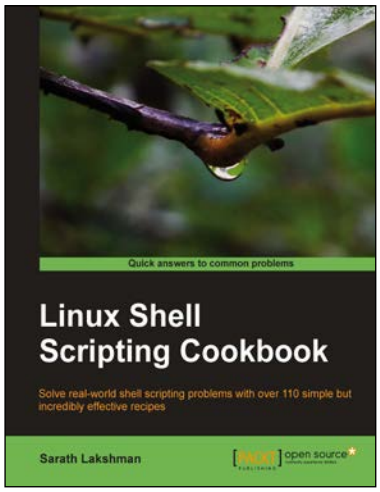
In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] open source
community experience distilled
PUBLISHING



Linux Shell Scripting Cookbook

ISBN: 978-1-84951-376-0 Paperback: 360 pages

Solve real-world shell scripting problems with over 110 simple but incredibly effective recipes

1. Master the art of crafting one-liner command sequence to perform tasks such as text processing, digging data from files, and lot more
2. Practical problem solving techniques adherent to the latest Linux platform
3. Packed with easy-to-follow examples to exercise all the features of the Linux shell scripting language
4. Part of Packt's Cookbook series: Each recipe is a carefully organized sequence of instructions to complete the task as efficiently as possible

---

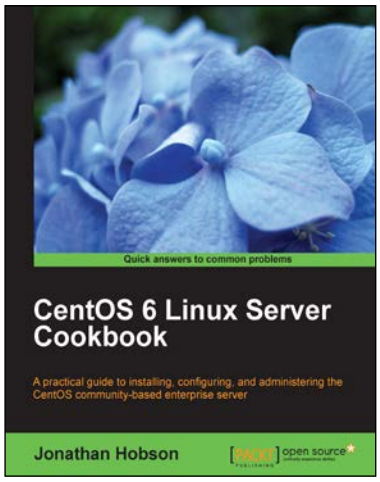
Linux Email

ISBN: 978-1-84719-864-8 Paperback: 376 pages

Set up, maintain, and secure a small office e-mail server

1. Covers all the information you need to easily set up your own Linux email server
2. Learn how to provide web access to email, virus and spam protection, and more
3. Thoroughly covers open source tools like PostFix, Courier, SpamAssassin, and ProcMail
4. A step-by-step approach where the reader is taken through examples with ample screenshots and clear explanations to facilitate learning

[PACKT] open source
community experience distilled
PUBLISHING

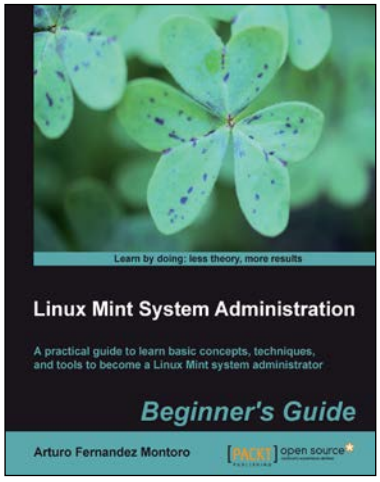


CentOS 6 Linux Server Cookbook

ISBN: 978-1-84951-902-1 Paperback: 374 pages

A practical guide to installing, and administrating the CentOS cmmunity-based enterprise server

1. Delivering comprehensive insight into CentOS server with a series of starting points that show you how to build, configure, maintain and deploy the latest edition of one of the world's most popular community based enterprise servers.
2. Providing beginners and more experienced individuals alike with the opportunity to enhance their knowledge by delivering instant access to a library of recipes that addresses all aspects of CentOS server and put you in control.



Linux Mint System Administrator's Beginner's Guide

ISBN: 978-1-84951-960-1 Paperback: 146 pages

A practical guide to learn basic concepts, techniques, and tools to become a Linux Mint sysytem administrator

1. Discover Linux Mint and learn how to install it
2. Learn basic shell commands and how to deal with user accounts
3. Find out how to carry out system administrator tasks such as monitoring, backups, and network configuration