PRAISE FOR *A PRACTICAL GUIDE TO RED HAT® LINUX®, SECOND EDITION*

"Since I'm in an educational environment, I found the content of Sobell's book to be right on target and very helpful for anyone managing Linux in the enterprise. His style of writing is very clear. He builds up to the chapter exercises, which I find to be relevant to real-world scenarios a user or admin would encounter. An IT/IS student would find this book a valuable complement to their education. The vast amount of information is extremely well balanced and Sobell manages to present the content without complicated asides and meandering prose. This is a 'must have' for anyone managing Linux systems in a networked environment or anyone running a Linux server. I would also highly recommend it to an experienced computer user who is moving to the Linux platform."

—Mary Norbury
IT Director
Barbara Davis Center/
University of Colorado at Denver
from a review posted on slashdot.org

"I had the chance to use your UNIX books when I when was in college years ago at Cal Poly San Luis Obispo, CA. I have to say that your books are among the best! They're quality books that teach the theoretical aspects and applications of the operating system."

—Benton Chan
IS Engineer

"The book has more than lived up to my expectations from the many reviews I read, even though it targets FC2. I have found something very rare with your book: It doesn't read like the standard technical text, it reads more like a story. It's a pleasure to read and hard to put down. Did I say that?! :-)"

—David Hopkins
Business Process Architect

"Thanks for your work and for the book you wrote. There are really few books that can help people to become more efficient administrators of different workstations. We hope (in Russia) that you will continue bringing us a new level of understanding of Linux/UNIX systems."

—Anton Petukhov

"Mark Sobell has written a book as approachable as it is authoritative."

—*Jeffrey Bianchine*
Advocate, Author, Journalist

"Excellent reference book, well suited for the sysadmin of a Linux cluster, or the owner of a PC contemplating installing a recent stable Linux. Don't be put off by the daunting heft of the book. Sobell has striven to be as inclusive as possible, in trying to anticipate your system administration needs."

—*Wes Boudville*
Inventor

"*A Practical Guide to Red Hat® Linux®* is a brilliant book. Thank you Mark Sobell."

—*C. Pozrikidis*
University of California at San Diego

"This book presents the best overview of the Linux operating system that I have found. . . . [It] should be very helpful and understandable no matter what the reader's background is: traditional UNIX user, new Linux devotee, or even Windows user. Each topic is presented in a clear, complete fashion and very few assumptions are made about what the reader knows. . . . The book is extremely useful as a reference, as it contains a 70-page glossary of terms and is very well indexed. It is organized in such a way that the reader can focus on simple tasks without having to wade through more advanced topics until they are ready."

—*Cam Marshall*
Marshall Information Service LLC
Member of Front Range UNIX
Users Group [FRUUG]
Boulder, Colorado

"Conclusively, this is THE book to get if you are a new Linux user and you just got into RH/Fedora world. There's no other book that discusses so many different topics and in such depth."

—*Eugenia Loli-Queru*
Editor in Chief
OSNews.com

Excerpts of Chapters from

A Practical Guide to Red Hat® Linux®

THIRD EDITION

Fedora™ Core and
Red Hat Enterprise Linux

Mark G. Sobell

ISBN 0-13-228027-2

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

3

Step-by-Step Installation

In This Chapter

Chapter 2 covered planning the installation of Red Hat Linux: determining the requirements; performing an upgrade versus a clean installation; planning the layout of the hard disk; obtaining the files you need for the installation, including how to download and burn CD and DVD ISO images; and collecting the information about the system you will need during installation. This chapter focuses on installing Fedora Core. The process of installing Red Hat Enterprise Linux is similar. Frequently the installation is quite simple, especially if you have done a good job of planning. Sometimes you may run into a problem or have a special circumstance; this chapter gives you tools to use in these cases.

THE ANACONDA INSTALLER

Anaconda, which is written in Python and C, identifies the hardware, builds the filesystems, and installs or upgrades the Red Hat Linux operating system. Anaconda can run in textual or graphical (default) interactive mode or in batch mode (see "Using the Kickstart Configurator" on page 63).

Anaconda does not write to the hard disk until it displays the Begin Installation screen

tip While you are installing Red Hat Linux, until Anaconda displays the Begin Installation screen (Figure 3-9, page 55), you can press CONTROL-ALT-DEL to abort the installation process and reboot the system without making any changes to the hard disk. However, if Anaconda displays the initialize warning dialog box (page 49), when you click **Yes**, it writes to the disk immediately.

Exactly which screens Anaconda displays depends on whether you are installing Fedora Core or Red Hat Enterprise Linux and which parameters you specified following the **boot:** prompt. With some exceptions—most notably if you are running a textual installation—Anaconda probes the video card and monitor and starts a native X server with a log in **/tmp/X.log**. (This log is not preserved unless you complete the installation.)

While it is running, Anaconda opens the virtual consoles (page 113) shown in Table 3-1. You can display a virtual console by pressing CONTROL-ALT-F*x*, where *x* is the virtual console number and F*x* is the function key that corresponds to the virtual console number.

Table 3-1 Virtual console assignments during installation

Virtual console	Information displayed during installation
1	Installation dialog
2	Shell
3	Installation log
4	System messages
5	Miscellaneous messages
7	GUI interactive installation

At any time during the installation, you can switch to virtual console 2 (CONTROL-ALT-F2) and give commands to see what is going on. Do not give any commands that change any part of the installation process. To switch back to the graphical installation screen, press CONTROL-ALT-F7.

USING ANACONDA

Anaconda provides a **Next** button at the lower-right corner of each of the installation screens and a **Back** button next to it on most screens. When you have completed the entries on an installation screen, click **Next** or, from a textual installation, press the TAB key until the **Next** button is highlighted and then press RETURN. Select **Back** to return to the previous screen. Click **Release Notes** at the lower-left corner of the screen to display the release notes for the version of Red Hat Linux you are installing.

ANACONDA SCREENS

Anaconda displays different screens depending on which commands you give and which choices you make. During a graphical installation, when you leave the CD Found screen, Anaconda starts, loads drivers, and probes for the devices it will use during installation. After probing, it starts the X server. This section describes the screens that Anaconda displays during a default installation and explains the choices you can make on each of them.

Logo Anaconda displays the Logo screen (Figure 3-1) after it obtains enough information to start the X Window System. There is nothing for you to do on this screen except display the release notes. Select **Next**.

Language Select the language you want to use for the installation. This language is not necessarily the same language that the installed system will display.

Keyboard Select the type of keyboard attached to the system.

Monitor Anaconda displays the Monitor screen only if it cannot probe the monitor successfully. Select the brand and model of the monitor attached to the system. Select a generic LCD or CRT display if the monitor is not listed. You can specify the Sync

Figure 3-1 The Logo screen

frequencies in place of the monitor brand and model, but be careful: Specifying the wrong values can ruin some older hardware.

Initialize warning

This warning is displayed if the hard disk has not been used before. The dialog box says that the partition table on the device was unreadable and asks if you want to initialize the drive. When you initialize a drive, all data on the drive is lost. Click **Yes** if it is a new drive or if you do not need the data on the drive. Anaconda initializes the hard disk immediately.

