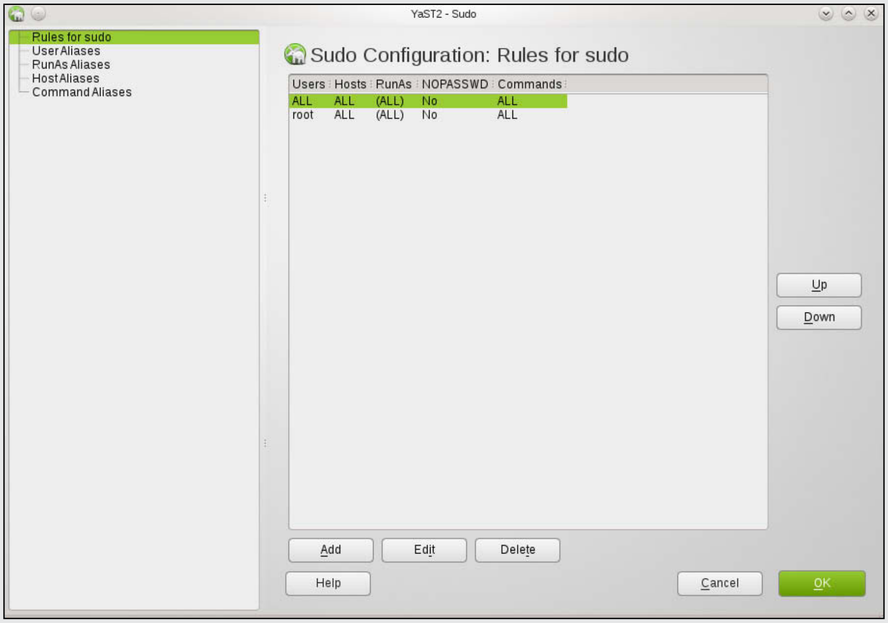
Book V

Administration



Sudo Configuration: Rules for sudo

Users	Hosts	PunAs	NOPASSWD	Commands
ALL	ALL	(ALL)	No	ALL
root	ALL	(ALL)	No	ALL

Visit `www.dummies.com/extras/linuxaio` for great Dummies content online.

Contents at a Glance

Chapter 1: Introducing Basic System Administration 301

Taking Stock of System Administration Tasks 301
Introducing Some GUI Sysadmin Tools....303
How to Become root 305
Understanding How Linux Boots....308
Taking Stock of Linux System Configuration Files....314
Monitoring System Performance....317
Viewing System Information with the /proc File System 322
Understanding Linux Devices....325
Managing Loadable Driver Modules....328
Scheduling Jobs in Linux....330

Chapter 2: Managing Users and Groups 337

Adding User Accounts....337
Understanding the /etc/passwd File....343
Managing Groups 344
Other User and Group Administration Values....345
Exploring the User Environment....347
Changing User and Group Ownership of Files....350

Chapter 3: Managing File Systems 351

Exploring the Linux File System....351
Sharing Files with NFS....358
Backing Up and Restoring Files 361
Accessing a DOS or Windows File System....368

Chapter 4: Working with Samba and NFS 373

Sharing Files with NFS....373
Setting Up a Windows Server Using Samba....377

Chapter 1: Introducing Basic System Administration

In This Chapter

- **Introducing the GUI sysadmin tools**
- **Becoming root**
- **Understanding the system startup process**
- **Taking stock of the system configuration files**
- **Viewing system information through the /proc file system**
- **Monitoring system performance**
- **Managing devices**
- **Scheduling jobs**

System administration, or *sysadmin,* refers to whatever has to be done to keep a computer system up and running. The *system administrator* (the *sysadmin*) is whoever is in charge of taking care of these tasks.

If you're running Linux at home or in a small office, you're most likely the system administrator for your systems. Or maybe you're the system administrator for an entire LAN full of Linux systems. Regardless of your position or title, this chapter will introduce you to basic system administration procedures and show you how to perform some common tasks.

Taking Stock of System Administration Tasks

So what are system administration tasks? An off-the-cuff reply is *anything you have to do to keep the system running well.* More accurately, though, a system administrator's duties include

- **Adding and removing user accounts:** You have to add new user accounts and remove unnecessary user accounts. If a user forgets the password, you have to change the password.
- **Managing the printing system:** You have to turn the print queue on or off, check the print queue's status, and delete print jobs if necessary.
- **Installing, configuring, and upgrading the operating system and various utilities:** You have to install or upgrade parts of the Linux operating system and other software that are part of the operating system.

- **Installing new software:** You have to install software that comes in various package formats, such as RPM or DEB. You also have to download and unpack software that comes in source-code form — and then build executable programs from the source code.
- **Managing hardware:** Sometimes you have to add new hardware and install drivers so the devices work properly.
- **Making backups:** You have to back up files, whether to a DVD drive, a USB memory stick, an external hard drive, or tape.
- **Mounting and unmounting file systems:** When you want to access the files on a CD-ROM, for example, you have to mount that CD-ROM's file system on one of the directories in your Linux file system. You may also have to mount old floppy disks from the archive closet, in both Linux format and DOS format.
- **Automating tasks:** You have to schedule Linux tasks to take place automatically (at specific times) or periodically (at regular intervals).
- **Monitoring the system's performance:** You may want to keep an eye on system performance to see where the processor is spending most of its time and to see the amount of free and used memory in the system.
- **Starting and shutting down the system:** Although starting the system typically involves nothing more than powering up the PC, you do have to take some care when you shut down your Linux system. If your system is set up for a graphical login screen, you can perform the shutdown operation by choosing a menu item from the login screen. Otherwise, use the `shutdown` command to stop all programs before turning off your PC's power switch.
- **Monitoring network status:** If you have a network presence (whether a LAN, a DSL line, or a cable modem connection), you may want to check the status of various network interfaces and make sure your network connection is up and running.
- **Setting up host and network security:** You have to make sure that system files are protected and that your system can defend itself against attacks over the network.
- **Monitoring security:** You have to keep an eye on any intrusions, usually by checking the log files.

That's a long list of tasks! Not all these items are covered in this chapter, but the rest of this minibook describes most of these tasks. The focus in this chapter is on some of the basics, such as using the GUI tools, explaining how to become `root` (the superuser), describing the system configuration files, and showing you how to monitor system performance, manage devices, and set up periodic jobs.

Introducing Some GUI Sysadmin Tools

Each Linux distribution comes with GUI tools for performing system administration tasks. The GUI tools prompt you for input and then run the necessary Linux commands to perform the task. Although there are slight differences among them, the tools have become more and more uniform as time has passed.

For example, Figure 1-1 shows the YaST Control Center available in SUSE, and Figure 1-2 shows the System Settings interface in Ubuntu. Aside from YaST having more options, you should notice that there is great similarity between the interfaces.

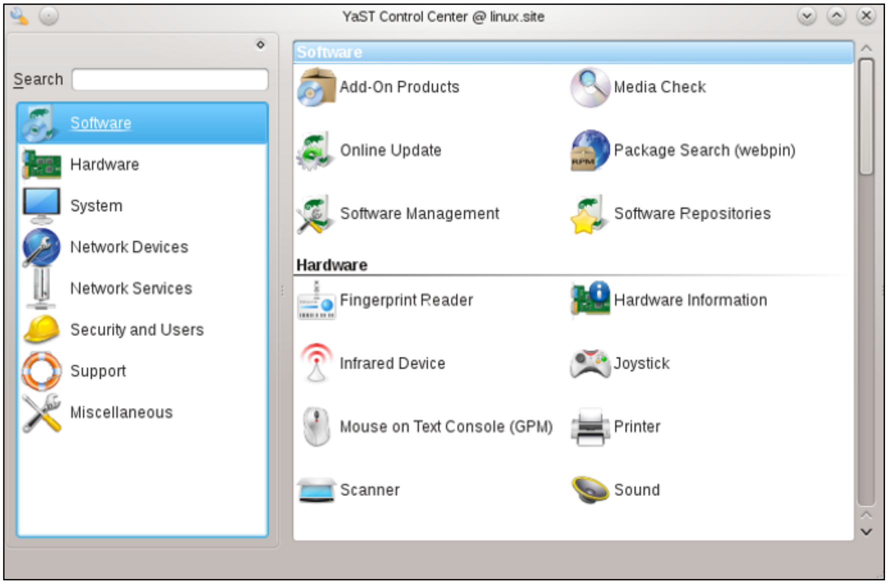


Figure 1-1: YaST Control Center is your starting point for many sysadmin tasks in SUSE.

The left side of the YaST Control Center shows icons for the categories of tasks you can perform. The right side shows icons for specific tasks in the currently selected category. When you click an icon on the right side of the YaST Control Center, a new YaST window appears and enables you to perform that task.

As you can see from the entries in the YaST Control Center, it is truly meant to be a one-stop-shopping spot for all your sysadmin chores.

Figure 1-3 shows the Settings interface in Fedora and you can see that while the icons differ a bit from those found in the other distributions, most of the major chores an administrator deals with are capable of being addressed from here.

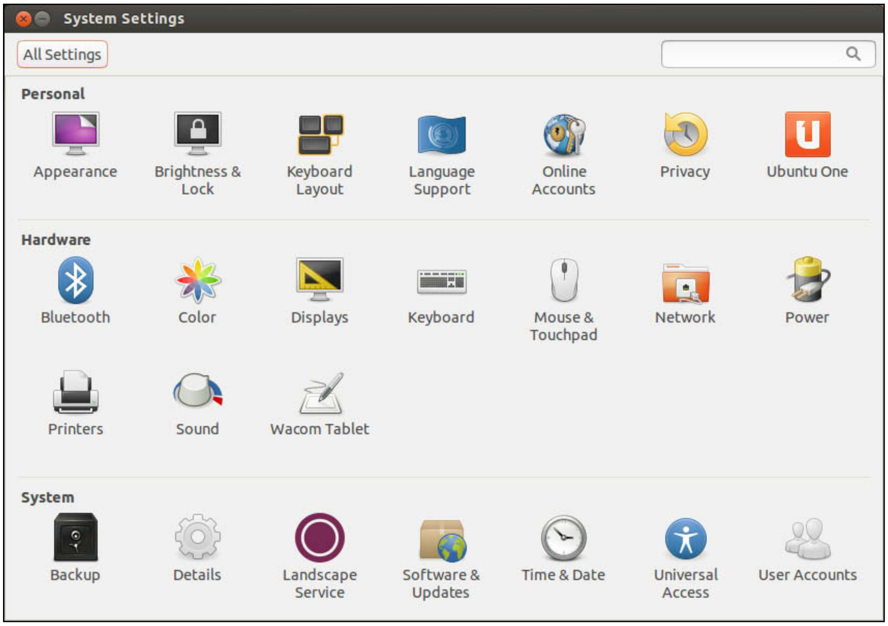


Figure 1-2: The Settings interface in Ubuntu offers many tools similar to those in YaST.

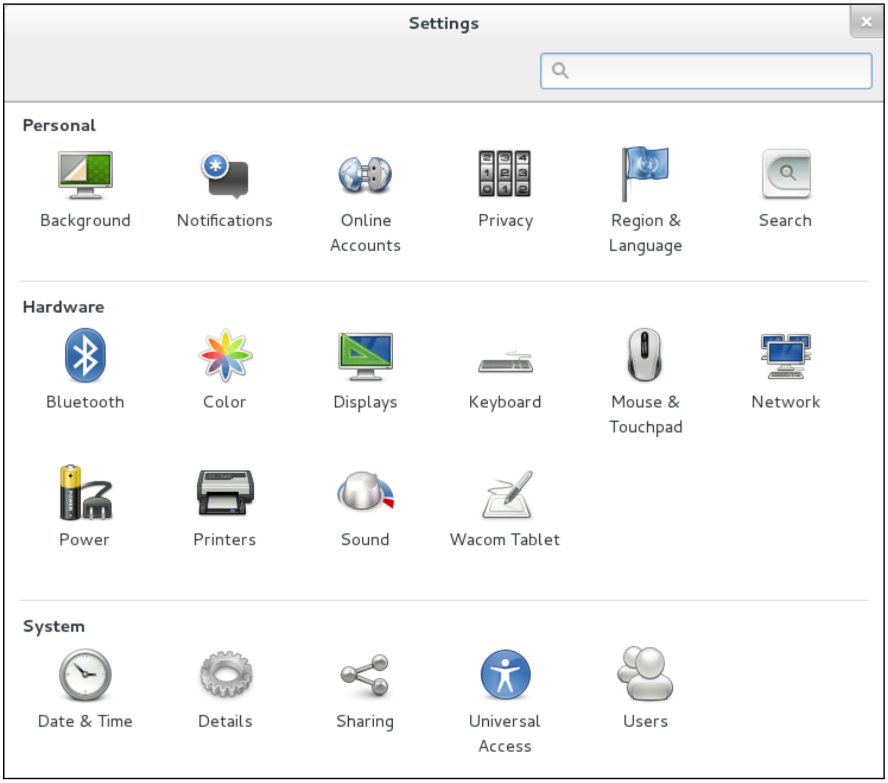


Figure 1-3: The Settings interface in Fedora differs a bit from Ubuntu.

Since YaST is the most complete of the lot, Table 1-1 lists the common tasks found in the Control Center.

Table 1-1 Tasks by Category in the YaST Control Center

This Category	*Enables You to Configure or Manage the Following*
Software	Online Update, Installation Source, Installation in Xen Environment, Installation into Directory, Media Check, Patch CD Update, Software Management, System Update
Hardware	Bluetooth, CD-ROM Drives, Disk Controller, Graphics Card and Monitor, Hardware Information, IDE DMA Mode, Infrared Device, Joystick, Keyboard Layout, Mouse Model, Printer, Scanner, Sound, TV Card
System	`/etc/sysconfig` Editor, Boot Loader Configuration, Boot or Rescue, Date and Time, LVM, Language, PCI Device Drivers, Partitioner, Power Management, Powertweak, Profile Manager, System Backup, System Restoration, System Services (run level)
Network Devices	DSL, Fax, ISDN, Modem, Network Card, Phone Answering Machine
Network Services	DHCP Server, DNS Server, DNS Host and Name, HTTP Server, Host Names, Kerberos Client, LDAP Client, Mail Transfer Agent, NFS Client, NFS Server, NIS Client, NIS Server, NTP Client, Network Services (`xinetd`), Proxy, Remote Administration, Routing, SLP Browser, Samba Client, Samba Server, TFTP Server
Security and Users	Firewall, Group Management, Local Security, User Management
Support	Where you can find documentation and help resources
Miscellaneous	Autoinstallation, Post a Support Query, Vendor Driver CD, View Start-up Log, View System Log

How to Become root

You have to log in as `root` to perform system administration tasks. The `root` user is the superuser and the only account with all the privileges needed to do anything in the system.

Common wisdom says you should *not* normally log in as `root`. When you're `root`, you can easily delete all the files with one misstep — especially when you're typing commands. For example, you type the command `rm *.html` to delete all files that have the `.html` extension. But what if you accidentally press the spacebar after the asterisk (`*`)? The shell takes the command to be `rm * .html` and — because `*` matches any filename — deletes everything in the current directory. Seems implausible until it happens to you!

Using the su - command

If you're logged in as a normal user, how do you do any system administration chores? Well, you become root for the time being. If you're working at a terminal window or console, type

```
su -
```

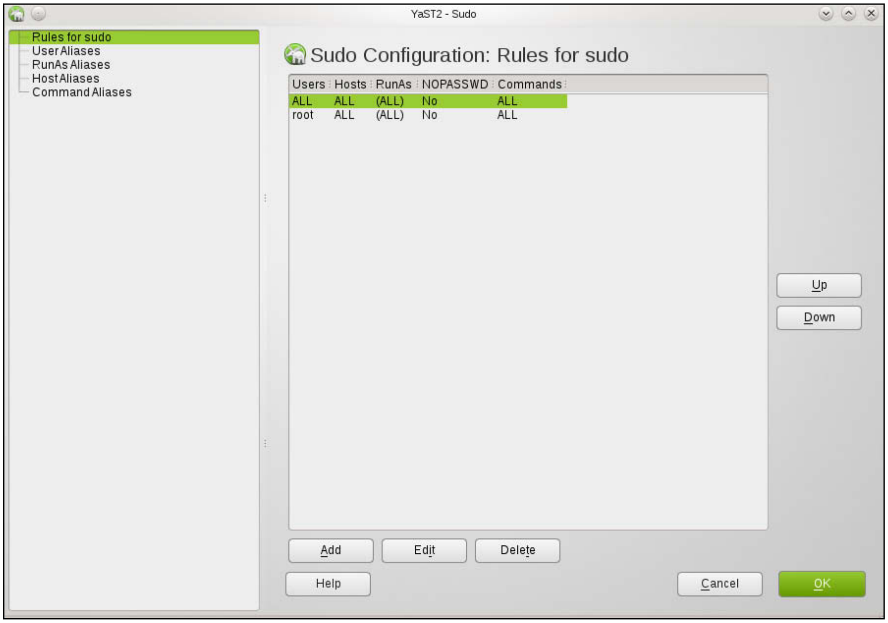
Then enter the root password in response to the prompt. From this point, you're root. Do whatever you have to do. To return to your usual self, type

```
exit
```

That's it! It's that easy.

By the way, Knoppix has a root user but doesn't have a root password, so you can become root by simply typing **su -** at the shell prompt in a terminal window. Also, Ubuntu doesn't have a root user. To perform any task that requires root privileges in Ubuntu, you must type **sudo** followed by the command and then provide your normal user password when prompted.

In the Security minibook, the issue will be addressed more, but you can control who can use sudo through configuration files (or the YaST interface, as Figure 1-4 shows).



YaST2 - Sudo

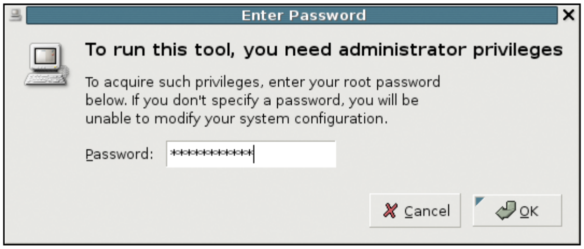
Sudo Configuration: Rules for sudo

Users	Hosts	RunAs	NOPASSWD	Commands
ALL	ALL	(ALL)	No	ALL
root	ALL	(ALL)	No	ALL

Figure 1-4: Configuring who can use sudo.

Becoming root for the GUI utilities

Most Linux distributions include GUI utilities to perform system administration chores. If you use any of these GUI utilities to perform a task that requires you to be `root`, the utility typically pops up a dialog box that prompts you for the `root` password, as shown in Figure 1-5 (except in Ubuntu, where the GUI tools prompt for your normal user password). Just type the password and press Enter. If you don't want to use the utility, click Cancel.



Enter Password

To run this tool, you need administrator privileges

To acquire such privileges, enter your root password below. If you don't specify a password, you will be unable to modify your system configuration.

Password: ************

Figure 1-5: Type the root password and press Enter to gain root privileges.

Recovering from a forgotten root password

To perform system administration tasks, you have to know the `root` password. What happens if you forget the `root` password? Not to worry. You can reset the `root` password by following these steps:

1. **Reboot the PC (select Reboot as you log out of the GUI screen) or power up as usual.**

   As soon you see the graphical GRUB boot loader screen that shows the names of the operating systems, you can boot. If your system runs the LILO boot loader, press Ctrl+X, type **linux single** at the `boot:` prompt, and press Enter. Then proceed to Step 4. If you don't see the graphical loader screen on reboot, it might not be installed (which can occasionally occur when choosing to install from a Live CD). If this is the case, it is recommended that you reinstall from the CD.

2. **If you have more than one operating system installed, use the arrow key to select Linux as your operating system and then press the A key.**

   GRUB prompts you for commands to add to its default boot command.

3. **Press the spacebar, and then type** single **and press Enter.**

   Linux starts as usual but runs in a single-user mode that doesn't require you to log in. After Linux starts, you see the following command-line prompt similar to the following:

   ```
   sh-3.00#
   ```

4. **Type the** `passwd` **command to change the** `root` **password as follows:**

   ```
   sh-3.00# passwd
   Changing password for user root.
   New password:
   ```

5. **Type the new** `root` **password that you want to use (it doesn't appear onscreen) and then press Enter.**

   Linux asks for the password again, like this:

   ```
   Retype new password:
   ```

6. **Type the password again and press Enter.**

   If you enter the same password both times, the `passwd` command changes the `root` password.

7. **Now type** reboot **and press Enter to reboot the PC.**

   After Linux starts, it displays the familiar login screen. Now you can log in as `root` with the new password.

DISTRIBUTION SPECIFIC

In SUSE Linux, in Step 3, type **single init=/bin/sh** (instead of **single**) and before proceeding to Step 4, at the command-line prompt, type **mount / -n -0 remount,rw**. Then perform Steps 4 through 6 to change the `root` password. After changing the password, type **mount / -n -o remount,ro**. Then continue to Step 7 and reboot the PC.

WARNING!

Make sure that your Linux PC is *physically* secure. As these steps show, anyone who can physically access your Linux PC can simply reboot, set a new `root` password, and do whatever he or she wants with the system. Another way to protect against resetting the password is to set a GRUB password, which causes GRUB to require a valid password before it boots Linux. Of course, you must then remember to enter the GRUB password every time you boot your system!

Understanding How Linux Boots

Knowing the sequence in which Linux starts processes as it boots is important. You can use this knowledge to start and stop services, such as the web server and Network File System (NFS). The next few sections provide you with an overview of how Linux boots and starts the initial set of processes. These sections also familiarize you with the shell scripts that start various services on a Linux system.

Understanding the init process

When Linux boots, it loads and runs the core operating system program from the hard drive. The core operating system is designed to run other programs. A process named `init` starts the initial set of processes on your Linux system.

To see the processes currently running on the system, type

```
ps ax | more
```

You get an output listing that starts like this:

```
PID TTY STAT TIME COMMAND
1 ? S 0:22 init [2]
```

The first column, with the heading `PID`, shows a number for each process. *PID* stands for process ID (identification) — a sequential number assigned by the Linux kernel. The first entry in the process list, with a PID of `1`, is the `init` process. It's the first process, and it starts all other processes in your Linux system. That's why `init` is sometimes referred to as the *mother of all processes.*

What the `init` process starts depends on

- **The run level,** an identifier that identifies a system configuration in which only a selected group of processes can exist.
- **The contents of the** `/etc/inittab` **file,** a text file that specifies which processes to start at different run levels.
- **A number of shell scripts that are executed at specific run levels.** (The scripts are located in the `/etc/init.d` directory and a number of subdirectories in `/etc` — these subdirectories have names that begin with `rc`.)

Most Linux distributions use seven run levels — `0` through `6`. The meaning of the run levels differs from one distribution to another. Table 1-2 shows the meanings of the run levels and points out some of the actions specific to Fedora, Debian, SUSE, Ubuntu, and Xandros.

Table 1-2 Run Levels in Linux

Run Level	*Meaning*
0	Shut down the system.
1	Run in single-user standalone mode (no one else can log in; you work at the text console).
2	Run in multiuser mode (Debian, Ubuntu, and Xandros use run level 2 as the default run level).
3	Run in full multiuser mode (used for text mode login in Fedora and SUSE).
4	Run in full multiuser mode (unused in Fedora and SUSE).
5	Run in full multiuser mode (used as the default run level with graphical login in Fedora and SUSE).
6	Reboot the system.

The current run level together with the contents of the /etc/inittab file control which processes init starts in Linux. The default run level is 2 in Debian, Ubuntu, and Xandros. In Fedora and SUSE, run level 3 is used for text mode login screens and 5 for the graphical login screen. You can change the default run level by editing a line in the /etc/inittab file.

To check the current run level, type the following command in a terminal window:

```
/sbin/runlevel
```

In Debian, the runlevel command prints an output like this:

```
N 2
```

The first character of the output shows the previous run level (N means no previous run level), and the second character shows the current run level (2). In this case, the system started at run level 2. If you're in a GUI desktop in Fedora, the runlevel command should show 5 as the current run level.

Examining the /etc/inittab file

The /etc/inittab file is the key to understanding the processes that init starts at various run levels. You can look at the contents of the file by using the more command, as follows:

```
more /etc/inittab
```

To see the contents of the /etc/inittab file with the more command, you don't have to log in as root.

To interpret the contents of the /etc/inittab file, follow these steps:

1. **Look for the line that contains the phrase initdefault.**

   Here's that line from the /etc/inittab file from a Debian system:

   ```
   id:2:initdefault:
   ```

   That line shows the default run level. In this case, it's 2.

2. **Find all the lines that specify what init runs at run level 2.**

   Look for a line that has a 2 between the first two colons (:). Here's a relevant line in Debian:

   ```
   l2:2:wait:/etc/init.d/rc 2
   ```

   This line specifies that init executes the file /etc/init.d/rc with 2 as an argument.

If you look at the file /etc/init.d/rc in a Debian system, you find it's a shell script. You can study this file to see how it starts various processes for run levels 1 through 5.

Each entry in the /etc/inittab file tells init what to do at one or more run levels — you simply list all run levels at which the process runs. Each inittab entry has four fields — separated by colons — in the following format:

```
id:runlevels:action:process
```

Table 1-3 shows what each field means.

Table 1-3 Fields in Each inittab Entry

Field	*Meaning*
id	A unique one- or two-character identifier. The init process uses this field internally. You can use any identifier you want, as long as you don't use the same identifier on more than one line.
runlevels	A sequence of zero or more characters, each denoting a run level. For example, if the runlevels field is 12345, that entry applies to each of the run levels 1 through 5. This field is ignored if the action field is set to sysinit, boot, or bootwait.
action	What the init process will do with this entry. If this field is initdefault, for example, init interprets the runlevels field as the default run level. If this field is set to wait, init starts the program or script specified in the process field and waits until that process exits.
process	Name of the script or program that init starts. Some settings of the action field require no process field. For example, when the action field is initdefault, there's no need for a process field.

Trying a new run level with the init command

To try a new run level, you don't have to change the default run level in the /etc/inittab file. If you log in as root, you can change the run level (and, consequently, the processes that run in Linux) by typing **init** followed by the run level.

For example, to put the system in single-user mode, type the following:

```
init 1
```

Thus, if you want to try run level 3 without changing the default run level in /etc/inittab file, enter the following command at the shell prompt:

```
init 3
```

The system ends all current processes and enters run level 3. By default, the `init` command waits 20 seconds before stopping all current processes and starting the new processes for run level 3.

To switch to run level 3 immediately, type the command `init -t0 3`. The number after the `-t` option indicates the number of seconds init waits before changing the run level.

You can also use the `telinit` command, which is simply a symbolic link (a shortcut) to `init`. If you make changes to the `/etc/inittab` file and want `init` to reload its configuration file, use the command `telinit q`.

Understanding the Linux startup scripts

The `init` process runs a number of scripts at system startup. In the following discussions, a Debian system is used as an example, but the basic sequence is similar in other distributions — only the names and locations of the scripts may vary.

If you look at the `/etc/inittab` file in a Debian system, you find the following lines near the beginning of the file:

```
# Boot-time system configuration/initialization script.
si::sysinit:/etc/init.d/rcS
```

The first line is a comment line. The second line causes `init` to run the `/etc/init.d/rcS` script — the first Linux startup script that `init` runs in a Debian system. The `rcS` script performs many initialization tasks, such as mounting the file systems, setting the clock, configuring the keyboard layout, starting the network, and loading many other driver modules. The `rcS` script performs these initialization tasks by calling many other scripts and reading configuration files located in the `/etc/rcS.d` directory.

After executing the `/etc/init.d/rcS` script, the `init` process runs the `/etc/init.d/rc` script with the run level as an argument. For example, for run level 2, the following line in `/etc/inittab` specifies what `init` executes:

```
l2:2:wait:/etc/init.d/rc 2
```

This example says `init` executes the command `/etc/init.d/rc 2` and waits until that command completes.

The `/etc/init.d/rc` script is somewhat complicated. Here's how it works:

- It executes scripts in a directory corresponding to the run level. For example, for run level 2, the `/etc/init.d/rc` script runs the scripts in the `/etc/rc2.d` directory.

- ✦ In the directory that corresponds with the run level, /etc/init.d/rc looks for all files that begin with K and executes each of them with the stop argument. This argument kills any currently running processes. Then it locates all files that begin with S and executes each file with a start argument. This argument starts the processes needed for the specified run level.

To see it executed at run level 2, type the following command:

```
ls -l /etc/rc2.d
```

In the resulting listing, the K scripts — the files whose names begin with K — stop (or *kill*) servers, whereas the S scripts start servers. The /etc/init.d/rc script executes these files in the order in which they appear in the directory listing.

Manually starting and stopping servers

In Linux, the server startup scripts reside in the /etc/init.d directory. You can manually invoke scripts in this directory to start, stop, or restart specific processes — usually servers. For example, to stop the FTP server (the server program is vsftpd), type the following command:

```
/etc/init.d/vsftpd stop
```

If vsftpd is already running and you want to restart it, type the following command:

```
/etc/init.d/vsftpd restart
```

You can enhance your system administration skills by familiarizing yourself with the scripts in the /etc/init.d directory. To see its listing, type the following command:

```
ls /etc/init.d
```

The script names give you some clue about which server the script can start and stop. For example, the samba script starts and stops the processes required for Samba Windows networking services. At your leisure, you may want to study some of these scripts to see what each one does. You don't have to understand all the shell programming; the comments help you discover the purpose of each script.

Automatically starting servers at system startup

You want some servers to start automatically every time you boot the system. The exact commands to configure the servers vary from one distribution to another.

In Fedora and SUSE, use the `chkconfig` command to set up a server to start whenever the system boots into a specific run level. For example, if you start the SSH server, you want the `sshd` server to start whenever the system starts. You can make that happen by using the `chkconfig` command. To set `sshd` to start whenever the system boots into run level 3, 4, or 5, type the following command (while logged in as `root`):

```
chkconfig --level 345 sshd on
```

In Fedora and SUSE, you can also use the `chkconfig` command to check which servers are turned on or off. For example, to see the complete list of all servers for all run levels, type the following command:

```
chkconfig --list
```

In Debian, Ubuntu, and Xandros, you can use the `update-rc.d` command to enable a server to start automatically at system startup. For example, to set `sshd` to start automatically at the default run levels, type **update-rc.d sshd defaults** in a terminal window while logged in as `root`. You can also specify the exact run levels and the sequence number (the order in which each server starts). To find out more about the `update-rc.d` command, type **man update-rc.d** in a terminal window.

Taking Stock of Linux System Configuration Files

Linux includes a host of configuration files. All these files share text files that you can edit with any text editor. To edit these configuration files, you must log in as `root`. A selection of the most popular configuration files appears in Table 1-4, along with a brief description of each. This table gives you an idea of what types of configuration files a system administrator has to work with. In many cases, Linux includes GUI utility programs to set up many of these configuration files.

Table 1-4 Some Linux Configuration Files

Configuration File	*Description*
`/boot/grub`	Location of files for the GRUB boot loader.
`/boot/grub/menu.lst`	Configuration file for the boot menu that GRUB displays before it boots your system.
`/boot/System.map`	Map of the Linux kernel (maps kernel addresses into names of functions and variables).
`/boot/vmlinuz`	The Linux kernel (the operating system's core).

Configuration File	*Description*
`/etc/apache2/httpd.conf`	Configuration file for the Apache web server (Debian).
`/etc/apt/sources.list`	Configuration file that lists the sources — FTP or websites or CD-ROM — from which the Advanced Packaging Tool (APT) obtains packages (Debian, Ubuntu, and Xandros).
`/etc/at.allow`	Usernames of users allowed to use the `at` command to schedule jobs for later execution.
`/etc/at.deny`	Usernames of users forbidden to use the `at` command.
`/etc/bashrc`	System-wide functions and aliases for the `bash` shell (Fedora).
`/etc/bash.bashrc`	System-wide functions and aliases for the `bash` shell (Debian, SUSE, Ubuntu, and Xandros).
`/etc/cups/cupsd.conf`	Printer configuration file for the Common Unix Printing System (CUPS) scheduler.
`/etc/fonts`	Directory with font configuration files. (In particular, you can put local font configuration settings in the file `/etc/fonts/local.conf`.)
`/etc/fstab`	Information about file systems available for mounting and where each file system is to be mounted.
`/etc/group`	Information about groups.
`/etc/grub.conf`	The configuration for the GRUB boot loader in Fedora and SUSE.
`/etc/hosts`	List of IP numbers and their corresponding hostnames.
`/etc/hosts.allow`	Hosts allowed to access Internet services on this system.
`/etc/hosts.deny`	Hosts forbidden to access Internet services on this system.
`/etc/httpd/conf/httpd.conf`	Configuration file for the Apache web server (Fedora).
`/etc/init.d`	Directory with scripts to start and stop various servers.
`/etc/inittab`	Configuration file used by the `init` process that starts all the other processes.

(continued)

Table 1-4 *(continued)*

Configuration File	*Description*
`/etc/issue`	File containing a message to be printed before displaying the text mode login prompt (usually the distribution name and the version number).
`/etc/lilo.conf`	The configuration for the Linux Loader (LILO) — one of the boot loaders that can load the operating system from disk (present only if you use the LILO boot loader).
`/etc/login.defs`	Default information for creating user accounts, used by the `useradd` command.
`/etc/modprobe.conf`	Configuration file for loadable kernel modules, used by the `modprobe` command (Fedora and SUSE).
`/etc/modules.conf`	Configuration file for loadable modules (Debian and Xandros).
`/etc/mtab`	Information about currently mounted file systems.
`/etc/passwd`	Information about all user accounts. (Actual passwords are stored in `/etc/shadow`.)
`/etc/profile`	System-wide environment and startup file for the `bash` shell.
`/etc/profile.d`	Directory containing script files (with names that end in `.sh`) that the `/etc/profile` script executes.
`/etc/init.d/rcS`	Linux initialization script in Debian, SUSE, Ubuntu, and Xandros.
`/etc/rc.d/rc.sysinit`	Linux initialization script in Fedora.
`/etc/shadow`	Secure file with encrypted passwords for all user accounts (can be read by only `root`).
`/etc/shells`	List of all the shells on the system that the user can use.
`/etc/skel`	Directory that holds initial versions of files such as `.bash_profile` that copy to a new user's home directory.
`/etc/sysconfig`	Linux configuration files (Fedora and SUSE).
`/etc/sysctl.conf`	Configuration file with kernel parameters that are read in and set by `sysctl` at system startup.
`/etc/termcap`	Database of terminal capabilities and options (Fedora and SUSE).

Configuration File	Description
/etc/udev	Directory containing configuration files for udev — the program that provides the ability to dynamically name hot-pluggable devices and create device files in the /dev directory.
/etc/X11	Directory with configuration files for the X Window System (X11) and various display managers such as gdm and xdm.
/etc/X11/xorg.conf	Configuration file for X.Org X11 — the X Window System (Fedora, Ubuntu, and SUSE).
/etc/xinetd.conf	Configuration for the xinetd daemon that starts a number of Internet services on demand.
/etc/yum.conf	Configuration for the Yum package updater and installer (Fedora).
/var/log/apache2	Web server access and error logs (Debian).
/var/log/cron	Log file with messages from the cron process that runs scheduled jobs.
/var/log/boot.msg	File with boot messages (SUSE).
/var/log/dmesg	File with boot messages (Debian, Fedora, Ubuntu, and Xandros).
/var/log/httpd	Web server access and error logs (Fedora).
/var/log/messages	System log.

Monitoring System Performance

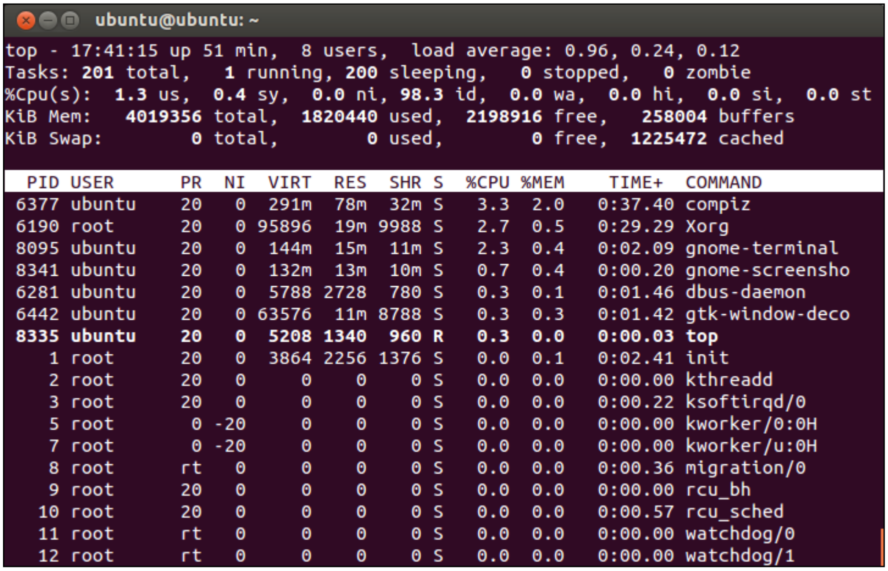
When you're the system administrator, you must keep an eye on how well your Linux system is performing. You can monitor the overall performance of your system by looking at information such as

- Central processing unit (CPU) usage
- Physical memory usage
- Virtual memory (swap-space) usage
- Hard drive usage

Linux comes with a number of utilities that you can use to monitor one or more of these performance parameters. The following sections introduce a few of these utilities and show you how to understand the information presented by said utilities.

Using the top utility

To view the top CPU processes — the ones that use most of the CPU time — you can use the text mode `top` utility. To start that utility, type **top** in a terminal window (or text console). The `top` utility then displays a text screen listing the current processes, arranged in the order of CPU usage, along with various other information, such as memory and swap-space usage. Figure 1-6 shows a typical output from the `top` utility.



```
top - 17:41:15 up 51 min,  8 users,  load average: 0.96, 0.24, 0.12
Tasks: 201 total,   1 running, 200 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.3 us,  0.4 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   4019356 total,  1820440 used,  2198916 free,   258004 buffers
KiB Swap:        0 total,        0 used,        0 free,  1225472 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 6377 ubuntu    20   0  291m  78m  32m S   3.3  2.0   0:37.40 compiz
 6190 root      20   0 95896  19m 9988 S   2.7  0.5   0:29.29 Xorg
 8095 ubuntu    20   0  144m  15m  11m S   2.3  0.4   0:02.09 gnome-terminal
 8341 ubuntu    20   0  132m  13m  10m S   0.7  0.4   0:00.20 gnome-screensho
 6281 ubuntu    20   0  5788 2728  780 S   0.3  0.1   0:01.46 dbus-daemon
 6442 ubuntu    20   0 63576  11m 8788 S   0.3  0.3   0:01.42 gtk-window-deco
 8335 ubuntu    20   0  5208 1340  960 R   0.3  0.0   0:00.03 top
    1 root      20   0  3864 2256 1376 S   0.0  0.1   0:02.41 init
    2 root      20   0     0    0    0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0.0  0.0   0:00.22 ksoftirqd/0
    5 root       0 -20     0    0    0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root       0 -20     0    0    0 S   0.0  0.0   0:00.00 kworker/u:0H
    8 root      rt   0     0    0    0 S   0.0  0.0   0:00.36 migration/0
    9 root      20   0     0    0    0 S   0.0  0.0   0:00.00 rcu_bh
   10 root      20   0     0    0    0 S   0.0  0.0   0:00.57 rcu_sched
   11 root      rt   0     0    0    0 S   0.0  0.0   0:00.00 watchdog/0
   12 root      rt   0     0    0    0 S   0.0  0.0   0:00.00 watchdog/1
```

Figure 1-6: You can see the top CPU processes by using the top utility.

TIP

The `top` utility updates the display every five seconds. If you keep `top` running in a window, you can continually monitor the status of your Linux system. To quit `top`, press Q or Ctrl+C or close the terminal window.

The first five lines of the output screen (see Figure 1-6) provide summary information about the system, as follows:

- The first line shows the current time, how long the system has been up, how many users are logged in, and three *load averages* — the average number of processes ready to run during the last 1, 5, and 15 minutes.
- The second line lists the total number of processes and the status of these processes.
- The third line shows CPU usage — what percentage of CPU time is used by user processes, what percentage by system (kernel) processes, and during what percentage of time the CPU is idle.
- The fourth line shows how the physical memory is being used — the total amount, how much is used, how much is free, and how much is allocated to buffers (for reading from the hard drive, for example).

- The fifth line shows how the virtual memory (or swap space) is being used — the total amount of swap space, how much is used, how much is free, and how much is being cached.

The table that appears below the summary information (refer to Figure 1-6) lists information about the current processes, arranged in decreasing order by amount of CPU time used. Table 1-5 summarizes the meanings of the column headings in the table that `top` displays.

Table 1-5 Column Headings in top Utility's Output

Heading	*Meaning*
`PID`	Process ID of the process.
`USER`	Username under which the process is running.
`PR`	Priority of the process.
`NI`	Nice value of the process — the value ranges from –20 (highest priority) to 19 (lowest priority) and the default is 0. (The *nice value* represents the relative priority of the process: The higher the value, the lower the priority and the nicer the process because it yields to other processes.)
`VIRT`	Total amount of virtual memory used by the process, in kilobytes.
`RES`	Total physical memory used by a task (typically shown in kilobytes, but an `m` suffix indicates megabytes).
`SHR`	Amount of shared memory used by process.
`S`	State of the process (`S` for sleeping, `D` for uninterruptible sleep, `R` for running, `Z` for zombies — processes that should be dead but are still running — or `T` for stopped).
`%CPU`	Percentage of CPU time used since last screen update.
`%MEM`	Percentage of physical memory used by the process.
`TIME+`	Total CPU time the process has used since it started.
`COMMAND`	Shortened form of the command that started the process.

Using the uptime command

You can use the `uptime` command to get a summary of the system's state. Just type the command like this:

```
uptime
```

It displays output similar to the following:

```
15:03:21 up 32 days, 57 min, 3 users, load average: 0.13, 0.23, 0.27
```

This output shows the current time, how long the system has been up, the number of users, and (finally) the three load averages — the average number of processes that were ready to run in the past 1, 5, and 15 minutes. Load averages greater than 1 imply that many processes are competing for CPU time simultaneously.

The load averages give you an indication of how busy the system is.

Using the vmstat utility

You can get summary information about the overall system usage with the `vmstat` utility. To view system usage information averaged over 5-second intervals, type the following command (the second argument indicates the total number of lines of output `vmstat` displays):

```
vmstat 5 8
```

You see output similar to the following listing:

```
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
r b swpd free buff cache si so bi bo in cs us sy id wa
0 0 31324 4016 18568 136004 1 1 17 16 8 110 33 4 61 1
0 1 31324 2520 15348 139692 0 0 7798 199 1157 377 8 8 6 78
1 0 31324 1584 12936 141480 0 19 5784 105 1099 437 12 5 0 82
2 0 31324 1928 13004 137136 7 0 1586 138 1104 561 43 6 0 51
3 1 31324 1484 13148 132064 0 0 1260 51 1080 427 50 5 0 46
0 0 31324 1804 13240 127976 0 0 1126 46 1082 782 19 5 47 30
0 0 31324 1900 13240 127976 0 0 0 0 1010 211 3 1 96 0
0 0 31324 1916 13248 127976 0 0 0 10 1015 224 3 2 95 0
```

The first line of output shows the averages since the last reboot. After that, `vmstat` displays the 5-second average data seven more times, covering the next 35 seconds. The tabular output is grouped as six categories of information, indicated by the fields in the first line of output. The second line shows further details for each of the six major fields. You can interpret these fields by using Table 1-6.

Table 1-6 Meaning of Fields in the vmstat Utility's Output

Field Name	*Description*
`procs`	Number of processes and their types: `r` = processes waiting to run, `b` = processes in uninterruptible sleep, `w` = processes swapped out but ready to run.
`memory`	Information about physical memory and swap-space usage (all numbers in kilobytes): `swpd` = virtual memory used, `free` = free physical memory, `buff` = memory used as buffers, `cache` = virtual memory that's cached.

Field Name	Description
`swap`	Amount of swapping (the numbers are in kilobytes per second): `si` = amount of memory swapped in from disk, `so` = amount of memory swapped to disk.
`io`	Information about input and output. (The numbers are in blocks per second, where the block size depends on the disk device.) `bi` = rate of blocks sent to disk, `bo` = rate of blocks received from disk.
`system`	Information about the system: `in` = number of interrupts per second (including clock interrupts), `cs` = number of context switches per second — how many times the kernel changed which process was running.
`cpu`	Percentages of CPU time used: `us` = percentage of CPU time used by user processes, `sy` = percentage of CPU time used by system processes, `id` = percentage of time CPU is idle, `wa` = time spent waiting for input or output (I/O).

In the `vmstat` utility's output, high values in the `si` and `so` fields indicate too much swapping. (*Swapping* refers to the copying of information between physical memory and the virtual memory on the hard drive.) High numbers in the `bi` and `bo` fields indicate too much disk activity.

Checking disk performance and disk usage

Linux comes with the `/sbin/hdparm` program to control IDE or ATAPI hard drives, which are common on most PCs. One feature of the `hdparm` program is that you can use the `-t` option to determine the rate at which data is read from the disk into a buffer in memory. For example, here's the result of typing **/sbin/hdparm -t /dev/hda** on one system:

```
/dev/hda:
Timing buffered disk reads: 178 MB in 3.03 seconds = 58.81 MB/sec
```

The command requires the IDE drive's device name (`/dev/hda` for the first hard drive and `/dev/hdb` for the second hard drive) as an argument. If you have an IDE hard drive, you can try this command to see how fast data is read from your system's disk drive.

To display the space available in the currently mounted file systems, use the `df` command. If you want a more readable output from `df`, type the following command:

```
df -h
```

Here's a typical output from this command:

```
Filesystem Size Used Avail Use% Mounted on
/dev/hda5 7.1G 3.9G 2.9G 59% /
/dev/hda3 99M 18M 77M 19% /boot
none 125M 0 125M 0% /dev/shm
/dev/scd0 2.6G 2.6G 0 100% /media/cdrecorder
```

As this example shows, the `-h` option causes the `df` command to display the sizes in gigabytes (G) and megabytes (M).

To check the disk space being used by a specific directory, use the `du` command — you can specify the `-h` option to view the output in kilobytes (K) and megabytes (M), as shown in the following example:

```
du -h /var/log
```

Here's a typical output of that command:

```
152K /var/log/cups
4.0K /var/log/vbox
4.0K /var/log/httpd
508K /var/log/gdm
4.0K /var/log/samba
8.0K /var/log/mail
4.0K /var/log/news/OLD
8.0K /var/log/news
4.0K /var/log/squid
2.2M /var/log
```

The `du` command displays the disk space used by each directory, and the last line shows the total disk space used by that directory. If you want to see only the total space used by a directory, use the `-s` option. For example, type **du -sh /home** to see the space used by the `/home` directory. The command produces output that looks like this:

```
89M /home
```

Viewing System Information with the /proc File System

Your Linux system has a special `/proc` file system. You can find out many things about your system from this file system. In fact, you can even change kernel parameters through the `/proc` file system (just by writing to a file in that file system), thereby modifying the system's behavior.

The `/proc` file system isn't a real directory on the hard drive but a collection of data structures in memory, managed by the Linux kernel, that appears to you as a set of directories and files. The purpose of `/proc` (also called the *process file system*) is to give you access to information about the Linux kernel as well as to help you find out about all processes currently running on your system.

You can access the /proc file system just as you access any other directory, but you have to know the meaning of various files to interpret the information. Typically, you can use the cat or more commands to view the contents of a file in /proc. The file's contents provide information about some aspect of the system.

As with any directory, start by looking at a detailed directory listing of /proc. To do so, log in as root and type **ls -l /proc** in a terminal window. In the output, the first set of directories (indicated by the letter d at the beginning of the line) represents the processes currently running on your system. Each directory that corresponds to a process has the process ID (a number) as its name.

Notice also a very large file named /proc/kcore; that file represents the *entire* physical memory of your system. Although /proc/kcore appears in the listing as a huge file, no single physical file occupies that much space on your hard drive — so don't try to remove the file to reclaim disk space.

Several files and directories in /proc contain interesting information about your Linux PC. The /proc/cpuinfo file, for example, lists the key characteristics of your system, such as processor type and floating-point processor information. You can view the processor information by typing **cat /proc/cpuinfo**. For example, here's what appears when cat /proc/cpuinfo is run on a sample system:

```
processor : 0
vendor_id : GenuineIntel
cpu family : 15
model : 3
model name : Intel(R) Celeron(R) CPU 2.53GHz
stepping : 3
cpu MHz : 2533.129
cache size : 256 KB
fdiv_bug : no
hlt_bug : no
f00f_bug : no
coma_bug : no
fpu : yes
fpu_exception : yes
cpuid level : 5
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
   mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss
   ht tm pbe pni monitor ds_cpl cid
bogomips : 4997.12
```

This output is from a 2.5-GHz Celeron system. The listing shows many interesting characteristics of the processor. Note the line that starts with `fdiv_bug`. Remember the infamous Pentium floating-point division bug? The bug is in an instruction called `fdiv` (for *floating-point division*). Thus, the `fdiv_bug` line indicates whether this particular Pentium has the bug.

The last line in the `/proc/cpuinfo` file shows the BogoMIPS for the processor, as computed by the Linux kernel when it boots. BogoMIPS is something that Linux uses internally to time-delay loops.

Table 1-7 summarizes some of the files in the `/proc` file system that provide information about your Linux system. You can view some of these files on your system to see what they contain, but note that not all files shown in Table 1-7 are present on your system. The specific contents of the `/proc` file system depend on the kernel configuration and the driver modules that are loaded (which, in turn, depend on your PC's hardware configuration).

Table 1-7 Some Files and Directories in /proc

File Name	*Content*
`/proc/acpi`	Information about Advanced Configuration and Power Interface (ACPI) — an industry-standard interface for configuration and power management on laptops, desktops, and servers.
`/proc/bus`	Directory with bus-specific information for each bus type, such as PCI.
`/proc/cmdline`	The command line used to start the Linux kernel (for example, `ro root=LABEL=/ rhgb`).
`/proc/cpuinfo`	Information about the CPU (the microprocessor).
`/proc/devices`	Available block and character devices in your system.
`/proc/dma`	Information about DMA (direct memory access) channels that are used.
`/proc/driver/rtc`	Information about the PC's real-time clock (RTC).
`/proc/filesystems`	List of supported file systems.
`/proc/ide`	Directory containing information about IDE devices.
`/proc/interrupts`	Information about interrupt request (IRQ) numbers and how they are used.
`/proc/ioports`	Information about input/output (I/O) port addresses and how they're used.
`/proc/kcore`	Image of the physical memory.
`/proc/kmsg`	Kernel messages.

File Name	*Content*
`/proc/loadavg`	Load average (average number of processes waiting to run in the last 1, 5, and 15 minutes).
`/proc/locks`	Current kernel locks (used to ensure that multiple processes don't write to a file at the same time).
`/proc/meminfo`	Information about physical memory and swap-space usage.
`/proc/misc`	Miscellaneous information.
`/proc/modules`	List of loaded driver modules.
`/proc/mounts`	List of mounted file systems.
`/proc/net`	Directory with many subdirectories that contain information about networking.
`/proc/partitions`	List of partitions known to the Linux kernel.
`/proc/pci`	Information about PCI devices found on the system.
`/proc/scsi`	Directory with information about SCSI devices found on the system (present only if you have a SCSI device).
`/proc/stat`	Overall statistics about the system.
`/proc/swaps`	Information about the swap space and how much is used.
`/proc/sys`	Directory with information about the system. You can change kernel parameters by writing to files in this directory. (Using this method to tune system performance requires expertise to do properly.)
`/proc/uptime`	Information about how long the system has been up.
`/proc/version`	Kernel version number.

TIP

You can navigate the `/proc` file system just as you'd work with any other directories and files in Linux. Use the `more` or `cat` commands to view the contents of a file.

Understanding Linux Devices

Linux treats all devices as files and uses a device just as it uses a file — opens it, writes data to it, reads data from it, and closes it when finished. This ability to treat every device as a file is possible because of *device drivers,* which are special programs that control a particular type of hardware. When the kernel writes data to the device, the device driver does whatever is appropriate for that device. For example, when the kernel writes data to the DVD drive, the DVD device driver puts that data onto the physical medium of the DVD disk.

Thus the device driver isolates the device-specific code from the rest of the kernel and makes a device look like a file. Any application can access a device by opening the file specific to that device.

Device files

Applications can access a device as if it were a file. These files, called *device files,* appear in the `/dev` directory in the Linux file system.

If you use the `ls` command to look at the list of files in the `/dev` directory, you see several thousand files. These files don't mean that your system has several thousand devices. The `/dev` directory has files for all possible types of devices — that's why the number of device files is so large.

So how does the kernel know which device driver to use when an application opens a specific device file? The answer is in two numbers called the *major* and *minor device numbers.* Each device file is mapped to a specific device driver through these numbers.

To see an example of the major and minor device numbers, type the following command in a terminal window:

```
ls -l /dev/hda
```

You see a line of output similar to the following:

```
brw-rw---- 1 root disk 3, 0 Aug 16 14:50 /dev/hda
```

In this line, the major and minor device numbers appear just before the date. In this case, the major device number is `3` and the minor device number is `0`. The kernel selects the device driver for this device file by using the major device number.

You don't have to know much about device files and device numbers, except to be aware of their existence.

In case you're curious, all the major and minor numbers for devices are assigned according to device type. The Linux Assigned Names And Numbers Authority (LANANA) assigns these numbers. You can see the current device list at `www.lanana.org/docs/device-list/devices-2.6+.txt`.

Block devices

The first letter in the listing of a device file also provides an important clue. For the `/dev/hda` device, the first letter is `b`, which indicates that `/dev/hda` is a *block device* — one that can accept or provide data in chunks (typically 512 bytes or 1K). By the way, `/dev/hda` refers to the first IDE hard drive on your system (the C: drive in Windows). Hard drives and CD-ROM drives are examples of block devices.

Character devices

If the first letter in the listing of a device file is `c`, the device is a *character device* — one that can receive and send data one character (one byte) at a time. For example, the serial port and parallel ports are character devices. To see the specific listing of a character device, type the following command in a terminal window:

```
ls -l /dev/ttyS0
```

The listing of this device is similar to the following:

```
crw-rw---- 1 root uucp 4, 64 Aug 16 14:50 /dev/ttyS0
```

Note that the very first letter is `c` because `/dev/ttyS0` — the first serial port — is a character device.

Network devices

Network devices that enable your system to interact with a network — for example, Ethernet and dial-up *Point-to-Point Protocol* (PPP) connections — are special because they need no file to correspond to the device. Instead, the kernel uses a special name for the device. For example, Ethernet devices are named `eth0` for the first Ethernet card, `eth1` for the second one, and so on. PPP connections are named `ppp0`, `ppp1`, and so on.

Because network devices aren't mapped to device files, no files corresponding to these devices are in the `/dev` directory.

Persistent device naming with udev

Starting with the Linux kernel 2.6, a new approach for handling devices was added, based on the following features:

- **`sysfs`:** The kernel provides the `sysfs` file system, which is mounted on the `/sys` directory of the file system. The `sysfs` file system displays all the devices in the system as well as lots of information about each device, including the location of the device on the bus, attributes such as name and serial number, and the major and minor numbers of the device.
- **`/sbin/hotplug`:** This program is called whenever a device is added or removed. It can then do whatever is necessary to handle the device.
- **`/sbin/udev`:** This program takes care of dynamically named devices based on device characteristics such as serial number, device number on a bus, or a user-assigned name based on a set of rules that are set through the text file `/etc/udev/udev.rules`.

The udev program's configuration file is /etc/udev/udev.conf. Based on settings in that configuration file, udev creates device nodes automatically in the directory specified by the udev_root parameter. For example, to manage the device nodes in the /dev directory, udev_root should be defined in /etc/udev/udev.conf as follows:

```
udev_root="/dev/"
```

Managing Loadable Driver Modules

To use any device, the Linux kernel must contain the driver. If the driver code is linked into the kernel as a *monolithic* program (a program in the form of a single, large file), adding a new driver means rebuilding the kernel with the new driver code. Rebuilding the kernel means you have to reboot the PC with the new kernel before you can use the new device driver. Luckily, the Linux kernel uses a modular design that does away with rebooting hassles. Linux device drivers can be created in the form of modules that the kernel can load and unload without having to restart the PC.

Driver modules are one type of a broader category of software modules called *loadable kernel modules.* Other types of kernel modules include code that can support new types of file systems, modules for network protocols, and modules that interpret different formats of executable files.

Loading and unloading modules

You can manage the loadable device driver modules by using a set of commands. You have to log in as root to use some of these commands. Table 1-8 summarizes a few commonly used module commands.

Table 1-8 Commands to Manage Kernel Modules

This Command	*Does the Following*
insmod	Inserts a module into the kernel.
rmmod	Removes a module from the kernel.
depmod	Determines interdependencies between modules.
ksyms	Displays a list of symbols along with the name of the module that defines the symbol.
lsmod	Lists all currently loaded modules.
modinfo	Displays information about a kernel module.
modprobe	Inserts or removes a module or a set of modules intelligently. (For example, if module A requires B, modprobe automatically loads B when asked to load A.)

If you have to use any of these commands, log in as `root` or type **su -** in a terminal window to become `root`.

To see what modules are currently loaded, type

```
lsmod
```

You see a long list of modules. The list that you see will depend on the types of devices installed on your system.

The list displayed by `lsmod` includes all types of Linux kernel modules, not just device drivers. For example, if you use the Ext3 file system, you typically find two modules — `jbd` and `ext3` — that are part of the Ext3 file system (the latest file system for Linux).

REMEMBER

Besides `lsmod`, one commonly used module command is `modprobe`. Use `modprobe` when you need to manually load or remove one or more modules. The best thing about `modprobe` is that you don't need to worry if a module requires other modules to work. The `modprobe` command automatically loads any other module needed by a module. For example, to manually load the sound driver, use the command

```
modprobe snd-card-0
```

This command causes `modprobe` to load everything needed to make sound work.

TIP

You can use `modprobe` with the `-r` option to remove modules. For example, to remove the sound modules, use the following command:

```
modprobe -r snd-card-0
```

This command gets rid of all the modules that the `modprobe snd-card-0` command had loaded.

Using the /etc/modprobe.conf file

How does the `modprobe` command know that it needs to load the `snd-intel8x0` driver module? The answer's in the `/etc/modprobe.conf` configuration file. That file contains a line that tells `modprobe` what it should load when it sees the module name `snd-card-0`.

To view the contents of `/etc/modprobe.conf`, type

```
cat /etc/modprobe.conf
```

For example, consider a /etc/modprobe.conf file that contains the following lines:

```
alias eth0 3c59x
alias snd-card-0 snd-intel8x0
alias usb-controller uhci-hcd
```

Each line that begins with the keyword alias defines a standard name for an actual driver module. For example, the first line defines 3c59x as the actual driver name for the alias eth0, which stands for the first Ethernet card. Similarly, the second line defines snd-intel8x0 as the module to load when the user uses the name snd-card-0.