Install or Upgrade

Anaconda displays the Install or Upgrade screen (Figure 3-2) only if it detects a version of Red Hat Linux on the hard disk that it can upgrade. Anaconda gives you the choice of upgrading the existing installation or overwriting the existing installation with a new one. Refer to "Installing a Fresh Copy or Upgrading an Existing Red Hat System?" on page 27 for help in making this selection.

Figure 3-2 The Install or Upgrade screen

Figure 3-3 The Partition the Disk screen

Partition the Disk

The Partition the Disk screen (Figure 3-3) allows you to specify partition information and to select which drives you want to install Red Hat Linux on (assuming the system has more than one drive). Specify which drives you want to install Linux on in the frame labeled **Select the drive(s) to use for this installation**. Anaconda presents the following options in a combo box; click the box and then click the choice you want:

- **Remove all partitions on selected drives and create default layout.** Deletes all data on the disk and creates a free space the size of the disk to work with, as though you were working with a new drive.
- **Remove linux partitions on selected drives and create default layout.** Removes all Linux partitions, deleting the data on those partitions and creating one or more chunks of *free space* (page 1033) on the disk. You can create new partitions using the free space. If there is only a Linux system on the disk, this choice is the same as the previous one.
- **Use free space on selected drives and create default layout.** Installs Red Hat Linux in the free space on the disk. Does not work if there is not enough free space.
- **Create custom layout.** Does not alter disk partitions. This choice causes Anaconda to run Disk Druid (page 58) so that you can preserve the partitions you want to keep and overwrite other partitions. It is good for installing Red Hat Linux over an existing system where you want to keep **/home**, for example, but want a clean install and not an upgrade.

The default layout that the first three choices create includes two logical volumes (LVs: swap and root [**/**]) and one standard partition (**/boot**). With this setup, most of the space on the disk is assigned to the root partition. For information on the Logical Volume Manager, see page 32.

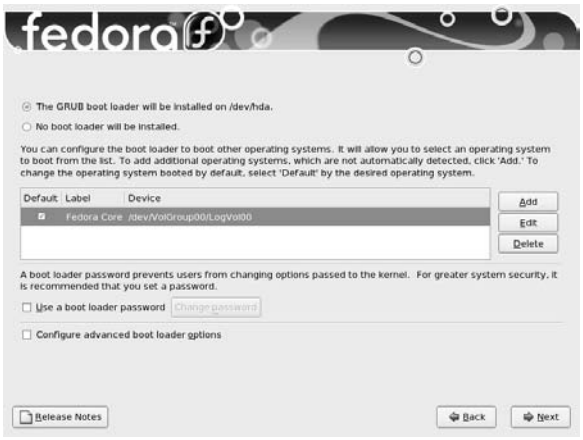

Figure 3-4 The Boot Loader Configuration screen

If you put a check mark in the box labeled **Review and modify partitioning layout** or if you select **Create custom layout** in the combo box, Anaconda runs Disk Druid (page 58) so that you can verify and modify the layout before it is written to the disk.

The disk is not partitioned until later

tip With one exception, Anaconda does not write to the hard disk when you specify partitions. Instead, it creates a table that specifies how you want the hard disk to be partitioned. The disk is actually partitioned and formatted when you click **Next** from the Begin installation screen (page 54). However, if Anaconda displays the initialize warning dialog box (page 49), when you click **Yes**, it writes to the disk immediately.

Disk Druid Anaconda runs Disk Druid only if you put a check mark in the box labeled **Review and modify partitioning layout** or if you select **Create custom layout** from the combo box as described in the previous section. See page 58 for information on the Disk Druid disk-partitioning program.

Warning Displays a warning if you are removing or formatting partitions. Click **Format** or **Yes** to proceed.

Boot Loader Configuration Anaconda displays the Boot Loader Configuration screen (Figure 3-4) only when you put a check mark in the box labeled **Review and modify partitioning layout** or select **Create custom layout** in the combo box in the Partition the Disk screen. By default, Anaconda installs the grub boot loader (page 533). If you do not want to install a boot loader, click the radio button next to **No boot loader will be installed**. When you install Red Hat Linux on a machine that already runs another operating system, Anaconda frequently recognizes the other operating system and sets up grub so you can boot from either operating system. Refer to "Setting Up a Dual-Boot System" on page 68. To manually add other operating systems to grub's list of bootable systems, click **Add** and specify a label and device to boot from. For a more secure system, specify a boot loader password.

8

Linux GUIs: X, GNOME, and KDE

This chapter covers the Linux graphical user interface (GUI). It continues where Chapter 4 left off, going into more detail about the X Window System, the basis for the Linux GUI. It presents a brief history of GNOME and KDE and discusses some of the problems and benefits of having two major Linux desktop environments. The section on GNOME covers the Nautilus file manager, including its spatial interface, and several important GNOME utilities. The final section covers KDE, presenting information about some of the more advanced features of Konqueror, and describing a few KDE utilities.

In This Chapter

USING GNOME

This section discusses the Nautilus file manager and several GNOME utilities.

THE NAUTILUS FILE MANAGER

Nautilus is a simple, powerful file manager. You can use it to create, open, view, move, and copy files and directories as well as execute programs and scripts. Nautilus gives you two ways to work with files: an innovative spatial view (Figure 8-2) and a traditional File Browser view (Figure 8-4 on page 244).

SPATIAL VIEW

The Nautilus object window presents a spatial view (Figure 8-2). This view has many powerful features but may take some getting used to. The spatial (as in "having the

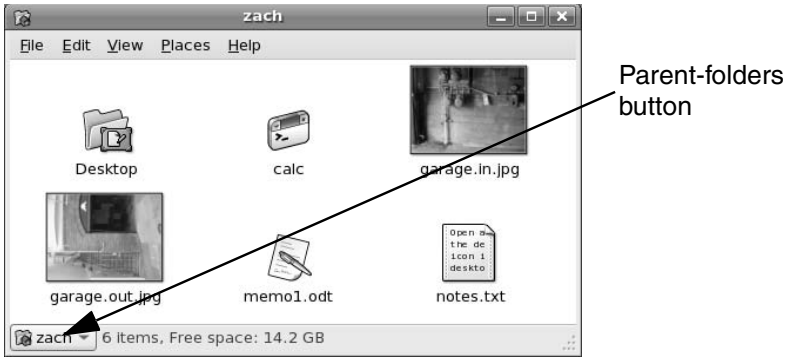

Figure 8-2 The Nautilus spatial view

nature of space") view always provides one window per folder. By default, when you open a folder, Nautilus displays a new window.

To open a spatial view of your home directory, double-click the home icon on the desktop and experiment as you read this section. If you double-click the Desktop icon in the spatial view, Nautilus opens a new window that displays the **Desktop** folder.

A spatial view can display icons or a list of filenames; select your preferred format by choosing one of the **View as** selections from View on the menubar. To create files to experiment with, right-click in the window (not on an icon) to display the Nautilus context menu and select **Create Folder** or **Create Document**.

GNOME desktop and Nautilus

tip The GNOME desktop is run from a back-end process that runs as part of Nautilus. If that process stops running, it usually restarts automatically. If it does not restart, give the command **nautilus** to restore the desktop. You do not have to keep the Nautilus window open to keep the desktop alive.