The modprobe command consults the /etc/modprobe.conf file to convert an alias to the real name of a driver module. It also consults the /etc/modprobe.conf file for other tasks, such as obtaining parameters for driver modules. For example, you can insert lines that begin with the options keyword to provide values of parameters that a driver may need.

For example, to set the debug-level parameter for the Ethernet driver to 5 (this parameter generates lots of information in /var/log/messages), add the following line to the /etc/modprobe.conf file:

```
options 3c59x debug=5
```

This line specifies 5 as the value of the debug parameter in the 3c59x module.

If you want to know the names of the parameters that a module accepts, use the modinfo command. For example, to view information about the 3c59x driver module, type

```
modinfo 3c59x | more
```

From the resulting output, it's possible to tell that debug is the name of the parameter for setting the debug level.

Unfortunately, the information displayed by the modinfo command can be cryptic. The only saving grace is that you may not have to do much more than use a graphical utility to configure the device, and the utility takes care of adding whatever is needed to configuration files, such as /etc/modprobe.conf.

Scheduling Jobs in Linux

As a system administrator, you may have to run some programs automatically at regular intervals or execute one or more commands at a specified time in the future. Your Linux system includes the facilities to schedule jobs to run at any future date or time you want. You can also set up the system to perform a task periodically or just once. Here are some typical tasks you can perform by scheduling jobs on your Linux system:

- Back up the files in the middle of the night
- Download large files in the early morning when the system isn't busy
- Send yourself messages as reminders of meetings
- Analyze system logs periodically and look for any abnormal activities

You can set up these jobs by using the `at` command or the `crontab` facility of Linux. The next few sections introduce these job-scheduling features of Linux.

Scheduling one-time jobs

If you want to run one or more commands at a later time, you can use the `at` command. The `atd` *daemon* — a program designed to process jobs submitted using `at` — runs your commands at the specified time and mails the output to you.

REMEMBER

Before you try the `at` command, you need to know that the following configuration files control which users can schedule tasks using the `at` command:

- `/etc/at.allow` contains the names of the users who may submit jobs using the `at` command.
- `/etc/at.deny` contains the names of users not allowed to submit jobs using the `at` command.

If these files aren't present or if you find an empty `/etc/at.deny` file, any user can submit jobs by using the `at` command. The default in Linux is an empty `/etc/at.deny` file; with this default in place, anyone can use the `at` command. If you don't want some users to use `at`, simply list their usernames in the `/etc/at.deny` file.

To use `at` to schedule a one-time job for execution at a later time, follow these steps:

1. **Run the `at` command with the date or time when you want your commands executed.**

   When you press Enter, the `at>` prompt appears, as follows:

   ```
   at 21:30
   at>
   ```

   This method is the simplest way to indicate the time when you want to execute one or more commands — simply specify the time in a 24-hour format. In this case, you want to execute the commands at 9:30 p.m. tonight (or tomorrow, if it's already past 9:30 p.m.). You can, however, specify the execution time in many different ways. (See Table 1-9 for examples.)

2. **At the `at>` prompt, type the commands you want to execute as if you were typing at the shell prompt. After each command, press Enter and continue with the next command. When you finish entering the commands you want to execute, press Ctrl+D to indicate the end.**

   Here's an example that shows how to execute the `ps` command at a future time:

   ```
   at> ps
   at> <EOT>
   job 1 at 2014-12-28 21:30
   ```

   After you press Ctrl+D, the `at` command responds with the `<EOT>` message, a job number, and the date and time when the job will execute.

Table 1-9 Formats for the at Command for the Time of Execution

Command	*When the Job Will Run*
`at now`	Immediately
`at now + 15 minutes`	15 minutes from the current time
`at now + 4 hours`	4 hours from the current time
`at now + 7 days`	7 days from the current time
`at noon`	At noontime today (or tomorrow, if already past noon)
`at now next hour`	Exactly 60 minutes from now
`at now next day`	At the same time tomorrow
`at 17:00 tomorrow`	At 5:00 p.m. tomorrow
`at 4:45pm`	At 4:45 p.m. today (or tomorrow, if it's already past 4:45 p.m.)
`at 3:00 Dec 28, 2014`	At 3:00 a.m. on December 28, 2014

After you enter one or more jobs, you can view the current list of scheduled jobs with the `atq` command:

```
atq
```

The output looks similar to the following:

```
4 2014-12-28 03:00 a root
5 2014-10-26 21:57 a root
6 2014-10-26 16:45 a root
```

The first field on each line shows the job number — the same number that the at command displays when you submit the job. The next field shows the year, month, day, and time of execution. The last field shows the jobs pending in the a queue.

If you want to cancel a job, use the atrm command to remove that job from the queue. When removing a job with the atrm command, refer to the job by its number, as follows:

```
atrm 4
```

This command deletes job 4 scheduled for 3:00 a.m. December 28, 2014.

TIP

When a job executes, the output is mailed to you. Type **mail** at a terminal window to read your mail and to view the output from your jobs.

Scheduling recurring jobs

Although at is good for running commands at a specific time, it's not useful for running a program automatically at repeated intervals. You have to use crontab to schedule such recurring jobs — for example, if you want to back up your files to tape at midnight every evening.

You schedule recurring jobs by placing job information in a file with a specific format and submitting this file with the crontab command. The cron daemon — crond — checks the job information every minute and executes the recurring jobs at the specified times. Because the cron daemon processes recurring jobs, such jobs are also referred to as *cron jobs.*

Any output from a cron job is mailed to the user who submits the job. (In the submitted job-information file, you can specify a different recipient for the mailed output.)

Two configuration files control who can schedule cron jobs using crontab:

- /etc/cron.allow contains the names of the users who may submit jobs using the crontab command.
- /etc/cron.deny contains the names of users not allowed to submit jobs using the crontab command.

If the /etc/cron.allow file exists, only users listed in this file can schedule cron jobs. If only the /etc/cron.deny file exists, users listed in this file can't schedule cron jobs. If neither file exists, the default Linux setup enables any user to submit cron jobs.

To submit a cron job, follow these steps:

1. **Prepare a shell script (or an executable program in any programming language) that can perform the recurring task you want to perform.**

   You can skip this step if you want to execute an existing program periodically.

2. **Prepare a text file with information about the times when you want the shell script or program (from Step 1) to execute and then submit this file by using crontab.**

   You can submit several recurring jobs with a single file. Each line with timing information about a job has a standard format, with six fields — the first five specify when the job runs, and the sixth and subsequent fields constitute the command that runs. For example, here's a line that executes the myjob shell script in a user's home directory at five minutes past midnight each day:

   ```
   5 0 * * * $HOME/myjob
   ```

   Table 1-10 shows the meaning of the first five fields. ***Note:*** An asterisk (*) means all possible values for that field. Also, an entry in any of the first five fields can be a single number, a comma-separated list of numbers, a pair of numbers separated by a hyphen (indicating a range of numbers), or an asterisk.

3. **Suppose the text file jobinfo (in the current directory) contains the job information. Submit this information to crontab with the following command:**

   ```
   crontab jobinfo
   ```

That's it! You're set with the cron job. From now on, the cron job runs at regular intervals (as specified in the job-information file), and you receive mail messages with the output from the job.

To verify that the job is indeed scheduled, type the following command:

```
crontab -l
```

The output of the crontab -l command shows the cron jobs currently installed in your name. To remove your cron jobs, type **crontab -r**.

If you log in as root, you can also set up, examine, and remove cron jobs for any user. To set up cron jobs for a user, use this command:

```
crontab _u username filename
```

Here *username* is the user for whom you install the cron jobs, and *filename* is the file that contains information about the jobs.

Table 1-10 Format for the Time of Execution in crontab Files

Field Number	*Meaning of Field*	*Acceptable Range of Values**
1	Minute	0–59
2	Hour of the day	0–23
3	Day of the month	0–31
4	Month	1–12 (1 means January, 2 means February, and so on) or the names of months using the first three letters — Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
5	Day of the week	0–6 (0 means Sunday, 1 means Monday, and so on) or the three-letter abbreviations of weekdays — Sun, Mon, Tue, Wed, Thu, Fri, Sat

** An asterisk in a field means all possible values for that field. For example, if an asterisk is in the third field, the job is executed every day.*

Use the following form of `crontab` command to view the `cron` jobs for a user:

```
crontab _u username -l
```

To remove a user's `cron` jobs, use the following command:

```
crontab -u username -r
```

Note: The `cron` daemon also executes the `cron` jobs listed in the system-wide `cron` job file `/etc/crontab`. Here's a typical `/etc/crontab` file from a Linux system (type **cat /etc/crontab** to view the file):

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

The first four lines set up several environment variables for the jobs listed in this file. The `MAILTO` environment variable specifies the user who receives the mail message with the output from the `cron` jobs in this file.

The line that begins with # is a comment line. The four lines following the run-parts comment execute the run-parts shell script (located in the /usr/bin directory) at various times with the name of a specific directory as argument. Each of the arguments to run-parts — /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, and /etc/cron.monthly — are directories. Essentially, run-parts executes all scripts located in the directory that you provide as an argument.

Table 1-11 lists the directories where you can find these scripts and when they execute. You have to look at the scripts in these directories to know what executes at these periodic intervals.

Table 1-11 Script Directories for cron Jobs

Directory Name	*Script Executes*
/etc/cron.hourly	Every hour
/etc/cron.daily	Each day
/etc/cron.weekly	Weekly
/etc/cron.monthly	Once each month

Chapter 2: Managing Users and Groups

In This Chapter

- **Managing a user account with a GUI user manager and commands**
- **Understanding the password file**
- **Managing your groups**
- **Working in the user environment**
- **Changing user and group ownerships of files and directories**

Linux is a multiuser system, so it has many user accounts. Even if you're the only user on your system, there will be a whole host of system user accounts. These aren't for people. They're just for running specific programs; many servers require a unique username and group name. For example, the FTP server runs under the username `ftp`.

User accounts can belong to one or more groups. Typically, each username has a corresponding private group name. By default, each user belongs to that corresponding private group. However, you can define other groups for the purpose of providing access to specific files and directories based on group membership.

User and group ownerships of files are a way to make sure that only the right people (or the right process) can access the right files and directories. Managing the user and group accounts is a typical job for system administration. It's not hard to do this part of the job, given the tools that come with Linux, as you discover in this chapter.

Adding User Accounts

You get the chance to add user accounts when you boot your system for the first time after installing Linux. Typically (depending on your distribution), the `root` account is the only one that must be created/set up during installation. If you don't add other user accounts when you start the system for the first time, you can add new users later on, using a GUI user account manager or the `useradd` command.

Creating other user accounts besides `root` is always a good idea. Even if you're the only user of the system, logging in as a less privileged user is good practice because that way you can't damage any important system files inadvertently. If necessary, you can type **su -** to log in as `root` and then perform any system administration tasks.

Managing user accounts by using a GUI user manager

Most Linux distributions come with a GUI tool to manage user accounts. You can use that GUI tool to add new user accounts. The tool displays a list of current user accounts and has an Add button for adding new users. For the purposes of illustration, the YaST interface from OpenSUSE is shown in Figure 2-1.

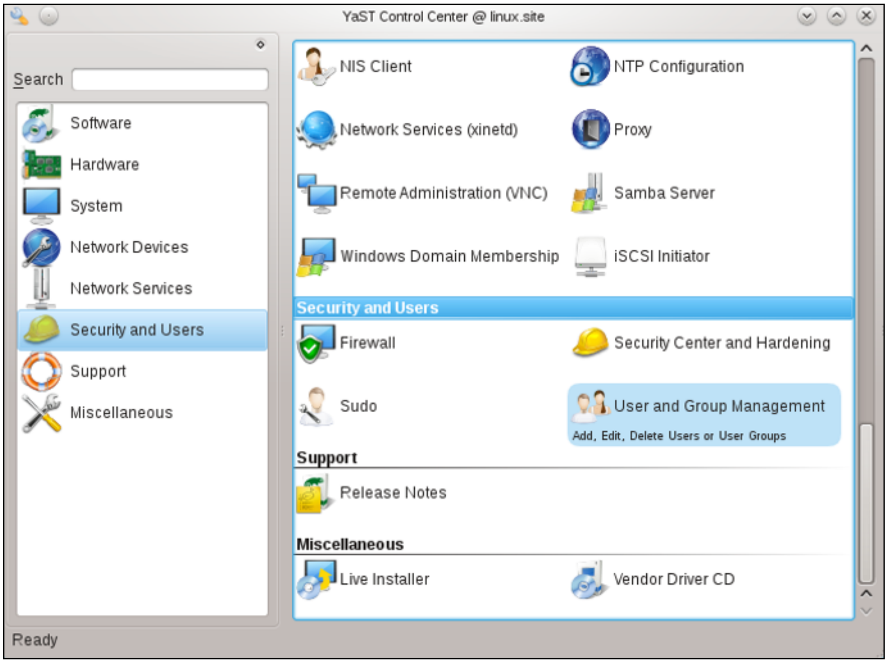
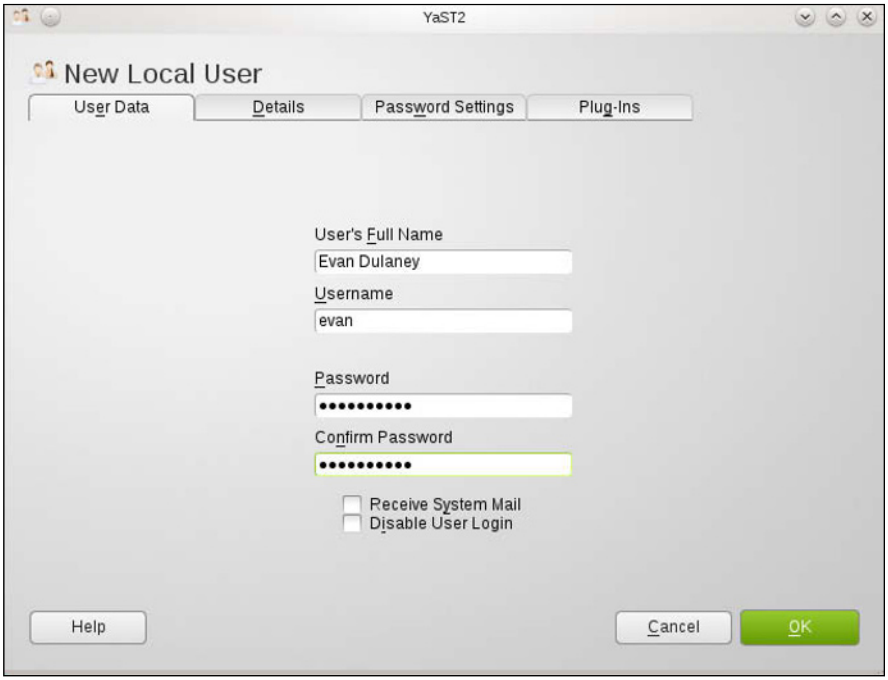


Figure 2-1: In SUSE, select the Security and Users category from the left side of the YaST Control Center, and then click the User and Group Management icon in the right side of the window.

The basic steps, regardless of the specific GUI tool, are as follows:

1. **Click the Add User button.**

   A dialog box prompts you for information about the username and password variables for the new user account, as shown in Figure 2-2.



User's Full Name: Evan Dulaney

Username: evan

Password: •••••••••••

Confirm Password: •••••••••••

□ Receive System Mail

□ Disable User Login

Figure 2-2: You can create the account by supplying the requested variables.

2. **Enter the requested information.**

   The GUI tool takes care of adding the new user account.

3. **(Optional) Click one of the other tabs for the user (refer to Figure 2-2) to configure additional information.**

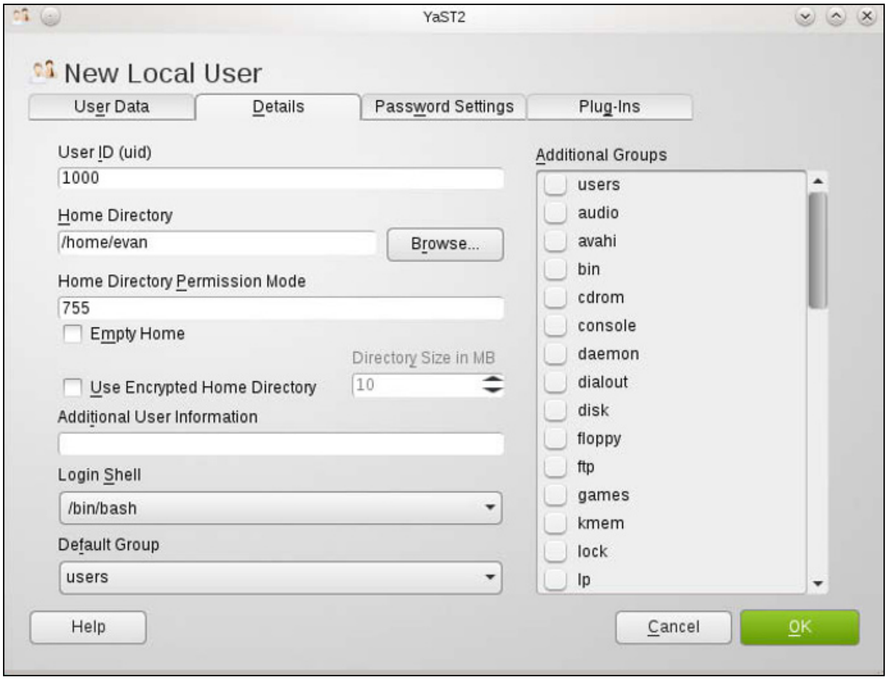
   The Details tab (shown in Figure 2-3) allows you to override the defaults for the home directory, shell, and ID information.

   The Password Settings tab (shown in Figure 2-4) allows you to override the defaults for the password configuration. Plug-Ins can be used for a number of parameters, but are often used with *quota configuration* (such as size limits on files and the number of inodes that can be created):

   - *Soft limits* warn the user.
   - *Hard limits* stop the user.

TIP

The tabs — other than User Data — are used to override the system defaults. If you want to change the system defaults, change the variables in the User and Group Administration interface.



User ID (uid): 1000

Home Directory: /home/evan

Home Directory Permission Mode: 755

□ Empty Home

Directory Size in MB: 10

□ Use Encrypted Home Directory

Additional User Information:

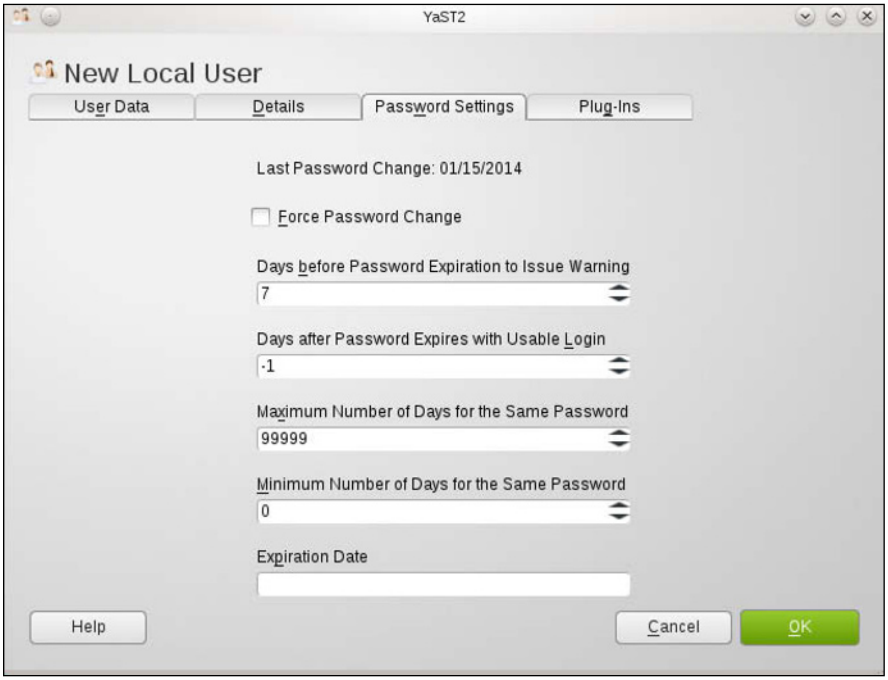
Login Shell: /bin/bash

Default Group: users

Additional Groups

□ users
□ audio
□ avahi
□ bin
□ cdrom
□ console
□ daemon
□ dialout
□ disk
□ floppy
□ ftp
□ games
□ kmem
□ lock
□ lp

Figure 2-3: Use Details to configure settings other than the defined defaults for user accounts.



Last Password Change: 01/15/2014

□ Force Password Change

Days before Password Expiration to Issue Warning: 7

Days after Password Expires with Usable Login: -1

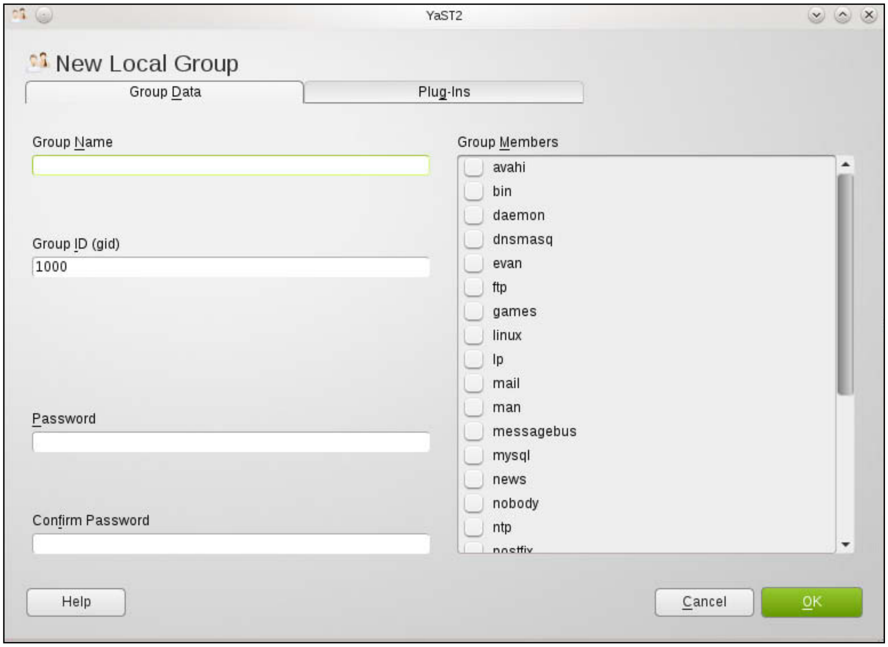
Maximum Number of Days for the Same Password: 99999

Minimum Number of Days for the Same Password: 0

Expiration Date:

Figure 2-4: Use Password Settings to configure settings other than the defined defaults for new user accounts.

Notice that the tool you use for adding new users is called User and Group Management because there are two types of accounts it can configure: Users and Groups. Selecting Groups instead of Users allows you to add new groups to `/etc/group`, as shown in Figure 2-5.



YaST2

New Local Group

Group Data | Plug-Ins

Group Name

Group ID (gid)
1000

Password

Confirm Password

Group Members
□ avahi
□ bin
□ daemon
□ dnsmasq
□ evan
□ ftp
□ games
□ linux
□ lp
□ mail
□ man
□ messagebus
□ mysql
□ news
□ nobody
□ ntp
□ postfix

Figure 2-5: Groups can be created and managed similarly to users.

To add a new user account, click the Add button and enter the information requested in the New Local User window. Fill in the requested information (including any add-ins such as for group qoutas) and then click the OK button.

Notice that the newly added user account from Figure 2-2 now appears in the list of users in the Group Members pane and can be added — along with others — to the group.

You can add more user or group accounts, if you like. When you finish, click the OK button to create any new accounts you've added; then you exit automatically.

By default, YaST places all local users in a group named `users`. Sometimes you want a user to be in another group as well, so that user can access the files owned by that group. Adding a user to another group is easy. For example, to add the username `kdulaney` to the group called `wheel`, type the following command in a terminal window:

```
usermod -G wheel kdulaney
```

To remove a user account, click the username in the list of user accounts and then click the Remove or the Delete button.

Managing user accounts by using commands

If you're working from a text console, you can create a new user account by using the `useradd` command. Follow these steps to add an account for a new user:

1. **Log in as** `root`**.**

   If you're not already logged in as `root`, type **su -** to become `root`.

2. **Type the following** `useradd` **command with the** `-c` **option to create the account:**

   ```
   /usr/sbin/useradd -c "Kristin Dulaney" kdulaney
   ```

3. **Set the password by using the** `passwd` **command, as follows:**

   ```
   passwd kdulaney
   ```

   You're prompted for the password twice. If you type a password that someone can easily guess, the `passwd` program will scold you and suggest that you use a more difficult password.

The `useradd` command consults the following configuration files to obtain default information about various parameters for the new user account:

- **`/etc/default/useradd`:** Specifies the default shell (`/bin/bash`) and the default home directory location (`/home`).
- **`/etc/login.defs`:** Provides system-wide defaults for automatic group and user IDs, as well as password-expiration parameters.
- **`/etc/skel`:** Contains the default files that `useradd` creates in the user's home directory.

Examine these files with the `cat` or `more` commands to see what they contain.

You can delete a user account by using the `userdel` command. Simply type **/usr/sbin/userdel *username*** at the command prompt where *username* is the name of the user you want to remove. To wipe out that user's home directory as well, type **+userdel -r *username***.

To modify any information in a user account, use the `usermod` command. For example, for user `kdulaney` to have `root` as the primary group, type the following:

```
usermod -g root kdulaney
```

TIP

To find out more about the useradd, userdel, and usermod commands, type **man useradd**, **man userdel**, or **man usermod**, respectively, in a terminal window.

Understanding the /etc/passwd File

The /etc/passwd file is a list of all user accounts. It's a text file and any user can read it — no special privileges needed. Each line in /etc/passwd has seven fields, separated by colons (:).

Here's a typical entry from the /etc/passwd file:

```
kdulaney:x:1000:1000:Kristin Dulaney,,,,:/home/kdulaney:/bin/bash
```

As the example shows, the format of each line in /etc/passwd looks like this:

```
username:password:UID:GID:GECOS:homedir:shell
```

Table 2-1 explains the meaning of the seven fields in each /etc/passwd entry.

Table 2-1 Fields in the /etc/passwd File

This Field	*Contains*
username	An alphanumeric username, usually 8 characters long and unique. (Linux allows usernames to be longer than 8 characters, but some other operating systems do not.)
password	When present, a 13-character encrypted password. (An *empty field* means that no password is required to access the account. An x means the password is stored in the /etc/shadow file, which is more secure.)
UID	A unique number that serves as the user identifier. (root has a UID of 0, and usually UIDs from 1 to 100 are reserved for non-human users such as servers; keeping the UID value to less than 32,767 is best.)
GID	The default group ID of the group to which the user belongs (GID 0 is for group root, other groups are defined in /etc/group, and users can be, and usually are, in more than one group at a time).
GECOS	Optional personal information about the user. (The finger command uses this field; *GECOS* stands for General Electric Comprehensive Operating System, a long-forgotten operating system that's immortalized by the name of this field in /etc/passwd.)

(continued)

Table 2-1 *(continued)*

This Field	*Contains*
`homedir`	The name of the user's home directory.
`shell`	The command interpreter (shell), such as `bash` (`/bin/bash`), which executes when this user logs in.

Managing Groups

A *group* is something to which users belong. A group has a name and an identification number (ID). After a group is defined, users can belong to one or more of these groups.

You can find all the existing groups listed in `/etc/group`. For example, here's the line that defines the group named `wheel`:

```
wheel:x:10:root,kdulaney
```

As this example shows, each line in `/etc/group` has the following format, with four fields separated by colons:

```
groupname:password:GID:membership
```

Table 2-2 explains the meaning of the four fields in a group definition.

Table 2-2 Meaning of Fields in /etc/group File

Field Name	*Meaning*
`Groupname`	The name of the group (for example, `wheel`)
`Password`	The group password (an `x` means that the password is stored in the `/etc/shadow` file)
`GID`	The numerical group ID (for example, `10`)
`Membership`	A comma-separated list of usernames that belong to this group (for example, `root,kdulaney`)

If you want to create a new group, you can simply use the `groupadd` command. For example, to add a new group called `class` with an automatically selected group ID, type the following command in a terminal window (you have to be logged in as `root`):

```
groupadd class
```

Then you can add users to this group with the usermod command. For example, to add the user kdulaney to the group named class, type the following commands:

```
usermod -G class kdulaney
```

If you want to remove a group, use the groupdel command. For example, to remove a group named class, type

```
groupdel class
```

Other User and Group Administration Values

One of the easiest ways to administer users and groups is to make certain you have the default values set to what you want them to be.

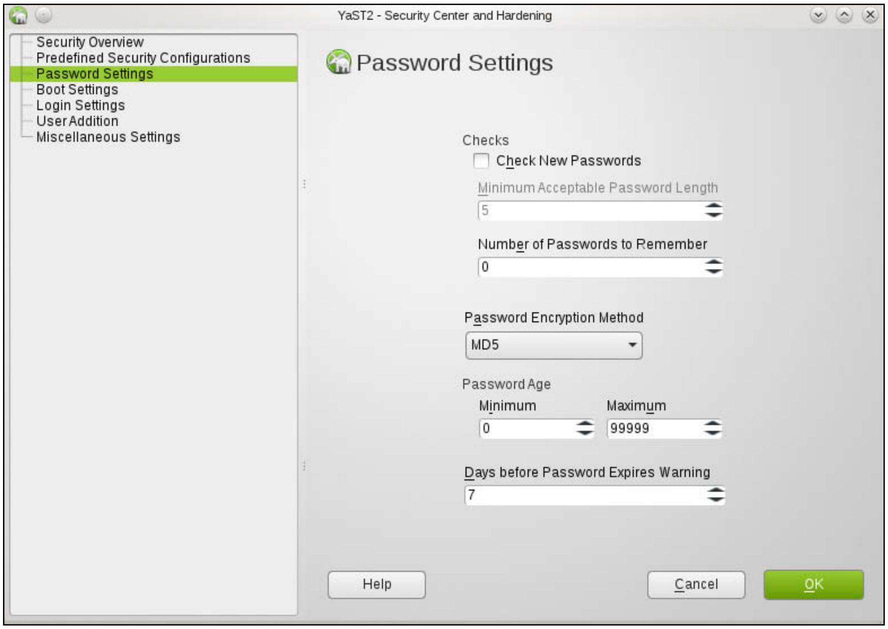
The concept of *least privilege* should be followed and, as the name implies, the goal of it is to give users the minimal privileges needed to do their jobs and nothing more.

Figure 2-6 shows the values that can be set for global password values. If you check the box Check New Passwords, then users will be prevented from using passwords that can be found in a dictionary, that are names, or use common words. The minimum password length can only be set if Check New Passwords is enabled; also consider the possible encryption methods seriously:

- DES is the default encryption method of many distributions, and although it works in almost any environment, it limits passwords to eight characters or fewer.
- MD5 lets you use longer passwords and is supported by all newer distributions, but can be a problem if you need to interact with older systems.
- SHA-512 is usually the other choice offered and it a strong hash method that is not compatible with many systems.

The default settings for new users, shown in Figure 2-7, can be set to increase security by choosing a different skeleton, a more secure shell, or a higher umask value.

Lastly, the authentication settings (shown in Figure 2-8) allow you to configure the connection settings that will take effect by default.



Password Settings

Checks

□ Check New Passwords

Minimum Acceptable Password Length: 5

Number of Passwords to Remember: 0

Password Encryption Method: MD5

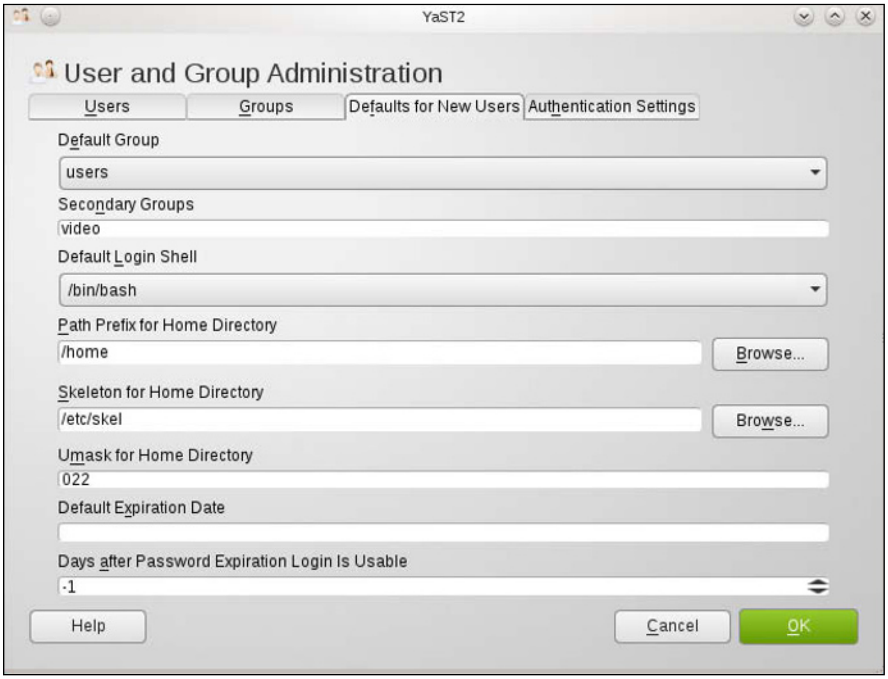
Password Age

Minimum: 0

Maximum: 99999

Days before Password Expires Warning: 7

Figure 2-6: Global password variables can greatly increase system security.



User and Group Administration

Default Group: users

Secondary Groups: video

Default Login Shell: /bin/bash

Path Prefix for Home Directory: /home

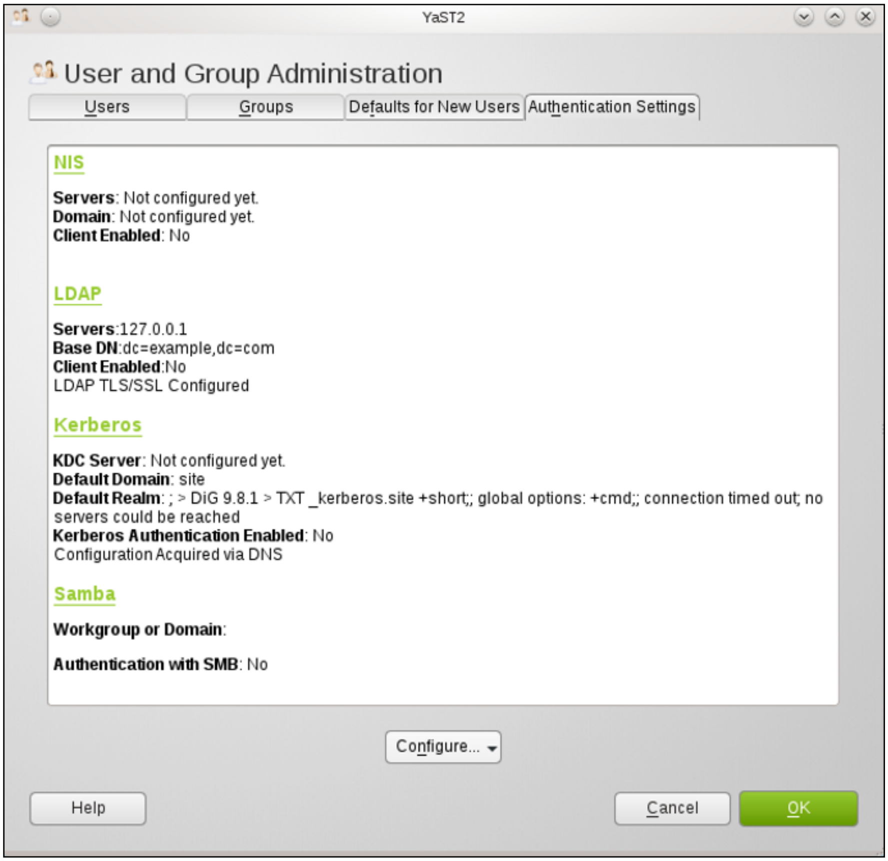
Skeleton for Home Directory: /etc/skel

Umask for Home Directory: 022

Default Expiration Date:

Days after Password Expiration Login Is Usable: -1

Figure 2-7: Default values for the new users can be changed.



NIS

Servers: Not configured yet.
Domain: Not configured yet.
Client Enabled: No

LDAP

Servers:127.0.0.1
Base DN:dc=example,dc=com
Client Enabled:No
LDAP TLS/SSL Configured

Kerberos

KDC Server: Not configured yet.
Default Domain: site
Default Realm: ; > DiG 9.8.1 > TXT _kerberos.site +short;; global options: +cmd;; connection timed out; no servers could be reached
Kerberos Authentication Enabled: No
Configuration Acquired via DNS

Samba

Workgroup or Domain:

Authentication with SMB: No

Figure 2-8: Default authentication setting values can be changed.

Exploring the User Environment

When you log in as a user, you get a set of environment variables that control many aspects of what you see and do on your Linux system. If you want to see your current environment, type the following command in a terminal window:

```
env
```

(By the way, the `printenv` command also displays the environment, but `env` is shorter.)

The env command prints a long list of lines. The collection of lines is the current environment; each line defines an environment variable. For example, the env command displays this typical line:

```
HOSTNAME=localhost.localdomain
```

This line defines the environment variable HOSTNAME as localhost.localdomain.

An *environment variable* is nothing more than a name associated with a string. For example, the environment variable named PATH is typically defined as follows for a normal user:

```
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/
    sbin:/sbin
```

The string to the right of the equal sign (=) is the value of the PATH environment variable. By convention, the PATH environment variable is a sequence of directory names, each name separated from the others by a colon (:).

Each environment variable has a specific purpose. For example, when the shell has to search for a file, it simply searches the directories listed in the PATH environment variable in the order of their appearance. Therefore, if two programs have the same name, the shell executes the one it finds first.

In a fashion similar to the shell's use of the PATH environment variable, an editor such as vi uses the value of the TERM environment variable to figure out how to display the file you edit with vi. To see the current setting of TERM, type the following command at the shell prompt:

```
echo $TERM
```

If you type this command in a terminal window, the output is as follows:

```
xterm
```

To define an environment variable in bash, use the following syntax:

```
export NAME=Value
```

Here NAME denotes the name of the environment variable and Value is the string representing its value. Therefore you set TERM to the value xterm by using the following command:

```
export TERM=xterm
```

TIP

After you define an environment variable, you can change its value by simply specifying the new value with the syntax `NAME=new-value`. For example, to change the definition of `TERM` to `vt100`, type **TERM=vt100** at the shell prompt.

With an environment variable such as `PATH`, typically you want to append a new directory name to the existing definition rather than define the `PATH` from scratch. For example, if you download and install the fictional XYZ 5 Development Kit, you have to add the location of the XYZ binaries to `PATH`. Here's how you accomplish that task:

```
export PATH=$PATH:/usr/xyz/xyz.5.0/bin
```

This command appends the string `:/usr/xyz/xyz.5.0/bin` to the current definition of the `PATH` environment variable. The net effect is to add `/usr/xyz/xyz.5.0/bin` to the list of directories in `PATH`.

Note: You also can write this `export` command as follows:

```
export PATH=${PATH}:/usr/xyz/xyz.5.0/bin
```

After you type that command, you can access programs in the `/usr/xyz/xyz.5.0/bin` directory that the interpreter can execute.

`PATH` and `TERM` are only two of a handful of common environment variables. Table 2-3 lists some of the environment variables for a typical Linux user.

Table 2-3 Typical Environment Variables in Linux

Environment Variable	*Contents*
`DISPLAY`	The name of the display on which the X Window System displays output (typically set to `:0.0`)
`HOME`	Your home directory
`HOSTNAME`	The host name of your system
`LOGNAME`	Your login name
`MAIL`	The location of your mail directory
`PATH`	The list of directories in which the shell looks for programs
`SHELL`	Your shell (`SHELL=/bin/bash` for `bash`)
`TERM`	The type of terminal

Changing User and Group Ownership of Files

In Linux, each file or directory has two types of owners: a user and a group. In other words, a user and group own each file and directory. The user and group ownerships can control who can access a file or directory.

To view the owner of a file or directory, use the `ls -l` command to see the detailed listing of a directory. For example, here's a typical file's information:

```
-rw-rw-r-- 1 kdulaney kdulaney 40909 Aug 16 20:37
   composer.txt
```

In this example, the first set of characters shows the file's permission setting — who can read, write, or execute the file. The third and fourth fields (in this example, `kdulaney kdulaney`) indicate the user and group owner of the file. Each user has a private group that has the same name as the username. Thus most files appear to show the username twice when you list user and group ownership.

As a system administrator, you may decide to change the group ownership of a file to a common group. For example, suppose you want to change the group ownership of the `composer.txt` file to the `class` group. To do that, log in as `root` and type the following command:

```
chgrp class composer.txt
```

This `chgrp` command changes the group ownership of `composer.txt` to `class`.

You can use the `chown` command to change the user owner. The command has the following format:

```
chown username filename
```

For example, to change the user ownership of a file named `sample.jpg` to `kdulaney`, type

```
chown kdulaney sample.jpg
```

The `chown` command can change both the user and group owner at the same time. For example, to change the user owner to `kdulaney` and the group owner to `class`, type

```
chown kdulaney.class composer.txt
```

In other words, you simply append the group name to the username with a period in between, and then use that as the name of the owner.

Chapter 3: Managing File Systems

In This Chapter

- **Navigating the Linux file system**
- **Sharing files with NFS**
- **Backing up and restoring files**
- **Mounting the NTFS file system**
- **Accessing MS-DOS files**

A *file system* refers to the organization of files and directories. As a system administrator, you have to perform certain operations to manage file systems on various storage media. For example, you have to know how to *mount* — add a file system on a storage medium by attaching it to the overall Linux file system. You also have to back up important data and restore files from a backup. Other file-system operations include sharing files with the Network File System (NFS) and accessing MS-DOS files. This chapter shows you how to perform all file-system management tasks.

Exploring the Linux File System

LXO-101 Exam

The files and directories in your PC store information in an organized manner, just like paper filing systems. When you store information on paper, typically you put several pages in a folder and then store the folder in a file cabinet. If you have many folders, you probably have some sort of filing system. For example, you may label each folder's tab and then arrange them alphabetically in the file cabinet. You might have several file cabinets, each with lots of drawers, which, in turn, contain folders full of pages.

Operating systems, such as Linux, organize information in your computer in a manner similar to your paper filing system. Linux uses a file system to organize all information in your computer. Of course, the storage medium isn't a metal file cabinet and paper. Instead, Linux stores information on devices such as hard drives, USB drives, and DVD drives.

To draw an analogy between your computer's file system and a paper filing system, think of a disk drive as the file cabinet. The drawers in the file cabinet correspond to the directories in the file system. The folders in each drawer are also directories — because a directory in a computer file system can contain other directories. You can think of files as the pages inside the folder — and that's where the actual information is stored. Figure 3-1 illustrates the analogy between a file cabinet and the Linux file system.

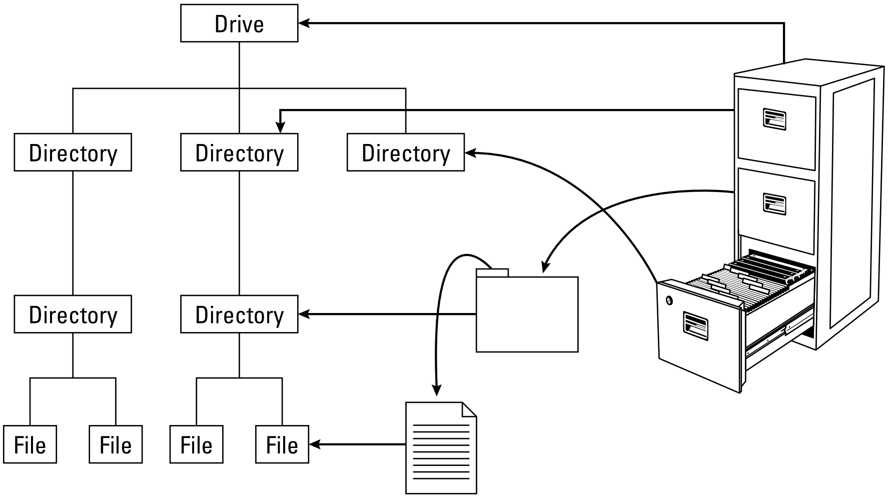


Figure 3-1: It's a bit of a stretch, but you can think of the Linux file system as similar to a filing cabinet.

The Linux file system has a *hierarchical* structure — directories can contain other directories, which in turn contain individual files.

Everything in your Linux system is organized in files and directories. To access and use documents and programs on your system, you have to be familiar with the file system.

Understanding the file-system hierarchy

LXO-101 Exam

The Linux file system is organized like a tree, with a `root` directory from which all other directories branch out. When you write a complete pathname, the `root` directory is represented by a single slash (/). Then there's a hierarchy of files and directories. Parts of the file system can be on different physical drives or in different hard drive partitions.

Linux uses a standard directory hierarchy. Figure 3-2 shows some of the standard parts of the Linux file system. You can create new directories anywhere in this structure.

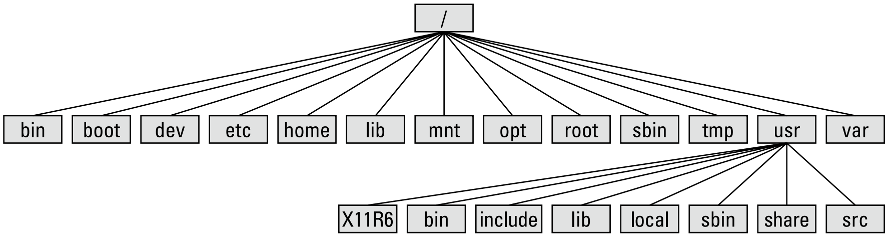


Figure 3-2: The Linux file system uses a standard directory hierarchy similar to this one.

Write the name of any file or directory by concatenating the names of directories that identify where that file or directory is and by using the forward slash (/) as a separator. For example, in Figure 3-2, the `usr` directory at the top level is written as `/usr` because the `root` directory (/) contains `usr`. On the other hand, the `X11R6` directory is inside the `usr` directory, which is inside the `root` directory (/). Therefore the `X11R6` directory is uniquely identified by the name `/usr/X11R6`. This type of full name is a *pathname* because the name identifies the path you take from the `root` directory to reach a file. Thus `/usr/X11R6` is a pathname.

TECHNICAL STUFF

The *Filesystem Hierarchy Standard (FHS)* specifies the organization of files and directories in Unix-like operating systems, such as Linux. FHS defines a standard set of directories and their intended use. The FHS, if faithfully adopted by all Linux distributions, should help improve the interoperability of applications, system administration tools, development tools, and scripts across all Linux distributions. FHS even helps the system documentation (as well as books like this one) because the same description of the file system applies to all Linux distributions. Version 2.3 of FHS was announced on January 29, 2004. FHS 2.3 is part of the Linux Standard Base version 3.x (LSB 3.0), which was released on July 1, 2005. The standard was updated with 3.1 on October 25, 2005, and 3.2 on January 28, 2008. LSB 3.x (see `www.linuxbase.org`) is a set of binary standards aimed at reducing variations among the Linux distributions and promoting portability of applications. As of this writing, the most current Base is 4.1, which came out in early 2011. To find out more about the Linux Standard Base, check out the home page at

`www.linuxfoundation.org/collaborate/workgroups/lsb`.

Each of the standard directories in the Linux file system has a specific purpose. Table 3-1, Table 3-2, and Table 3-3 summarize these directories.

Table 3-1 Standard Directories in Linux File System

Directory	*Used to Store*
/bin	Executable files for user commands (for use by all users)
/boot	Files needed by the boot loader to load the Linux kernel
/dev	Device files
/etc	Host-specific system configuration files
/home	User home directories
/lib	Shared libraries and kernel modules
/media	Mount point for removable media
/mnt	Mount point for a temporarily mounted file system
/opt	Add-on application software packages
/root	Home directory for the root user
/sbin	Utilities for system administration
/srv	Data for services (such as web and FTP) offered by this system
/tmp	Temporary files

Table 3-2 The /usr Directory Hierarchy

Directory	*Secondary Directory Hierarchy*
/usr/bin	Most user commands
/usr/include	Directory for *include files* — files that are inserted into source code of applications by using various directives — used in developing Linux applications
/usr/lib	Libraries used by software packages and for programming
/usr/libexec	Libraries for applications
/usr/local	Any local software
/usr/sbin	Nonessential system administrator utilities
/usr/share	Shared data that doesn’t depend on the system architecture (whether the system is an Intel PC or a Sun SPARC workstation)
/usr/src	Source code

Table 3-3 The /var Directory Hierarchy

Directory	*Variable Data*
`/var/cache`	Cached data for applications
`/var/lib`	Information relating to the current state of applications
`/var/lock`	Lock files to ensure that a resource is used by one application only
`/var/log`	Log files organized into subdirectories
`/var/mail`	User mailbox files
`/var/opt`	Variable data for packages stored in the `/opt` directory
`/var/run`	Data describing the system since it was booted
`/var/spool`	Data that's waiting for some kind of processing
`/var/tmp`	Temporary files preserved between system reboots
`/var/yp`	Network Information Service (NIS) database files

Mounting a device on the file system

LXO-101 Exam

The storage devices that you use in Linux contain Linux file systems. Each device has its own local file system consisting of a hierarchy of directories. Before you can access the files on a device, you have to attach the device's directory hierarchy to the tree that represents the overall Linux file system.

Mounting is the operation you perform to cause the file system on a physical storage device (a hard drive partition or a CD-ROM) to appear as part of the Linux file system. Figure 3-3 illustrates the concept of mounting.

Figure 3-3 shows each device with a name that begins with `/dev`. For example, `/dev/cdrom` is the first DVD/CD-ROM drive. Physical devices are mounted at specific mount points on the Linux file system. For example, the DVD/CD-ROM drive, `/dev/cdrom`, is mounted on `/media/cdrom` in the file system. After mounting the CD-ROM in this way, the Fedora directory on a CD-ROM or DVD-ROM appears as `/media/cdrom/Fedora` in the Linux file system.

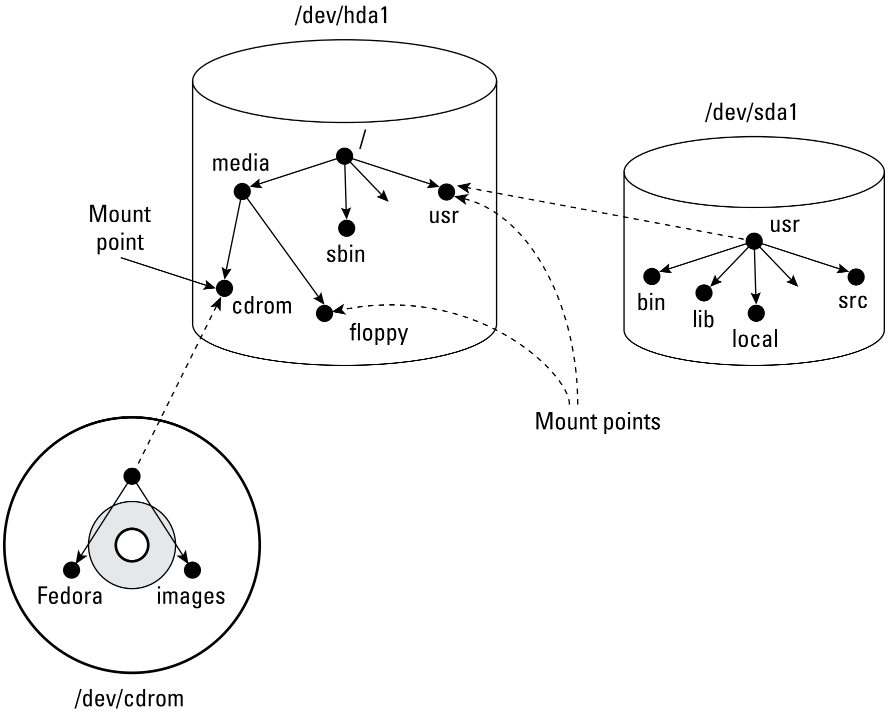


Figure 3-3: You have to mount a device on the Linux file system before accessing it.

You can use the `mount` command to manually mount a device on the Linux file system at a specified directory. That directory is the *mount point.* For example, to mount the DVD/CD-ROM drive at the `/media/cdrom` directory, type the following command (after logging in as `root`):

```
mount /dev/cdrom /media/cdrom
```

The `mount` command reports an error if the DVD/CD-ROM device is mounted already or if no CD or DVD media is in the drive. Otherwise the `mount` operation succeeds, and you can access the contents of the DVD or CD through the `/media/cdrom` directory.

REMEMBER

You can use any directory as the mount point. If you mount a device on a nonempty directory, however, you can't access the files in that directory until you unmount the device by using the `umount` command. Therefore always use an empty directory as the mount point.

TIP

To unmount a device when you no longer need it, use the `umount` command. For example, for a DVD/CD-ROM device with the device name `/dev/cdrom`, type the following command to unmount the device:

```
umount /dev/cdrom
```

The umount command succeeds as long as no one is using the DVD/CD-ROM. If you get an error when trying to unmount the DVD/CD-ROM, check to see if the current working directory is on the DVD or CD. If you're currently working in one of the DVD/CD-ROM's directories, that also qualifies as a use of the DVD/CD-ROM.

Examining the /etc/fstab file

The mount command has the following general format:

```
mount device-name mount-point
```

However, you can mount by specifying only the CD-ROM device name or the *mount-point* name, provided there's an entry in the /etc/fstab file for the CD-ROM mount point. That entry specifies the CD-ROM device name and the file-system type. That's why you can mount the CD-ROM with a shorter mount command.

For example, in Debian, you can mount the CD-ROM by typing one of the following commands:

```
mount /dev/cdrom
mount /media/cdrom
```

The /etc/fstab file is a *configuration file* — a text file containing information that the mount and umount commands use. Each line in the /etc/fstab file provides information about a device and its mount point in the Linux file system. Essentially, the /etc/fstab file associates various mount points within the file system with specific devices, which enables the mount command to work from the command line with only the mount point or the device as argument.

Here's a /etc/fstab file from a SUSE system. (The file has a similar format in other Linux distributions.)

```
/dev/hda7 /boot ext3 acl,user_xattr 1 2
/dev/hda6 /data1 auto noauto,user 0 0
/dev/hda9 /data2 auto noauto,user 0 0
/dev/hda10 /data3 auto noauto,user 0 0
/dev/hda5 /data4 auto noauto,user 0 0
/dev/hda2 /windows/C ntfs ro,users,gid=users,umask=0002,nls=utf8 0 0
/dev/hda8 swap swap pri=42 0 0
devpts /dev/pts devpts mode=0620,gid=5 0 0
proc /proc proc defaults 0 0
usbfs /proc/bus/usb usbfs noauto 0 0
sysfs /sys sysfs noauto 0 0
/dev/cdrecorder /media/cdrecorder subfs fs=cdfss,ro,procuid,nosuid,nodev,exec,ioc
    harset=utf8 0 0
```

The first field on each line shows a device name, such as a hard drive partition. The second field is the mount point, and the third field indicates the type of file system on the device. You can ignore the last three fields for now.

This /etc/fstab file shows that the /dev/hda8 device functions as a swap device for virtual memory, which is why both the mount point and the file-system type are set to swap.

TECHNICAL STUFF

The Linux operating system uses the contents of the /etc/fstab file to mount various file systems automatically. During Linux startup, the init process executes a shell script that runs the mount -a command. That command reads the /etc/fstab file and mounts all listed file systems (except those with the noauto option). The third field on each line of /etc/fstab specifies the type of file system on that device, and the fourth field shows a comma-separated list of options that the mount command uses when mounting that device on the file system. Typically, you find the defaults option in this field. The defaults option implies — among other things — that the device mounts at boot time, that only the root user can mount the device, and that the device mounts for both reading and writing. If the options include noauto, the device doesn't mount automatically when the system boots.

DISTRIBUTION SPECIFIC

In Fedora, you often find the managed option in the fourth field of /etc/fstab entries. The managed option indicates that the line was added to the fstab file by the HAL (hardware abstraction layer) daemon, which runs the fstab-sync command to add entries in the /etc/fstab file for each removable drive that it detects. You typically find that the entries for DVD/CD-ROM drive(s) (/dev/hdc in most systems) have the managed option in the fourth field.

Sharing Files with NFS

LX0-102 Exam

Sharing files through the NFS is simple and involves two basic steps:

- On the NFS server, export one or more directories by listing them in the /etc/exports file and by running the /usr/sbin/exportfs command. In addition, you must run the NFS server.
- On each client system, use the mount command to mount the directories the server has exported.

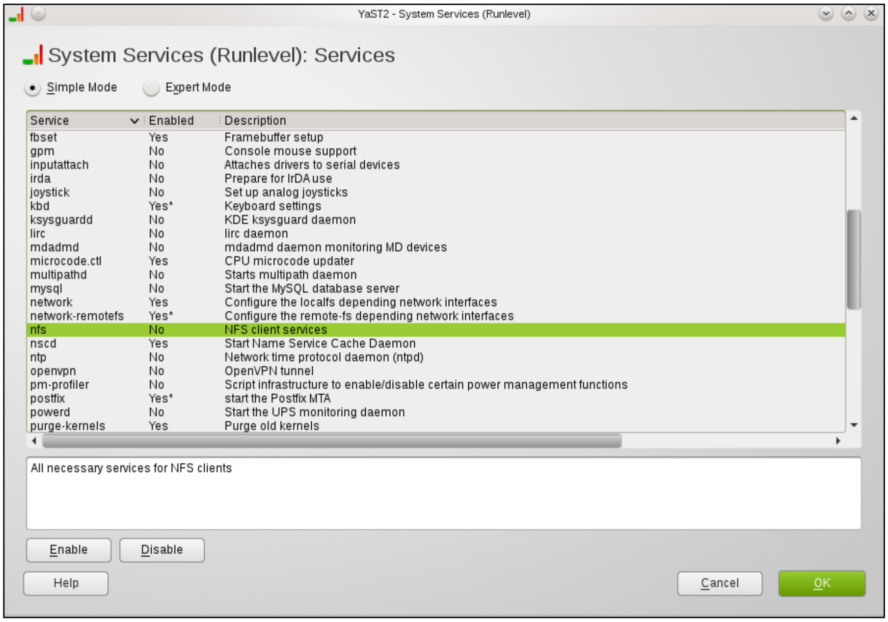
DISTRIBUTION SPECIFIC

How you start the NFS server depends on the Linux distribution. If a GUI sysadmin tool is available, you can start the NFS server from the GUI tool. Otherwise, you can type a command in a terminal window to start the NFS server:

- ✦ In Debian, you can type **invoke-rc.d nfs-kernel-server start** and **invoke-rc.d nfs-common start** to start the NFS server.
- ✦ In Fedora, type **service nfs start**.
- ✦ To start the NFS server in SUSE, you can use the YaST Control Center: From the main menu, choose the Kickoff Application Launcher➪YaST➪System➪System Services (run level).