Use SHIFT to close the current window as you open another window

tip If you hold the SHIFT key down when you double-click to open a new window, Nautilus closes the current window as it opens the new one. This behavior may be more familiar and can help keep the desktop from becoming overly cluttered. If you do not want to use the keyboard, you can achieve the same result by double-clicking the middle mouse button.

Window memory

Move the window by dragging the titlebar. The spatial view has *window memory*. The next time you open that folder, Nautilus opens it at the same size in the same location. Even the scrollbar will be in the same position.

Parent-folders button

The key to closing the current window and returning to the window of the parent directory is the **Parent-folders** button (Figure 8-2). Click this button to display the

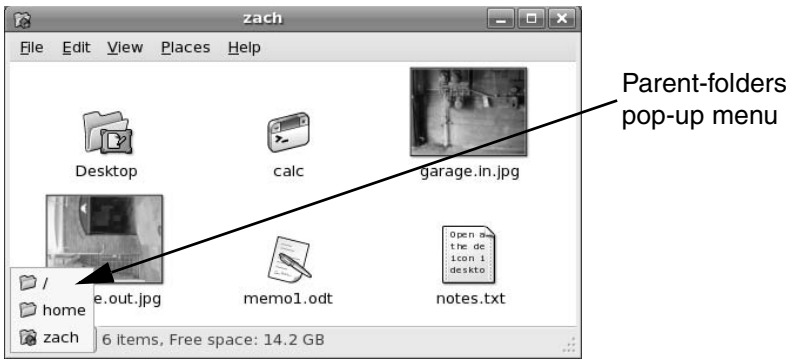

Figure 8-3 The Parent-folders pop-up menu

Parent-folders pop-up menu (Figure 8-3). Select the directory you want to open from this menu. Nautilus then displays in a spatial view the directory you specified.

From a spatial view, you can open a folder in a traditional view by right-clicking the folder and selecting **Browse Folder.**

TRADITIONAL VIEW

Figure 8-4 shows the traditional, or file browser, window with a Side pane (sometimes called a *sidebar*), View pane, menubar, toolbar, and location bar. To open a Browser view of your home directory, either right-click the home icon on the desktop and select **Browse Folder** or select **Applications: System Tools⇨File Browser.**

SIDE PANE

Click the button at the top of the Side pane to display the Side pane menu. From this menu select the type of items you want Nautilus to display in the Side pane:

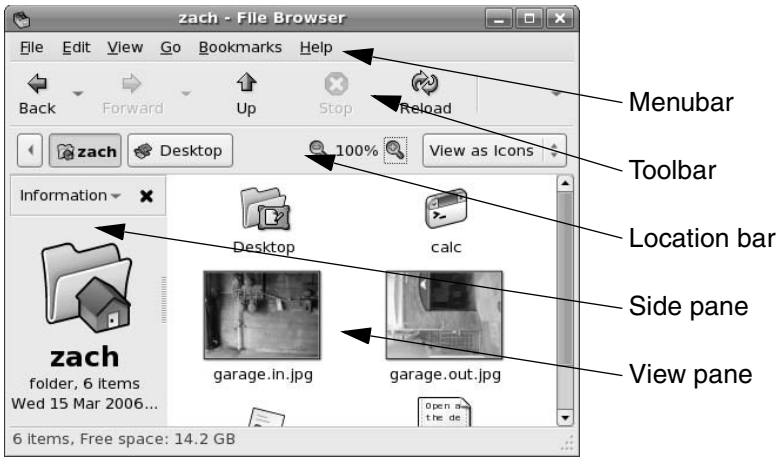

Figure 8-4 Nautilus traditional (file browser) window

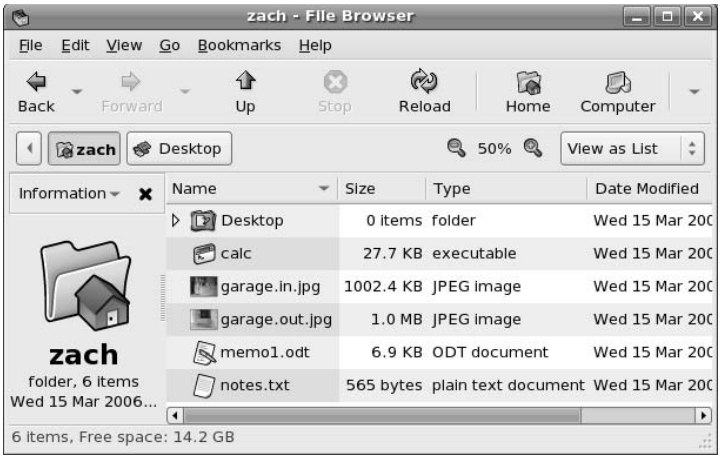

Figure 8-5 Nautilus displaying a List view

Places, Information (information about the folder displayed in the View pane; see Figure 8-4), Tree (directory hierarchy), History (list of recent locations displayed by Nautilus), Notes, or Emblems (drag emblems [page 247] to files in the View pane).

VIEW PANE

You can display icons or a list of filenames in the View pane. Choose your preferred view by making a selection from the drop-down menu that appears at the right end of the location bar. **View as Icons** is shown in Figure 8-4 and **View as List** is shown in Figure 8-5.

CONTROL BARS

This section discusses the three control bars—menubar, toolbar, and location bar—that initially appear at the top of a Nautilus browser window (Figure 8-4).

Menubar The menubar presents a drop-down menu when you click one of its selections. The menu selections depend on what Nautilus is displaying in the View pane.

Toolbar The Nautilus toolbar holds navigation tool icons: Back, Forward, Up, Stop, Reload, Home, Computer, and Search. Click the down arrow button at the right end of the toolbar to display and select icons that do not fit on the toolbar.

Location bar The buttons on the location bar depict the pathname of the directory that is displayed in the View pane and highlighted in the Tree tab, when it is displayed in the Side pane. You can display a text box in the location bar by pressing CONTROL-L. In this text box you can specify a local directory that you want to display in the View pane. When you enter the absolute pathname of the directory and press RETURN, Nautilus displays the contents of the directory.

The location bar also holds the magnification selector and the **View as** drop-down menu. To change the magnification of the display in the View pane, click the plus or minus sign on either side of the magnification percentage; click the magnification

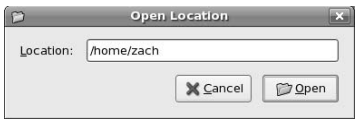

Figure 8-6 Open Location dialog box

percentage itself to return to 100% magnification. Click **View as** (to the right of the magnifying glass) to display and choose from the viewing selections.

FEATURES AVAILABLE FROM BOTH SPATIAL AND TRADITIONAL VIEWS

Open Location dialog box

You can display deeply nested directories quickly by using the Open Location dialog box (Figure 8-6) or the location bar. Press CONTROL-L while the cursor is over a Nautilus window to display the Open Location dialog box (spatial view) or to move the cursor to the location bar (traditional view). Enter the absolute pathname of the directory you want to display. Nautilus provides assistance by completing directory names as you type. Press TAB to accept a suggested completion or keep typing to ignore it.