  Doing so brings up a screen such as that shown in Figure 3-4.
- ✦ In Xandros, you can start the NFS server from the Xandros Control Center (Main Menu➪Control Center) or by typing **invoke-rc.d nfs-user-server start** in a terminal window.

The only problem in using NFS is that each client system must support it. Most PCs don't come with NFS — that means you have to buy NFS software separately if you want to share files by using NFS. If, however, all systems on your LAN run Linux (or other variants of Unix with built-in NFS support), using NFS makes sense.



System Services (Runlevel): Services

● Simple Mode ○ Expert Mode

Service	Enabled	Description
fbset	Yes	Framebuffer setup
gpm	No	Console mouse support
inputattach	No	Attaches drivers to serial devices
irda	No	Prepare for IrDA use
joystick	No	Set up analog joysticks
kbd	Yes*	Keyboard settings
ksysguardd	No	KDE ksysguard daemon
lirc	No	lirc daemon
mdadmd	No	mdadmd daemon monitoring MD devices
microcode.ctl	Yes	CPU microcode updater
multipathd	No	Starts multipath daemon
mysql	No	Start the MySQL database server
network	Yes	Configure the localfs depending network interfaces
network-remotefs	Yes*	Configure the remote-fs depending network interfaces
nfs	No	NFS client services
nscd	Yes	Start Name Service Cache Daemon
ntp	No	Network time protocol daemon (ntpd)
openvpn	No	OpenVPN tunnel
pm-profiler	No	Script infrastructure to enable/disable certain power management functions
postfix	Yes*	start the Postfix MTA
powerd	No	Start the UPS monitoring daemon
purge-kernels	Yes	Purge old kernels

All necessary services for NFS clients

Figure 3-4: You can start NFS in openSUSE from YaST.

NFS has security vulnerabilities. Therefore don't set up NFS on systems directly connected to the Internet.

The upcoming section walks you through an NFS setup, using an example of two Linux PCs on a LAN.

Exporting a file system with NFS

To export a file system with NFS, start with the server system that *exports* — makes available to the client systems — the contents of a directory. On the server, you must run the NFS service and also designate one or more file systems to be exported to the client systems.

You have to add an appropriate entry to the `/etc/exports` file. For example, suppose you want to export the `/home` directory and you want to enable the hostname `LNBP75` to mount this file system for read and write operations. (You can use a host's IP address in place of the hostname.) You can do so by adding the following entry to the `/etc/exports` file:

```
/home LNBP75(rw)
```

If you use the IP address of a host, the entry might look like this:

```
/home 192.168.1.200(rw)
```

This specifies that 192.168.1.200 is the IP address of the host that's allowed full access to the `/home` directory.

After adding the entry in the `/etc/exports` file, start the NFS server using a method appropriate for your Linux distribution. For example, in Fedora, log in as `root` and type the following command in a terminal window:

```
service nfs start
```

When the NFS service is up, the server side of NFS is ready. Now you can try to mount the exported file system from a client system and access the exported file system.

TIP

If you ever make any changes to the exported file systems listed in the `/etc/exports` file, remember to restart the NFS service. For example, in Fedora, type **service nfs restart** in a terminal window. In Xandros, type **invoke-rc.d nfs-user-server restart**.

Mounting an NFS file system

LXO-102 Exam

To access an exported NFS file system on a client system, you have to mount that file system on a *mount point* — which is, in practical terms, nothing more than a local directory. For example, suppose you want to access the `/home/public` directory exported from the server named `LNBP200` at the local directory `/mnt/lnbp200` on the client system. To do so, follow these steps:

1. **Log in as** `root` **and create the directory with the following command:**

   ```
   mkdir /mnt/lnbp200
   ```

2. **Type the following command to perform the** `mount` **operation:**

   ```
   mount lnbp200:/home/public /mnt/lnbp200
   ```

   If you know only the IP address of the server, replace the hostname (in this case, `lnbp200`) with the IP address.

3. **Change the directory to** `/mnt/lnbp200` **with the command** cd /mnt/lnbp200**.**

   Now you can view and access exported files from this directory.

TIP

To confirm that the NFS file system is indeed mounted, log in as `root` on the client system and type **mount** in a terminal window. You see a line similar to the following about the NFS file system:

```
lnbp200:/home/public on /mnt/lnbp200 type nfs (rw,addr=192.168.1.200)
```

Backing Up and Restoring Files

LXO-101 Exam

Backing up and restoring files is a crucial system administration task. If something happens to your system's hard drive, you have to rely on the backups to recover important files. The following discussion presents some backup strategies, describes several backup media, and explains how to back up and restore files by using the tape archiver (`tar`) program that comes with Linux. Also, you find out how to perform incremental and automatic backups on tapes.

REMEMBER

If you have a CD burner, you can back up files also by recording them on a CD-R. Consult Book II, Chapter 5, for information on what application you can use to burn a data CD.

Selecting a backup strategy and media

Your Linux system's hard drive contains everything you need to keep the system running — as well as other files (such as documents and databases) that keep your business running. You have to back up these files so you can recover quickly and bring the system back to normal in case the hard drive crashes. Typically, you have to follow a strict regimen of regular backups because you can never tell when the hard drive may fail or the file system may get corrupted. To implement such a regimen, first decide which files you want to back up, how often, and what backup storage media to use. This process is what is meant by selecting a backup strategy and backup media.

Your choice of backup strategy and backup media depends on your assessment of the risk of business disruption due to hard drive failure. Depending on how you use your Linux system, a disk failure may or may not have much effect on you.

For example, if you use your Linux system as a learning tool (to find out more about Linux or programming), all you may need are backup copies of some system files required to configure Linux. In this case, your backup strategy can be to save important system configuration files on your preferred storage media every time you change any system configuration.

On the other hand, if you use your Linux system as an office server that provides shared file storage for many users, the risk of business disruption due to disk failure is much higher. In this case, you have to back up all the files every week and back up any new or changed files every day. You can perform these backups in an automated manner (with the job-scheduling features described in Chapter 1 of this minibook). Also, you probably need a backup storage medium that can store many gigabytes of data. In other words, for high-risk situations, your backup strategy has to be more elaborate and requires additional equipment (such as a high-capacity external hard drive).

Your choice of backup media depends on the amount of data you have to back up. For a small amount of data (such as system configuration files), you can use USB flash drives as backup media. If your PC has an old Zip drive, you can use Zip disks as backup media; these are good for backing up a single-user directory. To back up entire servers, use an external hard drive (which could be attached to the computer or the network) or other storage device.

When backing up files to these media, you have to refer to the backup device by name. You will find these devices under the /dev directory.

Commercial backup utilities for Linux

The next section explains how to back up and restore files using the tape archiver (`tar`) program that comes with Linux. Although you can manage backups with `tar`, a number of commercial backup utilities come with graphical user interfaces and other features to simplify backups. Here are some well-known commercial backup utilities for Linux:

- **BRU:** A backup and restore utility from the TOLIS Group, Inc. (`www.tolisgroup.com`)
- **LONE-TAR:** Tape backup software package from Lone Star Software Corp. (`www.cactus.com`)

- **Arkeia:** Backup and recovery software for heterogeneous networks from Arkeia (www.arkeia.com)
- **CA ARCserve Backup for Linux:** Data-protection technology for Linux systems from Computer Associates (http://www.arcserve.com)

Using the tape archiver — tar

You can use the tar command to archive files to a device, such as a hard drive or tape. The tar program creates an archive file that can contain other directories and files and (optionally) compress the archive for efficient storage. The archive is then written to a specified device or another file. Many software packages are distributed in the form of a compressed tar file.

The command syntax of the tar program is as follows:

```
tar options destination source
```

Here *options* are usually specified by a sequence of single letters, with each letter specifying what tar does. The *destination* is the device name of the backup device. And *source* is a list of file or directory names denoting the files to back up.

Backing up and restoring a single-volume archive

Suppose you want to back up the contents of the /etc/X11 directory on a hard drive. Log in as root, and type the following command, where *xxx* represents your drive:

```
tar zcvf /dev/xxx /etc/X11
```

The tar program displays a list of filenames as each file is copied to the compressed tar archive. In this case, the options are zcvf, the destination is /dev/xxx (the drive), and the source is the /etc/X11 directory (which implies all its subdirectories and their contents). You can use a similar tar command to back up files to a tape — simply replace the hard drive location with that of the tape device — such as /dev/st0 for a SCSI tape drive.

Table 3-4 defines a few common tar options.

Table 3-4 Common tar Options

Option	*Does the Following*
c	Creates a new archive.
f	Specifies the name of the archive file or device on the next field in the command line.

(continued)

Table 3-4 *(continued)*

Option	*Does the Following*
`M`	Specifies a multivolume archive. (The next section describes multivolume archives.)
`t`	Lists the contents of the archive.
`v`	Displays verbose messages.
`x`	Extracts files from the archive.
`z`	Compresses the `tar` archive using `gzip`.

To view the contents of the `tar` archive you create on the drive, type the following command (replacing *xxx* with the drive device):

```
tar ztf /dev/xxx
```

You see a list of filenames (each begins with `/etc/X11`) indicating what's in the backup. In this `tar` command, the `t` option lists the contents of the `tar` archive.

To extract the files from a `tar` backup, follow these steps while logged in as `root`:

1. **Change the directory to `/tmp` by typing this command:**

   ```
   cd /tmp
   ```

   This step is where you can practice extracting the files from the `tar` backup. For a real backup, change the directory to an appropriate location (typically, you type **cd /**).

2. **Type the following command:**

   ```
   tar zxvf /dev/xxx
   ```

   This `tar` command uses the `x` option to extract the files from the archive stored on the device (replace *xxx* with the drive).

Now if you check the contents of the `/tmp` directory, you notice that the `tar` command creates an `etc/X11` directory tree in `/tmp` and restores all the files from the `tar` archive into that directory. The `tar` command strips the leading `/` from the filenames in the archive and restores the files in the current directory. If you want to restore the `/etc/X11` directory from the archive, use this command (substituting the device name for *xxx*):

```
tar zxvf /dev/xxx -C /
```

The `-C` does a `cd` to the directory specified (in this case, the `root` directory of `/`) before doing the `tar`; the `/` at the end of the command denotes the directory where you want to restore the backup files.

You can use the `tar` command to create, view, and restore an archive. You can store the archive in a file or in any device you specify with a device name.

Backing up and restoring a multivolume archive

Sometimes the capacity of a single storage medium is less than the total storage space needed to store the archive. In this case, you can use the `M` option for a multivolume archive — meaning the archive can span multiple tapes or even floppies (if you happen to be using an older machine that still has them). Note, however, that you can't create a compressed, multivolume archive. That means you have to drop the `z` option.

Note: The `M` tells `tar` to create a multivolume archive. The `tar` command prompts you for a second media when the first one is filled. Take out the first media, and insert another when you see the following prompt:

```
Prepare volume #2  and hit return:
```

When you press Enter, the `tar` program continues with the second media. For larger archives, the `tar` program continues to prompt for new media as needed.

To restore from this multivolume archive, type **`cd /tmp`** to change the directory to `/tmp`. (I use the `/tmp` directory for illustrative purposes, but you have to use a real directory when you restore files from archive.) Then type (replacing *xxx* with the device you are using)

```
tar xvfM /dev/xxx
```

The `tar` program prompts you to feed the media as necessary.

TIP

Use the `du -s` command to determine the amount of storage you need for archiving a directory. For example, type **du -s /etc** to see the total size of the `/etc` directory in kilobytes. Here's a typical output of that command:

```
35724 /etc
```

The resulting output shows that the `/etc` directory requires at least 35,724 kilobytes of storage space to back up.

Backing up on tapes

Although backing up on tapes is as simple as using the right device name in the `tar` command, you do have to know some nuances of the tape device to use it well. When you use `tar` to back up to the device named `/dev/st0` (the first SCSI tape drive), the tape device automatically rewinds the tape after the `tar` program finishes copying the archive to the tape. The `/dev/st0` device is called a *rewinding tape device* because it rewinds tapes by default.

If your tape can hold several gigabytes of data, you may want to write several `tar` archives — one after another — to the same tape (otherwise much of the tape may be left empty). If you plan to do so, your tape device can't rewind the tape after the `tar` program finishes. To help you with scenarios like this one, several Linux tape devices are nonrewinding. The nonrewinding SCSI tape device is called `/dev/nst0`. Use this device name if you want to write one archive after another on a tape.

After each archive, the nonrewinding tape device writes an end-of-file (EOF) marker to separate one archive from the next. Use the `mt` command to control the tape — you can move from one marker to the next or rewind the tape. For example, after you finish writing several archives to a tape using the `/dev/nst0` device name, you can force the tape to rewind with the following command:

```
mt -f /dev/nst0 rewind
```

After rewinding the tape, you can use the following command to extract files from the first archive to the current disk directory:

```
tar xvf /dev/nst0
```

After that, you must move past the EOF marker to the next archive. To do so, use the following `mt` command:

```
mt -f /dev/nst0 fsf 1
```

This positions the tape at the beginning of the next archive. Now use the `tar xvf` command again to read this archive.

If you save multiple archives on a tape, you have to keep track of the archives yourself. The order of the archives can be hard to remember, so you may be better off simply saving one archive per tape.

Performing incremental backups

Suppose you use `tar` to back up your system's hard drive on a tape. Because such a full backup can take quite some time, you don't want to repeat this task every night. (Besides, only a small number of files may have changed during the day.) To locate the files that need backing up, you can use the `find` command to list all files that have changed in the past 24 hours:

```
find / -mtime -1 -type f -print
```

This command prints a list of files that have changed within the last day. The `-mtime -1` option means you want the files that were last modified less than one day ago. You can now combine this `find` command with the `tar` command to back up only those files that have changed within the last day:

```
tar cvf /dev/st0 'find / -mtime -1 -type f -print'
```

When you place a command between single back quotes, the shell executes that command and places the output at that point in the command line. The result is that the `tar` program saves only the changed files in the archive. This process gives you an incremental backup of only the files that have changed since the previous day.

Performing automated backups

Chapter 1 of this minibook shows how to use `crontab` to set up recurring jobs (called *`cron` jobs*). The Linux system performs these tasks at regular intervals. Backing up your system is a good use of the `crontab` facility. Suppose your backup strategy is as follows:

- Every Sunday at 1:15 a.m., your system backs up the entire hard drive on the tape.
- Monday through Saturday, your system performs an incremental backup at 3:10 a.m. by saving only those files that have changed during the past 24 hours.

To set up this automated backup schedule, log in as `root` and type the following lines in a file named `backups` (this example assumes that you are using a SCSI tape drive):

```
15 1 * * 0 tar zcvf /dev/st0 /
10 3 * * 1-6 tar zcvf /dev/st0 'find / -mtime -1 -type f -print'
```

Next, submit this job schedule by using the following `crontab` command:

```
crontab backups
```

Now you're set for an automated backup. All you need to do is to place a new tape in the tape drive every day. Remember also to give each tape an appropriate label.

Accessing a DOS or Windows File System

If you're using a legacy machine that you just don't want to throw out and have a really old version of Microsoft Windows installed on your hard drive, you've probably already mounted the DOS or Windows partition under Linux. If not, you can easily mount DOS or Windows partitions in Linux. Mounting makes the DOS or Windows directory hierarchy appear as part of the Linux file system.

Mounting a DOS or Windows disk partition

LXO-101 Exam

To mount a DOS or Windows hard drive partition or storage device in Linux, use the `mount` command but include the option `-t vfat` to indicate the file-system type as DOS. For example, if your DOS partition happens to be the first partition on your IDE (Integrated Drive Electronics) drive and you want to mount it on `/dosc`, use the following `mount` command:

```
mount -t vfat /dev/hda1 /dosc
```

The `-t vfat` part of the `mount` command specifies that the device you mount — `/dev/hda1` — has an MS-DOS file system. Figure 3-5 illustrates the effect of this `mount` command.

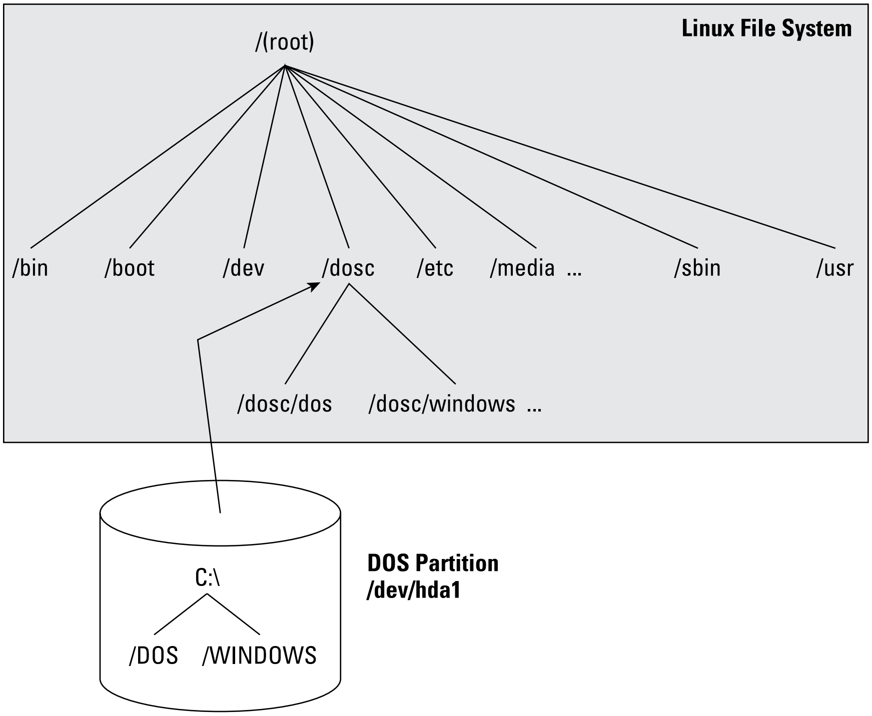


Figure 3-5: Here's how you mount a DOS partition on the /dosc directory.

Figure 3-5 shows how directories in your DOS partition map to the Linux file system. What was the `C:\DOS` directory under DOS becomes `/dosc/dos` under Linux. Similarly, `C:\WINDOWS` now is `/dosc/windows`. You probably can see the pattern. To convert a DOS filename to Linux (when you mount the DOS partition on `/dosc`), perform the following steps:

1. **Change the DOS names to lowercase.**
2. **Change** `C:\` **to** `/dosc/`**.**
3. **Change all backslashes (\) to slashes (/).**

Mounting those ancient DOS floppy disks

Just as you mount a DOS hard drive partition on the Linux file system, you can also mount a DOS floppy disk on a legacy machine. You must log in as `root` to mount a floppy, but you can set up your system so that any user can mount a DOS floppy disk. You also have to know the device name for the floppy drive. By default, Linux defines the following two generic floppy device names:

- `/dev/fd0` is the A drive (the first floppy drive)
- `/dev/fd1` is the B drive (the second floppy drive, if you have one)

You can use any empty directory in the file system as the mount point, but the Linux system comes with a directory, `/media/floppy`, specifically for mounting a floppy disk.

To mount a DOS floppy disk on the `/media/floppy` directory, put the floppy in the drive and type the following command:

```
mount -t vfat /dev/fd0 /media/floppy
```

After you mount the floppy, you can copy files to and from the floppy by using the Linux copy command (`cp`). To copy the file `gnome1.pcx` from the current directory to the floppy, type the following:

```
cp gnome1.pcx /media/floppy
```

Similarly, to see the contents of the floppy disk, type the following:

```
ls /media/floppy
```

If you want to remove the floppy disk from the drive, first unmount the floppy drive. *Unmounting* removes the association between the floppy disk's file system and the mount point on the Linux file system. Use the `umount` command to unmount the floppy disk like this:

```
umount /dev/fd0
```

You can set up your Linux system so that any user can mount a DOS floppy. To enable any user to mount a DOS floppy in the A drive on the `/a` directory, for example, perform the following steps:

1. **Log in as** `root`**.**
2. **Create the** `/a` **directory (the mount point) by typing the following command in a terminal window:**

   ```
   mkdir /a
   ```

3. **Edit the** `/etc/fstab` **file in a text editor (such as** `vi` **or** `emacs`**) by inserting the following line, and then save the file and quit the editor:**

   ```
   /dev/fd0 /a vfat noauto,user 0 0
   ```

   The first field in that line is the device name of the floppy drive (`/dev/fd0`); the second field is the mount directory (`/a`); the third field shows the type of file system (`vfat`). The user option (which appears next to `noauto`) enables all users to mount DOS floppy disks.

4. **Log out and then log back in as a normal user.**
5. **To confirm that you can mount a DOS floppy as a normal user and not just as** `root`**, insert a DOS floppy in the A drive and type the following command:**

   ```
   mount /a
   ```

   The `mount` operation succeeds, and you see a listing of the DOS floppy when you type the command **ls /a**.

6. **To unmount the DOS floppy, type** umount /a**.**

Mounting an NTFS partition

LX0-101 Exam

Nowadays, most PCs come with Windows 7 or Windows 8 pre-installed on the hard drive. Both of these versions of Windows, as well as Windows XP, typically use the NT File System (NTFS). Linux supports read-only access to NTFS partitions, and many distributions come with the `ntfs.ko` kernel module, which is needed to access an NTFS partition.

If you've installed Linux on a Windows system and want to access files on the NTFS partition but your distribution doesn't include the `ntfs.ko` module, you can build the kernel after enabling an NTFS module during the kernel configuration step.

After rebuilding and booting from the new kernel, log in as `root` and type the following command to create a mount point for the NTFS partition. (In this case, I'm creating a mount point in the `/mnt` directory.)

```
mkdir /mnt/windir
```

Now you can mount the NTFS partition with the following command:

```
mount /dev/hda2 /mnt/windir -t ntfs -r -o umask=0222
```

If necessary, replace /dev/hda2 with the device name for the NTFS partition on your system. On most PCs that come with Windows pre-installed, the NTFS partition is the second one (/dev/hda2) — the first partition (/dev/hda1) is usually a hidden partition used to hold files used for the Windows installation.

Chapter 4: Working with Samba and NFS

In This Chapter

- **Sharing files with Network File System**
- **Installing and configuring Samba**
- **Setting up a Windows server using Samba**

If your local area network is like many others, it needs the capability to share files between systems that run Linux and other systems that don't. Thus, Linux includes two prominent file-sharing services:

- **Network File System (NFS):** For sharing files with other Unix systems (or PCs with NFS client software)
- **Samba:** For file sharing and print sharing with Windows systems

This chapter describes how to share files using both NFS and Samba.

Sharing Files with NFS

Sharing files through NFS is simple and involves two basic steps:

- On the Linux system that runs the NFS server, you export (share) one or more directories by listing them in the `/etc/exports` file and by running the `exportfs` command. In addition, you must start the NFS server.
- On each client system, you use the `mount` command to mount the directories that your server has exported.

The only problem in using NFS is that each client system must support it. Microsoft Windows doesn't come with NFS, so you have to buy NFS software separately if you want to share files by using NFS. However, using NFS if all systems on your LAN run Linux (or other variants of Unix with built-in NFS support) makes sense.

WARNING!

NFS has security vulnerabilities. Therefore you should not set up NFS on systems directly connected to the Internet without using the RPCSEC_GSS security that comes with NFS version 4 (NFSv4).

The Linux 2.6 kernel includes support for NFSv4, which is built on earlier versions of NFS. But unlike earlier versions, NFSv4 has stronger security and was designed to operate in an Internet environment. (RFC 3510 describes NFSv4; see `www.ietf.org/rfc/rfc3530.txt`.) NFSv4 uses the RPCSEC_GSS (GSS stands for Generic Security Services) protocol for security. You can continue to use the older user ID- and group ID-based authentication with NFSv4, but if you want to use RPCSEC_GSS you have to run three additional services: `rpcsvcgassd` on the server, `rpsgssd` on the client, and `rpcidmapd` on both the client and the server. For more information about NFSv4 implementation in Linux, visit `www.citi.umich.edu/projects/nfsv4/linux`.

The next few sections walk you through NFS setup, using an example of two Linux PCs on a LAN.

Exporting a file system with NFS

Start with the server system that *exports* — makes available to the client systems — the contents of a directory. On the server, you must run the NFS service and also designate one or more file systems to export.

To export a file system, you have to add an appropriate entry to the `/etc/exports` file. For example, suppose that you want to export the `/home` directory and you want to enable the host named `LNBP75` to mount this file system for read and write operations. You can do so by adding the following entry to the `/etc/exports` file:

```
/home LNBP75(rw,sync)
```

If you want to give access to all hosts on a LAN such as `192.168.0.0`, you could change this line to

```
/home 192.168.0.0/24(rw,sync)
```

Every line in the `/etc/exports` file has this general format:

```
directory host1(options) host2(options) . . .
```

The first field is the directory being shared via NFS, followed by one or more fields that specify which hosts can mount that directory remotely and a number of options in parentheses. You can specify the hosts with names or IP addresses, including ranges of addresses.

The options in parentheses denote the kind of access each host is granted and how user and group IDs from the server are mapped to ID the client. (For example, if a file is owned by `root` on the server, what owner is that on the client?) Within the parentheses, commas separate the options. For example, if a host is allowed both read and write access — and all IDs are to be mapped to the anonymous user (by default, the anonymous user is named nobody) — the options look like this:

```
(rw,all_squash)
```

Table 4-1 shows the options you can use in the /etc/exports file. You find two types of options: general options and user ID mapping options.

Table 4-1 Options in /etc/exports

Option	*Description*
General Options	
`secure`	Allows connections only from ports 1024 or lower (default)
`insecure`	Allows connections from ports 1024 or higher
`ro`	Allows read-only access (default)
`rw`	Allows both read and write access
`sync`	Performs write operations (writing information to the disk) when requested (by default)
`async`	Performs write operations when the server is ready
`no_wdelay`	Performs write operations immediately
`wdelay`	Waits a bit to see whether related write requests arrive and then performs them together (by default)
`hide`	Hides an exported directory that's a subdirectory of another exported directory (by default)
`no_hide`	Causes a directory to not be hidden (opposite of `hide`)
`subtree_check`	Performs subtree checking, which involves checking parent directories of an exported subdirectory whenever a file is accessed (by default)
`no_subtree_check`	Turns off subtree checking (opposite of `subtree_check`)
`insecure_locks`	Allows insecure file locking
User ID Mapping Options	
`all_squash`	Maps all user IDs and group IDs to the anonymous user on the client
`no_all_squash`	Maps remote user and group IDs to similar IDs on the client (by default)
`root_squash`	Maps remote `root` user to the anonymous user on the client (by default)

(continued)

Table 4-1 *(continued)*

Option	*Description*
`no_root_squash`	Maps remote `root` user to the local `root` user
`anonuid=UID`	Sets the user ID of anonymous user to be used for the `all_squash` and `root_squash` options
`anongid=GID`	Sets the group ID of anonymous user to be used for the `all_squash` and `root_squash` options

After adding the entry in the `/etc/exports` file, manually export the file system by typing the following command in a terminal window:

```
exportfs -a
```

This command exports all file systems defined in the `/etc/exports` file.

Now you can start the NFS server processes.

In Debian, start the NFS server by logging in as `root` and typing **/etc/init.d/nfs-kernel-server start** in a terminal window. In Fedora, type **/etc/init.d/nfs start**. In SUSE, type **/etc/init.d/nfsserver start**. If you want the NFS server to start when the system boots, type **update-rc.d nfs-kernel-server defaults** in Debian. In Fedora, type **chkconfig - -level 35 nfs on**. In SUSE, type **chkconfig - -level 35 nfsserver on**. In Xandros, type **update-rc.d nfs-user-server defaults**.

When the NFS service is up, the server side of NFS is ready. Now you can try to mount the exported file system from a client system and then access the exported file system as needed.

If you ever make any changes to the exported file systems listed in the `/etc/exports` file, remember to restart the NFS service. To restart a service, invoke the script in the `/etc/init.d` directory with `restart` as the argument (instead of the `start` argument that you use to start the service).

Mounting an NFS file system

To access an exported NFS file system on a client system, you have to mount that file system on a mount point. The *mount point* is nothing more than a local directory. For example, suppose that you want to access the `/home` directory exported from the server named `LNBP200` at the local directory `/mnt/lnbp200` on the client system. To do so, follow these steps:

1. **Log in as `root` and create the directory with this command:**

   ```
   mkdir /mnt/lnbp200
   ```

2. **Type the following command to mount the directory from the remote system (`LNBP200`) on the local directory `/mnt/lnbp200`:**

   ```
   mount lnbp200:/home /mnt/lnbp200
   ```

After completing these steps, you can then view and access exported files from the local directory `/mnt/lnbp200`.

To confirm that the NFS file system is indeed mounted, log in as `root` on the client system and type **mount** in a terminal window. You see a line similar to the following about the NFS file system:

```
lnbp200:/home/public on /mnt/lnbp200 type nfs (rw,addr=192.168.0.4)
```

NFS supports two types of mount operations — hard and soft. By default, a mount is hard, which means that if the NFS server does not respond, the client keeps trying to access the server indefinitely until the server responds. You can soft mount an NFS volume by adding the `-o soft` option to the `mount` command. For a soft mount, the client returns an error if the NFS server fails to respond.

Setting Up a Windows Server Using Samba

If you rely on Windows for file sharing and print sharing, you probably use Windows in your servers and clients. If so, you can still move to a Linux PC as your server without losing Windows file-sharing and print-sharing capabilities; you can set up Linux as a Windows server. When you install Linux from this book's companion DVD-ROM, you also get a chance to install the Samba software package, which performs that setup. All you have to do is select the Windows File Server package group during installation.

After you install and configure Samba on your Linux PC, your client PCs — even if they're running an old Windows operating system or one of the more recent Windows versions — can access shared disks and printers on the Linux PC. To do so, they use the Common Internet File System (CIFS) protocol, the underlying protocol in Windows file and print sharing.

With the Samba package installed, you can make your Linux PC a Windows client, which means that the Linux PC can access the disks and printers that a Windows server manages. At the same time, your Linux PC can be a client to other Windows systems on the network.

The Samba software package has these major components:

- `/etc/samba/smb.conf`: The Samba configuration file that the SMB server uses.
- `/etc/samba/smbusers`: A Samba configuration file that shows the Samba usernames corresponding to usernames on the local Linux PC.

- `nmbd`: The NetBIOS name server, which clients use to look up servers. (NetBIOS stands for *Network Basic Input/Output System* — an interface that applications use to communicate with network transports, such as TCP/IP.)
- `nmblookup`: A command that returns the IP address of a Windows PC identified by its NetBIOS name.
- `smbadduser`: A program that adds users to the SMB (Server Message Block) password file.
- `smbcacls`: A program that manipulates Windows NT access control lists (ACLs) on shared files.
- `smbclient`: The Windows client, which runs on Linux and allows Linux to access the files and printer on any Windows server.
- `smbcontrol`: A program that sends messages to the `smbd`, `nmbd`, or `winbindd` processes.
- `smbd`: The SMB server, which accepts connections from Windows clients and provides file-sharing and print-sharing services.
- `smbmount`: A program that mounts a Samba share directory on a Linux PC.
- `smbpasswd`: A program that changes the password for an SMB user.
- `smbprint`: A script that enables printing on a printer on an SMB server.
- `smbstatus`: A command that lists the current SMB connections for the local host.
- `smbtar`: A program that backs up SMB shares directly to tape drives on the Linux system.
- `smbumount`: A program that unmounts a currently mounted Samba share directory.
- `testparm`: A program that ensures that the Samba configuration file is correct.
- `winbindd`: A server that resolves names from Windows NT servers.

The following sections describe how to configure and use Samba.

Installing Samba

You may have already installed Samba when you installed Linux. You can check first, and if you don't find Samba on your system, you can easily install it.

DISTRIBUTION SPECIFIC

To see whether Samba is installed, type **dpkg -l samba*** in Debian, Ubuntu, and Xandros or type **rpm -q samba** in Fedora and SUSE.

In Debian and Ubuntu, type **apt-get install samba** to install Samba. In Fedora, log in as `root` and type **yum install samba samba-swat**. This installs not only `samba` but also the web configuration interface, SWAT (Samba Web Administration Tool). In SUSE, click Software Management in the YaST Control Center's Software category. Then use YaST's search facility to look for `samba`, select the relevant packages, and install them. As for Xandros, you get Samba when you install Xandros.

After installing the Samba software, you have to configure Samba before you can use it.

Configuring Samba

To set up the Windows file-sharing and print-sharing services, you can either edit the configuration file manually or use a GUI tool. Using the GUI tool is much easier than editing a configuration file. Fedora and SUSE come with GUI tools for configuring the Samba server.

In Fedora, choose System Settings➪Advanced➪Samba from the KDE desktop to open the Samba Server Configuration window. Enter a valid username and password at the prompt, and the configuration interface that follows lets you create and edit entries in the configuration file `/etc/samba/smb.conf`.

In SUSE, you can configure Samba through the YaST Control Center — choose System➪Control Center (YaST) from the main menu. Click Network Services on the left side of the window and then click Samba Server on the right side of the window. In the window that appears, select a workgroup name (YaST displays the name of any existing Windows workgroup on your LAN) and click Next. Then you can select the server type, enable the server, and select what you want to share. After you exit the Samba server configuration utility, YaST stores the Samba settings in configuration files in the `/etc/samba` directory.

After configuring Samba, type the following command in a terminal window to verify that the Samba configuration file is okay:

```
testparm
```

If the command says that it loaded the files okay, you're all set to go. The `testparm` command also displays the contents of the Samba configuration file.

Samba uses the `/etc/samba/smb.conf` file as its configuration file. This is a text file with a syntax similar to that of a Microsoft Windows 3.1 `INI` file. You can edit that file in any text editor on your Linux system. Like the old Windows `INI` files, the `/etc/samba/smb.conf` file consists of sections, with a list of parameters in each section. Each section of the `smb.conf` file begins with the name of the section in brackets. The section continues until the next section begins or until the file ends. Each line uses the `name = value` syntax to specify the value of a parameter. As in Windows `INI` files,

Discovering more about Samba

This chapter is only an introduction to Samba. To find out more about Samba, you can consult the following resources:

- To view Samba documentation online, visit `www.samba.org/samba/docs/man/Samba-HOWTO-Collection`.
- *Using Samba,* 3rd Edition, by Jay Ts, Robert Eckstein, and David Collier-Brown (O'Reilly & Associates, 2007)

You should also visit `www.samba.org` to keep up with the latest news on Samba development. This site also has links to resources for learning Samba.

comment lines begin with a semicolon (`;`). In the `/etc/samba/smb.conf` file, comments may also begin with a hash mark (#).

To start the Samba services automatically when the system reboots, type **update-rc.d samba defaults** in Debian, Ubuntu, and Xandros or type **chkconfig - -level 35 smb on** in Fedora and SUSE. To start Samba immediately, type **/etc/init.d/smb start** in Fedora and SUSE or type **/etc/init.d/samba start** in Debian, Ubuntu, and Xandros.

Trying out Samba

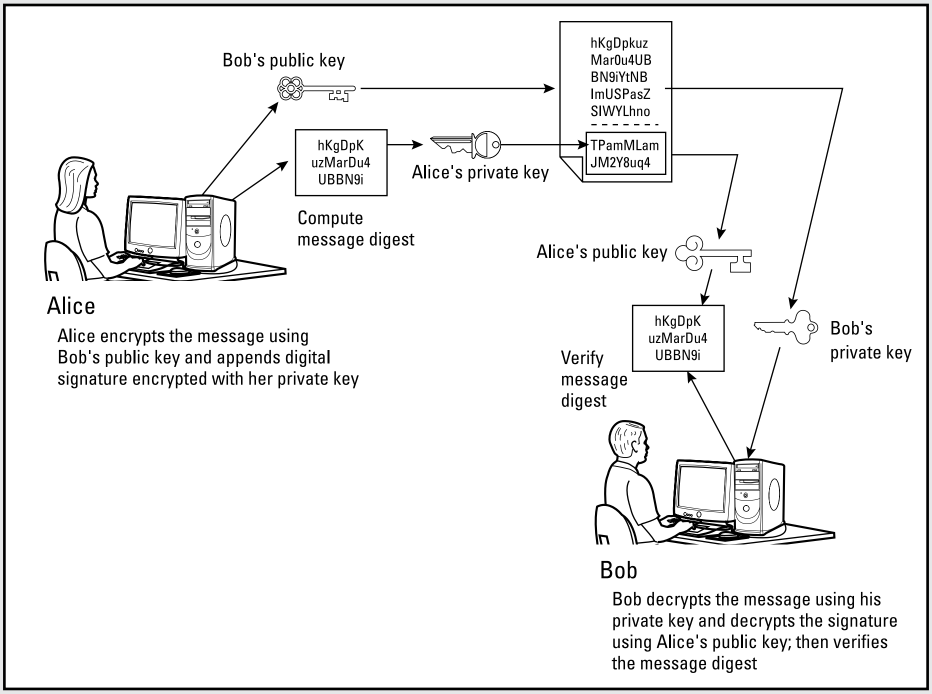
You can now access the Samba server on the Linux system from one of the Windows systems on the LAN. Double-click the Network Neighborhood icon on the Windows 95/98/ME desktop. On Windows XP, choose Start➪My Network Places and then click View Workgroup Computers. All the computers on the same workgroup are shown. In Windows, choose Start➪Computer➪Network.

When you see the Samba server, you can open it by double-clicking the icon. After you enter your Samba username and password, you can access the folders and printers (if any) on the Samba share.

You can use the `smbclient` program to access shared directories and printers on Windows systems on the LAN and to ensure that your Linux Samba server is working. One quick way to check is to type **smbclient -L** in a terminal window to view the list of services on the Linux Samba server itself.

Book VI

Security


Bob's public key
hKgDpkuz
Mar0u4UB
BN9iYtNB
ImUSPasZ
SIWYLhno
TPamMLam
JM2Y8uq4
hKgDpK
uzMarDu4
UBBN9i
Alice's private key
Compute
message digest
Alice's public key
Alice
Alice encrypts the message using Bob's public key and appends digital signature encrypted with her private key
hKgDpK
uzMarDu4
UBBN9i
Verify
message
digest
Bob's
private key
Bob
Bob decrypts the message using his private key and decrypts the signature using Alice's public key; then verifies the message digest

Contents at a Glance

Chapter 1: Introducing Linux Security 383

Why Worry about Security?....383
Establishing a Security Framework....384
Securing Linux....389
Delving into Computer Security Terminology & Tools....391
Keeping Up with Security News and Updates....397

Chapter 2: Securing Linux. 399

Securing Passwords....399
Protecting Files and Directories....402
Encrypting and Signing Files with GnuPG....406
Monitoring System Security....412
Securing Internet Services....413
Using Secure Shell (SSH) for Remote Logins....415
Setting Up Simple Firewalls....418
Security Files to Be Aware Of....426

Chapter 3: Computer Security Audits and Vulnerability Testing Types. 429

Understanding Security Audits....429
Implementing a Security Test Methodology....431
Vulnerability Testing Types....440
Exploring Security Testing Tools....440

Chapter 1: Introducing Linux Security

In This Chapter

- **Establishing a security policy and framework**
- **Understanding host security issues**
- **Understanding network security issues**
- **Translating computer security terminology**
- **Keeping up with security news and updates**

This chapter explains why you need to worry about security — and offers a high-level view of how to get a handle on security. The idea of an overall security framework is explained and the two key aspects of security — host security and network security — are discussed. This chapter ends by introducing you to the terminology used in discussing computer security.

LX0-102 Exam

According to the weighting, 15 percent of the questions on the LX0-102 exam fall under the Security domain. This number should be viewed as being very conservative since so much of administration involves security. You'll find topics related to it in domains such as Administrative Tasks, Essential System Services, Networking Fundamentals, and so on. Because of that, you'll find a lot of security-relevant information in the three chapters of Book 6 and in other chapters as well.

Why Worry about Security?

In today's networked world, you have to worry about your Linux system's security. For a standalone system or a system used in an isolated local area network (LAN), you have to focus on protecting the system from the users, and the users from one another. In other words, you don't want a user to modify or delete system files, whether intentionally or unintentionally — and you don't want a user destroying another user's files (or their own, if you can prevent it).

Since the odds are good that your Linux system is connected to the Internet, you have to secure the system from unwanted accesses over the Internet. These intruders — or *crackers,* as they're commonly known — typically impersonate a user, steal or destroy information, and even deny you access to your own system — known as a *Denial of Service (DoS),* or *Distributed Denial of Service (DDoS),* attack.

By its very nature, an Internet connection makes your system accessible to any other system on the Internet. After all, the Internet connects a huge number of networks across the globe. In fact, the client/server architecture of Internet services, such as HTTP (web) and FTP, rely on the wide-open network access the Internet provides. Unfortunately, the easy accessibility to Internet services running on your system also means that anyone on the Net can easily access your system.

If you operate an Internet host that provides information to others, you certainly want everyone to access your system's Internet services, such as FTP and web servers. However, these servers often have vulnerabilities that crackers may exploit to harm your system. You need to know about the potential security risks of Internet services — and the precautions you can take to minimize the risk of someone exploiting the weaknesses of your FTP or web server.

You also want to protect your company's internal network from outsiders, even though your goal is to provide information to the outside world through your web or FTP server. You can protect your internal network by setting up an Internet *firewall* — a controlled-access point to the internal network — and placing the web and FTP servers on a host outside the firewall.

Establishing a Security Framework

The first step in securing your Linux system is to set up a *security policy* — a set of guidelines that state what you enable users (as well as visitors over the Internet) to do on your Linux system. The level of security you establish depends on how you use the Linux system — and on how much is at risk if someone gains unauthorized access to your system.

If you're a system administrator for one or more Linux systems at an organization, you probably want to involve company management, as well as the users, in setting up the security policy. Obviously, you can't create a draconian policy that blocks all access. (That would prevent anyone from effectively working on the system.) On the other hand, if the users are creating or using data valuable to the organization, you have to set up a policy that protects the data from disclosure to outsiders. In other words, the security policy should strike a balance between the users' needs and the need to protect the system.

For a standalone Linux system or a home system that you occasionally connect to the Internet, the security policy can be just a listing of the Internet services that you want to run on the system and the user accounts that you plan to set up on the system. For any larger organization, you probably have one or more Linux systems on a LAN connected to the Internet — preferably through a firewall. (To reiterate, a *firewall* is a device that controls the flow of Internet Protocol — IP — packets between the LAN and the Internet.) In such cases, thinking of computer security systematically — across the entire organization — is best. Figure 1-1 shows the key elements of an organization-wide framework for computer security.

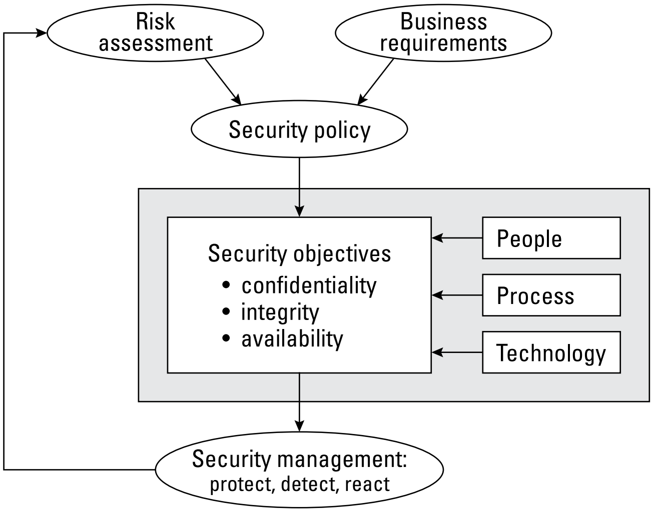


Figure 1-1: Start with an organization-wide framework for computer security.

The security framework outlined in Figure 1-1 focuses on

- ✦ Determining the business requirements for security
- ✦ Performing risk assessments
- ✦ Establishing a security policy
- ✦ Implementing a cybersecurity solution that includes people, process, and technology to mitigate identified security risks
- ✦ Continuously monitoring and managing security

The following sections discuss some of the key elements of the security framework.

Determining business requirements for security

The business requirements for security identify the computer resources and information you have to protect (including any requirements imposed by applicable laws, such as the requirement to protect the privacy of some types of data). Typical security requirements may include items such as the following:

- ✦ Enabling access to information by authorized users
- ✦ Implementing business rules that specify who has access to what information
- ✦ Employing a strong user-authentication system
- ✦ Denying execution to malicious or destructive actions on data

- Protecting data from end to end as it moves across networks
- Implementing all security and privacy requirements that applicable laws impose

Performing risk analysis

Risk analysis is all about identifying and assessing risks — potential events that can harm your Linux system. The analysis involves determining the following and performing some analysis to establish the priority for handling the risks:

- **Threats:** What you're protecting against
- **Vulnerabilities:** Weaknesses that may be exploited by threats (these are the risks)
- **Probability:** The likelihood that a threat will exploit the vulnerability
- **Impact:** The effect of exploiting a specific vulnerability
- **Mitigation:** What to do to reduce vulnerabilities

Typical threats

Some typical threats to your Linux system include the following:

- **Denial of Service:** The computer and network are tied up so legitimate users can't make use of the systems. For businesses, Denial of Service (DoS) can mean a loss of revenue. Since bringing a system to its knees with a single computer attack is a bit of a challenge these days, the more common tactic is to point a number of computers at a single site and let them do the dirty work. While the purpose and result are the same as ever, this ganging up is referred to as Distributed Denial of Service (DDoS) attack because more than one computer is attacking the host.
- **Unauthorized access:** Use of the computer and network by someone who isn't an authorized user. The unauthorized user can steal information or maliciously corrupt or destroy data. Some businesses may be hurt by the negative publicity resulting from the mere act of an unauthorized user gaining access to the system, even if the data shows no sign of explicit damage.
- **Disclosure of information to the public:** The unauthorized release of information to the public. For example, the disclosure of a password file enables potential attackers to figure out username and password combinations for accessing a system. Exposure of other sensitive information, such as financial and medical data, may be a potential liability for a business.

Typical vulnerabilities

The threats to your system and network come from exploitation of vulnerabilities in your organization's resources — both computer and people. Some common vulnerabilities follow:

- People's foibles (divulging passwords, losing security cards, and so on)
- Internal network connections (routers, switches)
- Interconnection points (gateways — routers and firewalls — between the Internet and the internal network)
- Third-party network providers (ISPs, long-distance carriers) with looser security
- Operating system security holes (potential holes in Internet servers, such as those associated with `sendmail`, `named`, and `bind`)
- Application security holes (known weaknesses in specific applications)

The 1-2-3 of risk analysis (probability and effect)

To perform risk analysis, assign a numeric value to the probability and effect of each potential vulnerability. To develop a workable risk analysis, do the following for each vulnerability or risk:

1. Assign subjective ratings of low, medium, and high to the probability. As the ratings suggest, low probability means a lesser chance that the vulnerability will be exploited; high probability means a greater chance.
2. Assign similar ratings to the effect. What you consider the effect is up to you. If the exploitation of a vulnerability will affect your business greatly, assign it a high effect rating.
3. Assign a numeric value to the three levels — low = 1, medium = 2, and high = 3 — for both probability and effect.
4. Multiply the probability by the effect — you can think of this product as the *risk level*. Then make a decision to develop protections for vulnerabilities that exceed a specific threshold for the product of probability and effect. For example, you may choose to handle all vulnerabilities that have a probability-times-effect value greater than 6.

If you want to characterize the probability and effect with finer gradations, use a scale of 1 through 5 (for example) instead of 1 through 3, and follow the same steps as before.

Establishing a security policy

Using risk analysis and any business requirements that you may have to address (regardless of risk level) as a foundation, you can craft a security policy for the organization. Such a security policy typically addresses high-level objectives such as ensuring the confidentiality, integrity, and availability of data and systems.

The security policy typically addresses the following areas:

- **Authentication:** What method is used to ensure that a user is the real user? Who gets access to the system? What is the minimum length and complexity of passwords? How often do users change passwords? How long can a user be idle before that user is logged out automatically?
- **Authorization:** What can different classes of users do on the system? Who can have the `root` password?
- **Data protection:** What data must be protected? Who has access to the data? Is encryption necessary for some data?
- **Internet access:** What are the restrictions on users (from the LAN) accessing the Internet? What Internet services (such as web, Internet Relay Chat, and so on) can users access? Are incoming e-mails and attachments scanned for viruses? Is there a network firewall? Are virtual private networks (VPNs) used to connect private networks across the Internet?
- **Internet services:** What Internet services are allowed on each Linux system? Are there any file servers, mail servers, or web servers? What services run on each type of server? What services, if any, run on Linux systems used as desktop workstations?
- **Security audits:** Who tests whether the security is adequate? How often is the security tested? How are problems found during security testing handled?
- **Incident handling:** What are the procedures for handling any computer security incidents? Who must be informed? What information must be gathered to help with the investigation of incidents?
- **Responsibilities:** Who is responsible for maintaining security? Who monitors log files and audit trails for signs of unauthorized access? Who maintains the security policy?

Implementing security solutions (mitigation)

After you analyze the risks — vulnerabilities — and develop a security policy, you have to select the *mitigation approach:* how to protect against specific vulnerabilities. This is where you develop an overall security solution based on security policy, business requirements, and available technology — a solution that makes use of people, process, and technology and includes the following:

- Services (authentication, access control, encryption)
- Mechanisms (username and password, firewalls)
- Objects (hardware, software)

Because it's impossible to protect computer systems from all attacks, solutions identified through the risk management process must support three integral concepts of a holistic security program:

- **Protection:** Provides countermeasures such as policies, procedures, and technical solutions to defend against attacks on the assets being protected.
- **Detection:** Monitors for potential breakdowns in the protective measures that could result in security breaches.
- **Reaction** or **Response:** Responds to detected breaches to thwart attacks before damage occurs; often requires human involvement.

Because absolute protection from attacks is impossible to achieve, a security program that doesn't incorporate detection and reaction is incomplete.

Managing security

In addition to implementing security solutions, you have to install security management that continually monitors, detects, and responds to any security incidents.

The combination of the risk analysis, security policy, security solutions, and security management provides the overall security framework. Such a framework helps establish a common level of understanding of security concerns — and a common basis for the design and implementation of security solutions.

Securing Linux

After you define a security policy, you can proceed to secure the system according to the policy. The exact steps depend on what you want to do with the system, whether the system is a server or workstation, and how many users must access the system.

To secure the Linux system, you have to handle two broad categories of security issues:

- **Host-security issues:** These issues relate to securing the operating system and the files and directories on the system.
- **Network-security issues:** These issues refer to the threat of attacks over the network connection.

If your host is connecting to a large network, Directory Services can become a significant issue. Directory Services security is outside the scope of this book, but you can find a number of sources addressing the issue with a Google search.

Understanding the host-security issues

Here are some high-level guidelines to address host security. (I cover some of these topics in detail in Chapter 2 of this minibook.)

LXO-102 Exam

- When installing Linux, select only the package groups that you need for your system. Don't install unnecessary software. For example, if your system is used as a workstation, you don't have to install most of the servers (web server, news server, and so on).
- Create initial user accounts and make sure that all passwords are strong enough that password-cracking programs can't guess them. Linux includes tools to enforce strong passwords.
- Set file ownerships and permissions to protect important files and directories.
- If mandatory access-control capabilities are available, enable them. Support for this feature has been incorporated, through Security Enhanced Linux (SELinux), since kernel 2.6.

LXO-102 Exam

- Use the GNU Privacy Guard (GnuPG) to encrypt or decrypt files with sensitive information and to authenticate files that you download from the Internet. GnuPG comes with Linux, and you can use the `gpg` command to perform tasks such as encrypting or decrypting a file and digitally signing a file. (See Chapter 2 of this minibook for an explanation of digital signatures.)
- Use file-integrity checking tools, such as Tripwire, to monitor any changes to crucial system files and directories. Visit `www.tripwire.com` for the commercial version.
- Periodically, check various log files for signs of any break-ins or attempted break-ins. These log files are in the `/var/log` directory of your system.
- Install security updates as soon as they are available and tested. These security updates fix known vulnerabilities in Linux. Be sure to test the update on nonproduction machines before rolling it out to your production servers.

Understanding network-security issues

The issue of security comes up as soon as you connect your organization's internal network to the Internet. You need to think of security even if you connect a single computer to the Internet, but security concerns are more pressing when an entire internal network is opened to the world.

If you're an experienced system administrator, you already know that the cost of managing an Internet presence doesn't worry corporate management; their main concern is security. To get your management's backing for the website, you have to lay out a plan to keep the corporate network secure from intruders.

You may think that you can avoid jeopardizing the internal network by connecting only external servers, such as web and FTP servers, to the Internet. However, employing this simplistic approach isn't wise. It's like deciding not to drive because you may have an accident. Not having a network connection between your web server and your internal network also has the following drawbacks:

- ✦ You can't use network file transfers, such as FTP, to copy documents and data from your internal network to the web server.
- ✦ Users on the internal network can't access the corporate web server.
- ✦ Users on the internal network don't have access to web servers on the Internet. Such a restriction makes a valuable resource — the web — inaccessible to the users in your organization.

A practical solution to this problem is to set up an Internet firewall and to put the web server on a highly secured host outside the firewall.

In addition to using a firewall, here are some other steps to take to address network security. (I explain these further in Chapter 2 of this minibook.)

- ✦ Enable only those Internet services you need on a system. In particular, don't enable services that aren't properly configured.
- ✦ Use Secure Shell (`ssh`) for remote logins. Don't use the `r` commands, such as `rlogin` and `rsh`.
- ✦ Secure any Internet services, such as FTP or TELNET, that you want to run on your system. You can use the *TCP wrapper* access-control files — `/etc/hosts.allow` and `/etc/hosts.deny` — to secure some of these services. (See Chapter 3 of this minibook for more on the TCP wrapper.)
- ✦ Promptly fix any known vulnerabilities of Internet services that you choose to run. Typically, you can download and install the latest security updates from your Linux distribution's online update sites.

Delving into Computer Security Terminology and Tools

Computer books, magazine articles, and experts on computer security use a number of terms that you need to know in order to understand discussions about computer security (and to communicate effectively with security vendors).

LX0-102 Exam

Table 1-1 describes some of the commonly used computer security terms. If you're taking the LX0-102 exam, port scanning and `setuid` are important.

Table 1-1 Common Computer Security Terminology

Term	*Description*
Application gateway	A proxy service that acts as a gateway for application-level protocols, such as FTP, HTTP, NNTP, and SSH.
Authentication	The process of confirming that a user is indeed who he or she claims to be. The typical authentication method is a challenge-response method wherein the user enters a username and secret password to confirm his or her identity.
Backdoor	A security weakness that a cracker places on a host to bypass security features.
Bastion host	A highly secured computer that serves as an organization's main point of presence on the Internet. A bastion host typically resides on the perimeter network, but a *dual-homed host* (with one network interface connected to the Internet and the other to the internal network) is also a bastion host.
Buffer overflow	A security flaw in a program that enables a cracker to send an excessive amount of data to that program and to overwrite parts of the running program with code in the data being sent. The result is that the cracker can execute arbitrary code on the system and possibly gain access to the system as a privileged user. The new `exec-shield` feature of the Linux kernel protects against buffer overflows.
Certificate	An electronic document that identifies an entity (such as an individual, an organization, or a computer) and associates a public key with that identity. A certificate contains the certificate holder's name, a serial number, an expiration date, a copy of the certificate holder's public key, and the digital signature of the certificate authority so a recipient can verify that the certificate is real.
Certificate authority (CA)	An organization that validates identities and issues certificates.
Confidentiality	Of data, a state of being accessible to no one but authorized users (usually achieved by encryption).
Cracker	A person who breaks into (or attempts to break into) a host, often with malicious intent.
Decryption	The process of transforming encrypted information into its original, intelligible form.

Term	*Description*
Denial of Service (DoS)	An attack that uses so many of the resources on your computer and network that legitimate users can't access and use the system. From a single source, the attack overwhelms the target computer with messages and blocks legitimate traffic. It can prevent one system from being able to exchange data with other systems or prevent the system from using the Internet.
Digital signature	A one-way MD5 (Message Digest algorithm 5) or SHA-1 (Secure Hash Algorithm-1) hash of a message encrypted with the private key of the message originator, used to verify the integrity of a message and ensure nonrepudiation.
Distributed Denial of Service (DDoS)	A variant of the Denial of Service attack that uses a coordinated attack from a distributed system of computers rather than a single source. It often makes use of worms to spread to — and take control of — multiple computers that can then attack the target.
DMZ	Another name for the perimeter network. (DMZ originally stood for *demilitarized zone,* the buffer zone separating the warring North and South in Korea and Vietnam.)
Dual-homed host	A computer with two network interfaces (think of each network as a home).
Encryption	The process of transforming information so it's unintelligible to anyone but the intended recipient. The transformation is performed by a mathematical operation between a key and the information.
Exploit tools	Publicly available and sophisticated tools that intruders of various skill levels can use to determine vulnerabilities and gain entry into targeted systems.
Firewall	A controlled-access gateway between an organization's internal network and the Internet. A dual-homed host can be configured as a firewall.
Hash	The result when a mathematical function converts a message into a fixed-size numeric value known as a *message digest* (or *hash*). The MD5 algorithm, for example, produces a 128-bit message digest; SHA-1 generates a 160-bit message digest. The hash of a message is encrypted with the private key of the sender to produce the digital signature.

(continued)

Table 1-1 ***(continued)***

Term	*Description*
Host	A computer on a network that's configured to offer services to other computers on the network.
Integrity	Of received data, a state of being the same as originally sent (that is, unaltered in transit).
IP spoofing	An attack in which a cracker figures out the IP address of a trusted host and then sends packets that appear to come from the trusted host. The attacker can send packets but can't see responses. However, the attacker can predict the sequence of packets and essentially send commands that set up a backdoor for future break-ins.
IPSec (IP Security Protocol)	A security protocol for the network layer of the OSI networking model, designed to provide cryptographic security services for IP packets. IPSec provides encryption-based authentication, integrity, access control, and confidentiality. (For information on IPSec for Linux, visit `www.ipsec-howto.org`.)
Logic bombs	A form of sabotage in which a programmer inserts code that causes the program to perform a destructive action when some triggering event occurs, such as terminating the programmer's employment.
Nonrepudiation	A security feature that prevents the sender of data from being able to deny ever having sent the data.
Packet	A collection of bytes, assembled according to a specific protocol, that serves as the basic unit of communication on a network. On TCP/IP networks, for example, the packet may be referred to as an *IP packet* or a *TCP/IP packet.*
Packet filtering	Selective blocking of packets according to type of packet (as specified by the source and destination IP address or port).
Perimeter network	A network between the Internet and the protected internal network. The perimeter network (also known as DMZ) is where the bastion host resides.
Port scanning	A method of discovering which ports are open (in other words, which Internet services are enabled) on a system, performed by sending connection requests to the ports, one by one. This procedure is usually a precursor to further attacks; two port-scanning tools to know are `nmap`, and `netstat`.