Zooming images

Use the Open Location dialog box or the location bar to display the directory named **/usr/share/backgrounds/images**. Double-click an image file to display that file in a preview window. Position the mouse pointer over the image and use the mouse wheel to zoom the image. When the image is bigger than the window, you can drag the image to view different parts of it.

Opening files

By default, you double-click a filename or icon to open it, or you can right-click the icon or filename and choose **Open** from the pop-up menu. When you open a file, Nautilus figures out the appropriate tool to use by determining the file's MIME type (page 96). GNOME associates each filename extension with a MIME type and each MIME type with a program. Initially GNOME uses the filename extension to try to determine a file's MIME type. For example, when you open a file with a filename extension of **ps**, Nautilus calls KGhostView, which displays the PostScript file in a readable format. When you open a text file, Nautilus opens a text editor that displays and allows you to edit the file. When you open a directory, Nautilus displays its contents. When you open an executable file such as Firefox, Nautilus runs the executable. When Nautilus cannot determine which tool to use to open a file, it asks you for assistance. See "Open With" on page 248 for information on how to change the program that GNOME associates with a MIME type.

PROPERTIES

You can view information about a file, such as ownership, permissions, size, ways to work with it, and so on, by right-clicking a filename or icon and selecting **Properties** from the drop-down menu. The Properties window initially displays some basic information. Click the tabs at the top of the window to display additional information. Different types of files display different sets of tabs, depending on what is

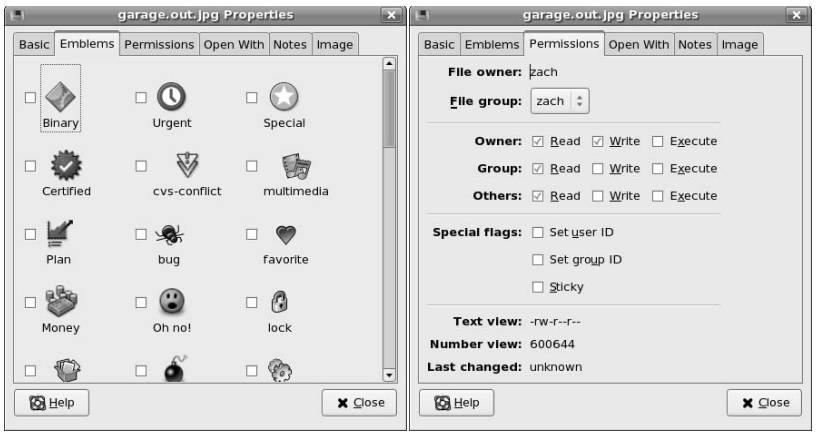

Figure 8-7 Properties window: Emblems tab (left); Permissions tab (right)

appropriate to the file. You can modify the settings in this window only if you have permission to do so.

Basic The Basic tab displays information about the file and enables you to select a custom icon for the file or change its name. To change the name of the file, make your changes in the text box. If the filename is not listed in a text box, you do not have permission to change it.

Emblems The Emblems tab (Figure 8-7, left) allows you to add or remove emblems associated with the file by placing (removing) a check mark in the box next to an emblem. Figure 8-8 shows some emblems on a file icon. Nautilus displays emblems in both its Icon and List views, although there may not be room for more than one icon in the List view. You can also place an emblem on an icon by dragging it from the Side pane Emblems tab to an icon in the View pane. Drag the Erase emblem to an icon to remove all emblems from the icon.

Permissions The Permissions tab (Figure 8-7, right) allows you to change file permissions (page 181). When the box to the left of the word **Read** in the Owner row (called *user* elsewhere; see the tip "chmod: **o** for other, **u** for owner" on page 183) has a

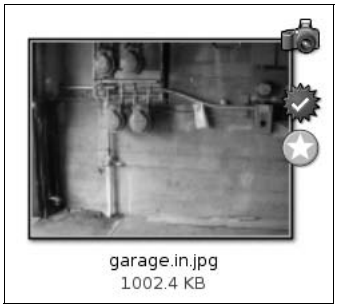

Figure 8-8 Emblems

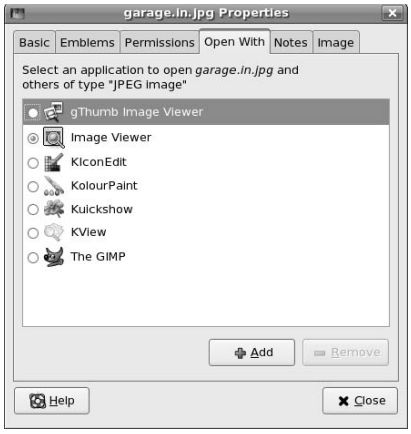

Figure 8-9 Properties window, Open With tab

check mark in it, the owner has permission to read the file. When you click the boxes in the Owner row so all of them contain check marks, the owner has read, write, and execute permissions. The owner of a file can change the group that the file is associated with to any other group the owner is associated with. When you run as Superuser, you can change the name of the user who owns the file and the group the file is associated with. Directory permissions work as explained on page 184. See page 183 for information on set user ID and set group ID permissions, and page 1057 for a description of the sticky bit.

Open With When you ask it to open a file that is not executable (by double-clicking its icon or right-clicking and selecting **Open with**), GNOME must figure out which application or utility to use when opening the file. GNOME uses several techniques to determine the MIME type (page 96) of a file and selects an appropriate application based on that determination.

The Open With tab (Figure 8-9) enables you to change which applications GNOME uses to open the current file and other files of the same MIME type (typically files with the same filename extension). Click **Add** to add an application and then click the radio button next to the application to cause GNOME to use that application to open the current file and others of the same MIME type. Highlight an application and click **Remove** to remove an application from the list. You cannot remove the default application.

GNOME UTILITIES

GNOME comes with numerous utilities intended to make your work on the desktop easier and more productive. This section covers several tools that are integral to the use of GNOME.

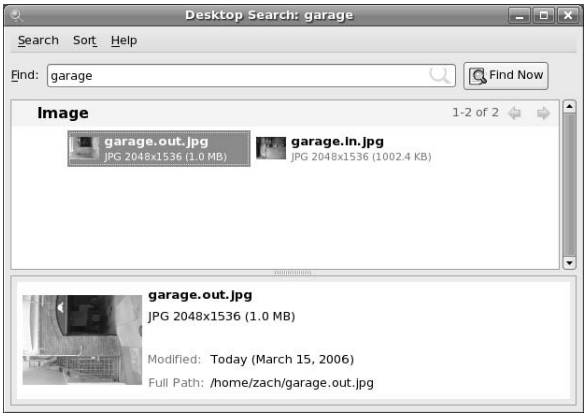

Figure 8-10 Desktop Search window

beagle: DESKTOP SEARCH (*FEDORA*)

To run beagle, enter **beagle-search** on a command line or select **Places: Search** on the panel at the top of the screen. This utility displays the Desktop Search window (Figure 8-10), from which you can search for a string of characters in a filename and in the contents of a file. To choose the type of files you want to search, make a selection from the menu displayed when you click **Search** on the menubar. By default, beagle searches files in your home directory hierarchy. Click **Search⇨Preferences⇨Indexing** to instruct beagle to search other directories. If, when you start beagle, it displays **Daemon not running**, click **Start the daemon** to start the **beagled** daemon.

To begin a search, enter the word or string of characters you want to search for in the Find text box. The beagle utility starts searching a moment after you stop typing. Click **Find Now** to display additional information about the selected file.

The Sort selection on the menubar allows you to sort the results of a search alphabetically by filename, by the date the file was modified, or by the relevance of each result.

FONT PREFERENCES

To display the GNOME Font Preferences window (Figure 8-11, next page), enter **gnome-font-properties** on a command line. You can also select **System: Preferences⇨Fonts** from the panel at the top of the screen. Click one of the five font bars in the upper part of the window to display the Pick a Font window (discussed in the next section). In this window you can change the font that GNOME uses for applications, documents, the desktop, window titles, or terminal emulators (fixed width).

Examine the four sample boxes in the Font Rendering frame in the lower part of the window and select the one with the best appearance. Subpixel smoothing is usually best for LCD monitors. Click **Details** to refine the font rendering further, again picking the box in each frame that has the best appearance.

Figure 8-11 Font Preferences window

PICK A FONT WINDOW

The Pick a Font window (Figure 8-12) appears when you need to choose a font. From this window you can select a font family, a style, and a size you want to use. A preview of your choice appears in the Preview frame in the lower part of the window. Click **OK** when you are satisfied with your choice.

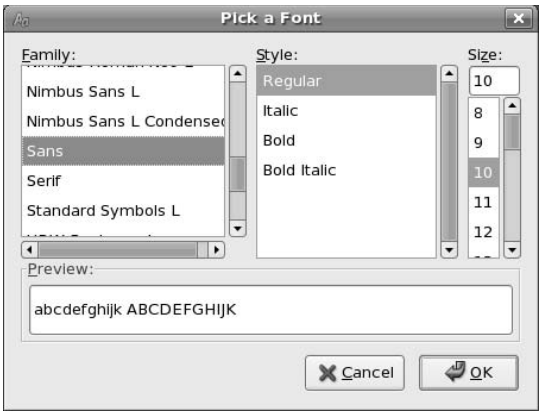

Figure 8-12 Pick a Font window

11

System Administration: Core Concepts

The job of a system administrator is to keep one or more systems in a useful and convenient state for users. On a Linux system, the administrator and user may both be you, with you and the computer being separated by only a few feet. Or the system administrator may be halfway around the world, supporting a network of systems, with you being simply one of thousands of users. A system administrator can be one person who works part-time taking care of a system and perhaps is also a user of the system. Or the administrator can be several people, all working full-time to keep many systems running.

In This Chapter

THE xinetd SUPERSERVER

The **xinetd** daemon is a more secure replacement for the **inetd** superserver that was originally shipped with 4.3BSD. The **xinetd** superserver listens for network connections. When one is made, it launches a specified server daemon and forwards the data from the socket (page 462) to the daemon's standard input.

The version of **xinetd** distributed with Red Hat Linux is linked against **libwrap.a**, so it can use the **/etc/hosts.allow** and **/etc/hosts.deny** files for access control (see "TCP Wrappers" on page 427 for more information). Using TCP wrappers can simplify configuration but hides some of the more advanced features of **xinetd**.

xinetd may not be installed

tip Working as **root**, give the following command to install **xinetd** on a *FEDORA* system:

```
# yum install xinetd
```

Use RHN to install it on a *RHEL* system.

The base configuration for **xinetd** is stored in the **/etc/xinetd.conf** file. If this file is not present, **xinetd** is probably not installed. See the preceding tip. The file supplied with Red Hat Linux is shown here:

```
$ cat /etc/xinetd.conf
#
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/

defaults
{
        instances               = 60
        log_type                = SYSLOG authpriv
        log_on_success          = HOST PID
        log_on_failure          = HOST
        cps                     = 25 30
}

includedir /etc/xinetd.d
```

The **defaults** section specifies the default configuration of **xinetd**; the files in the included directory, **/etc/xinetd.d**, specify server-specific configurations. Defaults can be overridden by server-specific configuration files.

In the preceding file, the **instances** directive specifies that no daemon may run more than 60 copies of itself at one time. The **log_type** directive specifies that **xinetd** send messages to the system log daemon (**syslogd**, page 562) using the **authpriv** facility. The next two lines specify what to log on success and on failure. The **cps** (connections per second) directive specifies that no more than 25 connections to a specific service should be made per second and that the service should be disabled for 30 seconds if this limit is exceeded.

The following **xinetd** configuration file allows telnet connections from the local system and any system with an IP address that starts with **192.168.**. This configuration file does not rely on TCP wrappers, so it does not rely on the **hosts.allow** and **hosts.deny** files.

```
$ cat /etc/xinetd.d/telnet
service telnet
{
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/sbin/in.telnetd
        only_from       = 192.168.0.0/16 127.0.0.1
        disable         = no
}
```

The **socket_type** indicates whether the socket uses TCP or UDP. TCP-based protocols establish a connection between the client and the server and are identified by the type **stream**. UDP-based protocols rely on the transmission of individual datagrams and are identified by the type **dgram**.

When **wait** is set to **no**, **xinetd** handles multiple concurrent connections to this service. Setting **wait** to **yes** causes **xinetd** to wait for the server process to complete before handling the next request for that service. In general, UDP services should be set to **yes** and TCP services to **no**. If you were to set **wait** to **yes** for a service such as telnet, only one person would be able to use the service at any given time.

The **user** specifies which user the server runs as. If this user is a member of multiple groups, you can also specify the group on a separate line with the keyword **group**. The **user** directive is ignored if **xinetd** is run as other than **root**. The **server** provides the pathname of the server program that **xinetd** runs for this service.

The **only_from** specifies which systems **xinetd** allows to use the service. It is a good idea to use IP addresses only—using hostnames can make the service unavailable if DNS fails. Zeros at the right of an IP address are treated as wildcards. For example, **192.168.0.0** allows access from any system in the **192.168** subnet.

The **disable** line can disable a service without removing the configuration file. As shipped by Red Hat, a number of services include an **xinetd** configuration file with **disable** set to **yes**. To run one of these services, change **disable** to **no** in the appropriate file in **xinetd.d** and restart **xinetd**:

```
# /sbin/service xinetd restart
Stopping xinetd:                                           [  OK  ]
Starting xinetd:                                           [  OK  ]
```

SECURING A SERVER

You may secure a server either by using TCP wrappers or by setting up a chroot jail.

TCP WRAPPERS: CLIENT/SERVER SECURITY (hosts.allow AND hosts.deny)

When you open a local system to access from remote systems, you must ensure that the following criteria are met:

- Open the local system only to systems you want to allow to access it.
- Allow each remote system to access only the data you want it to access.
- Allow each remote system to access data only in the appropriate manner (readonly, read/write, write only).

As part of the client/server model, TCP wrappers, which can be used for any daemon that is linked against **libwrap.a**, rely on the **/etc/hosts.allow** and **/etc/hosts.deny** files as the basis of a simple access control language. This access control language defines rules that selectively allow clients to access server daemons on a local system based on the client's address and the daemon the client tries to access.