Term	*Description*
Proxy server	A server on the bastion host that enables internal clients to access external servers (and enables external clients to access servers inside the protected network). There are proxy servers for various Internet services, such as FTP and HTTP.
Public key cryptography	An encryption method that uses a pair of keys — a private key and a public key — to encrypt and decrypt the information. Anything encrypted with the public key is decrypted only with the corresponding private key, and vice versa.
Public Key Infrastructure (PKI)	A set of standards and services that enables the use of public key cryptography and certificates in a networked environment. PKI facilitates tasks such as issuing, renewing, and revoking certificates, and generating and distributing public and private key pairs.
Screening router	An Internet router that filters packets.
`setuid` program	A program that runs with the permissions of the owner regardless of who runs the program. For example, if `root` owns a `setuid/suid` program, that program has `root` privileges regardless of who started the program. Crackers often exploit vulnerabilities in `setuid` programs to gain privileged access to a system. Similarly, `sgid` programs are used to run with the permissions of the group, regardless of who runs the program, and have their own similar vulnerabilities.
Sniffer	Synonymous with *packet sniffer* — a program that intercepts routed data and examines each packet in search of specified information, such as passwords transmitted in clear text.
Spyware	Any software that covertly gathers user information through the user's Internet connection and usually transmits that information in the background to someone else. Spyware can also gather information about e-mail addresses and even passwords and credit card numbers. Spyware is similar to a Trojan horse in that users are tricked into installing spyware when they install something else.
Symmetric key encryption	An encryption method wherein the same key is used to encrypt and decrypt the information.
Threat	An event or activity, deliberate or unintentional, with the potential for causing harm to a system or network.

(continued)

Table 1-1 *(continued)*

Term	*Description*
Trojan horse	A program that masquerades as a benign program but is really a backdoor used for attacking a system. Attackers often install a collection of Trojan horse programs that enable the attacker to freely access the system with `root` privileges, yet hide that fact from the system administrator. Such collections of Trojan horse programs are called *rootkits.*
Virus	A self-replicating program that spreads from one computer to another by attaching itself to other programs.
Vulnerability	A flaw or weakness that may cause harm to a system or network.
War-dialing	Simple programs that dial consecutive phone numbers looking for modems.
War-driving	A method of gaining entry into wireless computer networks that uses a laptop, antennas, and a wireless network card and involves patrolling locations to gain unauthorized access.
Worm	A self-replicating program that copies itself from one computer to another over a network.

LXO-102 Exam

Table 1-2 lists some of the commonly used computer security-related tools. Some of these you've seen before as they were discussed in other chapters where they related to the topics there; some others are new as they are relevant to security only.

Table 1-2 Common Computer Security Tools

Tool	*Description*
`chage`	With this command, you can modify the time between required password changes (both minimum and maximum number of days), the number of days of warning to be given that a change must be made, and expiration date.
`find`	One of the most powerful all-around tools, this command allows you to find almost anything on machine if you can come up with the right syntax. Among the plethora of choices, you can find files created by a user, by a group, on a certain date, with certain permissions.

Tool	*Description*
`lsof`	An acronym for list open files, this utility does just that. Depending on the parameters used, you can choose to see files opened by a process, or by a user.
`netstat`	To see the status of the network, including network connections, routing tables and statistics per interface, this tool does it all. A similar command, `ss`, is intended to replace much of the functionality here.
`nmap`	This tool is used to scan the network and essentially create a map of what is available on it. This capability makes it an ideal tool for port scanning and security auditing.
`passwd`	A utility (not the file by the same name that holds user account information), with which users can change their passwords at the command line whenever necessary. Many users don't know this utility exists, so they change their passwords when required, through one of the graphical interface tools.
`su`	To temporarily become another user, `su` can be used within the current user's session. Another shell is created; upon exiting from this second shell, the user goes back to the original session. This utility can be used to become the `root` user or any other user (provided the corresponding password is given).
`sudo`	Instead of creating a new session (as `su` requires) to perform a job with elevated privileges, `sudo` enables the user to just run that task.
`ulimit`	Resource limits on shells can be set or viewed using this command to keep one user from excessively hogging system resources.
`usermod`	This utility can be thought of as an enhanced version of `chage`. Not only can it be used to set/change password expiration parameters, it can also be used to specify a default shell, lock/unlock an account, and so on.

Keeping Up with Security News and Updates

To keep up with the latest security alerts, you may want to visit one or both of the following sites on a daily basis:

- CERT Coordination Center (CERT/CC) at `www.cert.org`
- United States Computer Emergency Readiness Team (US-CERT) at `www.us-cert.gov`

If you prefer to receive regular security updates through e-mail, you can also sign up for (subscribe to) various mailing lists:

- **Focus on Linux:** Fill out the form at `www.securityfocus.com/archive` to subscribe to this mailing list focused on Linux security issues.
- **US-CERT National Cyber Alert System:** Follow the directions at `www.us-cert.gov` to subscribe to this mailing list. The Cyber Alert System features four categories of security information through its mailing lists:
    - *Technical Cyber Security Alerts:* Alerts that provide technical information about vulnerabilities in various common software products.
    - *Cyber Security Alerts:* Alerts sent when vulnerabilities affect the general public. Each alert outlines the steps and actions that nontechnical home and corporate computer users can take to protect themselves from attacks.
    - *Cyber Security Bulletins:* Biweekly summaries of security issues and new vulnerabilities along with patches, workarounds, and other actions that users can take to help reduce risks.
    - *Cyber Security Tips:* Advice on common security issues for nontechnical computer users.

DISTRIBUTION SPECIFIC

Finally, check your distribution's website for updates that may fix any known security problems with that distribution:

- In Debian and Ubuntu, you can update the system with the commands `apt-get update` followed by `apt-get upgrade`.
- For Fedora, the website is `http://fedoraproject.org`.
- In SUSE, use YaST Online Update to keep your system up to date.
- In Xandros, obtain the latest updates from Xandros Networks.

Chapter 2: Securing Linux

In This Chapter

- ✓ **Securing passwords on your Linux system**
- ✓ **Protecting the system's files and directories**
- ✓ **Using GnuPG to encrypt and sign files**
- ✓ **Monitoring the security of your system**
- ✓ **Hardening Internet services**
- ✓ **Using Secure Shell for secure remote logins**
- ✓ **Setting up simple firewalls and enabling packet filtering**

To secure your Linux system, you have to pay attention to both host security and network security. The distinction between the two types of security is somewhat arbitrary because securing the network involves securing the applications on the host that relate to what Internet services your system offers.

This chapter first examines host security and then explains how you can secure network services (mostly by not offering unnecessary services), how you can use a firewall to stop unwanted network packets from reaching your network, and how to use Secure Shell for secure remote logins.

Host is the techie term for your Linux system — especially when you use it to provide services on a network. But the term makes sense even when you think of the computer by itself; it's the host for everything that runs on it: the operating system and all applications. A key aspect of computer security is to secure the host.

Securing Passwords

Historically, Unix passwords are stored in the `/etc/passwd` file, which any user can read. For example, a typical old-style `/etc/passwd` file entry for the `root` user looks like this:

```
root:t6Z7NWDK1K8sU:0:0:root:/root:/bin/bash
```

The fields are separated by colons (`:`), and the second field contains the password in encrypted form. To check whether a password is valid, the login program encrypts the plain-text password the user enters and compares the password with the contents of the `/etc/passwd` file. If they match, the user is allowed to log in.

Password-cracking programs work just like the login program, except these programs choose one word at a time from a dictionary, encrypt the word, and compare the encrypted word with the encrypted passwords in the `/etc/passwd` file for a match. To crack the passwords, the intruder needs the `/etc/passwd` file. Often crackers use weaknesses of various Internet servers (such as mail and FTP) to get a copy of the `/etc/passwd` file.

Passwords have become more secure in Linux due to several improvements, including shadow passwords and pluggable authentication modules, or PAMs (described in the next two sections). You can install shadow passwords or a PAM easily while you install Linux. During Linux installation, you typically get a chance to configure the authentication. If you enable MD5 security and enable shadow passwords, you automatically enable more secure passwords in Linux.

Shadow passwords

Obviously, leaving passwords lying around where anyone can get at them — even if the passwords are encrypted — is bad security. So instead of storing passwords in the `/etc/passwd` file (which any user can read), Linux now stores them in a *shadow password file,* `/etc/shadow`. Only the superuser (`root`) can read this file. For example, here's the entry for `root` in the new-style `/etc/passwd` file:

```
root:x:0:0:root:/root:/bin/bash
```

In this case, note that the second field contains an `x` instead of an encrypted password. The `x` is the *shadow password;* the actual encrypted password is now stored in the `/etc/shadow` file, where the entry for `root` is like this:

```
root:$1$AAAni/yN$uESHbzUpy9Cgfoo1Bf0tS0:11077:0:99999:7:-1:-1:134540356
```

The format of the `/etc/shadow` entries with colon-separated fields resembles the entries in the `/etc/passwd` file, but the meanings of most of the fields differ. The first field is still the username, and the second one is the encrypted password.

The remaining fields in each `/etc/shadow` entry control when the password expires. You don't have to interpret or change these entries in the `/etc/shadow` file. Instead, use the `chage` command to change the password expiration information. For starters, you can check a user's password expiration information by using the `chage` command with the `-l` option, as follows. (In such a case, you have to be logged in as `root`.)

```
chage -l root
```

This command displays expiration information, including how long the password lasts and how often you can change the password.

If you want to ensure that the user is forced to change a password at regular intervals, you can use the `-M` option to set the maximum number of days that a password stays valid. For example, to make sure that user `kdulaney` is prompted to change the password in 90 days, log in as `root` and type the following command:

```
chage -M 90 kdulaney
```

You can use the command for each user account to ensure that all passwords expire when appropriate and that all users must choose new passwords.

Pluggable authentication modules (PAMs)

In addition to improving the password file's security by using shadow passwords, Linux also improves the encryption of the passwords stored in the `/etc/shadow` file by using the MD5 message-digest algorithm described in RFC 1321 (`www.ietf.org/rfc/rfc1321.txt` or `www.cse.ohio-state.edu/cgi-bin/rfc/rfc1321.html`). MD5 reduces a message of any length to a 128-bit message digest (or *fingerprint*) of a document so that you can digitally sign it by encrypting it with your private key. MD5 works quite well for password encryption, too.

Another advantage of MD5 over older-style password encryption is that the older passwords were limited to a maximum of eight characters; new passwords (encrypted with MD5) can be much longer. Longer passwords are harder to guess, even if the `/etc/shadow` file falls into the wrong hands.

You can tell that MD5 encryption is in effect in the `/etc/shadow` file. The encrypted passwords are longer and they all sport the `$1$` prefix, as in the second field of the following sample entry:

```
root:$1$AAAni/yN$uESHbzUpy9Cgfoo1Bf0tS0:11077:0:99999:7:-1:-1:134540356
```

An add-on program module called a *pluggable authentication module* (PAM) performs the MD5 encryption. Linux PAMs provide a flexible method for authenticating users. By setting the PAM's configuration files, you can change your authentication method on-the-fly, without modifying vital programs that verify a user's identity (such as `login` and `passwd`).

Linux uses PAM capabilities extensively. The PAMs reside in many different modules (more on this momentarily); their configuration files are in the `/etc/pam.d` directory of your system. Check out the contents of this directory on your system by typing the following command:

```
ls /etc/pam.d
```

Each configuration file in this directory specifies how users are authenticated for a specific utility.

Protecting Files and Directories

One important aspect of securing the host is to protect important system files — and the directories that contain these files. You can protect the files through file ownership and the permission settings that control who can read, write, or (in the case of executable programs) execute the file.

The default Linux file security is controlled through the following settings for each file or directory:

- User ownership
- Group ownership
- Read, write, execute permissions for the owner
- Read, write, execute permissions for the group
- Read, write, execute permissions for others (everyone else)

Viewing ownerships and permissions

You can see settings related to ownership and permissions for a file when you look at a detailed listing with the `ls -l` command. For example, type the following command to see the detailed listing of the `/etc/inittab` file:

```
ls -l /etc/inittab
```

The resulting listing looks something like this:

```
-rw-r--r-- 1 root root 1666 Feb 16 07:57 /etc/inittab
```

The first set of characters describes the file permissions for user, group, and others. The third and fourth fields show the user and group that own this file. In this case, both user and group names are the same: `root`.

Changing file ownerships

You can set the user and group ownerships with the `chown` command. For example, if the file `/dev/hda` should be owned by the user `root` and the group `disk`, you type the following command as `root` to set up this ownership:

```
chown root.disk /dev/hda
```

To change the group ownership alone, use the `chgrp` command. For example, here's how you can change the group ownership of a file from whatever it was earlier to the group named `accounting`:

```
chgrp accounting ledger.out
```

Changing file permissions

Use the chmod command to set the file permissions. To use chmod effectively, you have to specify the permission settings. One way is to concatenate one or more letters from each column of Table 2-1, in the order shown (Who/Action/Permission).

Table 2-1 File Permission Codes

Who	*Action*	*Permission*
u (user)	+ (add)	r (read)
g (group)	- (remove)	w (write)
o (others)	= (assign)	x (execute)
a (all)	s (set user ID)	

To give everyone read and write access to all files in a directory, type **chmod a+rw ***. On the other hand, to permit everyone to execute a specific file, type **chmod a+x *filename***.

Another way to specify a permission setting is to use a three-digit sequence of numbers. In a detailed listing, the read, write, and execute permission settings for the user, group, and others appear as the sequence

```
rwxrwxrwx
```

with dashes in place of letters for disallowed operations. Think of rwxrwxrwx as three occurrences of the string rwx. Now assign the values r=4, w=2, and x=1. To get the value of the sequence rwx, simply add the values of r, w, and x. Thus rwx = 7. With this formula, you can assign a three-digit value to any permission setting. For example, if the user can read and write the file but everyone else can only read the file, the permission setting is rw-r--r-- (that's how it appears in the listing), and the value is 644. Thus, if you want all files in a directory to be readable by everyone but writable only by the user, use the following command:

```
chmod 644 *
```

Setting default permission

What permission setting does a file get when you (or a program) create a new file? The answer is in what is known as the *user file-creation mask,* which you can see and set by using the umask command.

Type **umask**, and the command prints a number showing the current file-creation mask. For the root user, the mask is set to 022, whereas the mask for other users is 002. To see the effect of this file-creation mask and to interpret the meaning of the mask, follow these steps:

1. **Log in as** `root` **and type the following command:**

   ```
   touch junkfile
   ```

   This command creates a file named `junkfile` with nothing in it.

2. **Type** ls -l junkfile **to see that file's permissions.**

   You see a line similar to the following:

   ```
   -rw-r--r-- 1 root root 0 Aug 24 10:56 junkfile
   ```

   Interpret the numerical value of the permission setting by converting each three-letter permission in the first field (excluding the very first letter) to a number between 0 and 7. For each letter that's present, the first letter gets a value of 4, the second letter is 2, and the third is 1. For example, `rw-` translates to 4+2+0 (because the third letter is missing), or 6. Similarly, `r--` is 4+0+0 = 4. Thus the permission string `-rw-r--r--` becomes 644.

3. **Subtract the numerical permission setting from 666 and what you get is the** `umask` **setting.**

   In this case, 666 – 644 results in a `umask` of 022.

Thus a `umask` of 022 results in a default permission setting of 666 – 022 = 644. When you rewrite 644 in terms of a permission string, it becomes `rw-r--r--`.

To set a new `umask`, type **umask** followed by the numerical value of the mask. Here is how you go about it:

1. **Figure out what permission settings you want for new files.**

   For example, if you want new files that can be read and written only by the owner and no one else, the permission setting looks like this:

   ```
   rw-------
   ```

2. **Convert the permissions into a numerical value by using the conversion method that assigns 4 to the first field, 2 to the second, and 1 to the third.**

   Thus, for files that are readable and writable only by their owner, the permission setting is 600.

3. **Subtract the desired permission setting from 666 to get the value of the mask.**

   For a permission setting of 600, the mask becomes 666 – 600 = 066.

4. **Use the** `umask` **command to set the file-creation mask by typing**

   ```
   umask 066
   ```

A default `umask` of 022 is good for system security because it translates to files that have read and write permission for the owner and read permissions for everyone else. The bottom line is that you don't want a default `umask` that results in files that are writable by the whole world.

Checking for set user ID permission

Another permission setting can be a security hazard. This permission setting, called the *set user ID* (or `setuid` and/or `suid` for short), applies to executable files. When the `suid` permission is enabled, the file executes under the user ID of the file's owner. In other words, if an executable program is owned by `root` and the `suid` permission is set, the program runs as if `root` is executing it — no matter who executed the program. The `suid` permission means that the program can do a lot more (for example, read all files, create new files, and delete files) than what a normal user program can do. Another risk is that if a `suid` program file has a security hole, crackers can do a lot more damage through such programs than through other vulnerabilities.

You can find all `suid` programs with a simple `find` command:

```
find / -type f -perm +4000
```

You see a list of files such as the following:

```
/bin/su
/bin/ping
/bin/eject
/bin/mount
/bin/ping6
/bin/umount
/opt/kde4/bin/fileshareset
/opt/kde4/bin/artswrapper
/opt/kde4/bin/kcheckpass
. . . lines deleted . . .
```

Many of the programs have the `suid` permission because they need it, but check the complete list and make sure that there are no strange `suid` programs (for example, `suid` programs in a user's home directory).

For example, if you type **ls -l /bin/su**, you see the following permission settings:

```
-rwsr-xr-x 1 root root 25756 Aug 19 17:06 /bin/su
```

The `s` in the owner's permission setting (`-rws`) tells you that the `suid` permission is set for the `/bin/su` file, which is the executable file for the `su` command that you can use to become `root` or another user.

Encrypting and Signing Files with GnuPG

Linux comes with the *GNU Privacy Guard* (GnuPG, or simply GPG) encryption and authentication utility. With GPG, you can create your public and private key pair, encrypt files using your key, and also digitally sign a message to authenticate that it's really from you. If you send a digitally signed message to someone who has your public key, the recipient can verify that it was you who signed the message.

Understanding public key encryption

The basic idea behind *public key encryption* is to use a pair of keys — one private and the other public — that are related but can't be used to guess one from the other. Anything encrypted with the private key can be decrypted only with the corresponding public key, and vice versa. The public key is for distribution to other people while you keep the private key in a safe place.

You can use public key encryption to communicate securely with others; Figure 2-1 illustrates the basic idea. Suppose Alice wants to send secure messages to Bob. Each of them generates public key and private key pairs, after which they exchange their public keys. Then, when Alice wants to send a message to Bob, she simply encrypts the message using Bob's public key and sends the encrypted message to him. Now the message is secure from eavesdropping because only Bob's private key can decrypt the message — and only Bob has that key. When Bob receives the message, he uses his private key to decrypt the message and read it.

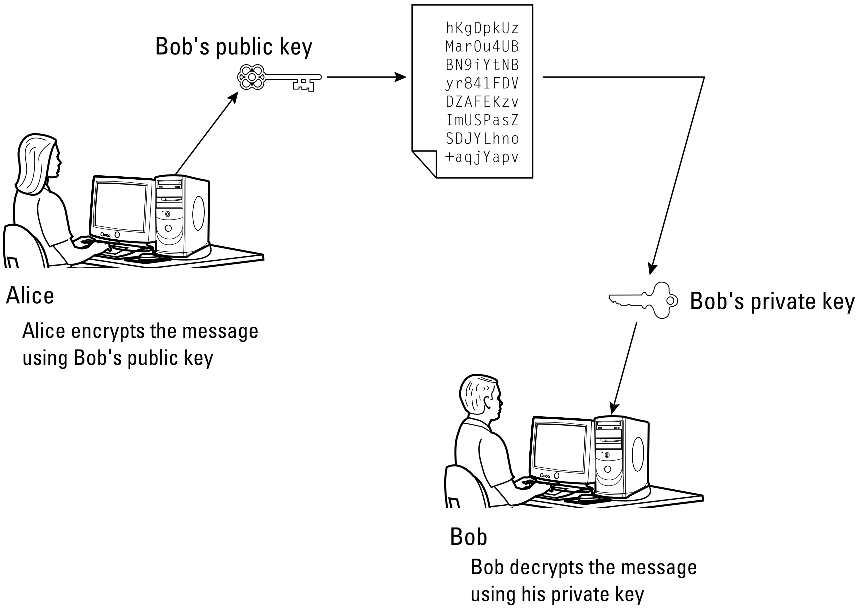


Figure 2-1: Bob and Alice can communicate securely with public key encryption.

At this point, you need to stop and think and say, “Wait a minute! How does Bob know the message really came from Alice? What if someone else uses Bob’s public key and sends a message as if it came from Alice?” This situation is where digital signatures come in.

Understanding digital signatures

The purpose of *digital,* or electronic, *signatures* is the same as pen-and-ink signatures, but how you sign digitally is different. Unlike a pen-and-ink signature, your digital signature depends on the message you’re signing. The first step in creating a digital signature is to apply a mathematical function to the message and reduce it to a fixed-size message digest (also called a *hash* or a *fingerprint*). No matter how big your message, the message digest is usually 128 or 160 bits, depending on the hashing function.

The next step is to apply public key encryption. Simply encrypt the message digest with your private key, and you get the digital signature for the message. Typically, the digital signature is added to the end of the message, and voilà — you get an electronically signed message.

What good does the digital signature do? Well, anyone who wants to verify that the message is indeed signed by you takes your public key and decrypts the digital signature. What that person gets is the message digest (the encrypted hash) of the message. Then he or she applies the same hash function to the message and compares the computed hash with the decrypted value. If the two match, no one has tampered with the message. Because your public key was used to verify the signature, the message must have been signed with the private key known only to you. So the message must be from you!

In the theoretical scenario of Alice sending private messages to Bob, Alice can digitally sign her message to make sure that Bob can tell that the message is really from her. Figure 2-2 illustrates the use of digital signatures along with normal public key encryption.

Here’s how Alice sends her private message to Bob with the assurance that Bob can really tell it’s from her:

1. Alice uses software to compute the message digest of the message and then encrypts the digest by using her private key. This is her digital signature for the message.
2. Alice encrypts the message (again, using some convenient software *and* Bob’s public key).
3. She sends both the encrypted message and the digital signature to Bob.
4. Bob decrypts the message, using his private key.
5. Bob decrypts the digital signature, using Alice’s public key. This gives him the message digest.

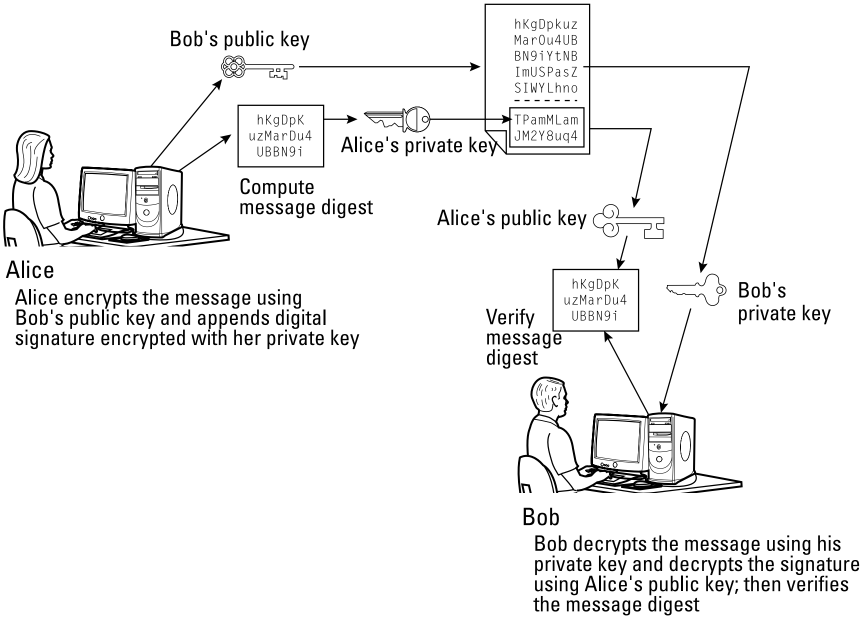

Bob's public key
hKgDpkuz
MarOu4UB
BN9iYtNB
ImUSPasZ
SIWYLhno
TPamMLam
JM2Y8uq4
hKgDpK
uzMarDu4
UBBN9i
Alice's private key
Compute message digest
Alice's public key
Alice
Alice encrypts the message using Bob's public key and appends digital signature encrypted with her private key
hKgDpK
uzMarDu4
UBBN9i
Bob's private key
Verify message digest
Bob
Bob decrypts the message using his private key and decrypts the signature using Alice's public key; then verifies the message digest

Figure 2-2: Alice can digitally sign her message so that Bob can tell it's really from her.

6. Bob computes the message digest of the message and compares it with what he got by decrypting the digital signature.
7. If the two message digests match, Bob can be sure that the message really came from Alice.

Using GPG

GPG includes the tools you need to use public key encryption and digital signatures. You can figure out how to use GPG gradually as you begin using encryption. The following shows you some of the typical tasks you can perform with GPG.

Generating the key pair

The steps for generating the key pairs are as follows:

1. **Type** gpg --gen-key**.**

   If you're using gpg for the first time, it creates a .gnupg directory in your home directory and a file named gpg.conf in that directory. Then GPG asks what kind of keys you want:

```
Please select what kind of key you want:
(1) DSA and ElGamal (default)
(2) DSA (sign only)
(4) RSA (sign only)
Your selection?
```

2. **Press Enter for the default choice because it's good enough.**

   GPG then prompts you for the key size (the number of bits).

3. **Press Enter again to accept the default value of 2,048 bits.**

   GPG asks you when the keys expire. The default is to never expire.

4. **If the default is what you want (and why not?), press Enter.**

5. **When GPG asks if you really want the keys to never expire, press the Y key to confirm.**

   GPG prompts you for your name, your e-mail address, and finally a comment to make it easier to associate the key pair with your name.

6. **Type each piece of requested information and press Enter.**

7. **When GPG gives you a chance to change the information or confirm it as is, confirm by typing** `o` **and pressing Enter.**

   GPG next prompts you for a passphrase that protects your private key.

8. **Type a long phrase that includes lowercase and uppercase letters, numbers, and punctuation marks — the longer the better — and then press Enter.**

   REMEMBER

   Be careful to choose a passphrase that you can easily remember.

   GPG generates the keys. It may ask you to perform some work on the PC so that the random-number generator can generate enough random numbers for the key-generation process.

Exchanging keys

To communicate with others, you have to give them your public key. You also have to get public keys from those who may send you a message (or someone who might sign a file and you want to verify the signature). GPG keeps the public keys in your key ring. (The *key ring* is simply the public keys stored in a file, but it sounds nice to call it a key ring because everyone has a key ring in the real world, and these are keys of a sort, right?) To list the keys in your key ring, type

```
gpg --list-keys
```

To send your public key to someone or to place it on a website, you have to export the key to a file. The best way is to put the key in what GPG documentation calls an *ASCII-armored* format, with a command like this:

```
gpg --armor --export kdualney@insightbb.com > kdulaneykey.asc
```

This command saves the public key in an ASCII-armored format (it basically looks like garbled text) in the file named `kdulaneykey.asc`. You would replace the e-mail address with your e-mail address (the one you used when you created the key) and replace the output filename to something different.

After you export the public key to a file, you can mail that file to others or place it on a website for use by others.

When you import a key from someone, you typically get it in an ASCII-armored format as well. For example, if you have a `us-cert@us-cert.gov` GPG public key in a file named `uscertkey.asc` (obtained from the link at `www.us-cert.gov/pgp/email.html`), you then import it into the key ring with the following command:

```
gpg --import uscertkey.asc
```

Use the `gpg --list-keys` command to verify that the key is in your key ring. For example, here's what you might see when typing **gpg –list-keys** on the system:

```
/home/kdulaney/.gnupg/pubring.gpg
-----------------------------
pub 1024D/7B38A728 2013-08-28
uid Kristin Dulaney <kdulaney@insightbb.com>
sub 2048g/3BD6D418 2013-08-28
pub 2048R/F0E187D0 2014-09-08 [expires: 2014-10-01]
uid US-CERT Operations Key <us-cert@us-cert.gov>
```

The next step is to check the fingerprint of the new key. Type the following command to get the fingerprint of the US-CERT key:

```
gpg --fingerprint us-cert@us-cert.gov
```

GPG prints the fingerprint:

```
pub 2048R/F0E187D0 2013-09-08 [expires: 2014-10-01]
Key fingerprint = 049F E3BA 240B 4CF1 3A76 06DC 1868 49EC F0E1 87D0
uid US-CERT Operations Key <us-cert@us-cert.gov>
```

At this point, you need to verify the key fingerprint with someone at the US-CERT organization.

If you think the key fingerprint is good, you can sign the key and validate it. Here's the command you use to sign the key:

```
gpg --sign-key us-cert@us-cert.gov
```

GPG asks for confirmation and then prompts you for your passphrase. After that, GPG signs the key.

Because key verification and signing is a potential weak link in GPG, be careful about what keys you sign. By signing a key, you basically say that you trust the key to be from that person or organization.

Signing a file

You may find signing files useful if you send a file to someone and want to assure the recipient that no one tampered with the file and that you did, in fact, send the file. GPG makes signing a file easy. You can compress and sign a file named `message` with the following command:

```
gpg -o message.sig -s message
```

To verify the signature, type

```
gpg --verify message.sig
```

To get back the original document, simply type

```
gpg -o message --decrypt message.sig
```

Sometimes you don't care about keeping a message secret, but you simply want to sign it to indicate that the message is from you. In such a case, you can generate and append a clear-text signature with the following command:

```
gpg -o message.asc --clearsign message
```

This command basically appends a clear-text signature to the text message. Here's a typical clear-text signature block:

```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.2 (GNU/Linux)
iD8DBQFDEhAtaHWlHHs4pygRAhiqAJ9Qj0pPMgKVBuokDyUZaEYVsp6RIQCfaoBm
9zCwrSAG9mo2DXJvbKS3ri8=
=2uc/
-----END PGP SIGNATURE-----
```

When a message has a clear-text signature appended, you can use GPG to verify the signature with the following command:

```
gpg --verify message.asc
```

If you had indeed signed the message, the last line of the output says that it's a good signature.

Encrypting and decrypting documents

To encrypt a message meant for a recipient, you can use the `--encrypt` (or `-e`) GPG command. Here's how you might encrypt a message for US-CERT using its GPG key:

```
gpg -o message.gpg -e -r us-cert@us-cert.gov message
```

The message is encrypted using the US-CERT public key (without a signature, but you can add the signature with the -s command).

When US-CERT receives the message.gpg file, the recipient has to decrypt it using US-CERT's private key. Here's the command someone at US-CERT can use:

```
gpg -o message --decrypt message.gpg
```

GPG then prompts for the passphrase to unlock the US-CERT private key, decrypts the message, and saves the output in the file named message.

If you simply want to encrypt a file and no one else has to decrypt the file, you can use GPG to perform *symmetric encryption.* In this case, you provide a passphrase to encrypt the file with the following GPG command:

```
gpg -o secret.gpg -c somefile
```

GPG prompts you for the passphrase and asks you to repeat the passphrase (to make sure that you didn't mistype anything). Then GPG encrypts the file using a key generated from the passphrase.

To decrypt a file encrypted with a symmetric key, type

```
gpg -o myfile --decrypt secret.gpg
```

GPG prompts you for the passphrase. If you enter the correct passphrase, GPG decrypts the file and saves the output (in this example) in the file named myfile.

Monitoring System Security

Even if you secure your system, you have to monitor the log files periodically for signs of intrusion. You may want to use Tripwire (a good tool for detecting any changes made to the system files) so that you can monitor the integrity of critical system files and directories. Your Linux system probably doesn't come with the Tripwire package. To use Tripwire, you have to buy it from www.tripwire.com. After you purchase and install Tripwire, you can configure it to monitor any changes to specified system files and directories on your system.

Periodically examine the log files in the /var/log directory and its subdirectories. Many Linux applications, including some servers, write log information by using the logging capabilities of syslogd or rsyslogd. On Linux systems, the log files written by syslogd and rsyslogd reside in the /var/log directory. Make sure that only the root user can read and write these files.

REMEMBER

The syslogd configuration file is /etc/syslog.conf, and the rsyslogd configuration file (existing on many newer systems) is /etc/rsyslog.conf. The default configuration of syslogd generates the necessary log files; however, if you want to examine and understand the configuration file, type **man syslog.conf** for more information.

Securing Internet Services

For an Internet-connected Linux system (or even one on a TCP/IP LAN that's not connected to the Internet), a significant threat is that someone could use one of many Internet services to gain access to your system. Each service — such as mail, web, or FTP — requires running a server program that responds to client requests arriving over the TCP/IP network. Some of these server programs have weaknesses that can allow an outsider to log in to your system — maybe with root privileges. Luckily, Linux comes with some facilities that you can use to make the Internet services more secure.

WARNING!

Potential intruders can employ a *port-scanning tool* — a program that attempts to establish a TCP/IP connection at a port and then looks for a response — to check which Internet servers are running on your system. Then, to gain access to your system, the intruders can potentially exploit any known weaknesses of one or more services.

Turning off standalone services

To provide Internet services, such as web, e-mail, and FTP, your Linux system has to run server programs that listen to incoming TCP/IP network requests. Some of these servers start when your system boots, and they run all the time. Such servers are *standalone servers.* The web server and mail server are examples of standalone servers.

Another server, xinetd, starts other servers that are configured to work under xinetd. Some Linux systems use the inetd server instead of xinetd to start other servers.

Some servers can be configured to run standalone or under a super server such as xinetd. For example, the vsftpd FTP server can be configured to run standalone or to run under the control of xinetd.

DISTRIBUTION SPECIFIC

In Debian, Ubuntu, and Xandros, use the update-rc.d command to turn off standalone servers and use the invoke-rc.d command to start or stop servers interactively. To get a clue about the available services, type **ls /etc/init.d** and look at all the script files designed to turn services on or off. You have to use these filenames when you want to turn a service on or off. For example, to turn off Samba service, type **update-rc.d -f samba remove**. If the service was already running, type **invoke-rc.d samba stop** to stop the service. You can use the invoke-rc.d command to stop any service in a similar manner.

In Fedora and SUSE, you can turn standalone servers on or off by using the `chkconfig` command. You can get the names of the service scripts by typing **ls /etc/init.d**. Then you can turn off a service (for example, Samba) by typing **chkconfig –del smb**. If the service was already running, type **/etc/init.d/smb stop** to stop the service. You can run scripts from the `/etc/init.d` directory with the `stop` argument to stop any service in a similar manner.

Configuring the Internet super server

In addition to standalone servers such as a web server or mail server, other servers — `inetd` or `xinetd` — have to be configured separately. These servers are *Internet super servers* because they can start other servers on demand.

Type **ps ax | grep inetd** to see which Internet super server — `inetd` or `xinetd` — your system runs.

Debian, Ubuntu, and Xandros use `inetd`, and Fedora and SUSE use `xinetd`.

The `inetd` server is configured through the `/etc/inetd.conf` file. You can disable a service by locating the appropriate line in that file and commenting it out by placing a pound sign (#) at the beginning of the line. After saving the configuration file, type **/etc/init.d/inetd restart** to restart the `inetd` server.

Configuring the `xinetd` server is a bit more complicated. The `xinetd` server reads a configuration file named `/etc/xinetd.conf` at startup. This file, in turn, refers to configuration files stored in the `/etc/xinetd.d` directory. The configuration files in `/etc/xinetd.d` tell `xinetd` which ports to listen to and which server to start for each port. Type **ls /etc/xinetd.d** to see a list of the files in the `/etc/xinetd.d` directory on your system. Each file represents a service that `xinetd` can start. To turn off any of these services, edit the file in a text editor and add a `disable = yes` line in the file. After you make any changes to the `xinetd` configuration files, you must restart the `xinetd` server; otherwise, the changes don't take effect. To restart the `xinetd` server, type **/etc/init.d/xinetd restart**. This command stops the `xinetd` server and then starts it again. When it restarts, it reads the configuration files, and the changes take effect.

Configuring TCP wrapper security

A security feature of both `inetd` and `xinetd` is their use of the TCP wrapper to start various services. The *TCP wrapper* is a block of code that provides an access-control facility for Internet services, acting like a protective package for your message. The TCP wrapper can start other services, such as FTP and TELNET; but before starting a service, it consults the `/etc/hosts.allow` file to see whether the host requesting the service is allowed to use that service. If nothing appears in /etc/hosts.allow about that host, the TCP wrapper checks the /etc/hosts.deny file to see if it denies the service. If both files are empty, the TCP wrapper provides access to the requested service.

Here are the steps to follow to tighten access to the services that inetd or xinetd are configured to start:

1. **Use a text editor to edit the** /etc/hosts.deny **file, adding the following line into that file:**

   ```
   ALL:ALL
   ```

   This setting denies all hosts access to any Internet services on your system.

2. **Edit the** /etc/hosts.allow **file and add to it the names of hosts that can access services on your system.**

   For example, to enable only hosts from the 192.168.1.0 network and the localhost (IP address 127.0.0.1) to access the services on your system, place the following line in the /etc/hosts.allow file:

   ```
   ALL: 192.168.1.0/255.255.255.0 127.0.0.1
   ```

3. **If you want to permit a specific remote host access to a specific Internet service, use the following syntax for a line in** /etc/hosts.allow**:**

   ```
   server_program_name: hosts
   ```

   Here *server_program_name* is the name of the server program, and *hosts* is a comma-separated list of the hosts that can access the service. You may also write *hosts* as a network address or an entire domain name, such as .mycompany.com.

Using Secure Shell (SSH) for Remote Logins

Linux comes with the *Open Secure Shell* (OpenSSH) software, a suite of programs that provides a secure replacement for the Berkeley r commands: rlogin (remote login), rsh (remote shell), and rcp (remote copy). OpenSSH uses public key cryptography to authenticate users and to encrypt the communication between two hosts, so users can securely log in from remote systems and copy files securely.

This section briefly describes how to use the OpenSSH software in Linux. To find out more about OpenSSH and read the latest news about it, visit www.openssh.com or www.openssh.org.

The OpenSSH software is installed during Linux installation. Table 2-2 lists the main components of the OpenSSH software.

Table 2-2 Components of the OpenSSH Software

Component	*Description*
/usr/sbin/sshd	This Secure Shell daemon must run on a host if you want users on remote systems to use the ssh client to log in securely. When a connection from the ssh client arrives, sshd performs authentication using public key cryptography and establishes an encrypted communication link with the ssh client.
/usr/bin/ssh	Users can run this Secure Shell client to log in to a host that is running sshd. Users can also use ssh to execute a command on another host.
/usr/bin/slogin	This component is a symbolic link to /usr/bin/ssh.
/usr/bin/scp	This secure-copy program works like rcp but securely. The scp program uses ssh for data transfer and provides the same authentication and security as ssh.
/usr/bin/ssh-keygen	You use this program to generate the public and private key pairs you need for the public key cryptography used in OpenSSH. The ssh-keygen program can generate key pairs for both RSA and DSA (Digital Signature Algorithm) authentication. (RSA comes from the initial of the last name of Ron Rivest, Adi Shamir, and Leonard Adleman — the developers of the RSA algorithm.)
/etc/ssh/sshd_config	This configuration file for the sshd server specifies many parameters for sshd, including the port to listen to, the protocol to use, and the location of other files. (There are two versions of SSH protocols: SSH1 and SSH2, both supported by OpenSSH.)
/etc/ssh/ssh_config	This configuration file is for the ssh client. Each user can also have an ssh configuration file named config in the .ssh subdirectory of the user's home directory.

OpenSSH uses public key encryption, in which the sender and receiver both have a pair of keys — a public key and a private key. The public keys are freely distributed, and each party knows the other's public key. The sender encrypts data by using the recipient's public key. Only the recipient's private key can then decrypt the data.

To use OpenSSH, you first need to start the `sshd` server and then generate the host keys. Here's how:

- If you want to support SSH-based remote logins on a host, start the `sshd` server on your system. Type **ps ax | grep sshd** to see if the server is already running. If not, log in as `root` and turn on the SSH service.
- Generate the host keys with the following command:

  ```
  ssh-keygen -d -f /etc/ssh/ssh_host_key -N ''
  ```

  The `-d` flag causes the `ssh-keygen` program to generate DSA keys, which the SSH2 protocol uses. If you see a message saying that the file `/etc/ssh/ssh_host_key` already exists, it means that the key pairs were generated during Linux installation. You can use the existing file without having to regenerate the keys.

A user who wants to log in using SSH can simply use the `ssh` command. For example:

```
ssh 192.168.0.4 -l kdulaney
```

where 192.168.0.4 is the IP address of the other Linux system. SSH then displays a message:

```
The authenticity of host '192.168.0.4 (192.168.0.4)' can't be established.
RSA key fingerprint is 7b:79:f2:dd:8c:54:00:a6:94:ec:fa:8e:7f:c9:ad:66.
Are you sure you want to continue connecting (yes/no)?
```

Type **yes** and press Enter. SSH then adds the host to its list of known hosts and prompts you for a password on the other Linux system:

```
kdulaney@192.168.0.4's password:
```

After entering the password, you have a secure login session with that system. You can also log in to this account with the following equivalent command:

```
ssh kdulaney@192.168.0.4
```

If you simply want to copy a file securely from another system on the LAN (identified by its IP address, 192.168.0.4), you can use `scp` like this:

```
scp 192.168.0.4:/etc/X11/xorg.conf
```

This command prompts for a password and securely copies the `/etc/X11/xorg.conf` file from the 192.168.0.4 host to the system from which the `scp` command was typed, as follows:

```
kdulaney@192.168.0.4's password: (type the password.)
xorg.conf 100% 2814 2.8KB/s 00:00
```

Setting Up Simple Firewalls

A *firewall* is a network device or host with two or more network interfaces — one connected to the protected internal network and the other connected to unprotected networks, such as the Internet. The firewall controls access to and from the protected internal network.

If you connect an internal network directly to the Internet, you have to make sure that every system on the internal network is properly secured — which can be nearly impossible because a single careless user can render the entire internal network vulnerable. A firewall is a single point of connection to the Internet: You can direct all your efforts toward making that firewall system a daunting barrier to unauthorized external users. Essentially, a firewall is like a protective fence that keeps unwanted external data and software out and sensitive internal data and software in. (See Figure 2-3.)

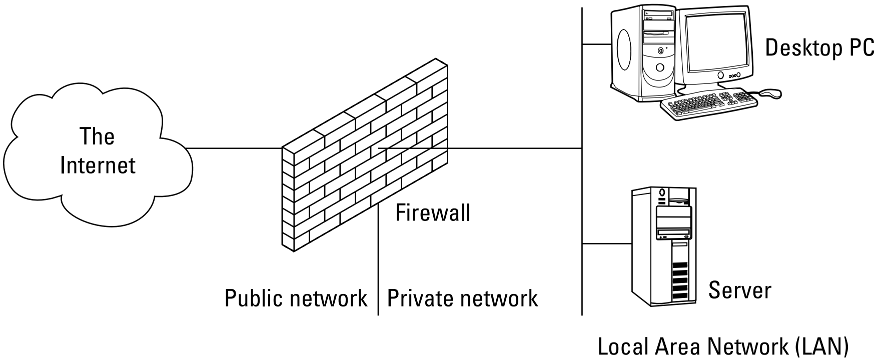


Figure 2-3: A firewall protects hosts on a private network from the Internet.

The firewall runs software that examines the network packets arriving at its network interfaces, and then takes appropriate action based on a set of rules. The idea is to define these rules so they allow only authorized network traffic to flow between the two interfaces. Configuring the firewall involves setting up the rules properly. A configuration strategy is to reject all network traffic and then enable only a limited set of network packets to go through the firewall. The authorized network traffic would include the connections necessary to enable internal users to do things such as visit websites and receive electronic mail.

To be useful, a firewall has the following general characteristics:

- ✦ It must control the flow of packets between the Internet and the internal network.
- ✦ It must *not* provide dynamic routing because dynamic routing tables are subject to route *spoofing* — the use of fake routes by intruders. Instead, the firewall uses static routing tables (which you can set up with the `route` command on Linux systems).

- It must not allow any external user to log in as root. That way, even if the firewall system is compromised, the intruder is blocked from using root privileges from a remote login.
- It must be kept in a physically secure location.
- It must distinguish between packets that come from the Internet and packets that come from the internal protected network. This feature allows the firewall to reject packets that come from the Internet but have the IP address of a trusted system on the internal network.
- It acts as the SMTP mail gateway for the internal network. Set up the sendmail software so that all outgoing mail appears to come from the firewall system.
- Its user accounts are limited to a few user accounts for those internal users who need access to external systems. External users who need access to the internal network should use SSH for remote login (see "Using Secure Shell (SSH) for Remote Logins," earlier in this chapter).
- It keeps a log of all system activities, such as successful and unsuccessful login attempts.
- It provides DNS name-lookup service to the outside world to resolve any hostnames that are known to the outside world.
- It provides good performance so that it doesn't hinder the internal users' access to specific Internet services (such as HTTP and FTP).

A firewall can take many different forms. Here are three common forms of a firewall:

- **Packet filter firewall:** This simple firewall uses a router capable of filtering (blocking or allowing) packets according to a number of their characteristics, including the source and destination IP addresses, the network protocol (TCP or UDP), and the source and destination port numbers. Packet filter firewalls are usually placed at the outermost boundary with an untrusted network, and they form the first line of defense. An example of a packet filter firewall is a network router that employs filter rules to screen network traffic.

  Packet filter firewalls are fast and flexible, but they can't prevent attacks that exploit application-specific vulnerabilities or functions. They can log only a minimal amount of information, such as source IP address, destination IP address, and traffic type. Also, they're vulnerable to attacks and exploits that take advantage of flaws within the TCP/IP protocol, such as IP address spoofing, which involves altering the address information in network packets to make them appear to come from a trusted IP address.
- **Stateful inspection firewall:** This type of firewall keeps track of the network connections that network applications are using. When an application on an internal system uses a network connection to create a session with a remote system, a port is also opened on the internal system. This port receives network traffic from the remote system. For successful

connections, packet filter firewalls must permit incoming packets from the remote system. Opening up many ports to incoming traffic creates a risk of intrusion by unauthorized users who abuse the expected conventions of network protocols such as TCP. Stateful inspection firewalls solve this problem by creating a table of outbound network connections, along with each session's corresponding internal port. This *state table* is then used to validate any inbound packets. This stateful inspection is more secure than a packet filter because it tracks internal ports individually rather than opening all internal ports for external access.

- **Application-proxy gateway firewall:** This firewall acts as an intermediary between internal applications that attempt to communicate with external servers such as a web server. For example, a web proxy receives requests for external web pages from web browser clients running inside the firewall and relays them to the exterior web server as though the firewall was the requesting web client. The external web server responds to the firewall, and the firewall forwards the response to the inside client as though the firewall was the web server. No direct network connection is ever made from the inside client host to the external web server.

  Application-proxy gateway firewalls have some advantages over packet filter firewalls and stateful inspection firewalls. First, application-proxy gateway firewalls examine the entire network packet rather than only the network addresses and ports. This enables these firewalls to provide more extensive logging capabilities than packet filters or stateful inspection firewalls. Another advantage is that application-proxy gateway firewalls can authenticate users directly, whereas packet filter firewalls and stateful inspection firewalls normally authenticate users on the basis of the IP address of the system (that is, source, destination, and protocol type). Given that network addresses can be easily spoofed, the authentication capabilities of application-proxy gateway firewalls are superior to those found in packet filter and stateful inspection firewalls.

  The advanced functionality of application-proxy gateway firewalls, however, results in some disadvantages when compared with packet filter or stateful inspection firewalls. First, because of the *full packet awareness* found in application-proxy gateways, the firewall is forced to spend significant time reading and interpreting each packet. Therefore application-proxy gateway firewalls are generally not well suited to high-bandwidth or real-time applications. To reduce the load on the firewall, a dedicated proxy server can be used to secure less time-sensitive services, such as e-mail and most web traffic. Another disadvantage is that application-proxy gateway firewalls are often limited in terms of support for new network applications and protocols. An individual application-specific proxy agent is required for each type of network traffic that needs to go through the firewall. Most vendors of application-proxy gateways provide generic proxy agents to support undefined network protocols or applications. However, those generic agents tend to negate many of the strengths of the application-proxy gateway architecture, and they simply allow traffic to *tunnel* through the firewall.

Most firewalls implement a combination of these firewall functionalities. For example, many vendors of packet filter firewalls or stateful inspection firewalls have also implemented basic application-proxy functionality to offset some of the weaknesses associated with their firewalls. In most cases, these vendors implement application proxies to provide better logging of network traffic and stronger user authentication. Nearly all major firewall vendors have introduced multiple firewall functions into their products in some manner.

In a large organization, you may also have to isolate smaller internal networks from the corporate network. You can set up such internal firewalls the same way that you set up Internet firewalls.

Using NAT

Network Address Translation (NAT) is an effective tool that enables you to *hide* the network addresses of an internal network behind a firewall. In essence, NAT allows an organization to use private network addresses behind a firewall while maintaining the ability to connect to external systems through the firewall.

Here are the three methods for implementing NAT:

- **Static:** In static NAT, each internal system on the private network has a corresponding external, routable IP address associated with it. This particular technique is seldom used because unique IP addresses are in short supply.
- **Hiding:** With hiding NAT, all systems behind a firewall share the same external, routable IP address, while the internal systems use private IP addresses. Thus, with a hiding NAT, a number of systems behind a firewall still appear to be a single system.
- **Port address translation:** With port address translation, you can place hosts behind a firewall system and still make them selectively accessible to external users.

In terms of strengths and weaknesses, each type of NAT — static, hiding, or port address translation — is applicable in certain situations; the variable is the amount of design flexibility offered by each type. Static NAT offers the most flexibility, but it's not always practical because of the shortage of IP addresses. Hiding NAT technology is seldom used because port address translation offers additional features. Port address translation is often the most convenient and secure solution.

Enabling packet filtering on your Linux system

The Linux kernel has built-in packet filtering software in the form of something called `netfilter`. You use the `iptables` command to set up the rules for what happens to the packets based on the IP addresses in their header and the network connection type.

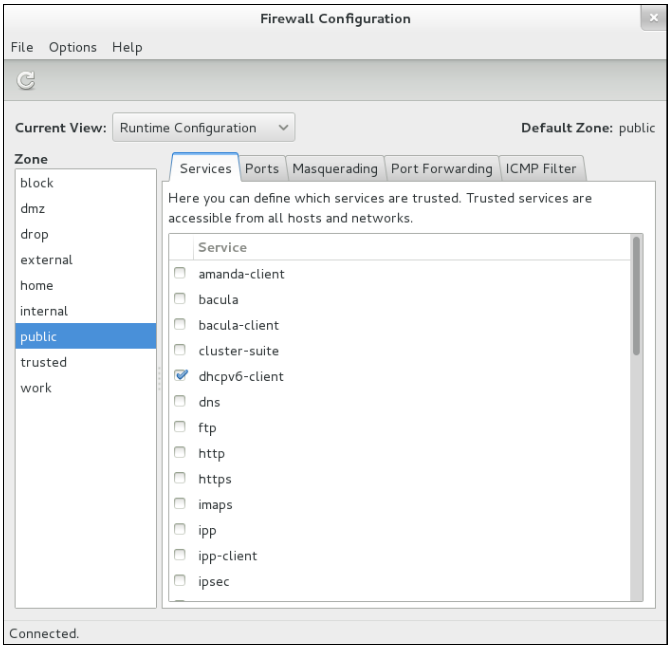
To find out more about netfilter and iptables, visit the documentation section of the netfilter website at www.netfilter.org/documentation.

The built-in packet filtering capability is handy when you don't have a dedicated firewall between your Linux system and the Internet. This is the case, for example, when you connect your Linux system to the Internet through a DSL or cable modem. Essentially, you can have a packet filtering firewall inside your Linux system, sitting between the kernel and the applications.

Using the security level configuration tool

Some Linux distributions, such as Fedora and SUSE, include GUI tools to turn on a packet filtering firewall.

In Fedora, you can turn on different levels of packet filtering through the graphical Firewall Configuration tool. To run the tool, log in as root and choose Activities then type in Firewall. The Firewall Configuration window appears (see Figure 2-4) along with an authentication window.



Current View: Runtime Configuration

Default Zone: public

Zone

- block
- dmz
- drop
- external
- home
- internal
- public
- trusted
- work

Here you can define which services are trusted. Trusted services are accessible from all hosts and networks.

Service

- □ amanda-client
- □ bacula
- □ bacula-client
- □ cluster-suite
- ☑ dhcpv6-client
- □ dns
- □ ftp
- □ http
- □ https
- □ imaps
- □ ipp
- □ ipp-client
- □ ipsec

Connected.

Figure 2-4: In Fedora, you can configure the firewall with this tool.

From the Firewall Configuration dialog box, you can select two predefined levels of simple firewalling (more precisely, packet filtering):

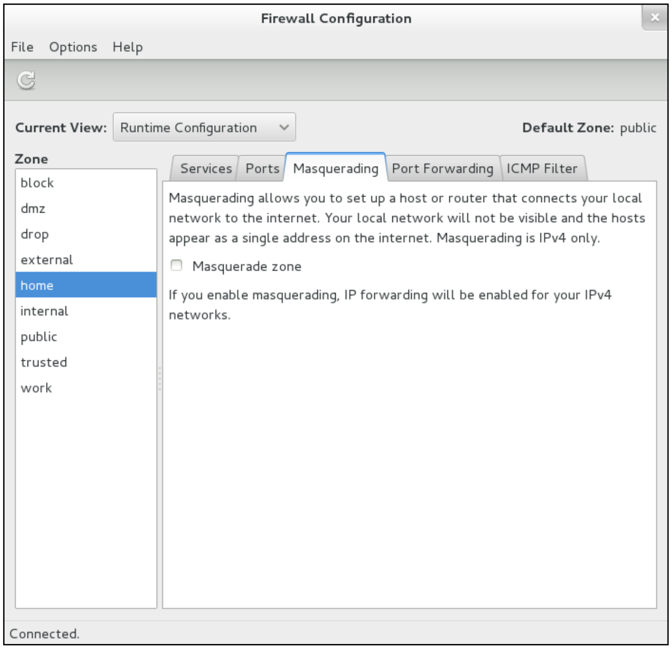
- **Disabled:** This option doesn't perform any filtering and allows all connections. (You can still turn off Internet services by not running the servers or disabling them in the xinetd configuration files.) This security level is fine if your Linux system is inside a protected local area network or if you have a separate firewall device.

- **Enabled:** This option turns on packet filtering. You can then select the services that you want to allow and the network devices that you trust.

You can allow incoming packets meant for specific Internet services such as SSH, TELNET, and FTP. If you select a network interface such as `eth0` (the first Ethernet card) as trusted, all network traffic over that interface is allowed without any filtering.

In SUSE, to set up a firewall, choose Main Menu➪System➪YaST. In the YaST Control Center window that appears, click Security and Users on the left side of the window and then click Firewall on the right side. YaST opens a window that you can use to configure the firewall.

You can designate network interfaces (by device name, such as `eth0`, `ppp0`, and so on) to one of three zones: internal, external, or demilitarized zone. Then, for that zone, you can specify what services (such as HTTP, FTP, and SSH) are allowed. If you have two or more network interfaces and you use the Linux system as a gateway (a router), you can enable forwarding packets between network interfaces (a feature called *masquerading*). Figure 2-5 shows an example of this feature in the Firewall Configuration tool included with Fedora.



Current View: Runtime Configuration

Default Zone: public

Masquerading allows you to set up a host or router that connects your local network to the internet. Your local network will not be visible and the hosts appear as a single address on the internet. Masquerading is IPv4 only.

□ Masquerade zone

If you enable masquerading, IP forwarding will be enabled for your IPv4 networks.

Connected.

Figure 2-5: In Fedora, you can turn on masquerading with a single click.

You can also turn on different levels of logging (for example, logging all dropped packets that attempt connection at specific ports). If you make changes to firewall settings, click the Startup category and click Save Settings and Restart Firewall Now.

Using the iptables command

The GUI firewall configuration tools use the `iptables` command to implement the firewall. If your Linux system doesn't have a GUI tool, you can use `iptables` directly to configure firewalling on your Linux system.

Using the `iptables` command is somewhat complex. The `iptables` command uses the concept of a *chain,* which is a sequence of rules. Each rule says what to do with a packet if the header contains certain information (such as the source or destination IP address). If a rule doesn't apply, `iptables` consults the next rule in the chain. By default, there are three chains:

- **`INPUT` chain:** The first set of rules against which packets are tested. The packets continue to the next chain only if the `INPUT` chain doesn't specify `DROP` or `REJECT`.
- **`FORWARD` chain:** Contains the rules that apply to packets attempting to pass through this system to another system (when you use your Linux system as a router between your LAN and the Internet, for example).
- **`OUTPUT` chain:** Includes the rules applied to packets before they are sent out (either to another network or to an application).

When an incoming packet arrives, the kernel uses `iptables` to make a routing decision based on the destination IP address of the packet. If the packet is for this server, the kernel passes the packet to the `INPUT` chain. If the packet satisfies all the rules in the `INPUT` chain, the packet is processed by local processes such as an Internet server that is listening for packets of this type.

If the kernel has IP forwarding enabled and the packet has a destination IP address of a different network, the kernel passes the packet to the `FORWARD` chain. If the packet satisfies the rules in the `FORWARD` chain, it's sent out to the other network. If the kernel doesn't have IP forwarding enabled and the packet's destination address isn't for this server, the packet is dropped.

If the local processing programs that receive the input packets want to send network packets out, those packets pass through the `OUTPUT` chain. If the `OUTPUT` chain accepts those packets, they're sent out to the specified destination network.

You can view the current chains, add rules to the existing chains, or create new chains of rules by using the `iptables` command. When you view the current chains, you can also save them to a file. For example, if you had configured nothing else and your system has no firewall configured, typing **iptables -L** should show the following:

```
Chain INPUT (policy ACCEPT)
target prot opt source destination
Chain FORWARD (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

In this case, all three chains — `INPUT`, `FORWARD`, and `OUTPUT` — show the same `ACCEPT` policy, which means everything is wide open.

If you're setting up a packet filter, the first thing you do is specify the packets that you want to accept. For example, to accept packets from the 192.168.0.0 network address, add the following rule to the `INPUT` chain:

```
iptables -A INPUT -s 192.168.0.0/24 -j ACCEPT
```

Now add a rule to drop everything except local loopback (the `lo` network interface) traffic and stop all forwarding with the following commands:

```
iptables -A INPUT -i ! lo -j REJECT
iptables -A FORWARD -j REJECT
```

The first `iptables` command, for example, appends to the `INPUT` chain (`-A INPUT`) the rule that if the packet does not come from the `lo` interface (`-i ! lo`), `iptables` rejects the packet (`-j REJECT`).

Before rejecting all other packets, you may also add more rules to each `INPUT` chain to allow specific packets in. You can select packets to accept or reject based on many parameters, such as IP addresses, protocol types (TCP, UDP), network interface, and port numbers.

You can do all sorts of specialized packet filtering with `iptables`. For example, suppose you set up a web server and want to accept packets meant for only HTTP (port 80) and Secure Shell (SSH) services. The Secure Shell service (port 22) is for you to securely log in and administer the server. Suppose the server's IP address is 192.168.0.10. Here is how you might set up the rules for this server:

```
iptables -P INPUT DROP
iptables -A INPUT -s 0/0 -d 192.168.0.10 -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -s 0/0 -d 192.168.0.10 -p tcp --dport 22 -j ACCEPT
```

In this case, the first rule sets up the default policy of the `INPUT` chain to `DROP`, which means that if none of the specific rules match, the packet will be dropped. The next two rules say that packets addressed to 192.168.0.10 and meant for ports 80 and 22 are accepted.

WARNING!

Don’t type `iptables` commands from a remote login session. A rule that begins denying packets from all addresses can also stop what you type from reaching the system; if that happens, you may have no way of accessing the system over the network. To avoid unpleasant surprises, always type `iptables` rules at the console — the keyboard and monitor connected directly to your Linux PC that is running the packet filter. If you want to delete all filtering rules in a hurry, type **iptables -F** to flush them. To change the default policy for the INPUT chain to `ACCEPT`, type **iptables -t filter -P INPUT ACCEPT**. This causes `iptables` to accept all incoming packets by default.

REMEMBER

Not every `iptables` command is discussed in this section. You can type **man iptables** to read a summary of the commands. You can also read about `netfilter` and `iptables` at `www.iptables.org`.

After you define the rules by using the `iptables` command, they’re in memory and are gone when you reboot the system. Use the `iptables-save` command to store the rules in a file. For example, you can save the rules in a file named `iptables.rules` by using the following command:

```
iptables-save > iptables.rules
```

Here’s a listing of the `iptables.rules` file generated on a Fedora system:

```
# Generated by iptables-save v1.3.0 on Sun Dec 28 16:10:12 2014
*filter
:FORWARD ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [6:636]
-A FORWARD -j REJECT --reject-with icmp-port-unreachable
-A INPUT -s 192.168.0.0/255.255.255.0 -j ACCEPT
-A INPUT -i ! lo -j REJECT --reject-with icmp-port-unreachable
COMMIT
# Completed on Sun Dec 28 16:10:12 2014
```

These rules correspond to the following `iptables` commands used to configure the filter:

```
iptables -A INPUT -s 192.168.0.0/24 -j ACCEPT
iptables -A INPUT -i ! lo -j REJECT
iptables -A FORWARD -j REJECT
```

If you want to load these saved rules into `iptables`, use the following command:

```
iptables-restore < iptables.rules
```

Security Files to Be Aware Of

Table 2-3 lists eleven files, or directories, that security administrators should be aware of and able to explain the purpose of.

Table 2-3 Key Security Files

File	*Description*
`/etc/nologin`	If this file exists, it denies login to all users except `root`. This can be handy when maintenance needs to be done and users need to stay off the system for a period of time. Removing the file restores login capability for all users. The file can be created as a text file with any editor, or you can often use the `nologin` command to create it.
`/etrc/passwd`	This file holds much of the user account information and is addressed heavily in this chapter.
`/etc/shadow`	When shadowing is turned on – which it almost always is – then password values (hashes) are stored in this file (which is more secure) as opposed to in `/etrc/passwd`.
`/etrc/xinetd.d/*`	This directory can be used to store configuration files used by `xinetd`, the server daemon.
`/etrc/xinetd.conf`	This is the main configuration file used by `xinetd`, the server daemon.
`/etrcxinetd.d/*`	This directory can be used to store configuration files used by `inetd`, the Internet daemon. In almost all distributions, `inetd` has been replaced by `xinetd`.
`/etrc/inetd.conf`	This is the main configuration file used by `inetd`, the Internet daemon.
`/etrc/inittab`	This is the initial startup (initialization) table used to identify what starts and stops as the system is booted and changes run states.
`/etrc/init.d/*`	This directory can hold configuration files that are used during the change of run states/level and referenced by the `inittab` file.
`/etrc/hosts.allow`	If this file exists, then it specifically lists the hosts that are allowed to network with this one. If the file does not exist, then the default is that all hosts are allowed to network with this one.
`/etrc/hosts.deny`	If this file exists, then it specifically lists the hosts that are not allowed to network with this one. If the file does not exist, then the default is that all hosts are allowed to network with this one. The three possibilities are that there is an allow file identifying only hosts that can, a deny file that identifies only hosts that cannot, or neither file – in which case all other hosts are allowed to network with this one.

Chapter 3: Computer Security Audits and Vulnerability Testing Types

In This Chapter

Understanding computer security audits

- **Learning a security test methodology**
- **Reviewing host and network security**
- **Appreciating vulnerability testing**
- **Exploring different security testing tools**

When you see the term *audit*, the odds are good you think of the kind involving taxes. In actuality, many types of audits exist, and one of them is a *computer security audit.* The purpose of a computer security audit, in its simplest form, is to test your system and network security. For larger organizations, an independent auditor (much like with the auditing of financial statements) can do the security audit. If you have only a few Linux systems or a small network, you can do the security audit as a self-assessment, just to figure out if you're doing everything okay.

This chapter explains how to perform computer security audits and shows you a number of free tools and resources to help you test your system's security.

Understanding Security Audits

An *audit* is simply an independent assessment of whatever it is you're auditing. So a *computer security audit* is an independent assessment of computer security. If someone conducts a computer security audit of your organization, he or she focuses typically on two areas:

- **Independent verification** of whether your organization complies with its existing policies and procedures for computer security. This part is the nontechnical aspect of the security audit.
- **Independent testing** of how effective your security controls (any hardware and software mechanisms you use to secure the system) are. This part is the technical aspect of the security audit.

Why do you need security audits? For the same reason you need financial audits — mainly to verify that everything is being done the way it's supposed to be done. For public as well as private organizations, management may want to have independent security audits done so as to assure themselves that their security is A-OK. Irrespective of your organization's size, you can always perform security audits on your own, either to prepare for independent security audits or simply to know that you're doing everything correctly.

No matter whether you have independent security audits or a self-assessment, here are some of the benefits you get from security audits:

- Periodic risk assessments that consider internal and external threats to systems and data
- Periodic testing of the effectiveness of security policies, security controls, and techniques
- Identification of any significant deficiencies in your system's security (so you know what to fix)
- In the case of self-assessments, preparation for any annual independent security testing that your organization might have to face

Nontechnical aspects of security audits

The nontechnical side of computer security audits focuses on your organization-wide security framework. The audit examines how well the organization has set up and implemented the policies, plans, and procedures for computer security. Here's a list of some items to be verified:

- Risks are periodically assessed.
- An entity-wide security program plan is in place.
- A security program-management structure is in place.
- Computer security responsibilities are clearly assigned.
- Effective security-related personnel policies are in place.
- The security program's effectiveness is monitored and changes are made when needed.

As you may expect, the nontechnical aspects of the security audit involve reviewing documents and interviewing appropriate individuals to find out how the organization manages computer security. For a small organization or a home PC, expecting plans and procedures in documents is ridiculous. In those cases, simply make sure that you have some technical controls in place to secure your system and your network connection.

Technical aspects of security audits

The technical side of computer security audits focuses on testing the technical controls that secure your hosts and network. The testing involves determining

- **How well the host is secured.** Are all operating system patches applied? Are the file permissions set correctly? Are user accounts protected? Are file changes monitored? Are log files monitored? And so on.
- **How well the network is secured.** Are unnecessary Internet services turned off? Is a firewall installed? Are remote logins secured with tools such as SSH? Are TCP wrapper access controls used? And so on.

Typically, security experts use automated tools to perform these two security reviews, for individual hosts and for the entire network.

Implementing a Security Test Methodology

A key element of a computer security audit is a security test that checks the technical mechanisms used to secure a host and the network. The security-test methodology follows these high-level steps:

1. Take stock of the organization's networks, hosts, network devices (routers, switches, firewalls, and so on), and Internet connection.
2. If there are many hosts and network connections, determine which are the important hosts and network devices that need to be tested. The importance of a host depends on the kinds of applications it runs. For example, a host that runs the corporate database would be more important than the hosts that serve as desktop systems.
3. Test the hosts individually. Typically, this step involves logging in as a system administrator and checking various aspects of host security, from passwords to system log files.
4. Test the network. This step is usually performed by attempting to break through the network defenses from another system on the Internet. If there's a firewall, the testing checks that the firewall is indeed configured correctly.
5. Analyze the test results of both host and network tests to determine vulnerabilities and risks.

Each of the two types of testing — host and network — focuses on three areas of overall computer security:

- **Prevention:** Includes the mechanisms (nontechnical and technical) that help prevent attacks on the system and the network.

- **Detection:** Refers to techniques such as monitoring log files, checking file integrity, and using *intrusion detection* systems that can detect when someone is about to break into (or has already broken into) your system.
- **Response:** Includes the steps for tasks such as reporting an incident to authorities and restoring important files from backup after a computer security incident occurs.

For host and network security, each of these areas has some overlaps. For example, prevention mechanisms for host security (such as good passwords or file permissions) can also provide network security. Nevertheless, thinking in terms of the three areas — prevention, detection, and response — does help.

Some common computer vulnerabilities

Before you can think of prevention, however, you have to know the types of problems you're trying to prevent — the common security vulnerabilities. The prevention and detection steps typically depend on the specific vulnerabilities. Basically, the idea is to check whether a host or a network has the vulnerabilities that crackers exploit.

Online resources on computer vulnerabilities

Several online resources identify and categorize computer security vulnerabilities:

- **SANS Institute** publishes a list of the top 20 most critical Internet security vulnerabilities — the Top Cyber Security Risks index — at `www.sans.org/top20`.
- **CVE** (Common Vulnerabilities and Exposures) is a list of standardized names of vulnerabilities. For more information on CVE, see `http://cve.mitre.org`. Using the CVE name to describe vulnerabilities is common practice.
- **National Vulnerability Database (NVD)** is a searchable index of information on computer vulnerabilities, published by the National Institute of Standards and Technology (NIST), a United States government agency. NVD is online at `http://nvd.nist.gov`.

Typical computer vulnerabilities

The SANS Internet security vulnerabilities list includes several types of vulnerabilities, such as Windows, cross-platform, and Unix. Of these, Unix and cross-platform vulnerabilities are relevant to Linux.

LXO-102 Exam

Table 3-1 summarizes some common Unix and cross-platform vulnerabilities that apply to Linux.

Table 3-1 Some Common Vulnerabilities to Unix Systems

Vulnerability Type	*Description*
BIND DNS	Berkeley Internet Name Domain (BIND) is a package that implements Domain Name System (DNS), the Internet's name service that translates a name to an IP address. Some versions of BIND have vulnerabilities.
Apache Web Server	Some Apache Web Server modules (such as `mod_ssl`) have known vulnerabilities. Any vulnerability in Common Gateway Interface (CGI) programs used with web servers to process interactive web pages can provide attackers a way to gain access to a system.
Authentication	User accounts often have no passwords or have weak passwords that are easily cracked by password-cracking programs.
CVS, Subversion	Concurrent Versions System (CVS) is a popular source-code control system used in Linux systems. Subversion is another version control system for Linux that is becoming popular. These version control systems have vulnerabilities that can enable an attacker to execute arbitrary code on the system.
`sendmail`	`sendmail` is a complex program used to transport mail messages from one system to another, and some versions of `sendmail` have vulnerabilities.
SNMP	Simple Network Management Protocol (SNMP) is used to remotely monitor and administer various network-connected systems ranging from routers to computers. SNMP lacks good access control, so an attacker may be able to reconfigure or shut down your system if it is running SNMP.
Open Secure Sockets Layer (OpenSSL)	Many applications, such as Apache Web Server, use OpenSSL to provide cryptographic security for a network connection. Unfortunately, some versions of OpenSSL have known vulnerabilities that could be exploited.

(continued)

Table 3-1 *(continued)*

Vulnerability Type	*Description*
Network File System (NFS) and Network Information Service (NIS)	Both NFS and NIS have many security problems (for example, buffer overflow, potential for denial-of-service attacks, and weak authentication). Also, NFS and NIS are often misconfigured, which could allow local and remote users to exploit the security holes.
Databases	Databases such as MySQL and PostgreSQL are complex applications and can be difficult to correctly configure and secure. These databases have many features that can be misused or exploited to compromise the confidentiality, availability, and integrity of data.
Linux kernel	The Linux kernel is susceptible to many vulnerabilities, such as denial of service, execution of arbitrary code, and `root`-level access to the system.

Host-security review

When reviewing host security, focus on assessing the security mechanisms in each of the following areas:

- **Prevention:** Install operating system updates, secure passwords, improve file permissions, set up a password for a boot loader, and use encryption.
- **Detection:** Capture log messages and check file integrity with Tripwire (a tool that can detect changes to system files).
- **Response:** Make routine backups and develop incident response procedures.

The following sections review a few of these host-security mechanisms.

Operating system updates

Linux distributions release updates soon. When security vulnerabilities are found, Linux distributions release an update to fix the problem. Many distributions offer online updates that you can enable and use to keep your system up to date. The details of updating the operating system depend on the distribution. (See Book V, Chapter 4 for information on how to update Linux online.)

File permissions

Protect important system files with appropriate file ownerships and file permissions. The key procedures in assigning file-system ownerships and permissions are as follows:

- Figure out which files contain sensitive information and why. Some files may contain sensitive data related to your work or business, whereas many other files are sensitive because they control the Linux system configuration.
- Maintain a current list of authorized users and what they are authorized to do on the system.
- Set up passwords, groups, file ownerships, and file permissions to allow only authorized users to access the files.

Table 3-2 lists some important system files in Linux, showing the typical numeric permission setting for each file (this may differ slightly, depending on the distribution). See Chapter 2 of this minibook for more on numeric permission settings.

Table 3-2 Important System Files and Their Permissions

File Pathname	*Permission*	*Description*
`/boot/grub/menu.lst`	600	GRUB boot loader menu file
`/etc/cron.allow`	400	List of users permitted to use `cron` to submit periodic jobs
`/etc/cron.deny`	400	List of users who can't use `cron` to submit periodic jobs
`/etc/crontab`	644	System-wide periodic jobs
`/etc/hosts.allow`	644	List of hosts allowed to use Internet services that are started using TCP wrappers
`/etc/hosts.deny`	644	List of hosts denied access to Internet services that are started using TCP wrappers
`/etc/logrotate.conf`	644	File that controls how log files rotate
`/etc/pam.d`	755	Directory with configuration files for pluggable authentication modules (PAMs)
`/etc/passwd`	644	Old-style password file with user account information but not the passwords

(continued)

Table 3-2 *(continued)*

File Pathname	*Permission*	*Description*
`/etc/rc.d`	755	Directory with system-startup scripts
`/etc/securetty`	600	TTY interfaces (terminals) from which `root` can log in
`/etc/security`	755	Policy files that control system access
`/etc/shadow`	400	File with encrypted passwords and password expiration information
`/etc/shutdown.allow`	400	Users who can shut down or reboot by pressing Ctrl+Alt+Delete
`/etc/ssh`	755	Directory with configuration files for the Secure Shell (SSH)
`/etc/sysconfig`	755	System configuration files
`/etc/sysctl.conf`	644	Kernel configuration parameters
`/etc/syslog.conf`	644	Configuration file for the `syslogd` server that logs messages
`/etc/udev/udev.conf`	644	Configuration file for `udev` — the program that provides the capability to dynamically name hot-pluggable devices and create the device files in the `/dev` directory
`/etc/vsftpd`	600	Configuration file for the Very Secure FTP server
`/etc/vsftpd.ftpusers`	600	List of users who are not allowed to use FTP to transfer files
`/etc/xinetd.conf`	644	Configuration file for the `xinetd` server
`/etc/xinetd.d`	755	Directory containing configuration files for specific services that the `xinetd` server can start
`/var/log`	755	Directory with all log files
`/var/log/lastlog`	644	Information about all previous logins
`/var/log/messages`	644	Main system message log file
`/var/log/wtmp`	664	Information about current logins

Another important check is to look for executable program files that have the setuid permission. If a program has setuid permission and is owned by root, the program runs with root privileges, no matter who actually runs the program. You can find all setuid programs with the following find command:

```
find / -perm +4000 -print
```

You may want to save the output in a file (just append > *filename* to the command) and then examine the file for any unusual setuid programs. For example, a setuid program in a user's home directory is unusual.

Password security

Verify that the password, group, and shadow password files are protected. In particular, the shadow password file has to be write-protected and readable only by root. The filenames and their recommended permissions are shown in Table 3-3.

Table 3-3 Ownership and Permission of Password Files

File Pathname	*Ownership*	*Permission*
/etc/group	root.root	644
/etc/passwd	root.root	644
/etc/shadow	root.root	400

Incident response

Incident response is the policy that answers the question of what to do if something unusual does happen to the system. The policy tells you how to proceed if someone breaks into your system.

Your response to an incident depends on how you use your system and how important it is to you or your business. For a comprehensive incident response, remember these key points:

- Figure out how critical and important your computer and network are — and identify who or what resources can help you protect your system.
- Take steps to prevent and minimize potential damage and interruption.
- Develop and document a comprehensive contingency plan.
- Periodically test the contingency plan and revise the procedures as appropriate.

Network-security review

A *network-security review* focuses on assessing the security mechanisms in each of the following areas:

- **Prevention:** Set up a firewall, enable packet filtering, disable unnecessary inetd or xinetd services, turn off unneeded Internet services, use TCP wrappers for access control, and use SSH for secure remote logins.
- **Detection:** Use network intrusion detection and capture system logs.
- **Response:** Develop incident response procedures.

Some key steps in assessing the network security are described in the following three subsections.

Services started by inetd or xinetd

Depending on your distribution, the inetd or xinetd server may be configured to start some Internet services such as TELNET and FTP. The decision to turn on some of these services depends on such factors as how the system connects to the Internet and how the system is being used. You can usually turn off most inetd and xinetd services by commenting out the line — just place a pound sign (#) at the beginning of the line.

If you are using xinetd, it is possible to see which services are turned off by checking the configuration files in the /etc/xinetd.d directory for all the configuration files that have a disable = yes line. (The line doesn't count if it's commented out, which is indicated by a # character at the beginning of the line.) You can add a disable = yes line to the configuration file of any service that you want to turn off.

Also check the following files for any access controls used with the inetd or xinetd services:

- /etc/hosts.allow lists hosts allowed to access specific services.
- /etc/hosts.deny lists hosts denied access to services.

Standalone services

Many services, such as apache or httpd (web server) and sendmail (mail server), start automatically at boot time, assuming they're configured to start that way.

In some distributions, you can use the chkconfig command to check which of these standalone servers are set to start at various run levels. (See Book V, Chapter 1 for more about run levels.) Typically, most systems start up at run level 3 (for text login) or 5 (for graphical login). Therefore, what matters is the setting for the servers in levels 3 and 5. To view the list of servers, type **chkconfig --list | more**. When you do a self-assessment of your network security and find that some servers shouldn't be running, you can turn them off for run levels 3 and 5 by typing **chkconfig –level 35 *servicename* off**, where *servicename* is the name of the service you want to turn off.

In some distributions, you can use a GUI tool to see which services are enabled and running at any run level. With YaST, for example, click System on the left side of the window, and then click Runlevel Editor on the right side of the window.

When you audit network security, make a note of all the servers that are turned on — and then try to determine whether they should really *be* on, according to what you know about the system. The decision to turn on a particular service depends on how your system is used (for example, as a web server or as a desktop system) and how it's connected to the Internet (say, through a firewall or directly).

Penetration test

A penetration test is the best way to tell what services are really running on a Linux system. *Penetration testing* involves trying to get access to your system from an attacker's perspective. Typically, you perform this test from a system on the Internet and try to break in or, at minimum, get access to services running on your Linux system.

One aspect of penetration testing is to see what ports are open on your Linux system. The *port number* is simply a number that identifies TCP/IP network connections to the system. The attempt to connect to a port succeeds only if a server is running, or "listening," on that port. A port is considered to be open if a server responds when a connection request for that port arrives.

The first step in penetration testing is to perform a port scan. The term *port scan* describes the automated process of trying to connect to each port number to see whether a valid response comes back. Many available automated tools can perform port scanning — you can install and use a popular port-scanning tool called `nmap` (described later in this chapter).

After performing a port scan, you know which ports are open and could be exploited. Not all servers have security problems, but many servers have well-known vulnerabilities. An open port provides a cracker a way to attack your system through one of the servers. In fact, you can use automated tools called *vulnerability scanners* to identify vulnerabilities that exist in your system (some vulnerability scanners are described in the following sections). Whether your Linux system is connected to the Internet directly (through DSL or cable modem) or through a firewall, use the port-scanning and vulnerability-scanning tools to figure out whether you have any holes in your defenses.

Vulnerability Testing Types

The number-one purpose of penetration testing is to identify vulnerabilities. When viewing such a test from this angle, it is important to understand that there are three ways of approaching it: *black, white,* or *gray*. These three approaches differ in the amount of information you assume you have in the beginning; you can use the color with almost any other word: *black box* versus *white box* if it is a piece of software doing the testing; *black hat* versus *white hat* if is an individual doing the testing; and so on. The following discussion focuses on the individual and uses *box* as the preferred noun.

- With *black-box testing,* the tests assume no knowledge of the network and look for vulnerabilities that an outsider might stumble across, such as open ports and weak passwords.

  Imagine that a bored miscreant came across your network at random and decided to bring it to its knees.

- With *white-box testing,* the test assumes that the attacker is a knowledgeable insider who's trying to break the system.

  Imagine that you just fired a system administrator and they want to get back at you by crashing your network.

- Between these two extremes rests the realm of *gray-box testing*. Here the assumption is that an insider is behind the problem.

  Imagine someone from shipping is angry about not getting the raise he or she thought was deserved, and so wants to make the company pay. The attacker doesn't have the knowledge an administrator would, but still knows more about the systems than a complete outsider would.

Exploring Security Testing Tools

Many automated tools are available to perform security testing. Some of these tools are meant for finding the open ports on every system in a range of IP addresses. Others look for the vulnerabilities associated with open ports. Yet other tools can capture (or *sniff*) those weaknesses and help you analyze them so that you can glean useful information about what's going on in your network.

You can browse a list of the top 100 security tools (based on an informal poll of `nmap` users) at `http://sectools.org`. Table 3-4 lists a number of these tools by category. A few of the freely available vulnerability scanners are described in the next few sections.

Table 3-4 Some Popular Computer-Security Testing Tools

Type	*Names of Tools*
Port scanners	`nmap`, Strobe
Vulnerability scanners	Nessus Security Scanner, SAINT, SARA, Whisker (CGI scanner), ISS Internet Scanner, CyberCop Scanner, Vetescan, Retina Network Security Scanner
Network utilities	Netcat, `hping2`, Firewalk, Cheops, `ntop`, `ping`, `ngrep`, AirSnort (802.11 WEP encryption-cracking tool)
Host-security tools	Tripwire, lsof
Packet sniffers	`tcpdump`, Ethereal, `dsniff`, `sniffit`
Intrusion detection	Snort, Abacus `portsentry`, `scanlogd`, NFR, LIDSSystems (IDSs)
Password-checking tools	John the Ripper, LC4
Log analysis and monitoring tools	`logcolorise`, `tcpdstats`, `nlog`, `logcheck`, LogWatch, Swatch

nmap

`nmap` (short for *network mapper*) is a port-scanning tool. It can rapidly scan large networks and determine what hosts are available on the network, what services they offer, what operating system (and the operating system version) they run, what type of packet filters or firewalls they use, and dozens of other characteristics. You can read more about `nmap` at `http://nmap.org`.

If `nmap` is not already installed, you can easily install it on your distribution either with the command `apt-get install nmap` or through the software search facility of YaST (find `nmap`) or any distribution-specific interface you may have.

If you want to try out `nmap` to scan your local area network, type a command similar to the following (replace the IP address range with addresses appropriate for your network):

```
nmap -O -sS 192.168.0.4-8
```

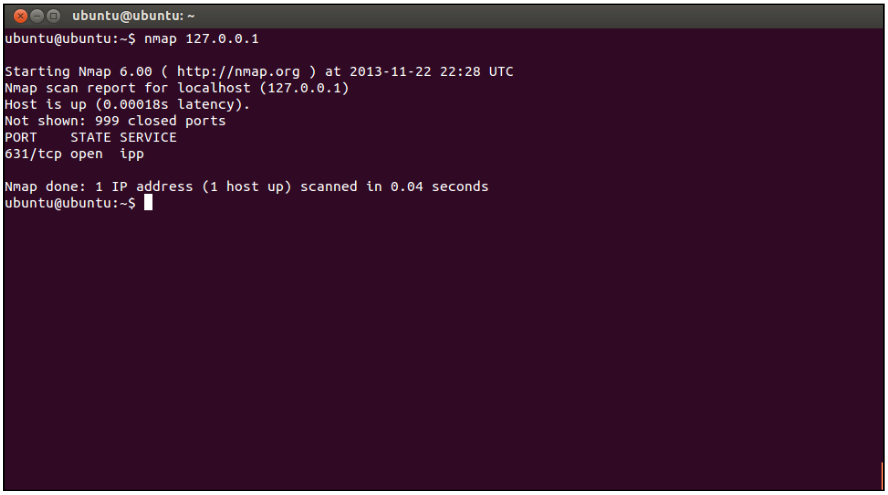
Here's a typical output listing from that command:

```
Starting nmap 6.40 ( http://www.insecure.org/nmap/ ) at 2013-08-28 16:20 EDT
Interesting ports on 192.168.0.4:
(The 1659 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
21/tcp open ftp
```

```
22/tcp open ssh
111/tcp open rpcbind
631/tcp open ipp
MAC Address: 00:C0:49:63:78:3A (U.S. Robotics)
Device type: general purpose
Running: Linux 3.9.X|3.9.X|3.9.X
OS details: Linux 2.4.18 - 2.6.7
Uptime 9.919 days (since Thu Aug 18 18:18:15 2013)
. . . Lines deleted . . .
Nmap finished: 5 IP addresses (5 hosts up) scanned in 30.846 seconds
```

As you can see, `nmap` displays the names of the open ports and hazards a guess at the operating system name and version number.

For a very quick scan of your own machine, you can use the IP address of 127.0.0.1 (as shown in Figure 3-1); hopefully the scan will verify that the ports are closed.

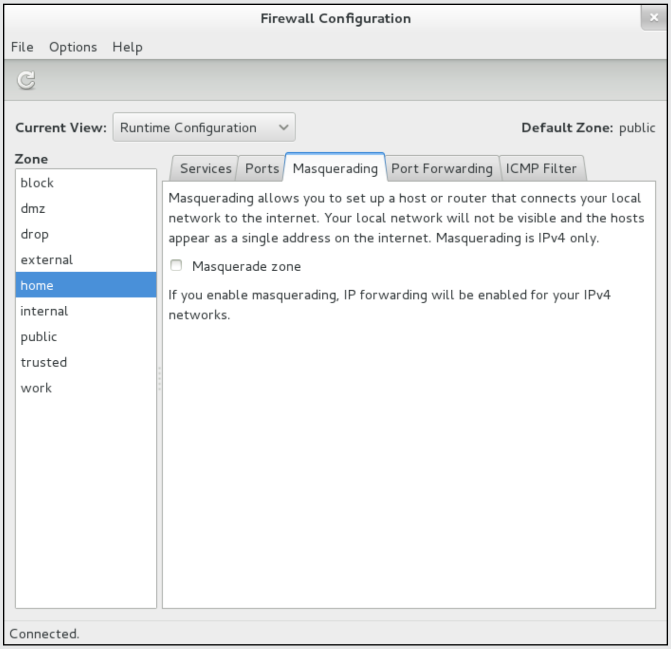


```
ubuntu@ubuntu:~$ nmap 127.0.0.1

Starting Nmap 6.00 ( http://nmap.org ) at 2013-11-22 22:28 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00018s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE
631/tcp open  ipp

Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
ubuntu@ubuntu:~$
```

Figure 3-1: You can view any open ports with nmap.

Book VII

Scripting



Current View: Runtime Configuration

Default Zone: public

Masquerading allows you to set up a host or router that connects your local network to the internet. Your local network will not be visible and the hosts appear as a single address on the internet. Masquerading is IPv4 only.

□ Masquerade zone

If you enable masquerading, IP forwarding will be enabled for your IPv4 networks.

Connected.

Visit `www.dummies.com/extras/linuxaio` for great Dummies content online.

Contents at a Glance

Chapter 1: Introductory Shell Scripting 445

Trying Out Simple Shell Scripts 445
Exploring the Basics of Shell Scripting 447

Chapter 2: Advanced Shell Scripting 457

Trying Out sed 457
Working with awk and sed 459
Final Notes on Shell Scripting 463

Chapter 3: Programming in Linux 465

An Overview of Programming 465
Exploring the Software-Development Tools in Linux 467
Understanding the Implications of GNU Licenses 484

Chapter 1: Introductory Shell Scripting

In This Chapter

- **Trying some simple shell scripts**
- **Discovering the basics of shell scripting**
- **Exploring bash's built-in commands**

Linux gives you many small and specialized commands, along with the plumbing necessary to connect these commands. Take *plumbing* to mean the way in which one command's output can be used as a second command's input. `bash` (short for Bourne-Again Shell) — the default shell in most Linux systems — provides this plumbing in the form of I/O redirection and pipes. `bash` also includes features such as the `if` statement that you can use to run commands only when a specific condition is true, and the `for` statement that repeats commands a specified number of times. You can use these features of `bash` when writing programs called *shell scripts* — task-oriented collections of shell commands stored in a file.

This chapter shows you how to write simple shell scripts, which are used to automate various tasks. For example, when your Linux system boots, many shell scripts stored in various subdirectories in the `/etc` directory (for example, `/etc/init.d`) perform many initialization tasks.

Trying Out Simple Shell Scripts

If you're not a programmer, you may feel apprehensive about programming. But shell *scripting* (or programming) can be as simple as storing a few commands in a file. In fact, you can have a useful shell program that has a single command.

Shell scripts are popular among system administrators. If you are a system administrator, you can build a collection of custom shell scripts that help you automate tasks you perform often. If a hard drive seems to be getting full, for example, you may want to find all files that exceed some size (say, 1MB) and that have not been accessed in the past 30 days. In addition, you may want to send an e-mail message to all users who have large files, requesting that they archive and clean up those files. You can perform all these tasks with a shell script. You might start with the following find command to identify large files:

```
find / -type f -atime +30 -size +1000k -exec ls -l {} \; > /tmp/largefiles
```

This command creates a file named /tmp/largefiles, which contains detailed information about old files taking up too much space. After you get a list of the files, you can use a few other Linux commands — such as sort, cut, and sed — to prepare and send mail messages to users who have large files to clean up. Instead of typing all these commands manually, place them in a file and create a shell script. That, in a nutshell, is the essence of shell scripts — to gather shell commands in a file so that you can easily perform repetitive system administration tasks.

bash scripts, just like most Linux commands, accept command-line options. Inside the script, you can refer to the options as $1, $2, and so on. The special name $0 refers to the name of the script itself.

Here's a typical bash script that accepts arguments:

```
#!/bin/sh
echo "This script's name is: $0"
echo Argument 1: $1
echo Argument 2: $2
```

The first line runs the /bin/sh program, which subsequently processes the rest of the lines in the script. The name /bin/sh traditionally refers to the Bourne shell — the first Unix shell. In most Linux systems, /bin/sh is a symbolic link to /bin/bash, which is the executable program for bash.

Save this simple script in a file named simple and make that file executable with the following command:

```
chmod +x simple
```

Now run the script as follows:

```
./simple
```

It displays the following output:

```
This script's name is: ./simple
Argument 1:
Argument 2:
```

The first line shows the script's name. Because you have run the script without arguments, the script displays no values for the arguments.

Now try running the script with a few arguments, like this:

```
./simple "This is one argument" second-argument third
```

This time the script displays more output:

```
This script's name is: ./simple
Argument 1: This is one argument
Argument 2: second-argument
```

As the output shows, the shell treats the entire string within the double quotation marks as a single argument. Otherwise the shell uses spaces as separators between arguments on the command line.

This sample script ignores the third argument because the script is designed to print only the first two arguments.

Exploring the Basics of Shell Scripting

Like any programming language, the bash shell supports the following features:

- Variables that store values, including special built-in variables for accessing command-line arguments passed to a shell script and other special values.
- The capability to evaluate expressions.
- Control structures that enable you to loop over several shell commands or to execute some commands conditionally.
- The capability to define functions that can be called in many places within a script. bash also includes many built-in commands that you can use in any script.

The next few sections illustrate some of these programming features through simple examples. (It's assumed that you're already running bash, in which case, you can try the examples by typing them at the shell prompt in a terminal window. Otherwise all you have to do is open a terminal window; bash runs and displays its prompt in that window.)

Storing stuff

You define variables in bash just as you define environment variables. Thus you may define a variable as follows:

```
count=12 # note no embedded spaces allowed
```

To use a variable's value, prefix the variable's name with a dollar sign ($). For example, $PATH is the value of the variable PATH. (This variable is the famous PATH environment variable that lists all the directories that bash searches when trying to locate an executable file.) To display the value of the variable count, use the following command:

```
echo $count
```

bash has some special variables for accessing command-line arguments. In a shell script, $0 refers to the name of the shell script. The variables $1, $2, and so on refer to the command-line arguments. The variable $* stores all the command-line arguments as a single variable, and $? contains the exit status of the last command the shell executes.

From a bash script, you can prompt the user for input and use the read command to read the input into a variable. Here is an example:

```
echo -n "Enter value: "
read value
echo "You entered: $value"
```

When this script runs, the read value command causes bash to read whatever you type at the keyboard and store your input in the variable called value.

Note: The -n option prevents the echo command from automatically adding a new line at the end of the string that it displays.

Calling shell functions

You can group a number of shell commands that you use consistently into a *function* and assign it a name. Later, you can execute that group of commands by using the single name assigned to the function. Here is a simple script that illustrates the syntax of shell functions:

```
#!/bin/sh
hello() {
echo -n "Hello, "
echo $1 $2
}
hello Jane Doe
```

When you run this script, it displays the following output:

```
Hello, Jane Doe
```

This script defines a shell function named hello. The function expects two arguments. In the body of the function, these arguments are referenced by $1 and $2. The function definition begins with hello() — the name of the function, followed by parentheses. The body of the function is enclosed in curly braces — { . . . }. In this case, the body uses the echo command to display a line of text.

The last line of the example shows how a shell function is called with arguments. In this case, the hello function is called with two arguments: Jane and Doe. The hello function takes these two arguments and prints a line that says Hello, Jane Doe.

Controlling the flow

In bash scripts, you can control the flow of execution — the order in which the commands are executed — by using special commands such as if, case, for, and while. These control statements use the exit status of a command to decide what to do next. When any command executes, it returns an exit status — a numeric value that indicates whether or not the command has succeeded. By convention, an exit status of zero means the command has succeeded. (Yes, you read it right: Zero indicates success!) A nonzero exit status indicates that something has gone wrong with the command.

For example, suppose that you want to make a backup copy of a file before editing it with the vi editor. More importantly, you want to avoid editing the file if a backup can't be made. Here's a bash script that takes care of this task:

```
#!/bin/sh
if cp "$1" "#$1"
then
vi "$1"
else
echo "Failed to create backup copy"
fi
```

This script illustrates the syntax of the if-then-else structure and shows how the exit status of the cp command is used by the if command to determine the next action. If cp returns zero, the script uses vi to edit the file; otherwise, the script displays an error message and exits. By the way, the script saves the backup in a file whose name is the same as that of the original, except for a hash mark (#) added at the beginning of the filename.

TIP

Don't forget the final fi that terminates the if command. Forgetting fi is a common source of errors in bash scripts.

You can use the test command to evaluate any expression and to use the expression's value as the exit status of the command. Suppose that you want a script that edits a file only if it already exists. Using test, you can write such a script as follows:

```
#!/bin/sh
if test -f "$1"
then
vi "$1"
else
echo "No such file"
fi
```

A shorter form of the test command is to place the expression in square brackets ([. . .]). Using this shorthand notation, you can rewrite the preceding script like this:

```
#!/bin/sh
if [ -f "$1" ]
then
vi "$1"
else
echo "No such file"
fi
```

Note: You must have spaces around the two square brackets.

Another common control structure is the for loop. The following script adds the numbers 1 through 10:

```
#!/bin/sh
sum=0
for i in 1 2 3 4 5 6 7 8 9 10
do
sum='expr $sum + $i'
done
echo "Sum = $sum"
```

This example also illustrates the use of the expr command to evaluate an expression.

The case statement is used to execute a group of commands based on the value of a variable. For example, consider the following script:

```
#!/bin/sh
echo -n "What should I do -- (Y)es/(N)o/(C)ontinue? [Y] "
read answer
case $answer in
y|Y|"")
echo "YES"
;;
c|C)
echo "CONTINUE"
;;
n|N)
echo "NO"
;;
*)
echo "UNKNOWN"
;;
esac
```

Save this code in a file named `confirm` and type **chmod +x confirm** to make it executable. Then try it out like this:

```
./confirm
```

When the script prompts you, type one of the characters `y`, `n`, or `c` and press Enter. The script displays `YES`, `NO`, or `CONTINUE`, respectively. For example, here's what happens when you type **c** (and then press Enter):

```
What should I do -- (Y)es/(N)o/(C)ontinue? [Y] c
CONTINUE
```

The script displays a prompt and reads the input you type. Your input is stored in a variable named `answer`. Then the `case` statement executes a block of code based on the value of the `answer` variable. For example, when you type **c**, the following block of commands executes:

```
c|C)
echo "CONTINUE"
;;
```

The `echo` command causes the script to display `CONTINUE`.

From this example, you can see that the general syntax of the `case` command is as follows:

```
case $variable in
value1 | value2)
command1
command2
. . . other commands . . .
;;
value3)
command3
command4
. . . other commands . . .
;;
esac
```

Essentially, the `case` command begins with the word `case` and ends with `esac`. Separate blocks of code are enclosed between the values of the variable, followed by a closing parenthesis and terminated by a pair of semicolons (`;;`).

Exploring bash's built-in commands

`bash` has more than 50 built-in commands, including common commands such as `cd` and `pwd`, as well as many others that are used infrequently. You can use these built-in commands in any `bash` script or at the shell prompt. Table 1-1 describes most of the bash built-in commands and their arguments. After looking through this information, type **help *command*** to read more about a specific built-in command. For example, to find out more about the built-in command `test`, type the following:

```
help test
```

Doing so displays the following information:

```
test: test [expr]
Exits with a status of 0 (true) or 1 (false) depending on
the evaluation of EXPR. Expressions may be unary or binary. Unary
expressions are often used to examine the status of a file. There
are string operators as well, and numeric comparison operators.
File operators:
-a FILE True if file exists.
-b FILE True if file is block special.
-c FILE True if file is character special.
-d FILE True if file is a directory.
-e FILE True if file exists.
-f FILE True if file exists and is a regular file.
-g FILE True if file is set-group-id.
-h FILE True if file is a symbolic link.
-L FILE True if file is a symbolic link.
-k FILE True if file has its 'sticky' bit set.
-p FILE True if file is a named pipe.
-r FILE True if file is readable by you.
-s FILE True if file exists and is not empty.
-S FILE True if file is a socket.
-t FD True if FD is opened on a terminal.
-u FILE True if the file is set-user-id.
-w FILE True if the file is writable by you.
-x FILE True if the file is executable by you.
-O FILE True if the file is effectively owned by you.
-G FILE True if the file is effectively owned by your group.
( . . . Lines deleted . . .)
```

Where necessary, the online help from the `help` command includes a considerable amount of detail.

Table 1-1 Summary of Built-in Commands in bash Shell

This Function	*Does the Following*
. *filename* [*arguments*]	Reads and executes commands from the specified *filename* using the optional *arguments*. (Works the same way as the `source` command.)
: [*arguments*]	Expands the *arguments* but does not process them.
[*expr*]	Evaluates the expression *expr* and returns zero status if `expr` is true.
alias [*name*[=*value*] . . .]	Allows one *value* to equal another. For example, you could set *xyz* to run *bg*.

This Function	Does the Following
`bg [job]`	Puts the specified *job* in the background. If no *job* is specified, it puts the currently executing command in the background.
`break [n]`	Exits from a `for`, `while`, or `until` loop. If *n* is specified, the *n*th enclosing loop is exited.
`cd [dir]`	Changes the current directory to *dir*.
`command [-pVv] cmd [arg . . . ]`	Runs the command *cmd* with the specified arguments (ignoring any shell function named `cmd`).
`continue [n]`	Starts the next iteration of the `for`, `while`, or `until` loop. If *n* is specified, the next iteration of the *n*th enclosing loop is started.
`declare [-frxi] [name[=value]]`	Declares a variable with the specified *name* and optionally, assigns it a *value*.
`dirs [-l] [+/-n]`	Displays the list of currently remembered directories.
`echo [-neE] [arg . . . ]`	Displays the arguments, *arg . . .* , on standard output.
`enable [-n] [-all]`	Enables or disables the specified built-in commands.
`eval [arg . . . ]`	Concatenates the arguments, *arg . . .* , and executes them as a command.
`exec [command [arguments]]`	Replaces the current instance of the shell with a new process that runs the specified *command*. with the given *arguments*.
`exit [n]`	Exits the shell with the status code *n*.
`export [-nf] [name[=word]] . . .`	Defines a specified environment variable and exports it to future processes.
`fc -s [pat=rep] [cmd]`	Re-executes the command after replacing the pattern *pat* with *rep*.
`fg [jobspec]`	Puts the specified job, *jobspec*, in the foreground. If no job is specified, it puts the most recent job in the foreground.

(continued)

Table 1-1 *(continued)*

This Function	*Does the Following*
`hash [-r] [name]`	Remembers the full pathname of a specified command.
`help [cmd . . . ]`	Displays help information for specified built-in commands, *cmd. . . .*
`history [n]`	Displays past commands or past *n* commands, if you specify a number *n*.
`jobs [-lnp] [ jobspec . . . ]`	Lists currently active jobs.
`kill [-s sigspec | -sigspec] [pid | jobspec] . . . let arg [arg . . . ]`	Evaluates each argument and returns 1 if the last *arg* is 0.
`local [name[=value] . . . ]`	Creates a local variable with the specified *name* and *value* (used in shell functions).
`logout`	Exits a login shell.
`popd [+/-n]`	Removes the specified number of entries from the directory stack.
`pushd [dir]`	Adds a specified directory, *dir*, to the top of the directory stack.
`pwd`	Prints the full pathname of the current working directory.
`read [-r] [name . . . ]`	Reads a line from standard input and parses it.
`readonly [-f] [name . . . ]`	Marks the specified variables as read-only so that the variables cannot be changed later.
`return [n]`	Exits the shell function with the return value *n*.
`set [--abefhkmnptuvxldCHP] [-o option] [arg . . . ]`	Sets various flags.
`shift [n]`	Makes the *n*+1 argument `$1`, the *n*+2 argument `$2`, and so on.
`times`	Prints the accumulated user and system times for processes run from the shell.
`trap [-l] [cmd] [sigspec]`	Executes *cmd* when the signal *sigspec* is received.

This Function	*Does the Following*
type [-all] [-type |-*path*] *name* [*name* . . .]	Indicates how the shell interprets each *name*.
ulimit [-SHacdfmstpnuv [limit]]	Controls resources available to the shell.
umask [-S] [*mode*]	Sets the *file creation mask* — the default permission to the *mode* specified for the files.
unalias [-a] [*name* . . .]	Undefines a specified alias.
unset [-fv] [*name* . . .]	Removes the definition of specified variables.
wait [*n*]	Waits for a specified process (*n* represents its PID) to terminate.

Some external programs may have the same name as bash built-in commands. If you want to run any such external program, you have to specify explicitly the full pathname of that program. Otherwise bash executes the built-in command of the same name.

Chapter 2: Advanced Shell Scripting

In This Chapter

- **Trying out the sed command**
- **Working with the awk and sed commands**
- **Reading some final notes on shell scripting**

The preceding chapter introduces you to some of the power available through shell scripting. All the scripts in that chapter are simple bash routines that allow you to run commands and repeat operations a number of times.

This chapter builds upon that knowledge by showing how to incorporate two powerful tools — sed and awk — into your scripts. These two utilities move your scripts to the place where the only limit to what you can do becomes your ability to figure out how to ask for the output you need. Although sed is the stream editor and awk is a quick programming language, they complement each other so well that it's not uncommon to use one with the other. The best way to show how these tools work is to walk through some examples.

Trying Out sed

The following are sample lines of a colon-delimited employee database that has five fields: unique id number, name, department, phone number, and address.

```
1218:Kris Cottrell:Marketing:219.555.5555:123 Main Street
1219:Nate Eichhorn:Sales:219.555.5555:1219 Locust Avenue
1220:Joe Gunn:Payables:317.555.5555:21974 Unix Way
1221:Anne Heltzel:Finance:219.555.5555:652 Linux Road
1222:John Kuzmic:Human Resources:219.555.5555:984 Bash Lane
```

This database has been in existence since the beginning of the company and has grown to include everyone who now works, or has ever worked, for the company. A number of proprietary scripts read from the database, and the company cannot afford to be without it. The problem is that the telephone company has changed the 219 prefix to 260, so all entries in the database need to be changed.

This is precisely the task for which sed was created. As opposed to standard (interactive) editors, a *stream editor* works its way through a file and makes changes based on the rules it is given. The rule in this case is to change 219 to 260. It's not quite that simple, however, because if you use the command

```
sed 's/219/260/'
```

the result is not completely what you want (changes are in **bold**):

```
1218:Kris Cottrell:Marketing:260.555.5555:123 Main Street
1260:Nate Eichhorn:Sales:219.555.5555:1219 Locust Avenue
1220:Joe Gunn:Payables:317.555.5555:26074 Unix Way
1221:Anne Heltzel:Finance:260.555.5555:652 Linux Road
1222:John Kuzmic:Human Resources:260.555.5555:984 Bash Lane
```

The changes in the first, fourth, and fifth lines are correct. But in the second line, the first occurrence of 219 appears in the employee id number rather than in the phone number and was changed to 260. If you wanted to change more than the very first occurrence in a line, you could slap a g (for *global*) into the command:

```
sed 's/219/260/g'
```

That is *not* what you want to do in this case, however, because the employee id number should not change. Similarly, in the third line, a change was made to the address because it contains the value that is being searched for; no change should have been made because the employee does not have the 219 telephone prefix.

The first rule of using sed is to identify what makes the location of the string you are looking for unique. If the telephone prefix were encased in parentheses, it would be much easier to isolate. In this database, though, that is not the case; the task becomes a bit more complicated.

If you said that the telephone prefix must appear at the beginning of the field (denoted by a colon), the result would be much closer to what you want:

```
sed 's/:219/:260/'
```

Again, bolding has been added to show the changes:

```
1218:Kris Cottrell:Marketing:260.555.5555:123 Main Street
1219:Nate Eichhorn:Sales:260.555.5555:1219 Locust Avenue
1220:Joe Gunn:Payables:317.555.5555:26074 Unix Way
1221:Anne Heltzel:Finance:260.555.5555:652 Linux Road
1222:John Kuzmic:Human Resources:260.555.5555:984 Bash Lane
```

The accuracy has increased, but there is still the problem of the third line. Because the colon helped to identify the start of the string, it may be tempting to turn to the period to identify the end:

```
sed 's/:219./:260./'
```

But the result still isn't what was hoped for (note the third line):

```
1218:Kris Cottrell:Marketing:260.555.5555:123 Main Street
1219:Nate Eichhorn:Sales:260.555.5555:1219 Locust Avenue
1220:Joe Gunn:Payables:317.555.5555:260.4 Unix Way
1221:Anne Heltzel:Finance:260.555.5555:652 Linux Road
1222:John Kuzmic:Human Resources:260.555.5555:984 Bash Lane
```

Because the period has a special meaning of *any character,* a match is found whether the 219 is followed by a period, a 7, or any other single character. Whatever the character, it is replaced with a period. The replacement side of things isn't the problem; the search needs to be tweaked. By using the \ character, we can override the special meaning of the period and specify that you are indeed looking for a period and not any single character:

```
sed 's/:219\./:260./'
```

The result becomes:

```
1218:Kris Cottrell:Marketing:260.555.5555:123 Main Street
1219:Nate Eichhorn:Sales:260.555.5555:1219 Locust Avenue
1220:Joe Gunn:Payables:317.555.5555:21974 Unix Way
1221:Anne Heltzel:Finance:260.555.5555:652 Linux Road
1222:John Kuzmic:Human Resources:260.555.5555:984 Bash Lane
```

And the mission is accomplished.

Working with awk and sed

The second example involves a database of books that includes the ISBN number of each title. In the old days, ISBN numbers were ten digits and included an identifier for the publisher and a unique number for each book. ISBN numbers are now thirteen digits for new books. Old books (those published before the first of 2007) have both the old 10-digit and a new 13-digit number that can be used to identify them. For this example, the existing 10-digit number will stay in the database and a new field — holding the ISBN-13 number — will be added to the end of each entry.

To come up with the ISBN-13 number for the existing entries in the database, you start with 978, then use the first 9 digits of the old ISBN number. The thirteenth digit is a mathematical calculation (a *check digit*) obtained by doing the following:

1. Add all odd-placed digits (the first, the third, the fifth, and so on).
2. Multiply all even-placed digits by 3 and add them.
3. Add the total of Step #2 to the total of Step #1.
4. Find out what you need to add to round the number up to the nearest 10. This value becomes the thirteenth digit.

For example, consider the 10-digit ISBN 0743477103. It first becomes 978074347710, and then the steps work out like this:

1. 9+8+7+3+7+1=35
2. 7*3=21 ; 0*3=0; 4*3=12; 4*3=12; 7*3=21; 0*3=0; 21+0+12+12+21+0=66
3. 66+35=101
4. 110-101=9. The ISBN-13 thus becomes 9780743477109.

The beginning database resembles:

```
0743477103:Macbeth:Shakespeare, William
1578518520:The Innovator's Solution:Christensen, Clayton M.
0321349946:(SCTS) Symantec Certified Technical Specialist:Alston, Nik
1587052415:Cisco Network Admission Control, Volume I:Helfrich, Denise
```

And you want the resulting database to change so each line resembles something like this:

```
0743477103:Macbeth:Shakespeare, William:9780743477109
```

The example that follows accomplishes this goal. It's not the prettiest thing ever written, but it walks through the process of tackling this problem, illustrating the use of awk and sed. I have also included writing to temporary files so you can examine those files and see the contents at various stages. Clean programming would mitigate the use of temporary files everywhere possible, but that practice also makes it difficult to follow the action at times. That said, here is one solution out of dozens. Read on.

Step 1: Pull out the ISBN

Given the database as it now exists, the first order of business is to pull out the existing ISBN — only the first nine digits because the tenth digit, which was just a checksum, no longer matters — and slap 978 onto the beginning. The nine digits we want are the first nine characters of each line, so we can pull them out by using the cut utility:

```
cut -c1e-9 books
```

Because a mathematical operation will be performed on the numbers comprising this value, and that operation works with each digit, I'll add a space between each number and the next in the new entry:

```
sed 's/[0-9]/& /g'
```

Now it's time to add the new code to the beginning of each entry (the start of every line):

```
sed 's/^/9 7 8 /'
```

And finally, I do an extra step: removing the white space at the end of the line just to make the entry a bit cleaner:

```
sed 's/ $//'
```

Then I write the results to a temporary file that can be examined to make sure all is working as it should. The full first step then becomes

```
cut -c1-9 books | sed 's/[0-9]/& /g' | sed 's/^/9 7 8 /' | sed 's/ $//' > isbn2
```

Note: the `sed` operations could be combined in a script file to increase speed and decrease cycles. However, I am walking through each operation step-by-step to show what's going on, and am not worried about creating script files for this one-time-only operation.

Examining the temporary file, the contents are as follows:

```
9 7 8 0 7 4 3 4 7 7 1 0
9 7 8 1 5 7 8 5 1 8 5 2
9 7 8 0 3 2 1 3 4 9 9 4
9 7 8 1 5 8 7 0 5 2 4 1
```

Step 2: Calculate the 13th digit

We've taken care of the first 12 digits of the ISBN number. Now we need to compute those 12 digits to figure out the thirteenth value. Because the numbers are separated by a space, `awk` can interpret them as fields. The calculation will take several steps:

1. Add all the odd-placed digits: x=$1+$3+$5+$7+$9+$11.
2. Add all the even-placed digits and multiply by 3:

   y=($2+$4+$6+$8+$10+$12)*3.
3. Add the total of Step #2 to the total of Step #1: x=x+y.
4. Find out what you need to add to round the number up to the nearest 10 by computing the modulo when divided by 10, and then subtracting it from 10. The following `awk` command gets everything in place except the transformation:

   ```
   awk '{ x=$1+$3+$5+$7+$9+$11 ; y=$2+$4+$6+$8+$10+$12 ; y=y*3 ; x=x+y ;
       y=x%10 ; print y }'
   ```

Everything is finished except subtracting the final result from 10. This is the hardest part. If the modulo is 7, for example, the check digit is 3. If the modulo is 0, however, the check digit does not become 10 (10 – 0), but stays 0. My solution is to use the transform function of `sed`:

```
sed 'y/12346789/98764321/'
```

Combining the two operations into one, the second step thus becomes

```
awk '{ x=$1+$3+$5+$7+$9+$11 ; y=$2+$4+$6+$8+$10+$12 ; y=y*3 ; x=x+y ; y=x%10 ;
    print y }' | sed 'y/12346789/98764321/' > isbn3
```

Examining the temporary file, the contents are

```
9
4
1
5
```

Step 3: Add the 13th digit to the other 12

The two temporary files can now be combined to get the correct 13-digit ISBN number. Just as `cut` was used in the earlier step, `paste` can be used now to combine the files. The default delimiter for `paste` is a tab, but we can change that to anything with the `-d` option. I use a space as the delimiter, and then use `sed` to strip the spaces (remember that the `isbn2` file has spaces between the digits so that they can be read as fields):

```
paste -d" "isbn2 isbn3 | sed 's/ //g'
```

Finally, I want to add a colon as the first character of each entry to make it easier to append the newly computed ISBN to the existing file:

```
sed 's/^/:/'
```

And the entire command becomes

```
paste -d" "isbn2 isbn3 | sed 's/ //g' | sed 's/^/:/' > isbn4
```

Examining the temporary file, the contents are

```
:9780743477109
:9781578518524
:9780321349941
:9781587052415
```

Step 4: Finish the process

The only operation remaining is to append the values in the temporary file to the current database. I'll use the default tab delimiter in the entry, and then strip it out. Technically, I could specify a colon as the delimiter and avoid the last part of the last steps. However, I would rather have my value complete there and be confident that I am stripping characters that don't belong (tabs) instead of running the risk of adding more characters than should be there. The final command is

```
paste books isbn4 | sed 's/\t//g' > newbooks
```

The final file looks like this:

```
0743477103:Macbeth:Shakespeare, William:9780743477109
1578518520:The Innovator's Solution:Christensen, Clayton M.:9781578518524
0321349946:(SCTS) Symantec Certified Technical Specialist:Alston,
    Nik:9780321349941
1587052415:Cisco Network Admission Control, Volume I:Helfrich,
    Denise:9781587052415
```

Again, this result can be accomplished in many ways. This solution is not the cleanest, but it does illustrate the down-and-dirty use of sed and awk.

Final Notes on Shell Scripting

As with any other aspect of computing, it takes a while to get used to shell scripting. After you become comfortable writing scripts, however, you'll find that you can automate any number of operations and simplify your task as an administrator. The following tips can be helpful to keep in mind:

- After you create a script, you can run it automatically on a one-time basis by using at, or on a regular basis by using cron.
- You can use *conditional expressions,* such as if, while, and until, to look for events to occur (such as certain users accessing a file they should not) or to let you know when something that should be there goes away (for example, a file is removed or a user terminates).
- You can set permissions on shell scripts in the same way you set permissions for other files. For example, you can create scripts that are shared by all members of your administrative group (use case to create menus based upon LOGNAME).

Chapter 3: Programming in Linux

In This Chapter

- **Figuring out programming**
- **Exploring the software-development tools in Linux**
- **Compiling and linking programs with GCC**
- **Using make**
- **Debugging programs with gdb**
- **Understanding the implications of GNU, GPL, and LGPL**

Linux comes loaded with all the tools you need to develop software. (All you have to do is install them.) In particular, it has all the GNU software-development tools, such as GCC (C and C++ compiler), GNU `make`, and the GNU debugger. Whereas the previous two chapters look at some simple tools and shell scripts, this chapter introduces you to programming, describes the software-development tools, and shows you how to use them. Although I provide examples in the C and C++ programming languages, the focus is not on showing you how to program in those languages but on showing you how to use various software-development tools (such as compilers, `make`, and debugger).

The chapter concludes with a brief explanation of how the Free Software Foundation's GNU General Public License (GPL) may affect any plans you might have to develop Linux software. You need to know about the GPL because you use GNU tools and GNU libraries to develop software in Linux.

An Overview of Programming

If you've written computer programs in any programming language, even the shell scripts from the previous two chapters, you can start writing programs on your Linux system quickly. If you've never written a computer program, however, you need two basic resources before you begin to write code: a look at the basics of programming and a quick review of computers and their major parts. This section offers an overview of computer programming — just enough to get you going.

At its simplest, a *computer program* is a sequence of instructions for performing a specific task, such as adding two numbers or searching for some text in a file. Consequently, computer programming involves *creating* that list of instructions, telling the computer how to complete a specific task. The exact instructions depend on the programming language that you use. For most programming languages, you have to go through the following steps to create a computer program:

1. **Use a text editor to type the sequence of commands from the programming language.**

   This sequence of commands accomplishes your task. This human-readable version of the program is called the *source file* or *source code.* You can create the source file with any application (such as a word processor) that can save a document in plain-text form.

   Always save your source code as plain text. (The filename depends on the type of programming language.) Word processors can sometimes put extra instructions in their documents that tell the computer to display the text in a particular font or other format. Saving the file as plain text deletes any and all such extra instructions. Trust me, your program is much better off without such stuff.

2. **Use a *compiler* program to convert that text file — the source code — from human-readable form into machine-readable *object code.***

   Typically, this step also combines several object code files into a single machine-readable computer program, something that the computer can run.

3. **Use a special program called a *debugger* to track down any errors and find which lines in the source file might have caused the errors.**

4. **Go back to Step 1 and use the text editor to correct the errors, and repeat the rest of the steps.**

These steps are referred to as the *edit-compile-debug cycle* of programming because most programmers have to repeat this sequence several times before a program works correctly.

In addition to knowing the basic programming steps, you also need to be familiar with the following terms and concepts:

- *Variables* are used to store different types of data. You can think of each variable as being a placeholder for data — kind of like a mailbox, with a name and room to store data. The content of the variable is its *value*.
- *Expressions* combine variables by using operators. One expression may add several variables; another may extract a part of a *string* (series of sequential characters).
- *Statements* perform some action, such as assigning a value to a variable or printing a string.
- *Flow-control statements* allow statements to execute in various orders, depending on the value of some expression. Typically, flow-control statements include `for`, `do-while`, `while`, and `if-then-else` statements.

- *Functions* (also called *subroutines* or *routines*) allow you to group several statements and give the group a name. You can use functions to execute the same set of statements over and over by invoking the function that represents those statements. Typically, a programming language provides many predefined functions to perform tasks, such as opening (and reading from) a file.

Exploring the Software-Development Tools in Linux

Linux includes the following traditional Unix software-development tools:

- **Text editors** such as `vi` and `emacs` for editing the source code. (To find out more about `vi`, see Book II, Chapter 6.)
- A **C compiler** for compiling and linking programs written in C — the programming language of choice for writing Unix applications (though nowadays, many programmers are turning to C++ and Java). Linux includes the GNU C and C++ compilers. Originally the GNU C compiler was known as GCC — which now stands for *GNU Compiler Collection.* (See a description at `http://gcc.gnu.org`.)
- The **GNU `make` utility** for automating the software *build process* — the process of combining object modules into an executable or a library. (The operating system can load and run an *executable*; a *library* is a collection of binary code that can be used by executables.)
- A **debugger** for debugging programs. Linux includes the GNU debugger `gdb`.
- A **version control system** to keep track of various revisions of a source file. Linux comes with RCS (Revision Control System) and CVS (Concurrent Versions System). Nowadays, most open source projects use CVS as their version control system, but a recent version control system called Subversion is being developed as a replacement for CVS.

DISTRIBUTION SPECIFIC

You can install these software-development tools in any Linux distribution:

- **Xandros:** Usually the tools are installed by default.
- **Fedora:** Select the Development Tools package during installation.
- **Debian:** Type `apt-get install gcc` and then `apt-get install libc6-dev` in a terminal window.
- **SUSE:** Choose Main Menu➪System➪YaST, click Software on the left side of the window, and then click Install and Remove Software. Type `gcc` in the search field in YaST, select the relevant packages from the search results, and click Accept to install. If you find any missing packages, you can install them in a similar manner.

The next few sections briefly describe how to use these software-development tools to write applications for Linux.

GNU C and C++ compilers

The most important software-development tool in Linux is GCC — the GNU C and C++ compiler. In fact, GCC can compile three languages: C, C++, and Objective-C (a language that adds object-oriented programming capabilities to C). You use the same gcc command to compile and link both C and C++ source files. The GCC compiler supports ANSI-standard C, making it easy to port any ANSI C program to Linux. In addition, if you've ever used a C compiler on other Unix systems, you should feel right at home with GCC.

Using GCC

Use the gcc command to invoke GCC. By default, when you use the gcc command on a source file, GCC preprocesses, compiles, and links to create an executable file. However, you can use GCC options to stop this process at an intermediate stage. For example, you might invoke gcc by using the -c option to compile a source file and to generate an object file, but not to perform the link step.

Using GCC to compile and link a few C source files is easy. Suppose you want to compile and link a simple program made up of two source files. To accomplish this task, use the following program source code; the task that is stored in the file area.c computes the area of a circle whose radius is specified at the command line:

```
#include <stdio.h>
#include <stdlib.h>
/* Function prototype */
double area_of_circle(double r);
int main(int argc, char **argv)
{
if(argc < 2)
{
printf("Usage: %s radius\n", argv[0]);
exit(1);
}
else
{
double radius = atof(argv[1]);
double area = area_of_circle(radius);
printf("Area of circle with radius %f = %f\n",
radius, area);
}
return 0;
}
```

You need another file that actually computes the area of a circle. Here's the listing for the circle.c file, which defines a function that computes the area of a circle:

```
#include <math.h>
#define SQUARE(x) ((x)*(x))
double area_of_circle(double r)
```

```
{
return 4.0 * M_PI * SQUARE(r);
}
```

For such a simple program, of course, we could place everything in a single file, but this example was contrived a bit to show you how to handle multiple files.

To compile these two files and to create an executable file named `area`, use this command:

```
gcc -o area area.c circle.c
```

This invocation of GCC uses the `-o` option to specify the name of the executable file. (If you don't specify the name of an output file with the `-o` option, GCC saves the executable code in a file named `a.out`.)

If you have too many source files to compile and link, you can compile the files individually and generate *object files* (that have the `.o` extension). That way, when you change a source file, you need to compile only that file — you just link the compiled file to all the object files. The following commands show how to separate the compile and link steps for the sample program:

```
gcc -c area.c
gcc -c circle.c
gcc -o area area.o circle.o
```

The first two commands run `gcc` with the `-c` option compiling the source files. The third `gcc` command links the object files into an executable named `area`.

Compiling C++ programs

GNU CC is a combined C and C++ compiler, so the `gcc` command also can compile C++ source files. GCC uses the file extension to determine whether a file is C or C++. C files have a lowercase `.c` extension, whereas C++ files end with `.C` or `.cpp`.

REMEMBER

Although the `gcc` command can compile a C++ file, that command doesn't automatically link with various class libraries that C++ programs typically require. Compiling and linking a C++ program by using the `g++` command is easy because it runs `gcc` with appropriate options.

Suppose you want to compile the following simple C++ program stored in a file named `hello.C`. (Using an uppercase `C` extension for C++ source files is customary.)