Each line in the **hosts.allow** and **hosts.deny** files has the following format:

daemon_list **:** *client_list [*: *command]*

where ***daemon_list*** is a comma-separated list of one or more server daemons (such as **portmap**, **vsftpd**, or **sshd**), ***client_list*** is a comma-separated list of one or more clients (see Table 11-3, "Specifying a client," on page 422), and the optional ***command*** is the command that is executed when a client from ***client_list*** tries to access a server daemon from ***daemon_list***.

When a client requests a connection with a local server, the **hosts.allow** and **hosts.deny** files are consulted as follows until a match is found:

1. If the daemon/client pair matches a line in **hosts.allow**, access is granted.
2. If the daemon/client pair matches a line in **hosts.deny**, access is denied.
3. If there is no match in either the **hosts.allow** or the **hosts.deny** files, access is granted.

The first match determines whether the client is allowed to access the server. When either **hosts.allow** or **hosts.deny** does not exist, it is as though that file was empty. Although it is not recommended, you can allow access to all daemons for all clients by removing both files.

Examples For a more secure system, put the following line in **hosts.deny** to block all access:

```
$ cat /etc/hosts.deny
...
ALL : ALL : echo '%c tried to connect to %d and was blocked' >> /var/log/tcpwrappers.log
```

This line prevents any client from connecting to any service, unless specifically permitted in **hosts.allow**. When this rule is matched, it adds a line to the file named **/var/log/tcpwrappers.log**. The **%c** expands to client information and the **%d** expands to the name of the daemon the client attempted to connect to.

With the preceding **hosts.deny** file in place, you can include lines in **hosts.allow** that explicitly allow access to certain services and systems. For example, the following **hosts.allow** file allows anyone to connect to the OpenSSH daemon (ssh, scp, sftp) but allows telnet connections only from the same network as the local system and users on the 192.168. subnet:

```
$ cat /etc/hosts.allow
sshd : ALL
in.telnet : LOCAL
in.telnet : 192.168.* 127.0.0.1
...
```

The first line allows connection from any system (ALL) to **sshd**. The second line allows connection from any system in the same domain as the server (LOCAL). The third line matches any system whose IP address starts **192.168.** and the local system.

SETTING UP A chroot JAIL

On early UNIX systems, the root directory was a fixed point in the filesystem. On modern UNIX variants, including Linux, you can define the root directory on a per-process basis. The chroot utility allows you to run a process with a root directory other than **/**.

The root directory appears at the top of the directory hierarchy and has no parent: A process cannot access any files above the root directory (because they do not exist). If, for example, you run a program (process) and specify its root directory as **/home/sam/jail**, the program would have no concept of any files in **/home/sam** or above: **jail** is the program's root directory and is labeled **/** (not **jail**).

By creating an artificial root directory, frequently called a (chroot) jail, you prevent a program from accessing or modifying—possibly maliciously—files outside the directory hierarchy starting at its root. You must set up a chroot jail properly to increase security: If you do not set up a chroot jail correctly, you can actually make it easier for a malicious user to gain access to a system than if there were no chroot jail.

USING chroot

Creating a chroot jail is simple: Working as **root**, give the command **/usr/sbin/chroot** ***directory***. The ***directory*** becomes the root directory and the process attempts to run the default shell. Working as **root** from the **/home/sam** directory, the following command sets up a chroot jail in the (existing) **/home/sam/jail** directory:

```
# /usr/sbin/chroot /home/sam/jail
/usr/sbin/chroot: cannot run command '/bin/bash': No such file or directory
```

This example sets up a chroot jail, but when it attempts to run the bash shell, it fails. Once the jail is set up, the directory that was named **jail** takes on the name of the root directory, **/**, so chroot cannot find the file identified by the pathname **/bin/bash**. In this situation the chroot jail is working but is not useful.

Getting a chroot jail to work the way you want is a bit more complicated. To have the preceding example run bash in a chroot jail, you need to create a **bin** directory in **jail** (**/home/sam/jail/bin**) and copy **/bin/bash** to this directory. Because the bash binary is dynamically linked to shared libraries (page 840), you need to copy these libraries into **jail** as well. The libraries go in **lib**. The next example creates the necessary directories, copies **bash**, uses ldd to display the shared library dependencies of bash, and copies the necessary libraries into **lib**. The **linux-gate.so.1** file is a dynamically shared object (DSO) provided by the kernel to speed system calls; you do not need to copy it to the **lib** directory.

```
$ pwd
/home/sam/jail
$ mkdir bin lib
$ cp /bin/bash bin
$ ldd bin/bash
        linux-gate.so.1 =>  (0x0089c000)
        libtermcap.so.2 => /lib/libtermcap.so.2 (0x00cdb000)
        libdl.so.2 => /lib/libdl.so.2 (0x00b1b000)
        libc.so.6 => /lib/libc.so.6 (0x009cb000)
        /lib/ld-linux.so.2 (0x009ae000)
$ cp /lib/{libtermcap.so.2,libdl.so.2,libc.so.6,ld-linux.so.2} lib
```

Now that everything is set up, you can start the chroot jail again. Although all of the setup can be done by an ordinary user, you have to run chroot as Superuser:

```
$ su
Password:
# /usr/sbin/chroot .
bash-3.1# pwd
/
bash-3.1# ls
bash: ls: command not found
bash-3.1#
```

This time the chroot finds and starts **bash**, which displays its default prompt (**bash-3.1#**). The pwd command works because it is a shell builtin (page 225). However, bash cannot find the ls utility (it is not in the chroot jail). You can copy **/bin/ls** and its libraries into the jail if you want users in the jail to be able to use ls.

To set up a useful chroot jail, first determine which utilities the users of the chroot jail will need. Then copy the appropriate binaries and their libraries into the jail. Alternatively you can build static copies of the binaries and put them in the jail without installing separate libraries. (The statically linked binaries are considerably larger than their dynamic counterparts. The base system with bash and the core utilities

exceeds 50 megabytes.) You can find the source code for most of the common utilities in the **bash** and **coreutils** SRPMS (source rpm) packages.

Whichever technique you choose, you must put a copy of su in the jail. The su command is required to run programs as a user other than **root**. Because **root** can break out of a chroot jail, it is imperative that you run a program in the chroot jail as a user other than **root**.

The dynamic version of su distributed by Red Hat requires PAM and will not work within a jail. You need to build a copy of su from the source to use in a jail. By default any copy of su you build does not require PAM. Refer to "GNU Configure and Build System" on page 491 for instructions on how to build packages such as **coreutils** (which includes su).

To use su, you must copy the relevant lines from the **/etc/passwd** and **/etc/shadow** files into files with the same names in the **etc** directory inside the jail.

Keeping multiple chroot jails

tip If you plan to deploy multiple chroot jails, it is a good idea to keep a clean copy of the **bin** and **lib** files somewhere other than in one of the active jails.

RUNNING A SERVICE IN A chroot JAIL

Running a shell inside a jail has limited usefulness. Instead you are more likely to need to run a specific service inside the jail. To run a service inside a jail, you must make sure all files needed by that service are inside the jail. The format of a command to start a service in a chroot jail is

```
# /usr/sbin/chroot jailpath /bin/su user daemonname &
```

where *jailpath* is the pathname of the jail directory, *user* is the username that runs the daemon, and *daemonname* is the path (inside the jail) of the daemon that provides the service.