```
#include <iostream>
int main()
{
using namespace std;
cout << "Hello from Linux!" << endl;
}
```

To compile and link this program into an executable program named `hello`, use this command:

```
g++ -o hello hello.C
```

The command creates the `hello` executable, which you can run as follows:

```
./hello
```

The program displays the following output:

```
Hello from Linux!
```

A host of GCC options controls various aspects of compiling C and C++ programs.

Exploring GCC options

Here's the basic syntax of the `gcc` command:

```
gcc options filenames
```

Each option starts with a hyphen (`-`) and usually has a long name, such as `-funsigned-char` or `-finline-functions`. Many commonly used options are short, however, such as `-c`, to compile only, and `-g`, to generate debugging information (needed to debug the program by using the GNU debugger, `gdb`).

You can view a summary of all GCC options by typing the following command in a terminal window:

```
man gcc
```

Then you can browse through the commonly used GCC options. Usually, you don't have to provide GCC options explicitly because the default settings are fine for most applications. Table 3-1 lists some of the GCC options you may use.

Table 3-1 Common GCC Options

Option	*Meaning*
`-ansi`	Supports only ANSI-standard C syntax. (This option disables some GNU C-specific features, such as the `__asm__` and `__typeof__` keywords.) When used with g++, supports only ISO-standard C++.
`-c`	Compiles and generates only the object file.
`-D`*`MACRO`*	Defines the macro with the string `"1"` as its value.

Option	*Meaning*
-D*MACRO*=*DEFN*	Defines the macro as DEFN, where DEFN is some text string.
-E	Runs only the C preprocessor.
-fallow-single-precision	Performs all math operations in single precision.
-fpcc-struct-return	Returns all struct and union values in memory, rather than in registers. (Returning values this way is less efficient, but at least it's compatible with other compilers.)
-fPIC	Generates position-independent code (PIC) suitable for use in a shared library.
-freg-struct-return	When possible, returns struct and union values registers.
-g	Generates debugging information. (The GNU debugger can use this information.)
-I *DIRECTORY*	Searches the specified directory for files that you include by using the #include preprocessor directive.
-L *DIRECTORY*	Searches the specified directory for libraries.
-l *LIBRARY*	Searches the specified library when linking.
-mcpu=*cputype*	Optimizes code for a specific processor. (cputype can take many different values — some common ones are i386, i486, i586, i686, pentium, pentiumpro, pentium2, pentium3, pentium4.)
-o *FILE*	Generates the specified output file (used to designate the name of an executable file).
-00 (two zeros)	Does not optimize.
-O or -O1 (letter O)	Optimizes the generated code.
-O2 (letter O)	Optimizes even more.
-O3 (letter O)	Performs optimizations beyond those done for -O2.
-Os (letter O)	Optimizes for size (to reduce the total amount of code).
-pedantic	Generates errors if any non-ANSI-standard extensions are used.
-pg	Adds extra code to the program so that, when run, this program generates information that the gprof program can use to display timing details for various parts of the program.

(continued)

Table 3-1 *(continued)*

Option	*Meaning*
`-shared`	Generates a shared object file (typically used to create a shared library).
`-U`*`MACRO`*	Undefines the specified macros.
`-v`	Displays the GCC version number.
`-w`	Doesn't generate warning messages.
`-W1,` *`OPTION`*	Passes the *`OPTION`* string (containing multiple comma-separated options) to the linker. To create a shared library named `libXXX.so.1`, for example, use the following flag: `-W1,-soname,libXXX.so.1`.

The GNU make utility

When an application is made up of more than a few source files, compiling and linking the files by manually typing the `gcc` command can get tiresome. Also, you don't want to compile every file whenever you change something in a single source file. These situations are where the GNU `make` utility comes to your rescue.

The `make` utility works by reading and interpreting a *`makefile`* — a text file that describes which files are required to build a particular program as well as how to compile and link the files to build the program. Whenever you change one or more files, `make` determines which files to recompile — and it issues the appropriate commands for compiling those files and rebuilding the program.

Makefile names

By default, GNU `make` looks for a `makefile` that has one of the following names, in the order shown:

- `GNUmakefile`
- `makefile`
- `Makefile`

In Unix systems, using `Makefile` as the name of the `makefile` is customary because it appears near the beginning of directory listings, where uppercase names appear before lowercase names.

When you download software from the Internet, you usually find a `Makefile`, together with the source files. To build the software, you only have to type **make** at the shell prompt and `make` takes care of all the steps necessary to build the software.

If your makefile doesn't have a standard name (such as Makefile), you have to use the -f option with make to specify the makefile name. If your makefile is called myprogram.mak, for example, you have to run make using the following command line:

```
make -f myprogram.mak
```

The makefile

For a program made up of several source and header files, the makefile specifies the following:

- The items that make creates — usually the object files and the executable. Using the term *target* to refer to any item that make has to create is common.
- The files or other actions required to create the target.
- Which commands to execute to create each target.

Suppose that you have a C++ source file named form.C that contains the following preprocessor directive:

```
#include "form.h" // Include header file
```

The object file form.o clearly depends on the source file form.C and the header file form.h. In addition to these dependencies, you must specify how make converts the form.C file to the object file form.o. Suppose that you want make to invoke g++ (because the source file is in C++) with these options:

- -c (compile only)
- -g (generate debugging information)
- -O2 (optimize some)

In the makefile, you can express these options with the following rule:

```
# This a comment in the makefile
# The following lines indicate how form.o depends
# on form.C and form.h and how to create form.o.
form.o: form.C form.h
g++ -c -g -O2 form.C
```

In this example, the first noncomment line shows form.o as the target and form.C and form.h as the dependent files.

WARNING!

The line following the dependency indicates how to build the target from its dependents. This line must start with a tab. Otherwise the make command exits with an error message, and you're left scratching your head because when you look at the makefile in a text editor, you can't tell the difference between a tab and a space. Now that you know the secret, the fix is to replace the space at the beginning of the offending line with a single tab.

The benefit of using make is that it prevents unnecessary compilations. After all, you can run g++ (or gcc) from a shell script to compile and link all the files that make up your application, but the shell script compiles everything, even if the compilations are unnecessary. GNU make, on the other hand, builds a target only if one or more of its dependents have changed since the last time the target was built. make verifies this change by examining the time of the last modification of the target and the dependents.

make treats the target as the name of a goal to be achieved; the target doesn't have to be a file. You can have a rule such as this one:

```
clean:
rm -f *.o
```

This rule specifies an abstract target named clean that doesn't depend on anything. This dependency statement says that to create the target clean, GNU make invokes the command rm -f *.o, which deletes all files that have the .o extension (namely, the object files). Thus, the effect of creating the target named clean is to delete the object files.

Variables (or macros)

In addition to the basic capability of building targets from dependents, GNU make includes many features that make it easy for you to express the dependencies and rules for building a target from its dependents. If you need to compile a large number of C++ files by using GCC with the same options, for example, typing the options for each file is tedious. You can avoid this repetitive task by defining a variable or macro in make as follows:

```
# Define macros for name of compiler
CXX= g++
# Define a macro for the GCC flags
CXXFLAGS= -O2 -g -mcpu=i686
# A rule for building an object file
form.o: form.C form.h
$(CXX) -c $(CXXFLAGS) form.C
```

In this example, CXX and CXXFLAGS are make variables. (GNU make prefers to call them *variables,* but most Unix make utilities call them *macros.*)

To use a variable anywhere in the makefile, start with a dollar sign ($) followed by the variable within parentheses. GNU make replaces all occurrences of a variable with its definition; thus, it replaces all occurrences of $(CXXFLAGS) with the string -O2 -g -mcpu=i686.

GNU make has several predefined variables that have special meanings. Table 3-2 lists these variables. In addition to the variables listed in Table 3-2, GNU make considers all environment variables (such as PATH and HOME) to be predefined variables as well.

Table 3-2 Some Predefined Variables in GNU make

Variable	*Meaning*
`$%`	Member name for targets that are archives. If the target is `libDisp.a(image.o)`, for example, `$%` is `image.o`.
`$*`	Name of the target file without the extension.
`$+`	Names of all dependent files with duplicate dependencies, listed in their order of occurrence.
`$<`	The name of the first dependent file.
`$?`	Names of all dependent files (with spaces between the names) that are newer than the target.
`$@`	Complete name of the target. If the target is `libDisp.a image.o)`, for example, `$@` is `libDisp.a`.
`$^`	Names of all dependent files, with spaces between the names. Duplicates are removed from the dependent filenames.
`AR`	Name of the archive-maintaining program (default value: `ar`).
`ARFLAGS`	Flags for the archive-maintaining program (default value: `rv`).
`AS`	Name of the assembler program that converts the assembly language to object code (default value: `as`).
`ASFLAGS`	Flags for the assembler.
`CC`	Name of the C compiler (default value: `cc`).
`CFLAGS`	Flags that are passed to the C compiler.
`CO`	Name of the program that extracts a file from RCS (default value: `co`).
`COFLAGS`	Flags for the RCS `co` program.
`CPP`	Name of the C preprocessor (default value: `$(CC) -E`).
`CPPFLAGS`	Flags for the C preprocessor.
`CXX`	Name of the C++ compiler (default value: `g++`).
`CXXFLAGS`	Flags that are passed to the C++ compiler.
`FC`	Name of the FORTRAN compiler (default value: `f77`).
`FFLAGS`	Flags for the FORTRAN compiler.
`LDFLAGS`	Flags for the compiler when it's supposed to invoke the linker `ld`.
`RM`	Name of the command to delete a file (Default value: `rm -f`).

A sample makefile

You can write a `makefile` easily if you use the predefined variables of GNU `make` and its built-in rules. Consider, for example, a `makefile` that creates the executable `xdraw` from three C source files (`xdraw.c`, `xviewobj.c`, and `shapes.c`) and two header files (`xdraw.h` and `shapes.h`). Assume that each source file includes one of the header files. Given these facts, here is what a sample makefile may look like:

```
#########################################################
# Sample makefile
# Comments start with '#'
#
#########################################################
# Use standard variables to define compile and link flags
CFLAGS= -g -O2
# Define the target "all"
all: xdraw
OBJS=xdraw.o xviewobj.o shapes.o
xdraw: $(OBJS)
# Object files
xdraw.o: Makefile xdraw.c xdraw.h
xviewobj.o: Makefile xviewobj.c xdraw.h
shapes.o: Makefile shapes.c shapes.h
```

This makefile relies on GNU make's implicit rules. The conversion of .c files to .o files uses the built-in rule. Defining the variable CFLAGS passes the flags to the C compiler.

The target named all is defined as the first target for a reason — if you run GNU make without specifying any targets in the command line (see the make syntax described in the following section), the command builds the first target it finds in the makefile. By defining the first target all as xdraw, you can ensure that make builds this executable file, even if you don't explicitly specify it as a target. Unix programmers traditionally use all as the name of the first target, but the target's name is immaterial; what matters is that it's the first target in the makefile.

How to run make

Typically you run make by simply typing the following command at the shell prompt:

```
make
```

When run this way, GNU make looks for a file named GNUmakefile, makefile, or Makefile — in that order. If make finds one of these makefiles, it builds the first target specified in that makefile. However, if make doesn't find an appropriate makefile, it displays the following error message and exits:

```
make: *** No targets specified and no makefile found. Stop.
```

If your makefile happens to have a different name from the default names, you have to use the -f option to specify the makefile. The syntax of the make command with this option is

```
make -f filename
```

where *filename* is the name of the makefile.

Even when you have a makefile with a default name such as Makefile, you may want to build a specific target out of several targets defined in the makefile. In that case, you have to use the following syntax when you run make:

```
make target
```

For example, if the makefile contains the target named clean, you can build that target with this command:

```
make clean
```

Another special syntax overrides the value of a make variable. For example, GNU make uses the CFLAGS variable to hold the flags used when compiling C files. You can override the value of this variable when you invoke make. Here's an example of how you can define CFLAGS as the option -g -O2:

```
make CFLAGS="-g -O2"
```

In addition to these options, GNU make accepts several other command-line options. Table 3-3 lists the GNU make options.

Table 3-3 Options for GNU make

Option	*Meaning*
-b	Ignores the variable given but accepts that variable for compatibility with other versions of make.
-C *DIR*	Changes to the specified directory before reading the makefile.
-d	Prints debugging information.
-e	Allows environment variables to override definitions of similarly named variables in the makefile.
-f *FILE*	Reads FILE as the makefile.
-h	Displays the list of make options.
-i	Ignores all errors in commands executed when building a target.
-I *DIR*	Searches the specified directory for included makefiles. (The capability to include a file in a makefile is unique to GNU make.)
-j *NUM*	Specifies the number of commands that make can run simultaneously.
-k	Continues to build unrelated targets, even if an error occurs when building one of the targets.
-l *LOAD*	Doesn't start a new job if load average is at least *LOAD* (a floating-point number).

(continued)

Table 3-3 *(continued)*

Option	*Meaning*
`-m`	Ignores the variable given but accepts that variable for compatibility with other versions of `make`.
`-n`	Prints the commands to execute but does not execute them.
`-o` *`FILE`*	Does not rebuild the file named *`FILE`*, even if it is older than its dependents.
`-p`	Displays the `make` database of variables and implicit rules.
`-q`	Does not run anything, but returns 0 (zero) if all targets are up to date, 1 if anything needs updating, or 2 if an error occurs.
`-r`	Gets rid of all built-in rules.
`-R`	Gets rid of all built-in variables and rules.
`-s`	Works silently (without displaying the commands as they execute).
`-t`	Changes the timestamp of the files.
`-v`	Displays the version number of `make` and a copyright notice.
`-w`	Displays the name of the working directory before and after processing the `makefile`.
`-W FILE`	Assumes that the specified file has been modified (used with `-n` to see what happens if you modify that file).

The GNU debugger

Although `make` automates the process of building a program, that part of programming is the least of your worries when a program doesn't work correctly or when a program suddenly quits with an error message. You need a debugger to find the cause of program errors. Linux includes `gdb` — the versatile GNU debugger with a command-line interface.

Like any debugger, `gdb` lets you perform typical debugging tasks, such as the following:

- Set a breakpoint so that the program stops at a specified line.
- Watch the values of variables in the program.
- Step through the program one line at a time.
- Change variables in an attempt to correct errors.

The `gdb` debugger can debug C and C++ programs.

Preparing to debug a program

If you want to debug a program by using gdb, you have to ensure that the compiler generates and places debugging information in the executable. The debugging information contains the names of variables in your program and the mapping of addresses in the executable file to lines of code in the source file. gdb needs this information to perform its functions, such as stopping after executing a specified line of source code.

TIP

To make sure that the executable is properly prepared for debugging, use the -g option with GCC. You can do this task by defining the variable CFLAGS in the makefile as

```
CFLAGS= -g
```

Running gdb

The most common way to debug a program is to run gdb by using the following command:

```
gdb progname
```

progname is the name of the program's executable file. After progname runs, gdb displays the following message and prompts you for a command:

```
GNU gdb (GDB) 7.5.91.20130417-cvs-ubuntu
Copyright (c) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
    <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change it and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
    and "show warranty" for details.
This GDB was configured as "i686--linux-gnu".
For bug reporting instructions, please see:
http://www.gnu.org/software/gdb/bugs/.
(gdb)
```

You can type gdb commands at the (gdb) prompt. One useful command, help, displays a list of commands, as the next listing shows:

```
(gdb) help
List of classes of commands:
aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands
```

```
Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
(gdb)
```

To quit `gdb`, type `q` and then press Enter.

`gdb` has a large number of commands, but you need only a few to find the cause of an error quickly. Table 3-4 lists the commonly used `gdb` commands.

Table 3-4 Common gdb Commands

This Command	*Does the Following*
`break` *NUM*	Sets a breakpoint at the specified line number, *NUM*. (The debugger stops at breakpoints.)
`bt`	Displays a trace of all stack frames. (This command shows you the sequence of function calls so far.)
`clear` *FILENAME: NUM*	Deletes the breakpoint at a specific line number, *NUM*, in the source file *FILENAME*. For example, `clear xdraw.c:8` clears the breakpoint at line 8 of file `xdraw.c`.
`continue`	Continues running the program being debugged. (Use this command after the program stops due to a signal or breakpoint.)
`display` *EXPR*	Displays the value of an expression, *EXPR* (consisting of variables defined in the program) each time the program stops.
`file` *FILE*	Loads the specified executable file, *FILE,* for debugging.
`help` *NAME*	Displays help on the command named *NAME*.
`info break`	Displays a list of current breakpoints, including information on how many times each breakpoint is reached.
`info files`	Displays detailed information about the file being debugged.
`info func`	Displays all function names.
`info local`	Displays information about local variables of the current function.
`info prog`	Displays the execution status of the program being debugged.
`info var`	Displays all global and static variable names.

This Command	*Does the Following*
`kill`	Ends the program you're debugging.
`list`	Lists a section of the source code.
`make`	Runs the `make` utility to rebuild the executable without leaving `gdb`.
`next`	Advances one line of source code in the current function without stepping into other functions.
`print` *`EXPR`*	Shows the value of the expression *`EXPR`*.
`quit`	Quits `gdb`.
`run`	Starts running the currently loaded executable.
`set variable` *`VAR=VALUE`*	Sets the value of the variable *`VAR`* to *`VALUE`*.
`shell` *`CMD`*	Executes the Unix command *`CMD`*, without leaving `gdb`.
`step`	Advances one line in the current function, stepping into other functions, if any.
`watch` *`VAR`*	Shows the value of the variable named *`VAR`* whenever the value changes.
`where`	Displays the call sequence. Use this command to locate where your program died.
`x/`*`F ADDR`*	Examines the contents of the memory location at address *`ADDR`* in the format specified by the letter *`F`*, which can be `o` (octal), `x` (hex), `d` (decimal), `u` (unsigned decimal), `t` (binary), `f` (float), `a` (address), `i` (instruction), `c` (char), or `s` (string). You can append a letter indicating the size of data type to the format letter. Size letters are `b` (byte), `h` (halfword, 2 bytes), `w` (word, 4 bytes), and `g` (giant, 8 bytes). Typically, *`ADDR`* is the name of a variable or pointer.

Finding bugs by using gdb

To understand how you can find bugs by using `gdb`, you need to see an example. The procedure is easiest to show with a simple example, so the following, `dbgtst.c`, is a contrived program that contains a typical bug.

```
#include <stdio.h>
static char buf[256];
void read_input(char *s);
int main(void)
{
char *input = NULL; /* Just a pointer, no storage for string */
read_input(input);
/* Process command. */
printf("You typed: %s\n", input);
/* . . ._*/
return 0;
}
void read_input(char *s)
{
printf("Command: ");
gets(s);
}
```

This program's main function calls the `read_input` function to get a line of input from the user. The `read_input` function expects a character array in which it returns what the user types. In this example, however, `main` calls `read_input` with an uninitialized pointer — that's the bug in this simple program.

Build the program by using `gcc` with the `-g` option:

```
gcc -g -o dbgtst dbgtst.c
```

Ignore the warning message about the `gets` function being dangerous; I'm trying to use the shortcoming of that function to show how you can use `gdb` to track down errors.

To see the problem with this program, run it and type `test` at the `Command:` prompt:

```
./dbgtst
Command: test
Segmentation fault
```

The program dies after displaying the `Segmentation fault` message. For such a small program as this one, you can probably find the cause by examining the source code. In a real-world application, however, you may not immediately know what causes the error. That's when you have to use `gdb` to find the cause of the problem.

To use `gdb` to locate a bug, follow these steps:

1. **Load the program under `gdb`.**

   For example, type `gdb dbgtst` to load a program named `dbgtst` in `gdb`.

2. **Start executing the program under `gdb` by typing the `run` command. When the program prompts for input, type some input text.**

   The program fails as it did previously. Here's what happens with the `dbgtst` program:

   ```
   (gdb) run
   Starting program: /home/edulaney/swdev/dbgtst
   Command: test
   Program received signal SIGSEGV, Segmentation fault.
   0x400802b6 in gets () from /lib/tls/libc.so.6
   (gdb)
   ```

3. **Use the `where` command to determine where the program died.**

   For the `dbgtst` program, this command yields this output:

   ```
   (gdb) where
   #0 0x400802b6 in gets () from /lib/tls/libc.so.6
   #1 0x08048474 in read_input (s=0x0) at dbgtst.c:16
   #2 0x08048436 in main () at dbgtst.c:7
   (gdb)
   ```

   The output shows the sequence of function calls. Function call `#0` — the most recent one — is to the `gets` C library function. The `gets` call originates in the `read_input` function (at line 16 of the file `dbgtst.c`), which in turn is called from the `main` function at line 7 of the `dbgtst.c` file.

4. **Use the `list` command to inspect the lines of suspect source code.**

   In `dbgtst`, you may start with line 16 of `dbgtst.c` file, as follows:

   ```
   (gdb) list dbgtst.c:16
   11 return 0;
   12 }
   13 void read_input(char *s)
   14 {
   15 printf("Command: ");
   16 gets(s);
   17 }
   18
   (gdb)
   ```

   After looking at this listing, you can tell that the problem may be the way `read_input` is called. Then you list the lines around line 7 in `dbgtst.c` (where the `read_input` call originates):

   ```
   (gdb) list dbgtst.c:7
   2 static char buf[256];
   3 void read_input(char *s);
   4 int main(void)
   5 {
   6 char *input = NULL; /* Just a pointer, no storage for string */
   7 read_input(input);
   8 /* Process command. */
   9 printf("You typed: %s\n", input);
   10 /* . . . */
   11 return 0;
   (gdb)
   ```

   At this point, you can narrow the problem to the variable named `input`. That variable is an array, not a `NULL` (which means zero) pointer.

Fixing bugs in gdb

Sometimes you can fix a bug directly in `gdb`. For the example program in the preceding section, you can try this fix immediately after the program dies after displaying an error message. An extra buffer named `buf` is defined in the `dbgtst` program, as follows:

```
static char buf[256];
```

We can fix the problem of the uninitialized pointer by setting the variable input to `buf`. The following session with `gdb` corrects the problem of the uninitialized pointer. (This example picks up immediately after the program runs and dies, due to the segmentation fault.)

```
(gdb) file dbgtst
A program is being debugged already. Kill it? (y or n) y
Load new symbol table from "/home/edulaney/sw/dbgtst"? (y or n) y
Reading symbols from /home/edulaney/sw/dbgtst . . . done.
(gdb) list
1 #include <stdio.h>
2 static char buf[256];
3 void read_input(char *s);
4 int main(void)
5 {
6 char *input = NULL; /* Just a pointer, no storage for string */
7 read_input(input);
8 /* Process command. */
9 printf("You typed: %s\n", input);
10 /* . . . */
(gdb) break 7
Breakpoint 2 at 0x804842b: file dbgtst.c, line 7.
(gdb) run
Starting program: /home/edulaney/sw/dbgtst
Breakpoint 1, main () at dbgtst.c:7
7 read_input(input);
(gdb) set var input=buf
(gdb) cont
Continuing.
Command: test
You typed: test
Program exited normally.
(gdb)q
```

As the preceding listing shows, if the program is stopped just before `read_input` is called and the variable named input is set to `buf` (which is a valid array of characters), the rest of the program runs fine.

After finding a fix that works in `gdb`, you can make the necessary changes to the source files and make the fix permanent.

Understanding the Implications of GNU Licenses

You have to pay a price for the bounty of Linux. To protect its developers and users, Linux is distributed under the GNU GPL (General Public License), which stipulates the distribution of the source code.

The GPL doesn't mean, however, that you can't write commercial software for Linux that you want to distribute (either for free or for a price) in binary form only. You can follow all the rules and still sell your Linux applications in binary form.

When writing applications for Linux, be aware of two licenses:

- The **GNU General Public License** (GPL), which governs many Linux programs, including the Linux kernel and GCC
- The **GNU Library General Public License** (LGPL), which covers many Linux libraries

The following sections provide an overview of these licenses and some suggestions on how to meet their requirements. Don't take anything in this book as legal advice. Instead, you should read the full text for these licenses in the text files on your Linux system, and then show these licenses to your legal counsel for a full interpretation and an assessment of their applicability to your business.

The GNU General Public License

The text of the GNU General Public License (GPL) is in a file named `COPYING` in various directories in your Linux system. For example, type the following command to find a copy of that file in your Linux system for various items:

```
find /usr -name "COPYING" -print
```

After you find the file, you can change to that directory and type `more COPYING` to read the GPL. These are examples of the license accompanying code, and you can find other examples at `http://www.gnu.org/copyleft/gpl.html`.

The GPL has nothing to do with whether you charge for the software or distribute it for free; its thrust is to keep the software free for all users. GPL requires that the software be distributed in source-code form, and stipulates that any user can copy and distribute the software in source-code form to anyone else. In addition, everyone is reminded that the software comes with absolutely no warranty.

The software that the GPL covers isn't in the public domain. Software covered by GPL is always copyrighted, and the GPL spells out the restrictions on the software's copying and distribution. From a user's point of view, of course, GPL's restrictions aren't really restrictions; the restrictions are benefits because the user is guaranteed access to the source code.

If your application uses parts of any software that the GPL covers, your application is considered a *derived work,* which means that your application is also covered by the GPL and you must distribute the source code to your application.

Although the GPL covers the Linux kernel, the GPL doesn't cover your applications that use the kernel services through system calls. Those applications are considered normal use of the kernel.

If you plan to distribute your application in binary form (as most commercial software is distributed), you must make sure that your application doesn't use any parts of any software the GPL covers. Your application may end up using parts of other software when it calls functions in a library. Most libraries, however, are covered by a different GNU license, which is described in the next section.

You have to watch out for only a few of the library and utility programs that the GPL covers. The GNU `dbm` (`gdbm`) database library is one of the prominent libraries that GPL covers. The GNU `bison` parser-generator tool is another utility that the GPL covers. If you allow `bison` to generate code, the GPL covers that code.

Other alternatives for the GNU `dbm` and GNU `bison` aren't covered by GPL. For a database library, you can use the Berkeley database library `db` in place of `gdbm`. For a parser-generator, you may use `yacc` instead of `bison`.

The GNU Library General Public License

The text of the GNU Library General Public License (LGPL) is in a file named `COPYING.LIB`. If you have the kernel source installed, a copy of `COPYING.LIB` file is in one of the source directories. To locate a copy of the `COPYING.LIB` file on your Linux system, type the following command in a terminal window:

```
find /usr -name "COPYING*" -print
```

This command lists all occurrences of `COPYING` and `COPYING.LIB` in your system. The `COPYING` file contains the GPL, whereas `COPYING.LIB` has the LGPL.

The LGPL is intended to allow use of libraries in your applications, even if you don't distribute source code for your application. The LGPL stipulates, however, that users must have access to the source code of the library you use — and that users can make use of modified versions of those libraries.

The LGPL covers most Linux libraries, including the C library (`libc.a`). Thus, when you build your application on Linux by using the GCC compiler, your application links with code from one or more libraries that the LGPL covers. If you want to distribute your application in only binary form, you need to pay attention to LGPL.

One way to meet the intent of the LGPL is to provide the object code for your application and a `makefile` that relinks your object files with any updated Linux libraries the LGPL covers.

REMEMBER

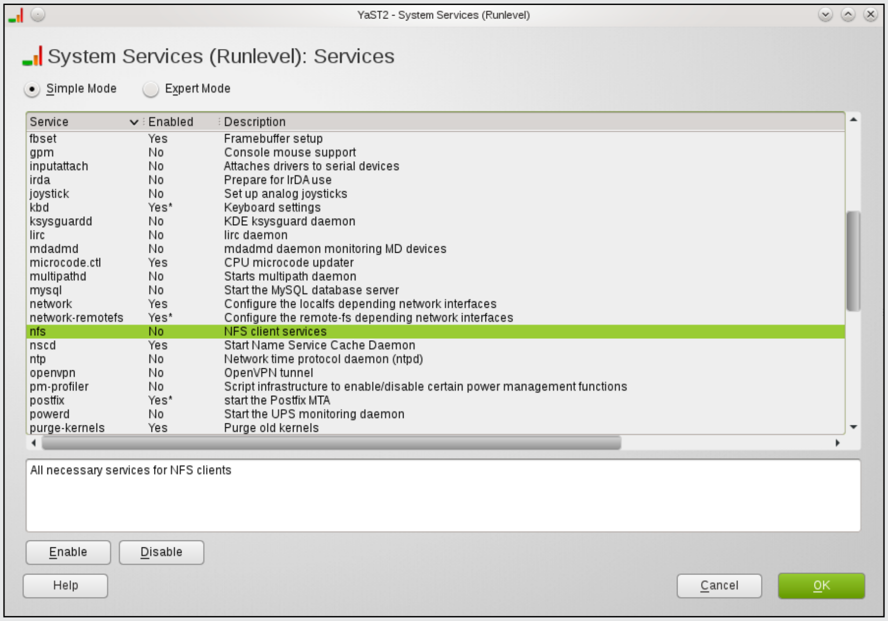
A better way to satisfy the LGPL is to use *dynamic linking,* in which your application and the library are separate entities, even though your application calls functions that reside in the library when it runs. With dynamic linking, users immediately get the benefit of any updates to the libraries without ever having to relink the application.

TIP

The newest version of the license is GPLv3 and a Quick Guide to it can be found at: `http://www.gnu.org/licenses/quick-guide-gplv3.html`.

Book VIII

Linux Certification



System Services (Runlevel): Services

● Simple Mode ○ Expert Mode

Service	Enabled	Description
fbset	Yes	Framebuffer setup
gpm	No	Console mouse support
inputattach	No	Attaches drivers to serial devices
irda	No	Prepare for IrDA use
joystick	No	Set up analog joysticks
kbd	Yes*	Keyboard settings
ksysguardd	No	KDE ksysguard daemon
lirc	No	lirc daemon
mdadmd	No	mdadmd daemon monitoring MD devices
microcode.ctl	Yes	CPU microcode updater
multipathd	No	Starts multipath daemon
mysql	No	Start the MySQL database server
network	Yes	Configure the localfs depending network interfaces
network-remotefs	Yes*	Configure the remote-fs depending network interfaces
nfs	No	NFS client services
nscd	Yes	Start Name Service Cache Daemon
ntp	No	Network time protocol daemon (ntpd)
openvpn	No	OpenVPN tunnel
pm-profiler	No	Script infrastructure to enable/disable certain power management functions
postfix	Yes*	start the Postfix MTA
powerd	No	Start the UPS monitoring daemon
purge-kernels	Yes	Purge old kernels

All necessary services for NFS clients

Visit www.dummies.com/extras/linuxaio for great Dummies content online.

Contents at a Glance

Chapter 1: Studying for the Linux Essentials Certification Exam . . . 491

Overview of Linux Essentials 491
The Linux Community and a Career in Open Source 492
Using the Command Line to Get Help 493
The Power of the Command Line 495
The Linux Operating System 496
Security and File Permissions 498

Chapter 2: Studying for the CompTIA Linux+ Powered by LPI Certification Exams . . . 501

Overview of the CompTIA Linux+ Exams 501
System Architecture 502
Linux Installation and Package Management 503
GNU and Unix Commands 505
Devices, Linux File systems, Filesystem Hierarchy Standard 507
Shells, Scripting, and Data Management 509
User Interfaces and Desktops 510
Administrative Tasks 511
Essential System Services 512
Networking Fundamentals 514
Security 515

Chapter 3: Other Linux Certifications . . . 517

Vendor-Neutral Certifications 517
Vendor-Specific Certifications 518

Chapter 1: Studying for the Linux Essentials Certification Exam

In This Chapter

- **Getting an overview of the Linux Essentials Exam**
- **Looking at the details of each domain**

Just as there are many levels of degree attainment you can achieve through an educational institution (associate's degree, bachelor's, master's, doctorate), there are multiple levels of Linux certification available. This book is written primarily for the Linux+ certification from CompTIA and the LPI LX0-101 and LX0-102 exams. As such, it covers everything you need to know to study and pass LPI's lower-level certification, known as Linux Essentials.

The following sections provide an overall look at the exam, and then explore the topics beneath each of the domains.

Overview of Linux Essentials

The Linux Essentials Certificate of Achievement was created by the Linux Professional Institute (LPI) to appeal to the academic sector. Students taking classes in Linux may not get through all the topics necessary to pass the two exams (LX0-101 and LX0-102) to gain the Level 1 certification, but LPI still wanted to recognize and authenticate their knowledge. This program was created through international collaboration with a classroom focus in mind. As of this writing, the program is live in Europe, the Middle East, and Africa, with plans to roll out to other regions in the near future.

Linux Essentials is a certificate of achievement, intended to be a much lower-level (subset) certification than the Level 1 certification. Although Linux Essentials is recommended, it's not required for any of the LPIC professional certifications.

There are five domains (called Topics) on the exam; Table 1-1 shows them, along with the weighting for each.

Table 1-1 Domains on the Linux Essentials Exam

Topic	*Weighting*
The Linux Community and a Career in Open Source	7
Finding Your Way on a Linux System	8
The Power of the Command Line	10
The Linux Operating System	8
Security and File Permissions	7

The sections that follow look at each of these topics in more detail.

The Linux Community and a Career in Open Source

Table 1-2 shows the subtopics, weight, description, and key knowledge areas for this topic.

Table 1-2 Breakout of Topic 1

Subtopic	*Weight*	*Description*	*Key Areas*
Linux Evolution and Popular Operating Systems	2	Knowledge of Linux development and major distributions	Open source philosophy; distributions; embedded systems
Major Open Source Applications	2	Awareness of major applications and their uses	Desktop applications; server applications; mobile applications; development languages; package management tools and repositories
Understanding Open Source Software and Licensing	1	Open communities and licensing Open Source Software for business	Licensing; Free Software Foundation (FSF), Open Source Initiative (OSI)
ICT Skills and Working in Linux	2	Basic Information and Communication Technology (ICT) skills and working in Linux	Desktop skills; getting to the command line; industry uses of Linux, cloud computing and virtualization

To adequately address these topics, focus on these files, terms, and utilities: Android; Apache; Audacity; Blender; BSD; C; CentOS; Creative Commons; Debian; DHCP; DNS; Firefox; FLOSS; FOSS; Gimp; GPR; ImageMagick; LibreOffice; MySQL; NFS; OpenLDAP; OpenOffice.org; Password issues; Perl; PHP; Python; Postfix; PostgreSQL; Privacy issues and tools; Samba; Shell; Terminal and Console; and Thunderbird.

In addition, focus on these topics: Use of common open source applications in presentations and projects; using a browser; privacy concerns; configuration options; searching the web; and saving content.

Here are the top ten items to know as you study for this domain:

1. Linux is the best-known example of open source software so far developed (and still in development).
2. The shell is the command interpreter that resides between the user and the operating system. While a number of shells are available, the most common today is the `bash` shell.
3. A plethora of applications and tools is available for use with the various Linux distributions. Many of these tools are also open source.
4. The Apache Software Foundation distributes open source software under the Apache license that is Free and Open Source (FOSS).
5. The Free Software Foundation (FSF) supports the free (open source) software movement and copyleft under the GNU General Public License. *Copyleft* makes it possible for modifications to be made to software while preserving the same rights in the produced derivatives.
6. The Open Source Initiative (OSI) also supports the open source software movement, as does the GNOME Foundation, Ubuntu Foundation, and many other organizations.
7. OpenOffice.org was a popular suite of open source office productivity software. LibreOffice is a fork of OpenOffice that has eclipsed it in popularity.
8. Samba makes it possible for Linux systems to share files with Windows-based systems.
9. Thunderbird is a popular mail and news client originally created by the Mozilla Foundation.
10. A number of web browsers are available for Linux. The most popular currently is Firefox.

Using the Command Line to Get Help

Table 1-3 shows the subtopics, weight, description, and key knowledge areas for this topic.

Table 1-3 Breakout of Topic 2

Subtopic	*Weight*	*Description*	*Key Areas*
Command Line Basics	2	Basics of using the Linux command line	Basic shell; Formatting commands; Working with options; Variables; Globbing; Quoting
Using the Command Line to Get Help	2	Running help commands and navigation of the various help systems	`man` files; `info` command
Using Directories and Listing Files	2	Navigation of home and system directories and listing files in various locations	Files, directories; hidden files and directories; home directory; absolute and relative paths
Creating, Moving, and Deleting Files	2	Create, move, and delete files under the home directory	Files and directories; case sensitivity; simple globbing and quoting

To adequately address these topics, focus on these files, terms, and utilities: `&&`; `.` and `..`; `/usr/share/doc`; `||`; `apropos`; `cd`; common options for `ls`; `cp`; `echo`; `history`; `Home` and `~`; `info`; `locate`; `man`; `man pages`; `mkdir`; `mv`; `PATH env` variable; recursive listings; `rm`; `rmdir`; substitutions; `touch`; `whatis`; `whereis`; and `which`.

Here are the top ten items to know as you study for this domain:

1. Regular expressions – often referred to as *globbing* – can be used with the shells available in Linux to match wildcard characters. Among the possible wildcards, the asterisk (`*`) will match any number of characters; the question mark (`?`) will match only one character.
2. Linux is a case-sensitive operating system.
3. Files can be "hidden" by preceding their names with a single period (`.`). In pathnames, however, a single period (`.`) specifies this directory and two periods (`..`) signifies the parent directory.
4. Absolute paths give the full path to a resource, while relative paths give directions from where you are currently working. An example of an absolute path would be `/tmp/eadulaney/file` whereas a relative link would be `../file`.

5. Files can be copied using cp or moved using mv. Files can be deleted with rm and directories (which are created with mkdir) can be removed with rmdir. Recursive deletion can be done with rm -r.

6. To change directories, use the cd command. When used without parameters, this will move you to your home directory. To see what directory you are presently working in, us the pwd (present working directory) command.

7. The ls command has a plethora of options to allow you to list files. The -a option will list all (including hidden) files.

8. Help is available through the manual pages (accessed with the man command) and info (which shows help files stored beneath /user/info).

9. The whatis command shows what manual pages are available for an entry while whereis shows the location of the file and all related files (including any manual pages).

10. Many standard utilities allow you to enter the name of the executable followed by "--help" to obtain help only on the syntax.

The Power of the Command Line

Table 1-4 shows the subtopics, weight, description, and key knowledge areas for this topic.

Table 1-4 Breakout of Topic 3

Subtopic	*Weight*	*Description*	*Key Areas*
Archiving Files on the Command Line	2	Archiving files in the user home directory	Files, directories; archives, compression
Searching and Extracting Data from Files	4	Search and extract data from files in the home directory	Command line pipes; I/O re-direction
Turning Commands into a Script	4	Turning repetitive commands into simple scripts	Basic text editing; basic shell scripting

To adequately address these topics, focus on the following files, terms, and utilities: /bin/sh; arguments; bash; bzip2; case; common tar options; cut; exit status; extracting individual files from archives; find; for loops; grep; gzip; head; if; less; nano; Partial POSIX Basic Regular Expressions

([^],^,$); Partial POSIX Extended Regular Expressions (+,(),|); pico; read; sort; tail; tar; test; unzip; variables; vi; wc; while; xargs; and zip.

Here are the top ten items to know as you study for this domain:

1. Standard input (stdin) is traditionally the keyboard and standard output (stdout) is traditionally the monitor. Both can be redirected, as can standard error (stderr) using the symbols >, >>, <, and |.
2. Commands can be joined on the command line by the semicolon (;) and each command will run independent of each other. You can also use the pipe (|) to send the output of one command as the input of another command.
3. The cut command can pull fields from a file and they can be combined using either paste or join. The latter offers more features than the former and can be used with conditions.
4. The wc command can count the number of lines, words, and characters in a file.
5. The grep utility (and its counterparts egrep and fgrep) can be used to find matches for strings within files.
6. The find command can be used to search the system for files/directories that meet any number of criteria. When these entities are found, the xargs command can be used to look deeper within them for other values (such as in conjunction with grep).
7. You can use the tar command (which can combine multiple files into a single archive) to do backups.
8. In addition to archiving, you can also compress files with the gzip or pack commands. To uncompress files, use uncompress, gunzip, or unpack.
9. Variables can be given at the command line and referenced as $1, $2, and so on, or entered into the executing file with the read command.
10. Logic can be added to scripts by testing conditions with test or [. Commands can execute using if-then-fi deviations or through looping (while, until, or for). You can use the exit command to leave a script or use break to leave a loop.

The Linux Operating System

Table 1-5 shows the subtopics, weight, description, and key knowledge areas for this topic.

Table 1-5 Breakout of Topic 4

Subtopic	*Weight*	*Description*	*Key Areas*
Choosing an Operating System	1	Knowledge of major operating systems and Linux distributions	Windows, Mac, Linux differences; distribution lifecycle management
Understanding computer hardware	2	Familiarity with the components that go into building desktop and server computers	Hardware
Where Data is Stored	3	Where various types of information are stored on a Linux system	Kernel; processes; `syslog`; `klog`; `dmesg`; `/lib`; `/usr/lib`; `/etc/`; `/var/log`
Your Computer on the Network	2	Querying vital networking settings and determining the basic requirements for a computer on a local area network (LAN)	Internet; network; routers; Domain Name Service; network configuration

To adequately address these topics, focus on the following files, terms, and utilities: desktop configuration (GUI versus command line); `dig`; display types; drivers; `free`; hard drives and partitions; `ifconfig`; IPv4; IPv6; libraries; maintenance cycles (beta and stable); memory addresses; motherboards; netstat; optical drives; peripherals; `ping`; power supplies; processes and process tables; processors; system messaging and logging; programs; packages and package databases; `ps`; `resolv.conf`; route; `ssh`; system configuration; and `top`.

Here are the top ten items to know as you study for this domain:

1. When run, every command spans at least one process; processes can be viewed with `ps` or `top` (which updates the display dynamically).
2. Jobs can run in the foreground or background and be moved between the two. Jobs running in the foreground can be suspended by pressing Ctrl+Z.

3. IPv4 uses 32-bit addresses, each divided into four octets. The first octet identifies the class of address (A, B, C). The address can be public or private.
4. The `ifconfig` utility can be used to see the current IP configuration of the network cards.
5. The `ping` utility is an all-purpose tool for testing connectivity. It sends echo messages to a specified host to see whether that host can be reached. You can use `ping` with the loopback address (127.0.0.1) to test internal configuration.
6. The `route` utility displays the routing table and allows you to configure it.
7. The `netstat` utility shows the current status of ports – those that are open, listening, and so on.
8. The system log is `/var/log/messages` and this is where the majority of events are written to by the system log daemon (`syslogd`). Messages routed there can be viewed with the `dmesg` command.
9. The `logrotate` command can be used to automatically archive log files and perform maintenance as configured in `/etc/syslog.conf`.
10. You can manually write entries to log files using the `logger` command.

Security and File Permissions

Table 1-6 shows the subtopics, weight, description, and key knowledge areas for this topic.

Table 1-6 Breakout of Topic 5

Subtopic	*Weight*	*Description*	*Key Areas*
Basic Security and Identifying User Types	2	Various types of users on a Linux system	Root and standard users; system users
Creating Users and Groups	2	Creating users and groups on a Linux system	User and group commands; user IDs
Managing Files Permissions and Ownership	2	Understanding and manipulating file permissions and ownership settings	File/directory permissions and owners
Special Directories and Files	1	Special directories and files on a Linux system, including special permissions	System files; libraries; symbolic links

To adequately address these topics, focus on the following files, terms, and utilities: `/etc`; `/etc/group`; `/etc/passwd`; `/etc/shadow`; `/tmp`; `/var`; `/var/tmp`; `chgrp`; `chmod`; `chown`; `groupadd`; `groupdel`; `groupmod`; hard links; `id`; `last`; `ls -d`; `ls -l`; `ls -s`; `passwd`; `setgid`; `setuid`; sticky bit; `su`; `sudo`; `useradd`; `userdel`; `usermod`; `w`; and `who`.

Here are the top ten items to know as you study for this domain:

1. File and directory permissions can be changed with the `chmod` command (which accepts numeric and symbolic values).
2. Adding 1000 to standard permissions turns on the "sticky bit"; adding 2000 turns on the `SGID` permission. Adding 4000 turns on the `SUID` permission.
3. Links are created with the `ln` command. A "hard" link is nothing more than an alias to a file (sharing the same inode). The `ln -s` command creates a symbolic link that is an actual file with its own inode. The symbolic link contains a pointer to the original file and can span across file systems; the hard link cannot.
4. User accounts can be added by manually editing the configuration files or by using the `useradd` command; they can be removed with `userdel`.
5. The `groupadd` utility can be used to create groups and `groupdel` can be used to remove groups. Groups can be modified with `groupmod` and users can change from one group to another with the `newgrp` command.
6. Passwords are changed with the `passwd` command. Older systems stored passwords in `/etc/passwd`; now passwords are stored in `/etc/shadow`, where they are more secure.
7. To see who logged on most recently and may currently still be on the network, you can use the `last` command.
8. The `su` command allows you to become another user (returning with `exit`). If no other username is specified, then the root user is implied, hence the su for superuser.
9. Use `sudo` instead of `su` when you want to run a command as another user (usually `root`) without becoming that user.
10. The `who` command shows who is logged on; the `w` command shows information combining `who` with `uptime`.

Chapter 2: Studying for the CompTIA Linux+ Powered by LPI Certification Exams

In This Chapter

- **Getting an overview of the CompTIA Linux+ Exams**
- **Looking into each domain in depth**

The previous chapter examines the Linux Essentials exam – LPI's lower-level certification. That certification should be viewed as a steppingstone to a higher-level exam. The Linux+ certification exam from CompTIA – consisting of the LPI LX0-101 and LX0-102 exams – is an ideal example of such a high-level test.

In the following sections, we provide an overview of the two exams and then explore the topics beneath each of the domains.

Overview of the CompTIA Linux+ Exams

The official name of the certification in question is "CompTIA Linux+ Powered by LPI" and although that's a mouthful to say, it's also a meaningful addition to a résumé. The certification is awarded by CompTIA; it consists of two exams by the Linux Professional Institute (LPI): LX0-101 and LX0-102. Accordingly, at the time of taking the exams, a candidate can choose to have the test scores forwarded to LPI — and gain the Level 1 certification (LPIC-1) at the same time.

TIP

Records are separately maintained by LPI and CompTIA. If you choose not to forward your scores, you can be Linux+-certified but not LPIC-1 certified.

Each of the two exams consists of 60 questions that must be answered in 90 minutes. The passing score is 500 on a scale from 200 to 800, and it is highly recommended – but not required – that candidates have 12 months of Linux administration experience.

There are four domains on one exam and six on the other. Table 2-1 shows the domains on each, along with their prospective weighting.

Table 2-1 Domains on the Linux+ Exams

Exam	*Domain*	*Weighting*
LX0-101	101 System Architecture	14%
	102 Linux Installation and Package Management	18%
	103 GNU and Unix Commands	43%
	104 Devices, Linux Filesystems, Filesystem Hierachy Standard	25%
LX0-102	105 Shells, Scripting, and Data Management	17%
	106 User Interfaces and Desktops	8%
	107 Administrative Tasks	20%
	108 Essential System Services	17%
	109 Networking Fundamentals	23%
	110 Security	15%

The sections that follow look at each of these topics in more detail.

System Architecture

Table 2-2 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-2 Breakout of Domain 101

Subtopic	*Weight*	*Description*	*Key Areas*
Determine and configure hardware settings	2	Conceptual understanding of kernel loading options and boot steps	Working with peripherals, storage devices, and the tools you use to configure them
Boot the system	3	How to walk through the boot process	Common boot commands; the boot sequence; boot logs
Change run levels and shut down or reboot system	3	Changing to single-user mode, shutting down and rebooting the system. Knowing that you should alert users to changes in run level and to the need for properly terminating processes	Default run level; changing run levels; how to terminate

To adequately address these topics, focus on the following files, terms, and utilities: /dev, /etc/init.d, /etc/inittab, /proc, /sys, /var/log/messages, BIOS, boot loader, dmesg, init, kernel, lsmod, lspci, lsusb, modprobe, shutdown, and telinit.

Here are the top ten items to know as you study for this domain:

1. The system log is /var/log/messages and this is where the majority of events are written to by the system log daemon (syslogd). Messages routed there can be viewed with the dmesg command.
2. The logrotate command can be used to automatically archive log files and perform maintenance as configured in /etc/syslog.conf.
3. You can manually write entries to log files using the logger command.
4. The init daemon is responsible for maintaining proper running of daemons at specified run levels. The system attempts to go to the run level specified as the default in the /etc/inittab file upon each boot.
5. Run levels can be changed with the init and shutdown commands.
6. Valid run levels defined as standards are: 0 (power off), 1 (single-user mode), 2 (multiple user without NFS), 3 (multiple user with NFS), 5 (X environment), and 6 (reboot).
7. The lsmod command is used to list loaded modules. The insmod command is used to install a module. The rmmod command is used to remove a module from the system. The modinfo command will show information about a module.
8. The modprobe utility can probe and install a module and its dependents, while the depmod utility will determine and show any module dependencies that exist.
9. Kernel software is typically named linux-*x.y.z* where *x.y.z* represents the version number.
10. The make config command executes a command-line-oriented view and allows you to respond interactively with the kernel build.

Linux Installation and Package Management

Table 2-3 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-3 Breakout of Domain 102

Subtopic	*Weight*	*Description*	*Key Areas*
Design hard disk layout	2	Designing a disk-partitioning scheme for a Linux system	Allocating file systems and swap space, tailoring the design to the intended use of the system
Install a boot manager	2	Select, install, and configure a boot manager	Boot loaders; GRUB and GRUB 2
Manage shared libraries	1	Determine the shared libraries that executable programs depend on and install them when necessary	Shared library locations and how to load
Use Debian package management	3	Know the Debian package tools	Install, upgrade, and uninstall Debian binary packages
Use RPM and Yum package management	3	Know the RPM and Yum tools	Install, upgrade, and remove packages with RPM and Yum

To adequately address these topics, focus on the following files, terms, and utilities: `/(root)` file system, `/boot/grub/menu.lst`, `/etc/apt/sources.list`, `/etc/ld.so.conf`, `/etc/yum.conf`, `/etc/yum.repos.d/`, `/home` file system, `/var` file system, `apt-cache`, `apt-get`, aptitude, `dpkg`, `dpkg-reconfigure`, `grub-install`, `LD_LIBRARY_PATH`, `ldconfig`, `ldd`, `MBR`, mount points, partitions, `rpm`, `rpm2cpio`, `superblock`, swap space, `yum`, `yumdownloader`.

Here are the top ten items to know as you study for this domain:

1. The `ldd` command is used to see what shared libraries a program is dependent upon.
2. The `ldconfig` command is used to update and maintain the cache of shared library data. You can see the current cache by using the command `ldconfig -p`.
3. Popular package managers include Red Hat's Package Manager (RPM) and Debian's (dpkg). The purpose of both is to simplify working with software.

4. Options available with RPM include `–i` (for installing packages), `-e` (for removing packages), `-q` (for querying what packages belong to what files), `-b` (for building a package), and `–p` (to print/display information).

5. With dpkg, you use the `dselect` command to use the graphical interface. You can also use command-line options that include `–i` (to install packages), `-l` (to list information about the package), `-r` (to remove the package), and `–c` (to list all files in the package).

6. The Advanced Packaging Tool (`APT`) was designed as a front end for `dpkg` but now works with both `.deb` and `.rpm` packages.

7. The Yellow dog Updater, Modified is more commonly known as `Yum` and can be used at the command line to download RPM packages.

8. The superblock contains information about the type of file system, the size, status, and metadata information.

9. The GRUB bootloader (an acronym for GNU's Grand Unified Bootloader) allows multiple operating systems to exist on the same machine and a user to choose which one they want to boot on startup. The latest version is GRUB 2.

10. Linux uses both a swap partition and a swap file for swap space. The `swapon` command can be used to toggle designated swap space on and off. Areas for swap space can be created with `mkswap`.

GNU and Unix Commands

Table 2-4 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-4 Breakout of Domain 103

Subtopic	*Weight*	*Description*	*Key Areas*
Work on the command line	4	Interact with shells and commands using the command line and the `bash` shell	Use single shell commands and one-line command sequences as well as modify the shell environment and use/edit command history
Process text streams using filters	3	Apply filters to text streams	Send text files and output streams through text utility filters
Perform basic file management	4	Use the basic Linux commands to manage files and directories	Copy, move, and delete files and directories individually and recursively

(continued)

Table 2-4 *(continued)*

Subtopic	*Weight*	*Description*	*Key Areas*
Use streams, pipes, and redirects	4	Redirect streams and connect them – including standard output, standard input, and standard error	Pipe the output of one command to the input of another
Create, monitor, and kill processes	4	Perform basic process management	Run jobs in the foreground and background; send signals to processes
Modify process execution priorities	2	Manage process execution priorities	Run programs with higher and lower priorities than the default
Search text files using regular expressions	2	Understand regular expressions and how to use them	Create simple regular expressions and perform searches
Perform basic file editing operations using `vi`	3	Understand `vi` navigation, editing, copying, deleting, etc.	Use basic `vi` modes to insert, edit, copy, delete, and find text.

To adequately address these topics, focus on the following files, terms, and utilities: `&`, `.`, `bash`, `bg`, `bzip2`, `cat`, `cp`, `cpio`, `cut`, `dd`, `echo`, `egrep`, `env`, `exec`, `expand`, `export`, `fg`, `fgrep`, file, file globbing, `find`, `fmt`, `free`, `grep`, `gunzip`, `gzip`, head, history, jobs, `join`, `kill`, `killall`, `ls`, `man`, `mkdir`, `mv`, `nice`, `nl`, `nohup`, `od`, `paste`, `pr`, `ps`, `pwd`, `regex(7)`, `renice`, `rm`, `rmdir`, `sed`, `set`, `sort`, `split`, `tail`, `tar`, `tee`, `top`, `touch`, `tr`, `uname`, `unexpand`, `uniq`, `unset`, `uptime`, `vi`, `wc`, and `xargs`.

Here are the top ten items to know as you study for this domain:

1. When run, every command spans at least one process and processes can be viewed with `ps` or `top` (which continues to update the display dynamically).
2. Jobs can run in the foreground or background and be moved between the two. Jobs running in the foreground can be suspended by pressing Ctrl+Z.
3. Files can be copied using `cp` or moved using `mv`. Files can be deleted with `rm` and directories (which are created with `mkdir`) can be removed with `rmdir`. Recursive deletion can be done with `rm -r`.
4. To change directories, use the `cd` command. When used without parameters, this will move you to your home directory. To see what directory you are presently working in, use the `pwd` (present working directory) command.

5. The ls command has a plethora of options to allow you to list files. The -a option will list all (including hidden) files.
6. The cut command can pull fields from a file and they can be combined using either paste or join. The latter offers more features than the former and can be used with conditions.
7. The wc command can count the number of lines, words, and characters in a file.
8. The grep utility (and its counterparts egrep and fgrep) can be used to find matches for strings within files.
9. The find command can be used to search the system for files/directories that meet any number of criteria. When these entities are found, the xargs command can be used to look deeper within them for other values (such as in conjunction with grep).
10. It's possible to convert data from one value to another by using a number of utilities. The most popular would include the tr (translate) utility and sed (the stream editor).

Devices, Linux File Systems, Filesystem Hierarchy Standard

Table 2-5 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-5 Breakout of Domain 104

Subtopic	*Weight*	*Description*	*Key Areas*
Create partitions and file-systems	2	Configure disk partitions	Use various mkfs commands to set-up partitions and create file-systems
Maintain the integrity of file-systems	2	Maintain a standard file-system and the extra data associated with journaling	File-system monitoring, integrity, and repair
Control mounting and unmounting of file-systems	3	Configure the mounting of file-systems	Manually mount and unmount and configure removable file-systems
Manage disk quotas	1	Manage user quotas	Configure quotas, edit and check reports

(continued)

Table 2-5 *(continued)*

Subtopic	*Weight*	*Description*	*Key Areas*
Manage file permissions and ownership	3	Control file access with permissions and ownership	Change the file-creation mask, work with special files
Create and change hard and symbolic links	2	Manage links to a file	Use links to support system administration
Find system files and place files in the correct location	2	Filesystem Hierarchy Standard (FHS)	Correct location of files and the purpose of important directories

To adequately address these topics, focus on the following files, terms, and utilities: /etc/fstab, /etc/updated.conf, /media, chgrp, chmod, chown, debugfs, df, du, dump32fs, e2fsck, edquota, ext2/ext3/ext4, find, fsck, ln, locate, mke2fs, mkfs, mkswap, mount, quota, quotaon, reiserfs v3, repquota, tune2fs, type, umask, umount, updated, vfat, whereis, which, xfs, xfs tools

Here are the top ten items to know as you study for this domain:

1. File and directory permissions can be changed with the `chmod` command (which accepts numeric and symbolic values).
2. The owner of a group can be changed with the `chown` command whereas the `chgrp` command allows changing he group associated with a file.
3. The `du` command can show how much of a disk is used.
4. The `df` command shows how much of a disk is free.
5. The main tool for troubleshooting disk issues is `fsck` which can check file-system structure, including inodes.
6. To mount file-systems, use the `mount` command and to unmount them use `umount`. To have mounting occur automatically at startup, add the entries to `/etc/fstab`.
7. Quotas can restrict the amount of space users or groups can use on the disk. Quotas are initialized with the `quota` command and they are toggled on and off with `quotaon` and `quotaoff`. They can be changed with `edquota` and reports can be generated with `repquota`.
8. When files are first created, the default permissions are equal to 666 minus any `umask` values. The default permissions for directories is equal to 777 minus any `umask` values.
9. The `mke2fs` utility can be used to make the file-system.
10. Linux supports numerous file-systems including ext2, ext3, ext4, and reiserfs.

Shells, Scripting, and Data Management

Table 2-6 shows the subtopics, weights, descriptions, and key knowledge areas for this topic — the first of the 102 exam.

Table 2-6 Breakout of Domain 105

Subtopic	*Weight*	*Description*	*Key Areas*
Customize and use the shell environment	4	Be able to modify global and user profiles	Set environment variables and work within bash
Customize or write simple scripts	4	Customize existing `bash` scripts and write new ones	Be able to write code that includes loops and tests
SQL data management	2	Query databases and manipulate data with SQL	Know basic SQL commands

To adequately address these topics, focus on the following files, terms, and utilities: `/etc/profile`, `~/.bash_login`, `~/.bash_logout`, `~/.bash_profile`, `~/.bashrc`, `~/.profile`, `alias`, `delete`, `env`, `export`, `for`, `from`, `function`, `group by`, `if`, `insert`, `join`, `lists`, `order by`, `read`, `select`, `seq`, `set`, `test`, `unset`, `update`, `where`, and `while`.

Here are the top ten items to know as you study for this domain:

1. Logic can be added to scripts by testing conditions with `test` or `[`. Commands can execute using `if-then-fi` deviations or through looping (`while`, `until`, or `for`). You can leave a script with the `exit` command or leave a loop with `break`.
2. Variables can be given at the command line and referenced as `$1`, `$2`, and so on, or entered into the executing file with the `read` command.
3. The `alias` command can be used to create an alias for a command to operate by another name (for example, being able to type `dir` and have `ls -l` performed).
4. Environmental variables can be viewed with the `env` command.
5. Variables can be added to the environment using the `set` command and `export`; they are removed using `unset`.
6. The `/etc/profile` configuration file is executed whenever a user logs in.
7. For those using the `bash` shell, the shell first looks for `.bash_profile`; if it does not file that profile, it looks for `.bash_login`.
8. When the `bash` user logs out, the shell will look for `.bash_logout` and execute any commands found there.

9. While other configuration files run only when the user logs in or out, the `.bashrc` file can execute each time a shell is run.

10. Shell scripts must have executable permissions to run, or be called by a shell (for example: `sh script`). The normal exit status of any script or application is 0 and anything else signifies a non-normal exit.

User Interfaces and Desktops

Table 2-7 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-7 Breakout of Domain 106

Subtopic	*Weight*	*Description*	*Key Areas*
Install and configure X11	2	Be able to install and configure X11	Basic understanding of X Window configuration
Set up a display manager	2	Set up and customize a display manager	Work with XDM (X Display Manager), GDM (Gnome Display Manager), and KDM (KDE Display Manager)
Accessibility	1	Knowledge and awareness of accessibility technologies	Assistive Technology (ATs) and keyboard settings

To adequately address these topics, focus on the following files, terms, and utilities: /etc/initab, /etc/x11/xorg.conf, braille display, DISPLAY, emacspeak, gdm configuration files, gestures, GOK, high contrast desktop themes, kdm configuration files, large screen desktop themes, mouse keys, on-screen reader, orca, screen magnifier, screen reader, slow/bounce/toggle keys, sticky/repeat keys, X, xdm configuration files, xdpyinfo, xhost, xwininfo

Here are the top ten items to know as you study for this domain:

1. The emacspeak speech interface is one of the most popular speech interfaces available for Linux.

2. The `xdpyinfo` utility can be used to view information about an X server. It can be used with the `all` option to see information about all the extensions supported by the server.

3. Window information for X can be viewed with the `xwininfo` utility. Using the `-all` option shows all the possible information.

4. The server access-control program for X is `xhost`. This is used to connect to a host across the network and work within the graphical interface.
5. The X Display Manager (XDM) is the default display manager included with the X Window System.
6. The `/etc/x11/xorg.conf` file is the X configuration file used for initial setup.
7. Several assistive technology projects have been developed for both KDE (the KDE Accessibility Project) and GNOME (the GNOME Accessibility Projects).
8. Orca is a screen reader from the GNOME project intended to help individuals who are blind or impaired. Orca will work with Firefox, Thunderbird, OpenOffice.org/LibreOffice, and other applications.
9. The GNOME onscreen keyboard reader (GOK) is another assistive technology. It works with XML files and can dynamically create keyboards to adapt to a user's needs.
10. *Slow keys* can be configured for a keyboard preference to accept input only if a key is held; this prevents accidental presses from counting as input. *Bounce keys* can be configured to ignore fast duplicate key presses; *sticky keys* can be used to simulate simultaneous key presses.

Administrative Tasks

Table 2-8 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-8 Breakout of Domain 107

Subtopic	*Weight*	*Description*	*Key Areas*
Manage user and group accounts and related system files	5	Add, remove, suspend, and change user accounts	Work with user and group accounts — including those for special purposes and limited accounts
Automate system administration tasks by scheduling jobs	4	Use `cron` and `anacron`	Run jobs at regular intervals and at specific times
Localization and Internationalization	3	Localize a system in a language other than English	Understand why `LANG=C` is useful in scripts

To adequately address these topics, focus on the following files, terms, and utilities: /etc/at.allow, /etc/at.deny, /etc/cron, /etc/cron.allow, /etc/cron.deny, /etc/crontab, /etc/group, /etc/localtime, /etc/passwd, /etc/shadow, /etc/skel, /etc/timezone, /usr/bin/locale, /usr/share/zoneinfo, /var/spool/cron/*, ASCII, at, atq, atrm, chage, crontab, date, environment variables, groupadd, groupdel, groupmod, iconv, ISO-8859, passwd, tzconfig, tzselect, Unicode, useradd, userdel, usermod, and UTF-8.

Here are the top ten items to know as you study for this domain:

1. Users can be added by manually editing the configuration files or by using the useradd command (and they can be removed with userdel).
2. The groupadd utility can be used to create groups and groupdel can be used to remove groups. Groups can be modified with groupmod and users can change between groups with the newgrp command.
3. To schedule a job to run only once in unattended mode, you can use the at command.
4. Scheduled jobs can be viewed with the atq command and deleted prior to execution with atrm.
5. Restrictions can be placed on who can use the at service (atd) by creating an at.allow file and only placing valid usernames beneath it.
6. You can create an at.deny file – instead of at.allow – and place in it the names of users who cannot use that at service (meaning that everyone not listed in there can still use it).
7. If you need to schedule an unattended job to run at any sort of regular interval, you can create a crontab (cron table) entry for it.
8. Crontab files are read by the cron daemon, which looks every minute to see whether any jobs need to run.
9. Restrictions can be placed on who can use cron by creating a cron.allow or a cron.deny file.
10. There are six fields to each entry in the cron tables: the minute the job is to run (0 to 59), the hour the job is to run (0 to 23), the day of the month (1 to 31), the month of the year (1 to 12), the day of the week (0 to 6), and the path to the executable that is to run.

Essential System Services

Table 2-9 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-9 Breakout of Domain 108

Subtopic	*Weight*	*Description*	*Key Areas*
Maintain system time	3	Properly maintain the system time and synchronize the clock	NTP
System logging	2	Configure the `syslog` daemon	Configure the `logging` daemon to send log output to a server
Mail Transfer Agent (MTA) basics	3	Commonly available MTA programs	Be able to perform basic forward and alias configuration on a host
Manage printers and printing	2	Manage print queues and user print jobs	Use both CUPS and LPD

To adequately address these topics, focus on the following files, terms, and utilities: /etc/cups, /etc/localtime, /etc/ntp.conf, /etc/timezone, /usr/share/zoneinfo, ~/.forward, CUPS config files/tools/utils, date, exim, hwclock, klogd, logger, lpd legacy interface (lpr, lprm, lpq), mail, mailq, newaliases, ntpd, ntpdate, pool.ntp.org, postfix, qmail, sendmail, syslog.conf, syslogd.

Here are the top ten items to know as you study for this domain:

1. The Network Time Protocol daemon (`ntpd`) maintains the time on all servers using NTP.
2. The `hwclock` command can be used to display the date and time of a system's hardware clock (also known as the *real-time clock*).
3. The time zone is configured in the `/etc/timezone` file. Local time is likewise configured in `/etc/localtime`.
4. The `sendmail` service is a general purpose SMTP program used for sending e-mail between servers.
5. The `mailq` command shows a list of messages in the mail queue and works `sendmail`.
6. The `newaliases` command builds a database for the mail aliases file.
7. Mail can be forwarded from one e-mail address to another using a `.forward` file.

8. Line printers are rarely used anymore, but support for them remains. The primary utilities associated with them were/are as follows: `lpr` (to submit a print job), `lpq` (to see the print queue), and `lprm` (to remove queued print jobs).
9. The Common Unix Printing System (CUPS) is the most common printing interface used on Linux today. It provides support for the line-printer daemon as well as for Server Message Block (SMB).
10. The kernel logging daemon (`klogd`) logs Linux kernel messages.

Networking Fundamentals

Table 2-10 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-10 Breakout of Domain 109

Subtopic	*Weight*	*Description*	*Key Areas*
Fundamentals of Internet protocols	4	Demonstrate a proper understanding of TCP/IP	Know networking fundamentals
Basic network configuration	4	Be able to view, change, and verify configuration settings on clients and hosts	Manually and automatically configure network interfaces
Basic network troubleshooting	4	Be able to troubleshoot networking issues on clients and hosts	Debug problems associated with the network configuration
Configure client side DNS	2	Be able to configure DNS on a client/host	Modify the order in which name resolution is done

To adequately address these topics, focus on the following files, terms, and utilities: `/etc/hostname`, `/etc/hosts`, `/etc/nsswitch.conf`, `/etc/resolv.conf`, `/etc/services`, `dig`, `ftp`, `host`, `hostname`, `ifconfig`, `ifdown`, `ifup`, `netstat`, `ping`, `route`, `telnet`, `tracepath`, and `traceroute`.

Here are the top ten items to know as you study for this domain:

1. IPv4 uses 32-bit addresses divided into four octets. The first octet identifies the class of address (A, B, C). The address can be public or private.

2. The `ifconfig` utility can be used to see the current IP configuration of the network cards.
3. The `ping` utility is an all purpose tool for testing connectivity. It will send echo messages to a specified host to see if it can be reached. It can be used with the loopback address (127.0.0.1) to test internal configuration.
4. Instead of using `ping`, one can use `traceroute` to see the route taken to reach a particular host.
5. The `route` utility will display the routing table and allow you to configure it.
6. The `netstat` utility will show the current status of ports — those that open, those that are listening, and so on.
7. The name of the network host is configured in `/etc/hostname` and can be viewed with the `hostname` command.
8. You can remotely log in to to another host with `telnet`, but it's highly recommended that this utility no longer be used due to very weak security.
9. FTP servers can be used to transfer files from one host to another.
10. DNS is used for resolving names to addresses. Utilities that can be used in conjunction with it include `dig` (for DNS lookup).

Security

Table 2-11 shows the subtopics, weights, descriptions, and key knowledge areas for this topic.

Table 2-11 Breakout of Domain 110

Subtopic	*Weight*	*Description*	*Key Areas*
Perform security administration tasks	3	Review system configuration to ensure host security	Understand local security policies
Setup host security	3	Know how to setup a basic level of host security	Understand TCP wrappers
Securing data with encryption	3	Key techniques that secure data	OpenSSh and GnuPG

To adequately address these topics, focus on the following files, terms, and utilities: `/etc/hosts.allow`, `/etc/hosts.deny`, `/etc/inetd.conf`, `/etc/inetd.d/*`, `/etc/init.d/*`, `/etc/inittab`, `/etc/nologin`, `/etc/passwd`, `/etc/shadow`, `/etc/ssh/ssh_host_dsa_key`, `/etc/`

ssh/ssh_host_rsa_key, /etc/ssh_known_hosts, /etc/sudoers, /etc/xinetd.conf, /etc/xinetd.d/*, ~/.gnupg/*, ~/.ssh/authorized_keys, ~/.ssh/id_dsa, ~/.ssh/id_rsa, chage, find, gpg, id_dsa.pub, id_rsa.pub, lsof, netstat, nmap, passwd, ssh, ssh_host_dsa_key.pub, ssh_host_rsa_key.pub, ssh-add, ssh-agent, ssh-keygen, su, sudo, ulimit, and usermod.

Here are the top ten items to know as you study for this domain:

1. Adding 1000 to standard permissions turns on the "sticky bit", whereas 2000 turns on the SGID permission and 4000 turns on the SUID permission.
2. Links are created with the ln command. A *hard link* is nothing more than an alias to a file (sharing the same inode). A symbolic link is created with ln -s and is an actual file with its own inode. The symbolic link contains a pointer to the original file and can span across file systems (while the hard link cannot).
3. Passwords are changed with the passwd command. While older systems stored passwords in /etc/passwd, they are now in /etc/shadow where they are more secure.
4. To see who logged on most recently and may currently still be logged on, you can use the last command.
5. The su command allows you to become another user (returning with exit). If no other username is specified, then the root user is implied, hence su for *superuser*.
6. To run a command as another user (usually root) rather than become them, sudo should be used instead of su.
7. The who command shows who is logged on; the w command shows information combining who with uptime.
8. You can limit which hosts can remotely connect by using either a hosts.allow file (only those hosts specifically listed can connect) or a hosts.deny file (only those hosts specifically listed cannot connect).
9. The ulimit utility can show the limit on the number of open files allowed in Linux. You can also change that value by using this same command.
10. The usermod command changes attributes for a user and modifies the user account.

Chapter 3: Other Linux Certifications

In This Chapter

- **Overview of vendor-neutral certifications**
- **Overview of vendor-specific certifications**

Previous chapters look at the Linux Essentials exam – LPI's lower-level certification – and then at the the higher entry-level CompTIA Linux+ certification (consisting of the LPI LX0-101 and LX0-102 exams). As important as those exams are, they are far from the only Linux certifications available.

In the following sections, we will look first at other vendor-neutral certifications, and then an overview of some of the more popular vendor-specific Linux certification.

Vendor-Neutral Certifications

Just as the CompTIA Linux+ Powered by LPI certification is a great entry-level authentication of basic knowledge, so too are most other certifications from CompTIA. The only operating system-specific one they have is Linux+, but they also offer such other certifications as A+ (hardware), Network+ (computer networking), Security+ (host and client security), and Green IT (sustainability). All are well recognized and represented in the market and good choices for adding to a resume.

LPI also offers a number of certifications above the LPIC-1 level. LPIC-2, which requires passing another two exams (referenced as 201 and 202), focuses on such advanced topics as network configuration, file storage, troubleshooting, and system security. LPIC-3 is for senior-level administrators and requires you to take one exam of your choice — on top of having attained the LPIC-2 certification. The exam you choose to take is the one that hones in on a specific area of expertise, such as virtualization and high availability or security.

Some other vendor-neutral Linux certifications once existed and were popular, but most of those have now fallen by the wayside – either no longer offered or no longer kept current. Although those certifications may hold value for those who currently hold them, they should be avoided by those who are currently looking to have a third-party authenticate their knowledge/skills.

Vendor-Specific Certifications

A number of vendors offer certifications that authenticate specialization in their specific distributions of Linux. The three most popular are discussed in this section.

One of the most recognized is those from Red Hat. At the entry level, the company offers the Red Hat Certified System Administrator (RHCSA) certification. The more recognized Red Hat Certified Engineer (RHCE) builds upon RHCSA, and the pinnacle certification is Red Hat Certified Architect (RHCA).

Novell/SUSE certification is available at four levels: SUSE Certified Linux Administrator (CLA), SUSE Certified Linux Desktop Administrator, SUSE Certified Linux Professional (CLP), and SUSE Certified Linux Engineer (CLE). Most of these certifications build on each other — CLA leads to CLP with another exam, and then you move on to CLE, and so on.

Oracle — now having acquired Sun Microsystems — offers both Oracle Certified Associate (OCA) and Oracle Certified Professional (OCP), and Certified Specialist (CS) certifications.

Index

Symbols and Numerics

- command, 144
; command, 144
: command, 452
:! command, 149
! command, 238
? command, 238
. command, 144, 452
' command, 144
{ command, 452
% command, 144
^ command, 144
+ command, 144
= command, 144
$ command, 144
/ directory, 111
$< variable, 475
$? variable, 475
$* variable, 475
$% variable, 475
$^ variable, 475
$+ variable, 475
* (asterisks), 95–96, 305, 494
@ (at sign), 116
\ (backslash), 91, 272
: (colon), 91, 142, 272, 343, 399
{} (curly braces), 249, 264, 448–449
$ (dollar sign), 289, 474
.. (dot-dot), 115, 494
= (equal sign), 348
! (exclamation point), 97, 273
/ (forward slash), 109, 111
> (greater-than sign), 93–94
# (hash mark), 249, 380, 449
- (hyphen), 117, 470
< (less-than sign), 93–94
. (period), 111, 119–120, 144, 494
| (pipe), 92, 105, 496
# (pound sign), 189, 208, 264, 271, 438
? (question marks), 95–96, 208, 494
; (semicolon), 380, 451, 496
/ (slash), 116, 143, 147, 352–353
[] (square brackets), 95–96
~ (tilde) character, 115, 146
-00 option, 471
10Base2, 173
10Base5, 173
10BaseT, 173
24-bit IV, 182
32-bit cyclic redundancy code (CRC-32), 182
127.0.0.1, reverse mapping, 295
802.11a standard, 180
802.11b standard, 179–180
802.11g standard, 180

A

a command, 145, 148
A command, 148
A resource record type, 290, 292
AAAA resource record type, 290
Abacus portsentry, 441
accessing DOS/Windows file system, 368–371
acpid service, 254
acpi=off boot option, 44
action field, 311
Active Window icon (KDE), 85
ad hoc modes, 180–181
adding newsgroups, 276–277
administrative tasks, in CompTIA Linux+ Exam, 511–512
ADSL (Asymmetric DSL), 157
Advanced Packaging Tool (APT), 50
AirSnort, 441
alias command, 98, 452, 509
allowcddma boot option, 44
allownewnews parameter, 272
all_squash option, 375
alt category, 219
anongid=GID option, 376
anonuid=UID option, 376
-ansi option, 470
apache service, 254, 438
Apache Web Server vulnerability, 433
apic boot option, 44
apmd service, 254
append command, 238
applets, 121
application gateway, 392
Application layer, of TCP/IP, 168
Application Starter, 82–85
application-proxy gateway firewall, 420
applications
  about, 19–21, 127–128
  calculators, 132–133
  calendars, 132
  graphics and imaging
    about, 136
    GIMP, 18, 20, 136–137
    GNOME Ghostview, 18, 138
  multimedia
    about, 133
    burning CDs/DVDs, 135–136
    digital camera, 133–134
    playing audio CDs, 134–135
    playing sound files, 135
  office, 128–132

Applications category (KickOff), 83
Applixware, 20
apropos command, 98
APT (Advanced Packaging Tool), 50
apt-get command, 11, 50, 128
AR variable, 475
ARFLAGS variable, 475
arguments, 61, 91
Arkeia utility, 363
ARPANET, 169, 279
AS variable, 290, 475
ascii command, 238
ASCII-armored format, 409
ASFLAGS variable, 475
associates, 244
asterisks (*), 95–96, 305, 494
Asymmetric DSL (ADSL), 157
async option, 375
at command, 331–333, 512
at sign (@), 116
atd service, 254, 331
atq command, 512
atrm command, 333, 512
audio CDs, playing, 134–135
audit, 429
authentication
 defined, 392
 in security policy, 387
 vulnerability, 433
authorization, in security policy, 387
autoconf software package, 17
autofs service, 254
automake software package, 17
automated backups, 367
automatic command completion, 95
automatically starting servers, 313–314
awk command, 459–463

B

b command, 149
-b option, 477
backdoor, 392
backing up
 files
  about, 361
  commercial backup utilities, 362–363
  selecting backup strategy and media, 361–362
  tar command, 363–367
 hard drive before installation, 33
backslash (\), 91, 272
bash shell. *See also* shell scripting
 about, 17, 90, 509
 asterisks in, 95–96
 automatic command completion, 95
 combining commands, 92
 command syntax, 90–92
 controlling command input and output, 92–94
 question marks in, 95–96
 repeating previously typed commands, 97
 starting, 60–61
bastion host, 392
batch files, 106
bc software package, 17
Berkeley Internet Name Domain (BIND)
 about, 282
 DNS vulnerability, 433
 named daemon, 254, 282–283, 296
 resolver library, 280, 283
 utility programs, 284–285
Berkeley sockets interface, 242
bg command, 100, 453
/bin directory, 111, 354
binary command, 238
BIND. *See* Berkeley Internet Name Domain (BIND)
Binutils software package, 17
bionet category, 219
BIOS, reconfiguring, 36
bison, 486
bit bucket, 94
bit category, 219
biz category, 220
black box, 440
block devices, 326
BogoMIPS, 58, 59
Bookmarks menu (Firefox), 212
/boot directory, 111, 354
boot loader, 57–58
boot messages, 58
boot: prompt, 44
bootable flash drive, creating, 35
/boot/grub file, 314
/boot/grub/menu.lst file, 314, 435
BOOTPROTO parameter, 186, 200
boots
 automatically starting servers, 313–314
 /etc/inittab file, 310–311
 init command, 311–312
 init process, 308–310
 manually starting/stopping servers, 313
 startup scripts, 312–313
/boot/System.map file, 314
/boot/vmlinuz file, 314
bottom panel (GNOME), 73–74
bounce keys, 511
break command, 453, 480, 509
BRU utility, 362
bt command, 480
buffer, 145

buffer overflow, 392
built-in commands, bash, 451–455
burning CDs/DVDs, 135–136
business requirements, for security, 385–386
bye command, 238

C

c command, 145
C command, 148
C++ compiler, 17, 468–472
C operator, 265
-c option, 470
-C option, 477
CA (certificate authority), 392
CA ARCserve Backup for Linux utility, 363
cable modem
 about, 155
 connecting with
  about, 161
  how it works, 161–163
  setup, 163–166
Cable Modem Termination System (CMTS), 162–163
cable/DSL router, 161, 165
cables, Ethernet, 173–175
caching name server
 configuring, 286–296
 starting and testing, 295–296
cal command, 101
Calc (LibreOffice.org), 130–131
calculator apps, 132–133
calendar apps, 132
careers, on Linux Essentials Certification Exam, 492–493
Carrier-Sense Multiple Access/Collision detection (CSMA/CD), 172
case statement, 450–451
cat command, 92, 95, 99
categories
 of shielded and unshielded twisted-pair cables, 173–174
 tasks by in YaST Control Center, 305
cc command, 148
CC variable, 475
cd command, 98, 115, 238, 453, 495, 506
cd images command, 121
cdrdao, 136
cdrecord, 136
CD-ROM, mounting, 357
CDs
 audio, playing, 134–135
 burning, 135–136
 installing from, 29–31
 mounting, 25
central processing unit (CPU), 9
CERT Coordination Center (CERT/CC) (website), 397
CERT resource record type, 291
certificate, defined, 392
certificate authority (CA), 392
certifications
 vendor-neutral, 517
 vendor-specific, 518
CFLAGS variable, 475
chage command, 396, 400–401
chain, 424
changing
 directories, 495
 file ownerships, 118–119, 350, 402
 file permissions, 118–119, 403
 home page in Firefox, 212–215
CHANNEL parameter, 186
character devices, 327
characters, substituting/deleting from files, 105–106
check digit, 459
checking
 connectivity to host, 195
 disk performance/disk usage, 321–322
 IP routing table, 194
 network interfaces, 193–194
 network status, 195–197
 PC hardware, 31–33
 for set user ID permission, 405
Cheops, 441
chgrp command, 402, 508
chkconfig command, 251–252, 314, 438
chmod command, 98, 118–119, 238, 499, 508
chown command, 98, 402, 508
chsh command, 100
Cinnamon, 69–70, 75–76
clari category, 220
clear command, 480
client architecture, 241
client/server model, sockets and, 243–244
close command, 238
CMTS (Cable Modem Termination System), 162–163
CNAME resource record type, 291
CO variable, 475
coaxial cable, 162

CodeWeavers CrossOver Office, 21
COFLAGS variable, 475
Collier-Brown, David (author)
  Using Samba, 3rd Edition, 380
colon (:), 91, 142, 272, 343, 399
colon command mode, 146
combining shell commands, 92
!command, 144
command command, 453
COMMAND heading, 319
command history, 97
command input/output, controlling, 92–94
command interpreter, 60
command line, 61, 91, 493–496, 495–496
command mode, 142
COMMAND.COM, 90
command-line
  FTP client, 236–240
  options, 91
commands
  - command, 144
  ; command, 144
  : command, 452
  :! command, 149
  ! command, 238
  ? command, 238
  . command, 144, 452
  ` command, 144
  { command, 452
  % command, 144
  ^ command, 144
  + command, 144
  = command, 144
  $ command, 144
  a, 145, 148
  A, 148
  at, 331–333, 512
  alias, 98, 452, 509
  append, 238
  apropos, 98
  apt-get, 11, 50, 128
  ascii, 238
  atq, 512
  atrm, 333, 512
  awk, 459–463
  b, 149
  bash built-in, 451–455
  bg, 100, 453
  binary, 238
  break, 453, 480, 509
  bt, 480
  bye, 238
  c, 145
  C, 148
  cal, 101
  cat, 92, 95, 99
  cc, 148
  cd, 98, 115, 238, 453, 495, 506
  cd images, 121
  chage, 396, 400–401
  chgrp, 402, 508
  chkconfig, 251–252, 314, 438
  chmod, 98, 118–119, 238, 499, 508
  chown, 98, 402, 508
  chsh, 100
  clear, 480
  close, 238
  !command, 144
  command command, 453
  compress, 100
  continue, 453, 480
  cp, 98, 119–120, 369–370, 506
  cpio, 17, 100
  crontab, 333–336, 367, 512
  Ctrl+D, 149
  Ctrl+L, 149
  Ctrl+U, 149
  cut, 99, 496, 507
  d, 143, 145
  D, 148, 268
  date, 101, 103–104
  dd, 44, 99, 148
  declare, 453
  delete, 239
  depmod, 328
  df, 101, 123–125, 508
  dG, 148
  diff, 17, 99
  dir, 239
  for directory navigation, 114–115
  dirs, 453
  disconnect, 239
  display, 480
  dselect, 505
  du, 101, 123–125, 365, 508
  dw, 148
  echo, 453
  enable, 453
  env, 348, 509
  eval, 453
  exec, 453
  exit, 453
  expand, 99
  export, 453, 509
  expr, 450
  failsafe Knoppix boot command, 40
  fb1280x1024 Knoppix boot command, 40
  fc, 453
  fdformat, 101
  fdisk, 101
  fg, 100, 453
  file, 99, 480
  find, 99, 121–122, 396, 496, 507
  fold, 99
  free, 100
  fsck, 101, 508
  gcc, 18, 468–469, 469–470

get, 239
gpg, 390, 408–412
grep, 18, 51, 62, 91, 92, 96, 99
groups, 100
gunzip, 100
gzip, 18, 100, 496
h, 149
halt, 100
hash, 239, 454
help, 239, 452, 454, 480
history, 97, 454
hwclock, 513
i, 143, 145, 148
I, 148
id, 101, 311
ifconfig, 193, 498, 515
image, 239
info, 98, 495
info break, 480
info files, 480
info func, 480
info local, 480
info prog, 480
info var, 480
init, 102, 311–312, 503
insmod, 328
iptables, 164, 421–422, 424–426
j, 149
J, 148
jobs, 454
join, 496, 507
k, 149
kill, 100, 454, 481
knoppix 1, 41
knoppix 2, 41
knoppix acpi=off, 41
knoppix atapicd, 41
knoppix desktop=wmname, 41
knoppix dma, 41
knoppix floppyconfig, 41
knoppix fromhd=/dev/hdal, 41
knoppix hsync=80, 41
knoppix lang=xx, 41
knoppix mem=256M, 41
knoppix myconf=/dev/hdal, 41
knoppix myconf=scan, 41
knoppix noeject, 42
knoppix noprompt, 42
knoppix nowheel, 42
knoppix noxxx, 42
knoppix pci=bios, 42
knoppix pnpbios=off, 42
knoppix screen=resolution, 42
knoppix testcd, 42
knoppix tohd=/dev/hdal, 42
knoppix toram, 42
knoppix vga=ext, 42
knoppix vsync=60, 42
knoppix wheelmouse, 42
knoppix xmodule=modname, 43
knoppix xserver=progname, 43
ksyms, 328
l, 149
last, 499, 516
lcd, 239
ldconfig, 504
ldd, 100, 504
less, 19, 99
Linux
  about, 62–63, 98–101
  date, 103–104
  managing processes, 102–103
  processing files, 104–106
  time, 103–104
list, 481
ln, 98, 499, 516
local, 454
locate, 99, 122
logger, 498, 503
logout, 454
logrotate, 498, 503
lpq, 514
lpr, 99, 514
lprm, 514
ls, 98, 116–118, 119, 239, 350, 402, 495, 507
lsmod, 328, 329, 503
mail, 258
mailq, 513
make, 19, 476–478, 481
make config, 503
man, 98, 495
managing user accounts with, 342–343
mdelete, 239
mdir, 239
mget, 239
mkdir, 98, 120–121, 239
mkfs, 101
mknod, 101
mkswap, 101
mls, 239
modinfo, 328, 503
modprobe, 328, 329, 330, 503
more, 63, 99
mount, 101, 356, 357, 358, 368–369, 508
mput, 239
mv, 98, 495, 506
-n, 144
n, 145
^n, 145
+n, 144
netstat, 195, 397, 498, 515
newaliases, 513
newgrp, 499, 512
next, 481
nice, 100
nl, 99
open, 240
p, 149
P, 149, 265
pack, 496

commands *(continued)*
  passwd, 101, 397, 516
  paste, 99, 496, 507
  patch, 19, 99
  ping, 195, 441, 498, 515
  popd, 454
  print, 481
  printenv, 100, 347
  prompt, 240
  ps, 63, 100, 496, 506
  pstree, 100
  pushd, 454
  put, 240
  pwd, 98, 115, 240, 454, 506
  q, 144, 145
  :q, 149
  Q, 145
  :q!, 149
  quit, 240, 481
  quota, 508
  r file, 145
  :r filename, 149
  read, 454, 509
  readonly, 454
  reboot, 100
  recv, 240
  rename, 240
  repeating previously typed, 97
  return, 454
  rm, 98, 120, 495, 506
  rmdir, 98, 240, 506
  rmmod, 328, 503
  run, 481
  runlevel, 310, 311
  rx, 148
  /sbin/ifconfig, 193
  /sbin/route, 194
  sed, 19, 99, 457–463, 507
  send, 240
  server, 250–251
  set, 454, 509
  set variable, 481
  shell, 61–62, 92
  shell, 481
  shift, 454
  shutdown, 100, 503
  size, 240
  s/old/new/, 145
  sort, 99, 105
  split, 99, 106
  status, 240
  step, 481
  /string, 149
  ?string, 149
  su, 101, 102, 306, 397, 499, 516
  swapoff, 101
  swapon, 101
  sync, 101, 375
  tac, 99
  tail, 99
  tar, 19, 100, 363–367, 496
  tcpdump, 197, 441
  telinit, 312
  test, 449–450, 509
  testparm, 378, 379
  /text/, 144
  ?text?, 144
  time, 19, 103–104
  times, 454
  top, 100, 496, 506
  touch, 98
  tr, 99, 105–106, 507
  trap, 454
  tty, 101
  type, 455
  u, 145, 149
  U, 149
  ulimit, 397, 455, 516
  umask, 403–405, 455
  umount, 101, 356–357
  unalias, 98, 455
  uname, 62, 100
  uncompress, 100
  uniq, 99
  unset, 455, 509
  update-rc.d, 252–253, 314
  uptime, 319–320, 516
  user, 240
  useradd, 499, 512
  userdel, 499
  w, 149, 499
  w file, 145
  W file, 145
  :w filename, 149
  wait, 455
  watch, 481
  wc, 100, 104–105, 496, 507
  whatis, 98, 495
  where, 481
  whereis, 99
  which, 99
  who, 499, 516
  :wq, 149
  x, 148, 481
  xargs, 496
  yy, 149
  zcat, 100
  zless, 100
  zmore, 100
commercial backup utilities, 362–363
Common Unix Printing System (CUPS), 514
Common Vulnerabilities and Exposures (CVE) (website), 432
community, on Linux Essentials Certification Exam, 492–493
comp category, 220
compress command, 100
CompTIA Linux+ exams
  about, 501–502
  administrative tasks, 511–512
  devices, Linux file systems, file system hierarchy standard, 507–508
  essential system services, 512–514
  GNU and Unix commands, 505–507

Linux installation and package management, 503–505
networking, 514–515
security, 515–516
shells, scripting, and data management, 509–510
system architecture, 502–503
user interfaces and desktops, 510–511
Computer category (KickOff), 83
computer program, 465
computer security, 391–397
computer security audit
about, 429–430
defined, 429
implementing methodology for
about, 431–432
common vulnerabilities, 432–434
host-security review, 434–437
network security review, 438–439
nontechnical aspects of, 430
security testing tools, 440–442
technical aspects of, 431
vulnerability testing types, 440
concatenate, 118
conditional expressions, 463
confidentiality, 392
configuration files, 314–317
configuration lists, 276
configuring. *See also* installing; troubleshooting
caching name server, 286–296
INN server, 269–275
Internet super server, 414
KDE bottom panel, 85–86
KDE desktop, 86–88
networks at boot time, 199–201