Some servers are already set up to take advantage of chroot jails. You can set up DNS so that **named** runs in a jail (page 750), and the **vsftpd** FTP server can automatically start chroot jails for clients (page 616).

SECURITY CONSIDERATIONS

Some services need to be run as **root**, but they release their **root** privilege once started (Procmail and **vsftpd** are examples). If you are running such a service, you do not need to put su inside the jail.

A process run as **root** could potentially escape from a chroot jail. For this reason, you should always su to another user before starting a program running inside the jail. Also, be careful about which setuid (page 183) binaries you allow inside a jail—a security hole in one of them could compromise the security of the jail. In addition, make sure the user cannot access executable files that he uploads.

DHCP: CONFIGURES HOSTS

Instead of storing network configuration information in local files on each system, DHCP (Dynamic Host Configuration Protocol) enables client systems to retrieve network configuration information each time they connect to the network. A DHCP server assigns an IP addresses from a pool of addresses to clients as needed. Assigned addresses are typically temporary, but need not be.

This technique has several advantages over storing network configuration information in local files:

- A new user can set up an Internet connection without having to deal with IP addresses, netmasks, DNS addresses, and other technical details. An experienced user can set up a connection more quickly.
- DHCP facilitates assignment and management of IP addresses and related network information by centralizing the process on a server. A system administrator can configure new systems, including laptops that connect to the network from different locations, to use DHCP; DHCP then assigns IP addresses only when each system connects to the network. The pool of IP addresses is managed as a group on the DHCP server.
- IP addresses can be used by more than one system, reducing the total number of IP addresses needed. This conservation of addresses is important because the Internet is quickly running out of IPv4 addresses. Although a particular IP address can be used by only one system at a time, many end-user systems require addresses only occasionally, when they connect to the Internet. By reusing IP addresses, DHCP lengthens the life of the IPv4 protocol. DHCP applies to IPv4 only, as IPv6 forces systems to configure their IP addresses automatically (called autoconfiguration) when they connect to a network (page 359).

DHCP is particularly useful for administrators who are responsible for maintaining a large number of systems because individual systems no longer need to store unique configuration information. With DHCP, the administrator can set up a master system and deploy new systems with a copy of the master's hard disk. In educational establishments and other open access facilities, the hard disk image may be stored on a shared drive, with each workstation automatically restoring itself to pristine condition at the end of each day.

MORE INFORMATION

Web www.dhcp.org

FAQ www.dhcp-handbook.com/dhcp_faq.html

HOWTO *DHCP Mini HOWTO*

How DHCP Works

The client daemon, **dhclient** (part of the **dhcp** package), contacts the server daemon, **dhcpd**, to obtain the IP address, netmask, broadcast address, nameserver address, and other networking parameters. The server provides a *lease* on the IP address to the client. The client can request the specific terms of the lease, including its duration; the server can, in turn, limit these terms. While connected to the network, a client typically requests extensions of its lease as necessary so its IP address remains the same. The lease can expire once the client is disconnected from the network, with the server giving the client a new IP address when it requests a new lease. You can also set up a DHCP server to provide static IP addresses for specific clients (refer to "Static Versus Dynamic IP Addresses" on page 354).

DHCP is broadcast based, so both client and server must be on the same subnet (page 357).

DHCP Client

A DHCP client requests network configuration parameters from the DHCP server and uses those parameters to configure its network interface.

Prerequisites

Install the following package:

- **dhclient**

dhclient: The DHCP Client

When a DHCP client system connects to the network, dhclient requests a lease from the DHCP server and configures the client's network interface(s). Once a DHCP client has requested and established a lease, it stores information about the lease in a file named **dhclient.leases**, which is stored in **/var/lib/dhcp** (*RHEL*) or **/var/lib/dhclient** (*FEDORA*). This information is used to reestablish a lease when either the server or the client needs to reboot. The DHCP client configuration file, **/etc/dhclient.conf**, is required only for custom configurations. The following **dhclient.conf** file specifies a single interface, **eth0**:

```
$ cat /etc/dhclient.conf
interface "eth0"
{
send dhcp-client-identifier 1:xx:xx:xx:xx:xx:xx;
send dhcp-lease-time 86400;
}
```

In the preceding file, the 1 in the **dhcp-client-identifier** specifies an Ethernet network and **xx:xx:xx:xx:xx:xx** is the *MAC address* (page 1041) of the device controlling that interface. See page 434 for instructions on how to display a MAC address. The **dhcp-lease-time** is the duration, in seconds, of the lease on the IP address. While the client is connected to the network, dhclient automatically renews the lease each time half of the lease is up. The lease time of 8,6400 seconds (or one day) is a reasonable choice for a workstation.

DHCP SERVER

The DHCP server maintains a list of IP addresses and other configuration parameters. When requested to do so, the DHCP server provides configuration parameters to a client.

PREREQUISITES

Install the following package:

- **dhcp**

Run chkconfig to cause **dhcpd** to start when the system enters multiuser mode:

```
# /sbin/chkconfig dhcpd on
```

Start **dhcpd**:

```
# /sbin/service dhcpd start
```

dhcpd: THE DHCP DAEMON

A simple DCHP server allows you to add clients to a network without maintaining a list of assigned IP addresses. A simple network, such as a home LAN sharing an Internet connection, can use DHCP to assign a dynamic IP address to almost all nodes. The exceptions are servers and routers, which must be at known network locations to be able to receive connections. If servers and routers are configured without DHCP, you can specify a simple DHCP server configuration in **/etc/dhcpd.conf**:

```
$ cat /etc/dhcpd.conf
default-lease-time 600;
max-lease-time 86400;

option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option routers 192.168.1.1;
option domain-name-servers 192.168.1.1;

subnet 192.168.1.0 netmask 255.255.255.0 {
   range 192.168.1.2 192.168.1.200;
}
```

The preceding configuration file specifies a LAN where the router and DNS are both located on **192.168.1.1**. The **default-lease-time** specifies the number of seconds the dynamic IP lease will remain valid if the client does not specify a duration. The **max-lease-time** is the maximum time allowed for a lease.

The information in the **option** lines is sent to each client when it connects. The names following the word **option** specify what the following argument represents. For example, the **option broadcast-address** line specifies the broadcast address of the network. The **routers** and **domain-name-servers** options allow multiple values separated by commas.

The **subnet** section includes a **range** line that specifies the range of IP addresses that the DHCP server can assign. If you define multiple subnets, you can define options, such as **subnet-mask**, inside the **subnet** section. Options defined outside all **subnet** sections are global and apply to all subnets.

The preceding configuration file assigns addresses in the range between 192.168.1.2 and 192.168.1.200. The DHCP server starts at the bottom of this range and attempts to assign a new IP address to each new client. Once the DHCP server reaches the top of the range, it starts reassigning IP addresses that have been used in the past, but are not currently in use. If you have fewer systems than IP addresses, the IP address of each system should remain fairly constant. You cannot use the same IP address for more than one system at a time.

Once you have configured a DHCP server, you can start (or restart) it by using the **dhcpd** init script:

```
# /sbin/service dhcpd restart
```

Once the server is running, clients configured to obtain an IP address from the server using DHCP should be able to do so.