primary name server, 296–298
resolver, 285–286
Samba, 379–380
TCP wrapper security, 414–415
wireless access point, 184
wireless networking, 185–188
connecting LAN to Internet, 177–178
connectionless protocols, 243
connection-oriented protocols, 242–243
console, 89
context menus (KDE)
desktop, 80–81
icons, 81–82
`continue` command, 453, 480
`controls` statement, 288
copying files, 495
Coreutils software package, 17
counting words/lines in text files, 104–105
`cp` command, 98, 119–120, 369–370, 506
`cpio` command, 17, 100
`CPP` variable, 475
`CPPFLAGS` variable, 475
CPU (central processing unit), 9
`cpu` field name, 321
`%CPU` heading, 319
cracker, 392
creating
bootable flash drive, 35
key pair, 408–409
working configuration file, 39
`cron` daemon, 254, 333–336
`crontab` command, 333–336, 367, 512
CrossOver Office, 21
CSMA/CD (Carrier-Sense Multiple Access/ Collision detection), 172
`ctlinnd`, 270
Ctrl+D command, 149
Ctrl+L command, 149
Ctrl+U command, 149
CUPS (Common Unix Printing System), 514
curly braces ({}), 249, 264, 448–449
current directory, 114–115
`cut` command, 99, 496, 507
CVE (Common Vulnerabilities and Exposures) (website), 432
CVS (Subversion, Concurrent Versions System), 23, 433
`CXX` variable, 475
`CXXFLAGS` variable, 475
CyberCop Scanner, 441

D

`d` command, 143, 145
`D` command, 148, 268
`-d` option, 477
daemon, 270
data management, in CompTIA Linux+ Exam, 509–510
data protection, in security policy, 387
databases, vulnerability of, 434
datagrams, 243
date, working with, 103–104
`date` command, 101, 103–104
`dd` command, 44, 99, 148

DDoS (Distributed Denial of Service), 383, 393
Debian
  about, 11
  BIND, 282
  configuring networks, 199
  distributions, 38
  init process, 310
  installing
    INN server, 268
    Samba, 379
    software in, 50–52
    software-development tools, 467
  mounting CD-ROM, 357
  runlevel command, 310
  starting
    innd, 275
    NFS server, 359, 376
    Samba, 380
    standalone servers in, 251
  stopping standalone servers in, 251
  turning off standalone servers, 413
  update-rc.d command, 252–253, 314
  updates, 398
debug parameter, 330
declare command, 453
decrypting
  defined, 392
  documents, 411–412
delay loop, 59
delete command, 239
deleting characters from files, 105–106
Denial of Service (DoS), 383, 386, 393
depmod command, 328
derived work, 485
DES, 345
desktop, GNOME, 70–71
desktop context menus (KDE), 80–81
Desktop Pager icon (KDE), 85
detection concept, in security programs, 389, 432, 434, 438
/dev directory, 111, 326, 354
device drivers, 325
device files, 325, 326–327
DEVICE parameter, 186
devices
  in CompTIA Linux+ Exam, 507–508
  Linux
    about, 325–326
    device files, 326–327
    persistent device naming with udev, 327–328
  mounting on file system, 355–357
  peripheral, 25–26
/dev/null file, 94
df command, 101, 123–125, 508
dG command, 148
DHCP (Dynamic Host Configuration Protocol), 170, 245
dial-up, compared with DSL and cable, 155
diff command, 17, 99
dig utility program, 284–285
digiKam app, 133–134
digital camera apps, 133–134
digital signatures, 393, 407–408
Digital Subscriber Line. *See* DSL (Digital Subscriber Line)
dir command, 239
directories
  about, 351–352
  changing, 495
  commands for listings and permissions, 116–118
  commands for working with, 120–121
  defined, 109
  navigating with commands, 114–115
  in /proc, 324–325
  protecting
    about, 402
    changing file ownerships, 402
    changing file permissions, 403
    checking for set user ID permission, 405
    setting default permission, 403–405
    viewing ownerships and permissions, 402
  standard in Linux file system, 354
  top-level, 111–112
directory path, 208
dirs command, 453
disclosure of information to public, 386
disconnect command, 239
disk partition, 29
disk-space usage, commands for, 123–125
display command, 480
DISPLAY environment variable, 349
display=IP_address:0 boot option, 44
distributed database, 279
Distributed Denial of Service (DDoS), 383, 393
distributed responsibility, 280
distribution list, 273
distribution media, 25
Distribution Specific icon, 4
distributions
  about, 16
  contents of, 16–24
  GNU software, 16–19

GUIs and applications, 19–21
Internet servers, 21–22
Linux, 10–14
networks, 21
online documentation, 22–23
software development, 22
distribution-specific version numbers, 15
DistroWatch (website), 11
DIX standard, 173
-DMACRO option, 470
-DMACRO=DEFN option, 471
DMZ, 393
DNAME resource record type, 291
dnl (delete through newline) macro, 260–261
DNS. *See* Domain Name System (DNS)
DOCSIS, 163
documents, decrypting and encrypting, 411–412
dollar sign ($), 289, 474
domain name, 208
domain name system (DNS)
about, 171, 279–280
Berkeley Internet Name Domain (BIND)
about, 282
named daemon, 282–283
resolver library, 283
utility programs, 284–285
configuring
caching name server, 286–296
primary name server, 296–298
resolver, 285–286
hierarchical domain names, 281–282
domain parameter, 272
domains, on Linux+ Exams, 502
DoS (Denial of Service), 383, 386, 393
DOS file system, accessing, 368–371
DOS floppy disks, mounting, 369–370
DOS/Windows disk partition, mounting, 368–369
dot-dot (..), 115, 494
dotted-decimal notation, 169
dotted-quad notation, 169
downstream data, 161
driverdisk boot option, 44
dselect command, 505
DSL (Digital Subscriber Line)
about, 154
connecting with
about, 156
ADSL, IDSL, SDSL, 157–158
how it works, 156–157
modem, 156
setup, 158–161
DSL/cable modem NAT router, 177
dsniff, 441
du command, 101, 123–125, 365, 508
dual-homed host, 393
Dummies (website), 5
dumps, 197
DVD drive, checking for installation, 31
DVDs
burning, 135–136
installing from, 29–31
mounting, 25
dw command, 148
Dynamic Host Configuration Protocol (DHCP), 170
dynamically linked, shared libraries, 23

E

-e option, 477
-E option, 471
echo command, 453
Eckstein, Robert (author)
Using Samba, 3rd Edition, 380
ed, 18, 141–145
Edit menu (Firefox), 212
802.11a standard, 180
802.11b standard, 179–180
802.11g standard, 180
electronic mail (e-mail), 21–22
emacs software package, 18, 467
e-mail, 153
enable command, 453
enabling packet filtering, 421–426
encrypting
defined, 393
documents, 411–412
files with GnuPG, 406–412
enforcing=0 boot option, 44
Enter, 145
env command, 348, 509
environment variable, 348
Epiphany, 209
equal sign (=), 348
error messages, saving in files, 94
Esc, 149
essential system services, in CompTIA Linux+ Exam, 512–514
ESSID parameter, 186
establishing security policies, 387–388
/etc directory, 111, 354
/etc/apache2/httpd.conf file, 315
/etc/apt/sources.list file, 315

/etc/at.allow file, 315
/etc.bashrc file, 315
/etc/cron.allow file, 435
/etc/cron.deny file, 435
/etc/crontab file, 435
/etc/cups/cupsd.conf file, 315
/etc/exports file, 375–376
/etc/fonts file, 315
/etc/fstab file, 315, 357–358
/etc/group file, 315
/etc/grub.conf file, 315
/etc/host.conf file, 190, 191
/etc/hosts file, 189–190, 315
/etc/hosts.allow file, 190, 192, 315, 435
/etc/hosts.deny file, 190, 192–193, 315, 435
/etc/httpd/conf/httpd.conf file, 315
/etc/init.d file, 315
/etc/init.d/rcs file, 316
/etc/inittab file, 310–311, 315
/etc/issue file, 316
/etc/lilo.conf file, 316
/etc/login.defs file, 316
/etc/logrotate.conf file, 435
/etc/mail directory, 267
/etc/modprobe.conf file, 95, 316, 329–330
/etc/modules.conf file, 316
/etc/mtab file, 316
/etc/named.conf file, 287–289
/etc/networks file, 190
/etc/nologin file, 427
/etc/nsswitch.conf file, 190, 193
/etc/pam.d file, 435
/etc/passwd file, 316, 343–344, 399–400, 435
/etc/profile file, 316, 509
/etc/profile.d file, 316
/etc.rc.d file, 436
/etc/resolv.conf file, 190, 191
/etc/samba/smb.conf, 377, 379
/etc/samba/smbusers, 377
/etc/securetty file, 436
/etc/security file, 436
/etc/shadow file, 316, 400–401, 427, 436
/etc/shells file, 316
/etc/shutdown.allow file, 436
/etc/skel file, 316
/etc/ssh file, 436
/etc/ssh/ssh_config, 416
/etc/ssh/sshd_config, 416
/etc/sysconfig file, 316, 436
/etc/sysctl.conf file, 316, 436
/etc/syslog.conf file, 436
/etc/termcap file, 316
/etc/udev file, 317
/etc/udev/udev.conf file, 436
/etc/vsftpd file, 436
/etc/vsftpd.ftpusers file, 436
/etc/X11 file, 317
/etc/X11/xorg.conf file, 317
/etc/xinetd.conf file, 317, 436
/etc/xinetd.d file, 436
/etc/yum.conf file, 317
Ethereal, 441
Ethernet
  about, 157, 172
  cables, 173–175
  how it works, 172–173
Ethernet hub, 174
Ethernet LAN, setting up, 172–175
  about, 172
  cables, 173–175
  how it works, 172–173
/etrc/hosts.allow file, 427
/etrc/hosts.deny file, 427
/etrc/inetd.conf file, 427
/etrc/init.d/* file, 427
/etrc/inittab file, 427
/etrc/passwd file, 427
/etrc/xinetd/conf file, 427
/etrc/xinetd.d/* file, 427
eval command, 453
Excel (Microsoft), 130–131
exchanging keys, 409–410
exclamation point (!), 97, 273
exec command, 453
exit command, 453
expand command, 99
expert boot option, 40, 45
expire, 270
exploit tools, 393
export command, 453, 509
exporting file systems with NFS, 360, 374–376
expr command, 450
expressions, 466

F

F operator, 265
-f option, 477
failsafe Knoppix boot command, 40

-fallow-single-precision option, 471
FAT (file allocation table), 36
Favorites category (KickOff), 83
fb1280x1024 Knoppix boot command, 40
fc command, 453
FC variable, 475
fdformat command, 101
fdisk command, 101
fdiv (floating point division), 324
FEATURE macro, 263
Fedora
  about, 12
  BIND, 282
  chkconfig command, 251–252, 314
  configuring
    networks, 199
    Samba, 379
  distributions, 38
  /etc/fstab file, 358
  init process, 310
  installing
    Samba, 379
    software in, 52–54
    software-development tools, 467
  reading RSS feeds, 230
  security level configuration tool, 422
  starting
    innd, 275
    NFS server, 359, 376
    Samba, 380
  turning off standalone servers, 414
  updates, 398
Fedora Core (website), 12
FFLAGS variable, 475
fg command, 100, 453
FHS (Filesystem Hierarchy Standard), 353
file allocation table (FAT), 36
file command, 99, 480
File menu (Firefox), 212
file sharing
  about, 26
  with NFS
    about, 358–359, 373–374
    exporting file systems, 360, 374–376
    mounting NFS file systems, 360–361, 376–377
file systems
  about, 26, 109–114
  for checking disk-space usage, 123–125
  managing
    about, 351–352, 361
    accessing DOS or Windows file system, 368–371
    backing up, 361–367
    /etc/fstab file, 357–358
    hierarchy, 352–355
    mounting devices, 355–357
    sharing with NFS, 358–361
  for mounting and unmounting, 122–123
  navigating with Linux commands
    about, 114
    for changing permissions and ownerships, 118–119
    for directory listings and permissions, 116–118
    for directory navigation, 114–115
    for finding files, 121–122
    for working with directories, 120–121
    for working with files, 119–120
File Transfer Protocol (FTP)
  about, 170, 231
  command-line client, 236–240
  graphical clients
    about, 231–232
    FileZilla, 234–236
    gFTP, 231, 232–233
  port numbers and, 245
file transfer utilities, 21–22
filenames
  about, 208
  in Linux, 110
files. *See also specific files*
  backing up
    about, 361
    commercial backup utilities, 362–363
    selecting backup strategy and media, 361–362
    tar command, 363–367
  batch, 106
  changing
    permissions, 118–119, 403
    user/group ownership of, 350, 402
  commands
    for filing, 121–122
    for working with, 119–120
  copying, 495
  deleting characters from, 105–106
  encrypting with GnuPG, 406–412
  getting command input from, 93
  getting command output from, 93–94
  hiding, 494
  listings, 495
  moving, 495
  permissions, security and, 435–437

files. *See also specific files (continued)*
in /proc, 324–325
processing, 104–106
protecting, 402–405
saving error messages in, 94
signing with GnuPG, 406–412
slitting into smaller files, 106
substituting characters from, 105–106
Filesystem Hierarchy Standard (FHS), 353
FileZilla, 234–236
find command, 99, 121–122, 396, 496, 507
finding bugs in gbd, 482–483
Findutils software package, 18
finger software package, 18
fingerprint, 401, 407
Firefox
changing home page, 212–215
user interface, 210–212
web surfing in, 215–216
Firewalk, 441
firewall
about, 384
defined, 27, 393
setting up
about, 418–421
enabling packet filtering, 421–426
NAT, 421
fixing bugs in gbd, 483–484
flash drive
creating bootable, 35
installing on a, 34–35
floating-point division, 324
flow, controlling in shell scripting, 449–451
flow-control statements, 466
FOCUS-LINUX mailing list, 398
fold command, 99
for loop, 450
forgotten root password, 307–308
FORWARD chain, 424–426
.forward file, 267–268
forward slash (/), 109, 111
-fpcc-struct-return option, 471
-fPIC option, 471
FQDN (fully qualified domain name), 281–282
framework, security
about, 384–385
business requirements, 385–386
managing, 389
mitigation, 388–389
policy, 387–388
risk analysis, 386–387
free command, 100
Free Software Foundation (FSF), 16
-freg-struct-return option, 471
fsck command, 101, 508
FSF (Free Software Foundation), 16
FTP. *See* File Transfer Protocol (FTP)
full packet awareness, 420
fully qualified domain name (FQDN), 281–282
functions
defined, 467
shell, 448–449

G

-g option, 471
Games subcategory (KickOff), 84
gawk software package, 18
gcc command, 18, 468–469, 469–470
gdb software package, 18, 478–484
gdbm software package, 18
GECOS, /etc/passwd file, 343
General Public License (GPL), 16
Gentoo Linux, 12
get command, 239
gets function, 482
gettext software package, 18
getting started, 27–28
gFTP, 231, 232–233
ghostscript software package, 18
ghostview software package, 18, 138
GID
/etc/passwd file, 343
in group definitions, 344
GIMP (GNU Image Manipulation Program), 18, 20, 136–137
The GIMP User Manual (GUM), 137
globbing, 494
GNOME
about, 11, 18, 20, 69–70
Add or Remove Software Utility, 52–54
bash shell, 60, 61
desktop, 70–71
logging out of, 65
onscreen keyboard reader, 511
panels, 71–74
text editor, 139–140
GNOME calculator, 132
GNOME Ghostview, 138
GNOME Strae (website), 229
GNU C compiler, 23, 468–472
GNU C++ compiler (g++), 23

GNU C Library software package, 18
GNU commands, in CompTIA Linux+ Exam, 505–507
GNU compiler for Java, 23
GNU debugger (gdb), 23, 467, 478–484
GNU emacs editor, 23
GNU General Public License (GPL), 485–486
GNU Library General Public License (LGPL), 485, 486–487
GNU licenses, 14, 484–487
GNU make utility, 23, 467, 472–478
GNU Privacy Guard (GnuPG), 390, 406–412
GNU profiling utility (gprof), 23
GNU Project, 16
GNU software, 16–19
GNUchess software package, 18
Gnumeric software package, 18
GnuPG, 390, 406–412
Google (website), 227
Google Groups (website), 220, 227, 228
gpg command, 390, 408–412
GPL (General Public License), 16
GPL (GNU General Public License), 485–486
GPLv3 (website), 487
gpm service, 254
graphical desktop, 11
graphical FTP clients
  about, 231–232
  FileZilla, 234–236
  gFTP, 231, 232–233
graphical user interfaces (GUIs)
  about, 19–21, 69–70
  Cinnamon, 75–76
  GNOME
    desktop, 70–71
    panels, 71–74
  MATE, 76–77
  Unity, 74–75
graphics and imaging applications
  about, 136
  GIMP, 18, 20, 136–137
  GNOME Ghostview, 18, 138
Graphics subcategory (KickOff), 84
gray box, 440
greater-than sign (>), 93–94
grep command, 18, 51, 62, 91, 92, 96, 99
grep utility, 496, 507
groff software package, 18
groupadd utility, 499, 512
groupdel utility, 499, 512
groupmod utility, 499, 512
Groupname, in group definitions, 344
groups. *See also* users
  about, 337
  administration values, 345–347
  changing ownership of files, 350
  managing, 344–345
groups command, 100
GRUB boot loader, 57, 505
gtk+ software package, 18
GUI service-configuration utility, 253–255
gui text editors, 139–141
GUI tools, 198, 303–305
GUI user manager, 338–342
GUI utilities, 307
GUIs. *See* graphical user interfaces (GUIs)
gunzip command, 100
gzip command, 18, 100, 496

H

h command, 149
H operator, 265
-h option, 477
halt command, 100
hard disk, 24
hard drive
  backing up, 33
  checking for installation, 31
hard link, 516
hardware
  checking for installation, 31–33
  defined, 9
  wireless, 183–184
hash, 393, 407
hash command, 239, 454
hash mark (#), 249, 380, 449. *See also* pound sign (#)
head end, 162
hello function, 448–449
help command, 239, 452, 454, 480
Help menu (Firefox), 212
hexadecimal digit, 184
hidden files, 494
hide option, 375
hierarchical domain names, 281–282
hierarchy
  directory, 111
  file system, 352–355
HINFO resource record type, 291
hiscachesize parameter, 272
history command, 97, 454
History menu (Firefox), 212
/home directory, 111, 354
HOME environment variable, 349
home page, 210, 212–215

homedir, /etc/passwd file, 344
host
  checking connectivity to, 195
  defined, 394
  security issues, 389, 390
host address, 169
host utility program, 284–285
HOSTNAME environment variable, 348, 349
host-security
  about, 434–437
  recommended tools, 441
hot plug, 26
hping2, 441
HTML anchor, 208
HTTP (HyperText Transfer Protocol), 171, 206, 245
httpd service, 22, 254, 438
HWADDR parameter, 186
hwclock command, 513
HyperText Markup Language (HTML), 206, 208
HyperText Transfer Protocol (HTTP), 171, 206, 245
hyphen (-), 117, 470

I

i command, 143, 145, 148
I command, 148
-i option, 477
-I option, 471, 477
i386, 10
IA32 architecture processors, 10
IBM, 10
IC (Integrity Check) field, 182
ICMP (Internet Control Message Protocol), 195
icon context menus (KDE), 81–82
icons, explained, 4
id command, 101, 311
ide=nodma boot option, 45
IDSL (ISDN DSL), 157
ieee category, 220
IETF (Internet Engineering Task Force), 170, 280
ifconfig command, 193, 498, 515
image command, 239
images
  accessing on USB digital camera, 134
  opening in GIMP, 137
IMAP (Internet Message Access Protocol), 171
implementing
  security solutions, 388–389
  security test methodology
    about, 431–432
    common vulnerabilities, 432–434
    host-security review, 434–437
    network security review, 438–439
Impress (LibreOffice.org), 131–132
incident handling, in security policy, 387
incident response, 437
incoming.conf file, 274
incremental backups, 366–367
indent software package, 19
inetd server, 247–248, 413, 415, 438
info break command, 480
info command, 98, 495
info files command, 480
info func command, 480
info local command, 480
info prog command, 480
info var command, 480
information sharing, 154
infrastructure, wireless Ethernet networks, 180–181
init command, 102, 311–312, 503
init process, 308–310
Initialization Vector (IV), 182
INN server
  configuring and starting, 269–275
  installing, 268–269
  website, 269
inn.conf file, 271–272
innd service, 22, 254, 270
innflags parameter, 272
input and output redirection (I/O redirection), 92–94
INPUT chain, 424–426
insecure option, 375
insecure_locks option, 375
insmod command, 328
installing
  about, 29, 50
  checking PC hardware, 31–33
  creating bootable flash drive, 35
  on a flash drive, 34–35
  INN server, 268–269
  Live CD, 34
  mail servers, 257–268
  Samba, 378–379
  software
    in Debian and Ubuntu, 50–52
    in Fedora, 52–54
    in SUSE, 54
    in Xandros, 55

space needed for Linux, 32–34
steps for, 29–31
troubleshooting
about, 40
Knoppix boot commands, 40–43
Linux kernel boot options, 44–47
PC reboot problem, 44
signal 11 error, 43
troubleshooting workstation, 36
working with new drive, 36
integrity, 394
Integrity Check (IC) field, 182
Intel 80x86 processor, 9, 10
in.telnetd, 22
InterBulletin (website), 227
Internet access, in security policy, 387
Internet Assigned Numbers Authority (website), 247
Internet connection
about, 153–154
cable modem
about, 161
how it works, 161–163
setup, 163–166
DSL
about, 156
ADSL, IDSL, SDSL, 157–158
how it works, 156–157
setup, 158–161
ways to connect, 154–155
Internet Control Message Protocol (ICMP), 195
Internet Engineering Task Force (IETF), 170, 280
Internet Message Access Protocol (IMAP), 171
Internet servers, 21–22
Internet services
about, 170–171, 241–242
Internet Super Server, 247–250
port numbers and, 245–247
securing, 413–415
in security policy, 387
standalone servers, 250–255
TCP/IP and sockets, 242–245
Internet subcategory (KickOff), 84
Internet Super Server, 247–250, 414
InterNetNews
components of, 270–274
startup, 274–275
internetworking, 169
intrusion detection, 441
io field name, 321
IP addresses, 169–170
IP routing table, 194
IP spoofing, 394
IPSec (IP Security Protocol), 394
iptables command, 164, 421–422, 424–426
IPv4, 497
IPv6, 170
IPV6INIT parameter, 186
isdn service, 254
Isof, 441
ISOLINUX boot loader, 57, 58
ISP, reading newsgroups from, 222–227
ISS Internet Scanner, 441
IV (Initialization Vector), 182

J

j command, 149
J command, 148
-j option, 477
jobs, scheduling
about, 330–331
one-time, 331–333
recurring, 333–336
jobs command, 454
John the Ripper, 441
join command, 496, 507

K

k command, 149
K Desktop Environment. *See* KDE (K Desktop Environment)
K operator, 265
-k option, 477
k12 category, 220
Kbps, 154
KDE, 11, 60, 61
logging out of, 65
KDE (K Desktop Environment), 20
about, 79–80
configuring
about, 86–88
KDE bottom panel, 85–86
desktop context menus, 80–81
icon context menus, 81–82
panel
about, 82
icons, 85
Main Menu button, 82–85
KDE Akregator (website), 229
KDE calculator, 132–133
key pair, generating, 408–409
KEY resource record type, 291
key ring, 409
keyboard, checking for installation, 31
keys, exchanging, 409–410

keyword, 50–51
KickOff application, 83–85
kill command, 100, 454, 481
klogd daemon, 514
KNode, 223, 226–227
Knopper, Klaus (developer), 12
Knoppix
  about, 12, 14
  boot commands, 40–43
  distributions, 38
  root user, 306
  update-rc.d command, 252–253
knoppix 1 Knoppix boot command, 41
knoppix 2 Knoppix boot command, 41
knoppix acpi=off Knoppix boot command, 41
knoppix atapicd Knoppic boot command, 41
knoppix desktop=wmname Knoppix boot command, 41
knoppix dma Knoppix boot command, 41
knoppix floppyconfig Knoppix boot command, 41
knoppix fromhd=/dev/hda1 Knoppix boot command, 41
knoppix hsync=80 Knoppix boot command, 41
knoppix lang=xx Knoppix boot command, 41
knoppix mem=256M Knoppix boot command, 41
knoppix myconf=/dev/hda1 Knoppix boot command, 41
knoppix myconf=scan Knoppix boot command, 41
knoppix noeject Knoppix boot command, 42
knoppix noprompt Knoppix boot command, 42
knoppix nowheel Knoppix boot command, 42
knoppix noxxx Knoppix boot command, 42
knoppix pci=bios Knoppix boot command, 42
knoppix pnpbios=off Knoppix boot command, 42
knoppix screen=resolution Knoppix boot command, 42
knoppix testcd Knoppix boot command, 42
knoppix tohd=/dev/hda1 Knoppix boot command, 42
knoppix toram Knoppix boot command, 42
knoppix vga=ext Knoppix boot command, 42
knoppix vsync=60 Knoppix boot command, 42
knoppix wheelmouse Knoppix boot command, 42
knoppix xmodule=modname Knoppix boot command, 43
knoppix xserver=progname Knoppix boot command, 43
Konqueror, 209
Kontact app, 132
ks boot option, 45
ks=kickstartfile boot option, 45
ksyms command, 328
KWrite text editor, 140–141

L

l command, 149
-l option, 471, 477
-L option, 471
LAN. *See* local area network (LAN)
LANANA (Linux Assigned Names And Numbers Authority), 326
last command, 499, 516
LC4, 441
lcd command, 239
ldconfig command, 504
ldd command, 100, 504
LDFLAGS variable, 475
least privilege, 345
Leave category (KickOff), 83
less command, 19, 99
less-than sign (<), 93–94
LGPL (GNU Library General Public License), 485, 486–487
/lib directory, 111, 354
libpng software package, 19
LibreOffice (website), 20, 21
LibreOffice Suite, 128–132
licenses, GNU, 484–487
LIDSSystems, 441
LILO boot loader, 57
links, 117, 206–208
Linspire, 12

Linux. *See also specific topics*
about, 1, 9–10
boots
automatically starting servers, 313–314
/etc/inittab file, 310–311
init command, 311–312
init process, 308–310
manually starting/stopping servers, 313
startup scripts, 312–313
commands
about, 62–63, 98–101
date, 103–104
managing processes, 102–103
navigating file system with, 114–121
processing files, 104–106
root, 101–102
time, 103–104
computer systems for, 10
configuration files, 314–317
devices
about, 325–326
device files, 326–327
persistent device naming with udev, 327–328
distributions, 10–14, 16–24
getting started, 27–28
installation and package management, in CompTIA Linux+ Exam, 503–505
installing, 29–36
managing PC with, 24–27
space needed for, 32–34
starting, 57–60
versions, 14–15
web browsing in, 209–216
Linux Assigned Names And Numbers Authority (LANANA), 326
linux category, 220
Linux Essentials Certification Exam
about, 491–492
career, 492–493
command line, 493–496
community, 492–493
operating system, 496–498
security and file permissions, 498–499
Linux IPv6 HOWTO, 170
Linux kernel. *See also* operating system
boot options, 44–47
defined, 9, 57
versions, 14
vulnerability, 434
Linux Professional Institute (LPI), 2
Linux Standard Base (LSB), 15–16, 353
list command, 481
listings
directory, 116–118
files, 495
Live CD, 34, 40
Live version, 14
ln command, 98, 499, 516
loadable driver modules, managing
about, 328
/etc/modprobe.conf file, 329–330
loading/unloading modules, 328–329
loadable kernel modules, 328
loading modules, 328–329
local area network (LAN)
about, 26, 167
configuring TCP/IP networking, 175–177
connecting to Internet, 177–178
Ethernet
about, 172
cables, 173–175
how it works, 172–173
TCP/IP
about, 167–169
Internet services, 170–171
IP, 170
IP addresses, 169–170
port numbers, 170–171
local command, 454
local loop, 156
localhost.zone file, 294–295
locate command, 99, 122
log analysis/monitoring tools, 441
logcheck, 441
logcolorise, 441
logger command, 498, 503
logic bombs, 394
LOGNAME environment variable, 349
logout command, 454
logrotate command, 498, 503
LogWatch, 441
LONE-TAR utility, 362
loop length, 158
loops per jiffy (LPJ), 59
/lost+found directory, 111
lowres boot option, 45
LPI (Linux Professional Institute), 2
LPJ (loops per jiffy), 59
lpq command, 514
lpr command, 99, 514
lprm command, 514
ls command, 98, 116–118, 119, 239, 350, 402, 495, 507
LSB (Linux Standard Base), 15–16

lsmod command, 328, 329, 503
lsof tool, 397
LXO-101 Exam icon, 4
LXO-102 Exam icon, 4

M

M operator, 265
-m option, 478
m4 software package, 19, 260–261
macro processor, 260
macros, 260, 474–475
mail command, 258
MAIL environment variable, 349
mail exchanger, 293
mail servers, 257–268
mail transport agent (MTA), 257
mail user agent (MUA), 257
mail-delivery mechanism, 259
mail-delivery test, 258
MAILER macro, 263
mailq command, 513
Main Menu button (KDE), 82–85
make command, 19, 476–478, 481
make config command, 503
makefile
  about, 473–474
  names, 472–473
  sample, 475–476
man command, 98, 495
managing
  CD-ROMs, 49–50
  CDs, 49–50
  command input/output, 92–94
  file systems
    about, 351–352, 361
    accessing DOS or Windows file system, 368–371
    backing up, 361–367
    /etc/fstab file, 357–358
    hierarchy, 352–355
    mounting devices, 355–357
    sharing with NFS, 358–361
  flash drives, 49–50
  flow in shell scripting, 449–451
  groups, 344–345
  loadable driver modules
    about, 328
    /etc/modprobe.conf file, 329–330
    loading/unloading modules, 328–329
  networks. *See* network management
  PC with Linux, 24–27
  processes, 102–103
  security, 389
  user accounts
    using commands, 342–343
    using GUI user manager, 338–342
Mandrakelinux, 12
Mandriva Linux One, 12
manually starting/stopping servers, 313
MASQUERADE_AS macro, 263
MASQUERADE_DOMAIN macro, 264
masquerading, 423
MATE, 69–70, 76–77
Mbps, 154
-mcpu=cputype option, 471
MD5 encryption, 345, 401
mdelete command, 239
mdir command, 239
media, selecting, 361–362
/media directory, 112, 354
/media/cdrom directory, 96
mediacheck boot option, 45
%MEM heading, 319
Membership, in group definitions, 344
memory field name, 320
mem=xxxM boot option, 45
menus, Firefox, 212
MEPIS Linux, 13, 14
message digest, 393, 401
Metcalfe, Robert (inventor), 173
mget command, 239
Microsoft Excel, 130–131
Microsoft PowerPoint, 131–132
Microsoft Word, 129
misc category, 220
misc.test newsgroup, 226
mitigation approach, 388–389
mkdir command, 98, 120–121, 239
mkfs command, 101
mknod command, 101
mkswap command, 101
mls command, 239
/mnt directory, 112, 354
Mockapetris, Paul (computer scientist), 280
MODE parameter, 186
modinfo command, 328, 503
modprobe command, 328, 329, 330, 503
monitor, checking for installation, 32
monitoring
  system performance
    about, 317
    checking disk performance/disk usage, 321–322

top utility, 318–319
uptime command, 319–320
vmstat utility, 320–321
system security, 412–413
monolithic program, 328
more command, 63, 99
mount command, 101, 356, 357, 358, 368–369, 508
mount point, 123, 356, 360, 376
mounting
CDs/DVDs, 25, 122–123, 355–357
DOS floppy disks, 369–370
DOS/Windows disk partition, 368–369
NFS file systems, 360–361, 376–377
NTFS partition, 370–371
mouse, checking for installation, 32
moving files, 495. *See also* mv command
mput command, 239
MTA (mail transport agent), 257
mta parameter, 271
MUA (mail user agent), 257
multi option, 191
multihomed, 191
multimedia, defined, 127
multimedia applications
about, 133
burning CDs/DVDs, 135–136
digital camera, 133–134
playing audio CDs, 134–135
playing sound files, 135
Multimedia subcategory (KickOff), 84
multivolume archive, backing up and restoring, 365
mv command, 98, 495, 506
MX resource record type, 291, 292

N

-n command, 144
n command, 145
^n command, 145
+n command, 144
-n option, 478
NAME environment variable, 348
NAME parameter, 187
name servers, 280, 283
named daemon, 254, 282–283, 296
namespace, 281
NAT (Network Address Translation), 159–161, 165–166, 421
National Vulnerability Database (NVD) (website), 432
Nautilus File Manager, 135
navigating file system
about, 114
for changing permissions and ownerships, 118–119
for directory listings and permissions, 116–118
for directory navigation, 114–115
for finding files, 121–122
for working with directories, 120–121
for working with files, 119–120
ncurses software package, 19
Nessus Security Scanner, 441
NETBIOS, 171
Netcat, 441
netfilter, 421–422
netstat command, 195, 397, 498, 515
network address, 169, 242
Network Address Translation (NAT), 159–161, 165–166, 421
Network Basic Input/Output System, 378
network card, checking for installation, 32
Network Configuration tool, 185
Network connection icon (KDE), 85
network devices, 327
Network File System (NFS), 26, 373, 434
Network Information Service (NIS), 434
Network layer, of TCP/IP, 168
network management
about, 189
configuring networks at boot time, 199–201
TCP/IP configuration files
about, 189
/etc/host.conf file, 190, 191
/etc/hosts file, 189–190
/etc/hosts.allow file, 190, 192, 315, 435
/etc/hosts.deny file, 190, 192–193, 315, 435
/etc/networks file, 190
/etc/nsswitch.conf file, 190, 193
/etc/resolv.conf file, 190, 191
TCP/IP networks
about, 193
connectivity to host, 195
GUI tools, 198
IP routing table, 194
network interfaces, 193–194
network status, 195–197
sniffing network packets, 197–198

Network News Transfer Protocol (NNTP), 171, 246
network packets, sniffing, 197–198
network security issues, 389, 390–391
network service, 254
networking, in CompTIA Linux+ Exam, 514–515
NETWORKING variable, 199
networks
  about, 21, 26–27
  checking status, 195–197
  information systems, 193
  interfaces, 193–194
  recommended utilities, 441
  security review, 438–439
newaliases command, 513
newgrp command, 499, 512
news, security and, 397–398
news category, 220
news servers, 268–269
news services, 21–22
newsfeeds file, 272–274
newsgroups
  about, 153, 217–218
  adding, 276–277
  hierarchy of, 218–219, 275–276
  Linux-related, 221–222
  posting news, 225–226
  reading and searching at websites, 227–228
  reading from ISP, 222–227
  reading with Thunderbird, 223–225
  setting up, 275–277
  subscriptions to, 225
  testing, 277
  top-level categories of, 219–220
NewsMonster (website), 229
newsreader, 222–223
Newzbot (website), 222
next command, 481
NFR, 441
NFS (Network File System)
  about, 26, 171, 373, 434
  file sharing with
    about, 373–374
    exporting file systems, 374–376
    mounting NFS file systems, 376–377
  port numbers and, 246
  sharing files with
    about, 358–359
    exporting file systems, 360
    mounting NFS file systems, 360–361
nfs service, 254
nfslock service, 255
NFSv4 implementation (website), 374
ngrep, 441
NI heading, 319
nice command, 100
NIS (Network Information Service), 434
nl command, 99
nlog, 441
nmap tool, 397, 441–442
nmbd, 378
nmblookup, 378
nmi_watchdog=1 boot option, 45
nnrpd, 270
NNTP (Network News Transfer Protocol), 171, 246
nntpsend, 270
no_all_squash option, 375
noapic boot option, 45
nofirewire boot option, 45
no_hide option, 375
noht boot option, 45
nomce boot option, 45
nomount boot option, 46
nonrepudiation, 394
nopass boot option, 46
nopcmcia boot option, 46
noprobe boot option, 46
no_root_squash option, 376
noshell boot option, 46
no_subtree_check option, 375
Notifications and Jobs icon (KDE), 85
nousb boot option, 46
nousbstorage boot option, 46
Novell certifications, 518
no_wdelay option, 375
NS resource record type, 291, 292, 295
NTFS partition, mounting, 370–371
ntop, 441
NTP, port numbers and, 246
ntpd, 513
NVD (National Vulnerability Database) (website), 432

O

O operator, 265
-o option, 471, 478
-O option, 471
-O1...O3 options, 471
object files, 469
office applications, 128–132
Office subcategory (KickOff), 84
ONBOOT parameter, 187, 200
10Base2, 173
10Base5, 173
10BaseT, 173
127.0.0.1, reverse mapping, 295
one-time job scheduling, 331–333

online documentation, 22–24
open command, 240
Open Secure Shell (OpenSSH) software, 415–417
Open Secure Sockets Layer (Open SSL), 433
Open Source Initiative (website), 11
open source projects, 11
Open Windows icon (KDE), 85
opening terminal windows and virtual consoles, 89–90
operating system
  defined, 9
  on Linux Essentials Certification Exam, 496–498
  updates to, 434
/opt directory, 112, 354
options statement, 288
Oracle certifications, 518
order option, 191
organization parameter, 271
-Os option, 471
OSTYPE macro, 263
OUTPUT chain, 424–426
ovmethod parameter, 271
ownerships
  changing, 118–119, 350, 402
  viewing, 402

P

p command, 149
P command, 149, 265
-p option, 478
pack command, 496
packet filter firewall, 419
packet filtering
  defined, 394
  enabling, 421–426
packet sniffers, recommended, 441
packets, 173, 394
PAMs (pluggable authentication modules), 401
Pan, 223
Panel toolbox (KDE), 85
panels
  defined, 80
  GNOME, 71–74
  KDE
    about, 82
    icons, 85
    Main Menu button, 82–85
PARC (Xerox Palo Alto Research Center), 173
Partial POSIX Extended Regular Expressions, 495–496
partition, 30
passwd command, 101, 397, 516
password, /etc/passwd file, 343
Password, in group definitions, 344
password security, 437
password-checking tools, recommended, 441
paste command, 99, 496, 507
patch command, 19, 99
PATH environment variable, 348, 349, 447–448
pathhost parameter, 271
pathname, 110–111
pathnews parameter, 272
patterns, newsgroup, 273
PC
  checking hardware for installation, 31–33
  managing with Linux, 24–27
  reboot problem, 44
pci=biosirq boot option, 46
pci=noacpi boot option, 46
pcmcia service, 255
-pedantic option, 471
PEERDNS parameter, 187
penetration test, 439
Pentium processor, 9, 10
performing risk analysis, 386–387
perimeter network, 394
period (.), 111, 119–120, 144, 494
peripheral, defined, 24
peripheral devices, 25–26
Perl, 23
permissions
  changing, 118–119, 403
  checking for set user ID, 405
  directory, 116–118
  security and file, 498–499
  setting default, 403–405
  viewing, 402
persistent device naming with udev, 327–328
-pg option, 471
physical console, 89
Physical layer, of TCP/IP, 168
PID heading, 319
ping command, 195, 441, 498, 515
pipe (|), 92, 105, 496
pkgname, 50
PKI (Public Key Infrastructure), 395
playing
  audio CDs, 134–135
  sound files, 135
pluggable authentication modules (PAMs), 401
plug-ins, 137
Point-to-Point Protocol (PPP), 327

POP3 (Post Office Protocol version 3), 171
popd command, 454
port numbers, 170–171, 242, 245–247, 439
port scanners
  defined, 394
  recommended, 441
  tool, 413
portmap service, 255
ports, 208
post access, 274
Post Office Protocol version 3 (POP3), 171
posting to newsgroups, 225–226
pound sign (#), 189, 208, 264, 271, 438. *See also* hash mark (#)
PowerPoint (Microsoft), 131–132
PPP (Point-to-Point Protocol), 327
PPP over Ethernet (PPPoE), 161
PPPoE (PPP over Ethernet), 161
PR heading, 319
prevention concept, in security programs, 434, 438
primary name server, configuring, 296–298
print command, 481
printenv command, 100, 347
printers
  about, 25
  checking for installation, 32
  setting up, 47–49
/proc directory, 112
/proc file system, 322–325
process field, 311
process file system, 323
process ID (PID), 102–103, 309
processes, managing, 102–103
processing files, 104–106
processors, 9, 10, 32
procs field name, 320
programming
  about, 465–467
  software-development tools
    about, 467
    C++ compiler, 468–472
    GNU C compiler, 468–472
    GNU debugger, 478–484
    GNU make utility, 472–478
    implications of GNU licenses, 484–487
prompt command, 240
protection concept, in security programs, 389, 431
protocols
  about, 207–208
  connectionless, 243
  connection-oriented, 242–243
  defined, 168
proxy server, 395
ps command, 63, 100, 496, 506
pstree command, 100
PTR resource record type, 291, 292
public key cryptography, 395
public key encryption, 406–407
Public Key Infrastructure (PKI), 395
pushd command, 454
put command, 240
pwd command, 98, 115, 240, 454, 506
Python, 23

Q

q command, 144, 145
:q command, 149
:q! command, 149
Q command, 145
-q option, 478
QTParted tool, 30
question marks (?), 95–96, 208, 494
quit command, 240, 481
quota command, 508
quotation marks, 91

R

r file command, 145
:r filename command, 149
R operator, 265
-r option, 478
-R option, 478
RAM (random-access memory), 24, 32
random-access memory (RAM), 24, 32
RATE parameter, 187
RC4 encryption algorithm, 182
RCS (Revision Control System), 23
rcs software package, 19
reaction concept, in security programs, 389
read access, 274
read command, 454, 509
readers.conf file, 274
reading. *See* newsgroups; RSS feeds
read_input function, 482
readonly command, 454
real-time clock, 513
reboot command, 100
reboot=b boot option, 46

rec category, 220
Recently Used category (KickOff), 83
reconfiguring BIOS, 36
recurring job scheduling, 333–336
recv command, 240
Red Hat certifications, 518
Red Hat Linux 9, 12
Red Hat Package Manager (RPM), 12, 52
regular expressions, 494
Remember icon, 4
remote access, 154
remote login, 21–22
rename command, 240
Rendezvous Directory Service, 171
repeating previously typed commands, 97
RES heading, 319
rescue boot option, 46
resolution=HHHxVVV *boot option,* 46
resolver, configuring, 285–286
resolver library, 280, 283
resource records (RR), 289–293
resources, Samba, 380
response concept, in security programs, 389, 432, 434, 438
responsibilities, in security policy, 387
Retina Network Security Scanner, 441
return command, 454
reverse mapping, zone file for, 295
Revision Control System (RCS), 23
rewinding tape device, 366
Rhythbox, 135
risk analysis, 386–387
rm command, 98, 120, 495, 506
RM variable, 475
rmdir command, 98, 240, 506
rmmod command, 328, 503
rndc utility, 288
ro option, 375
root, becoming
  about, 101–102, 305
  forgotten root password, 307–308
  for GUI utilities, 307
  su - command, 306
root access, 64
/root directory, 111, 112, 354
root domain, 281
root password, 55
root username, 58–59
root zone file, 293–294
root_squash option, 375
route utility, 498, 515
RP resource record type, 291
RPM (Red Hat Package Manager), 12, 52
RR (resource records), 289–293
RSS feeds
  about, 217–218, 228–229
  reading, 229–230
rsyslogd file, 413
run command, 481
runlevel command, 310, 311
running
  gbd, 479–481
  make, 476–478
rw option, 375
rwx group, 117–118
rx command, 148

S

S heading, 319
S operator, 265
-s option, 478
SAINT, 441
Samba
  about, 26, 373
  configuring, 379–380
  installing, 378–379
  resources, 380
  setting up Windows server using, 377–380
  trying, 380
samba service, 255
SANS Institute, 432
SARA, 441
saving error messages in files, 94
/sbin directory, 112, 354
/sbin/hdparm program, 321
/sbin/hotplug program, 327
/sbin/ifconfig command, 193
/sbin/route command, 194
/sbin/udev program, 327
sbpcd file, 92
scanlogd, 441
scheduling jobs
  about, 330–331
  one-time, 331–333
  recurring, 333–336
sci category, 220
screening router, 395
screenshot, 51
script directories, for cron jobs, 336
scripting, 445
scripts, writing, 106–108
SCSI controller, checking for installation, 32
SDSL (Symmetric DSL), 157

- searching newsgroups at websites, 227–228
- `secure` option, 375
- Secure Shell, 391
- Secure Shell (SSH), 171, 246
- Secure Sockets Layer (SSL), 267
- security
  - about, 383, 399
  - in CompTIA Linux+ Exam, 515–516
  - defined, 25
  - encrypting and signing files with GnuPG
    - digital signatures, 407–408
    - GPG, 408–412
    - public key encryption, 406–407
  - files and directories
    - about, 402
    - changing file ownership, 402
    - changing file permissions, 403
    - checking for set user ID permission, 405
    - setting default permission, 403–405
    - viewing ownerships and permissions, 402
  - files to be aware of, 426–427
  - framework for
    - about, 384–385
    - business requirements, 385–386
    - managing, 389
    - mitigation, 388–389
    - policy, 387–388
    - risk analysis, 386–387
  - importance of, 383–384
  - Internet services
    - about, 413
    - configuring Internet super server, 414
    - configuring TCP wrapper, 414–415
    - standalone services, 413–414
  - monitoring for system, 412–413
  - news, 397–398
  - passwords
    - about, 399–400, 437
    - pluggable authentication modules (PAMs), 401
    - shadow, 400–401
  - securing the Linux system, 389–391
  - setting up firewalls
    - about, 418–421
    - enabling packet filtering, 421–426
    - NAT, 421
  - terminology and tools, 391–397
  - testing tools, 440–442
  - updates, 397–398
  - using Secure Shell (SSH) for remote logins, 415–417
- security audits, in security policy, 387. *See also* computer security audit
- security level configuration tool, 422–424
- security policy, 384, 387–388
- `sed` command, 19, 99, 457–463, 507
- Segment Violation Signal (SIGSEGV), 43
- selecting backup strategies and media, 361–362
- `selinux=0` boot option, 46
- semicolon (;), 380, 451, 496
- `send` command, 240
- `sendmail`
  - about, 22
  - alias file, 268
  - configuration file, 259–264
  - files, 266–267
  - macro, 261–264
  - mail transport agent, 259
  - service, 255, 438, 513
  - using, 257–258
  - vulnerability, 433
- `sendmail/cf` file, 264–265
- `serial` boot option, 46
- `server` command, 250–251
- server process, 171, 246
- servers, automatically starting, 313–314
- services, Internet. *See* Internet services
- `set` command, 454, 509
- `set variable` command, 481
- setting default permission, 403–405
- `setuid` program, 395, 437
- setup
  - Ethernet LAN, 172–175
  - local newsgroups, 275–277
  - printers, 47–49
  - Windows server with Samba, 377–380
- `SGID` permission, 499
- SHA-512, 345
- shadow passwords, 400–401
- `-shared` option, 472
- sharing files. *See* file sharing
- Sharutils software package, 19
- shell
  - about, 17, 60, 89
  - `bash`
    - about, 90
    - asterisks, 95–96
    - automatic command completion, 95
    - combining commands, 92

command syntax, 90–92
controlling command input and output, 92–94
question marks, 95–96
repeating previously typed commands, 97
commands, 61–62
Linux commands
about, 62–63, 98–101
date, 103–104
managing processes, 102–103
processing files, 104–106
root, 101–102
time, 103–104
opening terminal windows and virtual consoles, 89–90
starting bash shell, 60–61
writing scripts, 106–108
shell, /etc/passwd file, 344
shell command, 481
SHELL environment variable, 349
shell functions, 448–449
shell scripting
about, 445–447
advanced
about, 463
awk, 459–463
sed, 457–463
basics of
about, 447
bash built-in commands, 451–455
controlling flow, 449–451
shell functions, 448–449
storing, 447–448
shell scripts, 61, 445, 509–510
shift command, 454
SHR heading, 319
shutdown command, 100, 503
shutting down, 64–65
Shuttleworth, Mark (developer), 74
SIG resource record type, 291
signal 11 error, 43, 115
SIGSEGV (Segment Violation Signal), 43
simple file, 107
Simple Mail Transfer Protocol (SMTP), 171, 246
Simple Network Management Protocol (SNMP), 171, 246, 433
single dot (.), 115
single-volume archive, backing up and restoring, 363–365
size command, 240
skipddc boot option, 46
Slackware Linux, 13
slash (/), 116, 143, 147, 352–353
Slashdot (website), 230
smbadduser, 378
smbcacls, 378
smbclient, 378
smbcontrol, 378
smbd, 378
smbmount, 378
smbpasswd, 378
smbprint, 378
smbstatus, 378
smbtar, 378
smbumount, 378
SMTP (Simple Mail Transfer Protocol), 171, 246
sniffer, defined, 395
sniffing network packets, 197–198
sniffit, 441
SNMP (Simple Network Management Protocol), 171, 246, 433
snmpd service, 255
Snort, 441
SOA resource record type, 291, 295
soc category, 220
sockets
client/server model and, 243–244
TCP/IP and, 242–245
software
defined, 9
GNU, 16–19
installing
about, 50
in Debian and Ubuntu, 50–52
in Fedora, 52–54
in SUSE, 54
in Xandros, 55
software development, 22
software-development tools
about, 467
C++ compiler, 468–472
GNU C compiler, 468–472
GNU debugger, 478–484
GNU make utility, 472–478
implications of GNU licenses, 484–487
s/old/new/ command, 145
sort command, 99, 105
sorting text files, 105
sound card
about, 26
checking for installation, 32
sound files, playing, 135
source code, 11
spamassassin service, 255
split command, 99, 106
splitting files into smaller files, 106
spoofing, 418
spyware, 395
square brackets ([]), 95–96
/srv directory, 112, 354
SRV resource record type, 291

- `ssh`, 391
- SSH (Secure Shell), 171, 246
- `ssh` service, 255
- `sshd`, 22
- SSL (Secure Sockets Layer), 267
- stable releases, 15
- standalone servers
  - about, 250
  - starting and stopping manually, 250–251
  - starting automatically, 251–255
  - turning off, 413–414
- standalone services, security of, 438–439
- standard error, 92–94
- standard input, 92–94
- standard output, 92–94
- standards, wireless Ethernet, 179–180
- starting
  - `bash` shell, 60–61
  - caching name server, 295–296
  - INN server, 269–275
  - Linux, 57–60
- STARTTLS extension, 267
- startup, InterNetNews, 274–275
- startup scripts, 312–313
- stateful inspection firewall, 419–420
- statements, 466
- statistical attacks, 182
- status bar, Firefox, 211
- `status` command, 240
- `stderr`, 92–94, 496
- `stdin`, 92–94, 496
- `stdout`, 92–94
- `step` command, 481
- sticky keys, 511
- storing, in shell scripting, 447–448
- stream cipher, 182
- stream editor, 458
- stream-based macro, 260
- `/string` command, 149
- `?string` command, 149
- Strobe, 441
- `su` command, 101, 102, 306, 397, 499, 516
- subscriptions, to newsgroups, 225
- substituting characters from files, 105–106
- `subtree_check` option, 375
- Subversion, Concurrent Versions System (CVS), 23, 433
- subversion, vulnerability, 433
- `sudo` tool, 397, 499, 516
- `suid` permission, 405
- superuser, 101–102, 516
- SUSE
  - about, 13
  - BIND, 282
  - certifications from, 518
  - `chkconfig` command, 251–252, 314
  - configuring
    - networks, 200
    - Samba, 379
    - wireless network in, 187
  - `directory` option, 288
  - distributions, 38
  - `/etc/fstab` file, 357–358
  - `init` process, 310
  - installing
    - INN server, 268
    - software in, 54
    - software-development tools, 467
  - reading RSS feeds, 230
  - recovering from forgotten `root` password, 308
  - security level configuration tool, 422, 423
  - starting
    - `innd`, 275
    - NFS server, 359, 376
    - Samba, 380
    - standalone servers in, 251
  - stopping standalone servers in, 251
  - turning off standalone servers, 414
  - updates, 398
- `swap` field name, 321
- `swapoff` command, 101
- `swapon` command, 101
- swapping, 321
- Swatch, 441
- symbolic links, 117, 252, 516
- Symmetric DSL (SDSL), 157
- symmetric key encryption, 395
- Synaptic Package Manager, 51
- `sync` command, 101, 375
- `/sys` directory, 112
- `sysfs` program, 327
- `syslog` service, 255
- `syslogd` file, 413
- system administration
  - about, 25
  - becoming `root`
    - about, 305
    - forgotten `root` password, 307–308
    - for GUI utilities, 307
    - `su` command, 306
  - GUI tools, 303–305
  - Linux boots
    - automatically starting servers, 313–314
    - `/etc/inittab` file, 310–311
    - `init` command, 311–312
    - `init` process, 308–310
    - manually starting/stopping servers, 313
    - startup scripts, 312–313
  - Linux configuration files, 314–317

Linux devices
about, 325–326
device files, 326–327
persistent device naming with udev, 327–328
managing loadable driver modules
about, 328
/etc/modprobe.conf file, 329–330
loading/unloading modules, 328–329
monitoring system performance, 318–321
scheduling jobs
about, 330–331
one-time, 331–333
recurring, 333–336
tasks, 301–302
viewing system information with /proc file system, 322–325
system architecture, in CompTIA Linux+ Exam, 502–503
system field name, 321
system information, viewing with /proc file system, 322–325
system performance, monitoring
about, 317
checking disk performance/disk usage, 321–322
top utility, 318–319
uptime command, 319–320
vmstat utility, 320–321
System subcategory (KickOff), 84

T

T operator, 265
-t option, 478
tabbed browsing, 210
tac command, 99
tail command, 99
talk category, 220
tapes, backing up on, 366
tar command, 19, 100, 363–367, 496
tasks
by category in YaST Control Center, 305
system administration, 301–302
TCP wrapper
access control files, 391
security, configuring, 414–415
tcpdstats, 441
tcpdump command, 197, 441
TCP/IP (Transmission Control Protocol/ Internet Protocol)
about, 21, 167–169
configuration files
about, 189
/etc/host.conf file, 190, 191
/etc/hosts file, 189–190
/etc/hosts.allow file, 190, 192
/etc/hosts.deny file, 190, 192–193
/etc/networks file, 190
/etc/nsswitch.conf file, 190, 193
/etc/resolv.conf file, 190, 191
configuring networking, 175–177
four-layer network model, 168–169
Internet services, 170–171
IP, 170
IP addresses, 169–170
networks
about, 193
checking connectivity to host, 195
checking IP routing table, 194
checking network interfaces, 193–194
checking network status, 195–197
sniffing network packets, 197–198
using GUI tools, 198
port numbers, 170–171
sockets and, 242–245
Technical Stuff icon, 4
telinit command, 312
telnet, 515
TELNET, 171, 246
Temporal Key Integrity Protocol (TKIP), 181
TERM environment variable, 348, 349
terminal, 89
terminal windows, opening, 89–90
test command, 449–450, 509
testing
caching name server, 295–296
newsgroups, 277
vulnerabilities, 440
testing releases, 15
testparm command, 378, 379

texinfo software package, 19
/text/ command, 144
?text? command, 144
text editors
  about, 139, 467
  with ed, 141–145
  GUI, 139–141
  with vi, 141, 145–149
text files
  counting words/lines in, 104–105
  sorting, 105
text input mode, 142, 146
text mode installation, 38
text terminal, 16
TFTP (Trivial File Transfer Protocol), 171, 246
thick Ethernet, 173
thicknet, thickwire, 173
thinwire, 173
32-bit cyclic redundancy code (CRC-32), 182
threats
  defined, 395
  typical, 386
Thunderbird, 222, 223–225
tilde (~) character, 115, 146
time, working with, 103–104
time command, 19, 103–104
TIME+ heading, 319
Time icon (KDE), 85
Time To Live (TTL), 289
times command, 454
Tip icon, 4
TKIP (Temporal Key Integrity Protocol), 181
/tmp directory, 112, 354
Tool Command Language and graphical toolkit (Tcl/Tl), 23
toolbars, Firefox, 211
tools, GUI, 198, 303–305
Tools menu (Firefox), 212
top command, 100, 496, 506
Top Cyber Security Risks index (website), 432
top panel (GNOME), 72
top utility, 318–319
top-level directories, 111–112
Torvalds, Linus (Linux developer), 1, 10
touch command, 98
tr command, 99, 105–106, 507
traceroute, 515
Transmission Control Protocol/Internet Protocol. *See* TCP/IP (Transmission Control Protocol/Internet Protocol)
Transport layer, of TCP/IP, 168
trap command, 454
Tripwire, 390, 412, 441
Trivial File Transfer Protocol (TFTP), 171, 246
Trojan horse, 396
troubleshooting
  about, 37
  CD-ROMs, 49–50
  DVDs, 49–50
  flash drives, 49–50
  installation
    about, 40
    Knoppix boot commands, 40–43
    Linux kernel boot options, 44–47
    PC reboot, 44
    signal 11 error, 43
  printer setup, 47–49
  text mode installation, 38
  workstation, 36
  X, 38–40
Ts, Jay (author)
  Using Samba, 3rd Edition, 380
TTL (Time To Live), 289
tty command, 101
turning off standalone servers, 413–414
24-bit IV, 182
TXT resource record type, 291
type command, 455
TYPE parameter, 187

U

u command, 145, 149
U command, 149
Ubuntu
  BIND, 282
  configuring networks, 199
  distributions, 38
  init process, 310
  installing
    INN server, 268
    Samba, 379
    software in, 50–52
  starting
    innd, 275
    Samba, 380
    standalone servers in, 251
  stopping standalone servers in, 251
  turning off standalone servers, 413
  update-rc.d command, 252–253, 314
  updates, 398
Ubuntu Linux, 13
udev, persistent device naming with, 327–328
UDP (User Datagram Protocol), 243
UID, /etc/passwd file, 343
ulimit command, 397, 455, 516
-UMACRO option, 472
umask command, 403–405, 455

umount command, 101, 356–357
unalias command, 98, 455
uname command, 62, 100
unauthorized access, 386
uncompress command, 100
Uniform Resource Locators (URLs), 206–208
uniq command, 99
United States Computer Emergency Readiness Team (website), 397
Unity, 69–70, 74–75
Unix commands, in CompTIA Linux+ Exam, 505–507
Unix-to-Unix Copy Protocol (UUCP), 218
unloading modules, 328–329
unmounting, 122–123, 369–370
unset command, 455, 509
unshielded twisted-pair cable (UTP), 173
unstable releases, 15
update-rc.d command, 252–253, 314
updates, security, 397–398
updating
  configuration lists, 276
  operating systems, 434
upstream data, 161
uptime command, 319–320, 516
URLs (Uniform Resource Locators), 206–208
USB interface, 26
US-CERT National Cyber Alert System mailing list, 398
Usenet, 218
Usenet Replayer (website), 227
user command, 240
User Datagram Protocol (UDP), 243
user file-creation mask, 403
USER heading, 319
user interfaces and desktops, in CompTIA Linux+ Exam, 510–511
useradd command, 499, 512
USERCTL parameter, 187
userdel command, 499
usermod tool, 397, 516
username, /etc/passwd file, 343
users. *See also* groups
  about, 337
  adding user accounts
    about, 337–338
    using commands, 342–343
    using GUI user manager, 338–342
  administration values, 345–347
  changing ownership of files, 350
  environment, 347–349
  /etc/passwd file, 343–344
Using Samba, 3rd Edition (Ts, Eckstein & Collier-Brown), 380
/usr directory, 112, 113, 354
/usr/bin/scp, 416
/usr/bin/slogin, 416
/usr/bin/ssh, 416
/usr/bin/ssh-keygen, 416
/usr/sbin/sshd, 416
utilities, commercial backup, 362–363. *See also specific utilities*
Utilities subcategory (KickOff), 84
utility programs, 284–285
UTP (unshielded twisted-pair cable), 173
UUCP (Unix-to-Unix Copy Protocol), 218

V

V operator, 265
-v option, 472, 478
/var directory, 112, 113–114, 354
variables
  about, 474–475
  defined, 466
/var/log dmesg file, 317
/var/log file, 412, 436
/var/log/apache2 file, 317
/var/log/boot.msg file, 317
/var/log/cron file, 317
var/log/httpd file, 317
/var/log/lastlog file, 436
/var/log/messages file, 317, 436, 503
/var/log/wtmp file, 436
vendor-neutral certifications, 517
vendor-specific certifications, 518
version-control system, 467
VERSIONID macro, 263
versions, 14–15, 228
Vetescan, 441
vi text editor, 141, 145–149, 467
video card, checking for installation, 32
View menu (Firefox), 212
viewing
  online help in vi, 147
  ownerships and permissions, 402
  system information with /proc file system, 322–325
VIRT heading, 319
virtual consoles, opening, 89–90

virus, 396
visual command mode, 146
vmstat utility, 320–321
vnc boot option, 46
Volume icon (KDE), 85
vsftpd service, 22, 255
vulnerabilities
  common, 432–434
  defined, 396
  testing, 440
  typical, 387
vulnerability scanners, 439, 441

W

w command, 149, 499
w file command, 145
W file command, 145
:w filename command, 149
-w option, 472, 478
-W option, 478
-W1 option, 472
wait command, 455
WAP (wireless access point), 184
war-dialing, 396
war-driving, 396
Warning! icon, 4
watch command, 481
wc command, 100, 104–105, 496, 507
wdelay option, 375
Web, 153
web browsing
  about, 205–206
  links, 206–208
  in Linux, 209–216
  URLs, 206–208
  web servers, 209
web servers, 209
web surfing, in Firefox, 215–216
websites. *See also specific websites*
  distance limits, 154
  downstream transfer speed, 163
  reading newsgroups at, 227–228
  searching newsgroups at, 227–228
  signal 11 error, 43
WEP (Wired Equivalent Privacy), 181–182
whatis command, 98, 495
where command, 481
whereis command, 99
which command, 99
Whisker, 441
white box, 440
who command, 499, 516
Wi-Fi (Wireless Fidelity), 179
Wi-Fi Alliance (website), 180
Wi-Fi Protected Access (WPA), 181
Wi-Fi Protected Access 2 (WPA2), 181
wildcards, 95–96, 494
wildmat pattern, 273
WIMP (windows, icons, mouse, and pointer), 60
winbind service, 255
winbindd, 378
windows, icons, mouse, and pointer (WIMP), 60
Windows file system, accessing, 368–371
Windows partition, reducing size of, 33–34
Windows server, setting up using Samba, 377–380
Wired Equivalent Privacy (WEP), 181–182
wireless access point (WAP), 184
wireless Ethernet networks
  about, 179–180
  ad hoc modes, 180–181
  infrastructure, 180–181
  Wired Equivalent Privacy (WEP), 181–182
wireless local area network (WLAN), 179
wireless networks
  configuring
    about, 185–188
    wireless access point, 184
  setting up wireless hardware, 183–184
  wireless Ethernet networks
    about, 179–180
    ad hoc modes, 180–181
    infrastructure, 180–181
    Wired Equivalent Privacy (WEP), 181–182
Wireshark (website), 198
WLAN (wireless local area network), 179
Word (Microsoft), 129
working configuration file, 39
workstation, troubleshooting, 36
World Wide Web, 21–22
worm, 396
WPA (Wi-Fi Protected Access), 181
:wq command, 149
Writer (LibreOffice.org), 129
writing shell scripts, 106–108

X

X, troubleshooting, 38–40
x command, 148, 481
X Window System, 11, 19–20

Xandros
about, 12
BIND, 282
configuring networks, 199
distributions, 38
init process, 310
installing
INN server, 268
software in, 55
software-development tools, 467
starting
innd, 275
NFS server, 359
Samba, 380
turning off standalone servers, 413
update-rc.d command, 252–253, 314
updates, 398
Xandros Desktop OS, 13
Xandros File Manager, 135
xargs command, 496
xdpyinfo utility, 510
Xerox Palo Alto Research Center (PARC), 173
XFree86 Project, 20
xfs service, 255
xinetd server, 248–250, 255, 413, 415, 438
XMMS, 135
xwininfo utility, 510

Y

YaST Control Center, 187, 259, 282, 303–305
Yellowdog Update, Modified (yum), 505
ypbind service, 255
yy command, 149

Z

zcat command, 100
zless command, 100
zmore command, 100
zone statement, 288, 293–294
zone transfer, 283

About the Author

Emmett Dulaney is the author of several books on operating systems and certifications, and an associate professor at Anderson University. Other books he has written include *CompTIA A+ Complete Study Guide* (Sybex) and the CompTIA *Security + Study Guide* (Sybex).

Emmett is a columnist for *Certification Magazine* and *Campus Technology*. He is also contributor to a number of other magazines.

Dedication

For Karen, Kristin, Evan, and Spencer.

Author's Acknowledgments

I would like to thank Naba Barkakati, who wrote the first two editions and did a fantastic job of condensing a wealth of information into a small tome.

I would also like to thank Elizabeth Zinkann for being one of the best technical editors in the business, and Pat O'Brien for keeping everything on track and on time.

Publisher's Acknowledgments

Senior Acquisitions Editor: Kyle Looper

Project Editor: Pat O'Brien

Copy Editor: Barry Childs-Helton

Technical Editor: Elizabeth Zinkann

Editorial Assistant: Annie Sullivan

Sr. Editorial Assistant: Cherie Case

Project Coordinator: Patrick Redmond

Cover Image: ©iStock.com/Coldimages

Apple & Mac

iPad For Dummies,
6th Edition
978-1-118-72306-7

iPhone For Dummies,
7th Edition
978-1-118-69083-3

Macs All-in-One
For Dummies, 4th Edition
978-1-118-82210-4

OS X Mavericks
For Dummies
978-1-118-69188-5

Blogging & Social Media

Facebook For Dummies,
5th Edition
978-1-118-63312-0

Social Media Engagement
For Dummies
978-1-118-53019-1

WordPress For Dummies,
6th Edition
978-1-118-79161-5

Business

Stock Investing
For Dummies, 4th Edition
978-1-118-37678-2

Investing For Dummies,
6th Edition
978-0-470-90545-6

Personal Finance
For Dummies, 7th Edition
978-1-118-11785-9

QuickBooks 2014
For Dummies
978-1-118-72005-9

Small Business Marketing
Kit For Dummies,
3rd Edition
978-1-118-31183-7

Careers

Job Interviews
For Dummies, 4th Edition
978-1-118-11290-8

Job Searching with Social
Media For Dummies,
2nd Edition
978-1-118-67856-5

Personal Branding
For Dummies
978-1-118-11792-7

Resumes For Dummies,
6th Edition
978-0-470-87361-8

Starting an Etsy Business
For Dummies, 2nd Edition
978-1-118-59024-9

Diet & Nutrition

Belly Fat Diet For Dummies
978-1-118-34585-6

Mediterranean Diet
For Dummies
978-1-118-71525-3

Nutrition For Dummies,
5th Edition
978-0-470-93231-5

Digital Photography

Digital SLR Photography
All-in-One For Dummies,
2nd Edition
978-1-118-59082-9

Digital SLR Video &
Filmmaking For Dummies
978-1-118-36598-4

Photoshop Elements 12
For Dummies
978-1-118-72714-0

Gardening

Herb Gardening
For Dummies, 2nd Edition
978-0-470-61778-6

Gardening with Free-Range
Chickens For Dummies
978-1-118-54754-0

Health

Boosting Your Immunity
For Dummies
978-1-118-40200-9

Diabetes For Dummies,
4th Edition
978-1-118-29447-5

Living Paleo For Dummies
978-1-118-29405-5

Big Data

Big Data For Dummies
978-1-118-50422-2

Data Visualization
For Dummies
978-1-118-50289-1

Hadoop For Dummies
978-1-118-60755-8

Language & Foreign Language

500 Spanish Verbs
For Dummies
978-1-118-02382-2

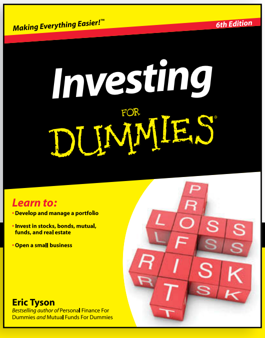
English Grammar
For Dummies, 2nd Edition
978-0-470-54664-2

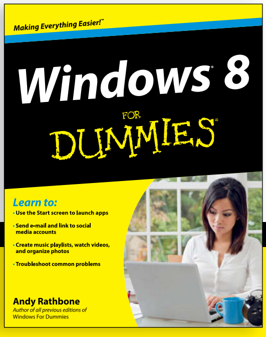
French All-in-One
For Dummies
978-1-118-22815-9

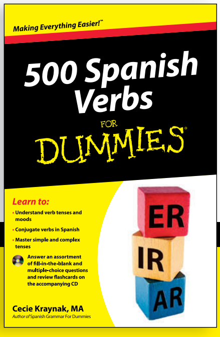
German Essentials
For Dummies
978-1-118-18422-6

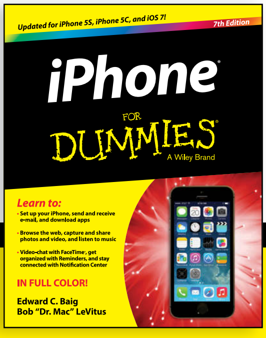
Italian For Dummies,
2nd Edition
978-1-118-00465-4

Available in print and e-book formats.









Available wherever books are sold. **For more information or to order direct visit www.dummies.com**

Math & Science

Algebra I For Dummies, 2nd Edition
978-0-470-55964-2

Anatomy and Physiology For Dummies, 2nd Edition
978-0-470-92326-9

Astronomy For Dummies, 3rd Edition
978-1-118-37697-3

Biology For Dummies, 2nd Edition
978-0-470-59875-7

Chemistry For Dummies, 2nd Edition
978-1-118-00730-3

1001 Algebra II Practice Problems For Dummies
978-1-118-44662-1

Microsoft Office

Excel 2013 For Dummies
978-1-118-51012-4

Office 2013 All-in-One For Dummies
978-1-118-51636-2

PowerPoint 2013 For Dummies
978-1-118-50253-2

Word 2013 For Dummies
978-1-118-49123-2

Music

Blues Harmonica For Dummies
978-1-118-25269-7

Guitar For Dummies, 3rd Edition
978-1-118-11554-1

iPod & iTunes For Dummies, 10th Edition
978-1-118-50864-0

Programming

Beginning Programming with C For Dummies
978-1-118-73763-7

Excel VBA Programming For Dummies, 3rd Edition
978-1-118-49037-2

Java For Dummies, 6th Edition
978-1-118-40780-6

Religion & Inspiration

The Bible For Dummies
978-0-7645-5296-0

Buddhism For Dummies, 2nd Edition
978-1-118-02379-2

Catholicism For Dummies, 2nd Edition
978-1-118-07778-8

Self-Help & Relationships

Beating Sugar Addiction For Dummies
978-1-118-54645-1

Meditation For Dummies, 3rd Edition
978-1-118-29144-3

Seniors

Laptops For Seniors For Dummies, 3rd Edition
978-1-118-71105-7

Computers For Seniors For Dummies, 3rd Edition
978-1-118-11553-4

iPad For Seniors For Dummies, 6th Edition
978-1-118-72826-0

Social Security For Dummies
978-1-118-20573-0

Smartphones & Tablets

Android Phones For Dummies, 2nd Edition
978-1-118-72030-1

Nexus Tablets For Dummies
978-1-118-77243-0

Samsung Galaxy S 4 For Dummies
978-1-118-64222-1

Samsung Galaxy Tabs For Dummies
978-1-118-77294-2

Test Prep

ACT For Dummies, 5th Edition
978-1-118-01259-8

ASVAB For Dummies, 3rd Edition
978-0-470-63760-9

GRE For Dummies, 7th Edition
978-0-470-88921-3

Officer Candidate Tests For Dummies
978-0-470-59876-4

Physician's Assistant Exam For Dummies
978-1-118-11556-5

Series 7 Exam For Dummies
978-0-470-09932-2

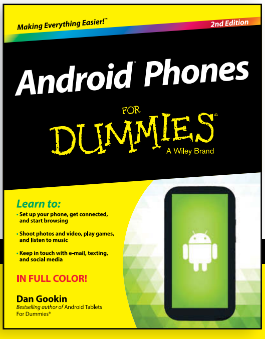
Windows 8

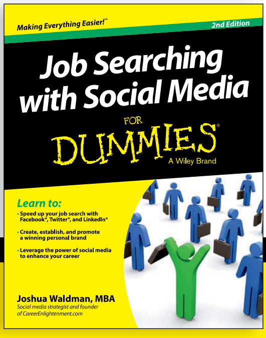
Windows 8.1 All-in-One For Dummies
978-1-118-82087-2

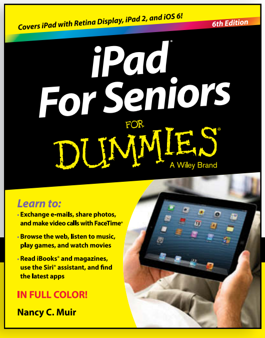
Windows 8.1 For Dummies
978-1-118-82121-3

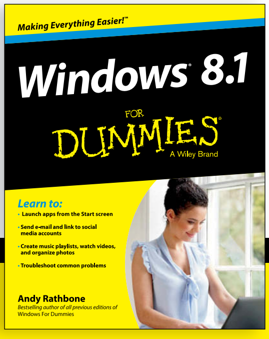
Windows 8.1 For Dummies, Book + DVD Bundle
978-1-118-82107-7

Available in print and e-book formats.









Available wherever books are sold. **For more information or to order direct visit www.dummies.com**

Take Dummies with you everywhere you go!

Whether you are excited about e-books, want more from the web, must have your mobile apps, or are swept up in social media, Dummies makes everything easier.

Visit Us
bit.ly/JE00

Like Us
on.fb.me/1f1ThNu

Follow Us
bit.ly/ZDytkR

Watch Us
bit.ly/gbOQHn

Join Us
linkd.in/1gurkMm

Pin Us
bit.ly/16caOLd

Circle Us
bit.ly/1aQTuDQ

Shop Us
bit.ly/4dEp9

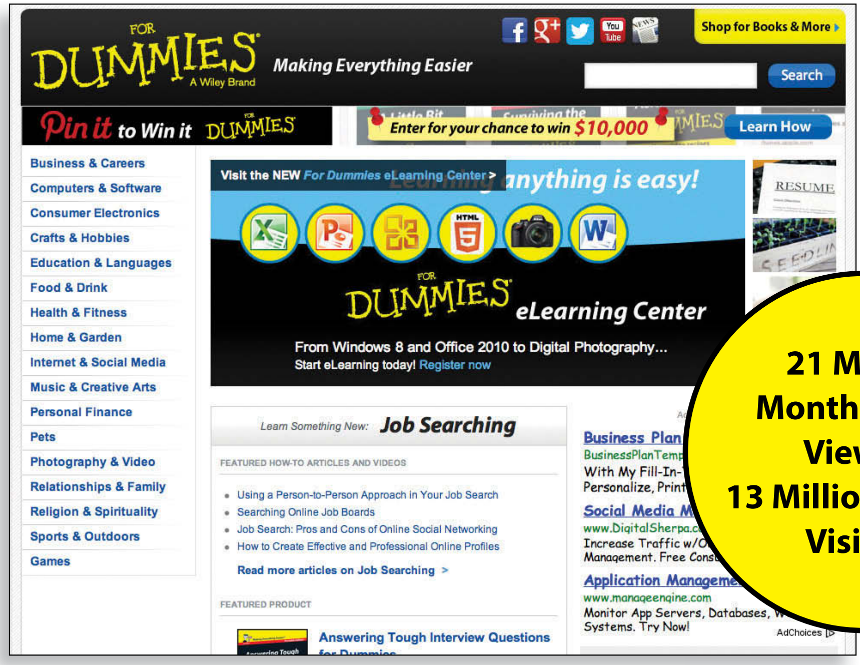
Leverage the Power

For Dummies is the global leader in the reference category and one of the most trusted and highly regarded brands in the world. No longer just focused on books, customers now have access to the For Dummies content they need in the format they want. Let us help you develop a solution that will fit your brand and help you connect with your customers.

Advertising & Sponsorships

Connect with an engaged audience on a powerful multimedia site, and position your message alongside expert how-to content.

Targeted ads • Video • Email marketing • Microsites • Sweepstakes sponsorship



From Windows 8 and Office 2010 to Digital Photography…
Start eLearning today! Register now

Learn Something New: Job Searching

FEATURED HOW-TO ARTICLES AND VIDEOS

- Using a Person-to-Person Approach in Your Job Search
- Searching Online Job Boards
- Job Search: Pros and Cons of Online Social Networking
- How to Create Effective and Professional Online Profiles

Read more articles on Job Searching >

FEATURED PRODUCT

Answering Tough Interview Questions for Dummies

Business Plan
BusinessPlanTemp
With My Fill-In-
Personalize, Print

Social Media M
www.DigitalSherpa.c
Increase Traffic w/O
Management. Free Cons

Application Manageme
www.manageengine.com
Monitor App Servers, Databases, W
Systems. Try Now!

AdChoices

21 Million Monthly Page Views & 13 Million Unique Visitors

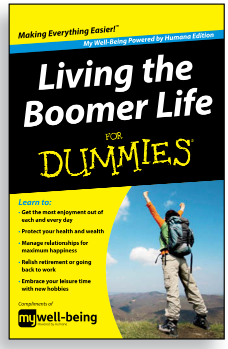
For Dummies is a registered trademark of John Wiley & Sons, Inc.

# of For Dummies

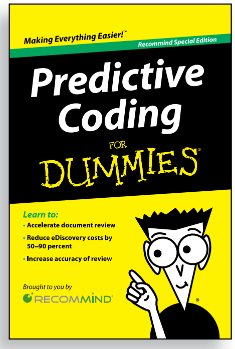
Custom Publishing

Reach a global audience in any language by creating a solution that will differentiate you from competitors, amplify your message, and encourage customers to make a buying decision.

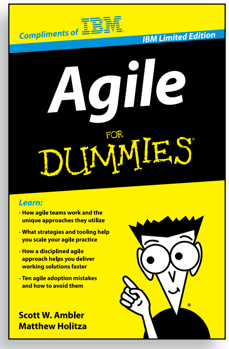
Apps • Books • eBooks • Video • Audio • Webinars

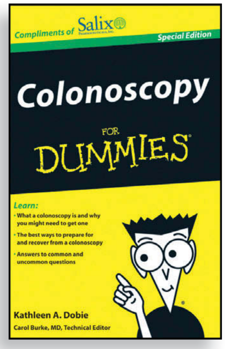


Making Everything Easier!
My Well-Being Powered by Humana Edition
Living the Boomer Life For Dummies
mywell-being



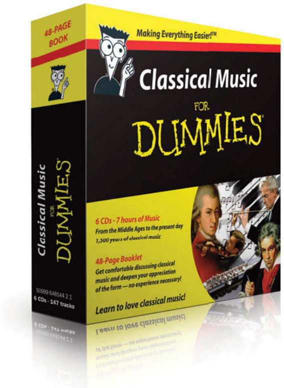
Making Everything Easier!
Recommind Special Edition
Predictive Coding For Dummies
RECOMMIND

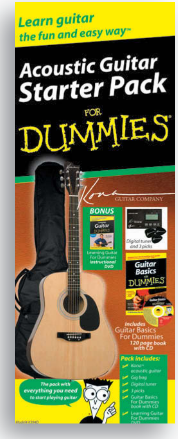




Brand Licensing & Content

Leverage the strength of the world's most popular reference brand to reach new audiences and channels of distribution.





Learn guitar the fun and easy way
Acoustic Guitar Starter Pack For Dummies

For more information, visit www.Dummies.com/biz

FOR DUMMIES
A Wiley Brand