STATIC IP ADDRESSES

As mentioned earlier, routers and servers typically require static IP addresses. While you can manually configure IP addresses for these systems, it may be more convenient to have the DHCP server provide them with static IP addresses.

When a system that requires a specific static IP address connects to the network and contacts the DHCP server, the server needs a way to identify the system so the server can assign the proper IP address to the system. The DHCP server uses the *MAC address* (page 1041) of the system's Ethernet card (NIC) as an identifier. When you set up the server, you must know the MAC address of each system that requires a static IP address.

Displaying a MAC address

You can use ifconfig to display the MAC addresses of the Ethernet cards (NICs) in a system. In the following example, the MAC addresses are the colon-separated series of hexadecimal number pairs following **HWaddr**:

```
$ /sbin/ifconfig | grep -i hwaddr
eth0      Link encap:Ethernet  HWaddr BA:DF:00:DF:C0:FF
eth1      Link encap:Ethernet  HWaddr 00:02:B3:41:35:98
```

Run ifconfig on each system that requires a static IP address. Once you have determined the MAC address of each of these systems, you can add a **host** section to the **/etc/dhcpd.conf** file for each system, instructing the DHCP server to assign a specific address to the system. The following **host** section assigns the address **192.168.1.1** to the system with the MAC address of **BA:DF:00:DF:C0:FF**:

```
$ cat /etc/dhcpd.conf
...
host router {
   hardware ethernet BA:DF:00:DF:C0:FF;
   fixed-address 192.168.1.1;
   option host-name router;
}
```

The name following **host** is used internally by **dhcpd**. The name specified after **option host-name** is passed to the client and can be a hostname or an FQDN.

After making changes to **dhcpd.conf**, restart **dhcpd** using service and the **dhcpd** init script (page 433).

nsswitch.conf: WHICH SERVICE TO LOOK AT FIRST

With the advent of NIS and DNS, finding user and system information was no longer a simple matter of searching a local file. Where once you looked in **/etc/passwd** to get user information and in **/etc/hosts** to find system address information, you can now use several methods to find this type of information. The **/etc/nsswitch.conf** (name service switch configuration) file specifies which methods to use and the order in which to use them when looking for a certain type of information. You can also specify what action the system takes based on whether a method works or fails.

Format Each line in **nsswitch.conf** specifies how to search for a piece of information, such as a user's password. A line in **nsswitch.conf** has the following format:

info: ***method*** *[[**action**]] [**method** [[**action**]]...]*

where ***info*** specifies the type of information that the line describes, ***method*** is the method used to find the information, and ***action*** is the response to the return status of the preceding ***method***. The action is enclosed within square brackets.

How nsswitch.conf Works

When called upon to supply information that **nsswitch.conf** describes, the system examines the line with the appropriate ***info*** field. It uses the methods specified on the line starting with the method on the left. By default, when it finds the desired information, the system stops searching. Without an ***action*** specification, when a method fails to return a result, the system tries the next action. It is possible for the search to end without finding the requested information.

INFORMATION

The **nsswitch.conf** file commonly controls searches for users (in **passwd**), passwords (in **shadow**), host IP addresses, and group information. The following list

describes most of the types of information (*info* in the format discussed earlier) that **nsswitch.conf** controls searches for.

automount	Automount (**/etc/auto.master** and **/etc/auto.misc**, page 690)
bootparams	Diskless and other booting options (See the **bootparam** man page.)
ethers	MAC address (page 1041)
group	Groups of users (**/etc/group**, page 451)
hosts	System information (**/etc/hosts**, page 452)
netgroup	Netgroup information (**/etc/netgroup**, page 453)
networks	Network information (**/etc/networks**)
passwd	User information (**/etc/passwd**, page 454)
protocols	Protocol information (**/etc/protocols**, page 455)
publickey	Used for NFS running in secure mode
rpc	RPC names and numbers (**/etc/rpc**, page 456)
services	Services information (**/etc/services**, page 456)
shadow	Shadow password information (**/etc/shadow**, page 456)

METHODS

Following is a list of the types of information that **nsswitch.conf** controls searches for (*method* in the format on page 435). For each type of information, you can specify one or more of the following methods:[2]

files	Searches local files such as **/etc/passwd** and **/etc/hosts**
nis	Searches the NIS database; **yp** is an alias for **nis**
dns	Queries the DNS (**hosts** queries only)
compat	**±** syntax in **passwd**, **group**, and **shadow** files (page 438)

SEARCH ORDER

The information provided by two or more methods may overlap: For example, **files** and **nis** may each provide password information for the same user. With overlapping information, you need to consider which method you want to be authoritative (take precedence), and put that method at the left of the list of methods.

The default **nsswitch.conf** file lists methods without actions, assuming no overlap (which is normal). In this case, the order is not critical: When one method fails, the system goes to the next one; all that is lost is a little time. Order becomes critical when you use actions between methods, or when overlapping entries differ.

The first of the following lines from **nsswitch.conf** causes the system to search for password information in **/etc/passwd** and, if that fails, to use NIS to find the information. If the user you are looking for is listed in both places, the information in the local file would be used and therefore would be authoritative. The second line uses

2. There are other, less commonly used methods. See the default **/etc/nsswitch.conf** file and the **nsswitch.conf** man page for more information. Although NIS+ belongs in this list, it is not implemented for Linux and is not discussed in this book.

NIS; if that fails, it searches **/etc/hosts;** if that fails, it checks with DNS to find host information.

```
passwd          files nis
hosts           nis files dns
```

ACTION ITEMS

Each method can optionally be followed by an action item that specifies what to do if the method succeeds or fails for any of a number of reasons. An action item has the following format:

[[!]STATUS=action]

where the opening and closing square brackets are part of the format and do not indicate that the contents are optional; ***STATUS*** (by convention uppercase although it is not case sensitive) is the status being tested for; and ***action*** is the action to be taken if ***STATUS*** matches the status returned by the preceding method. The leading exclamation point (**!**) is optional and negates the status.

STATUS Values for ***STATUS*** are as follows:

NOTFOUND The method worked but the value being searched for was not found. Default action is **continue**.

SUCCESS The method worked and the value being searched for was found; no error was returned. Default action is **return**.

UNAVAIL The method failed because it is permanently unavailable. For example, the required file may not be accessible or the required server may be down. Default action is **continue**.

TRYAGAIN The method failed because it was temporarily unavailable. For example, a file may be locked or a server overloaded. Default action is **continue**.

action Values for ***action*** are as follows:

return Returns to the calling routine with or without a value.

continue Continues with the next method. Any returned value is overwritten by a value found by the next method.

Example For example, the following line from **nsswitch.conf** causes the system first to use DNS to search for the IP address of a given host. The action item following the DNS method tests whether the status returned by the method is not (**!**) UNAVAIL.

```
hosts           dns [!UNAVAIL=return] files
```

The system takes the action associated with the ***STATUS*** (**return**) if the DNS method does not return UNAVAIL (**!**UNAVAIL)—that is, if DNS returns SUCCESS, NOTFOUND, or TRYAGAIN. The result is that the following method (**files**) is used only when the DNS server is unavailable: If the DNS server is *not un*available (read the two negatives as "is available"), the search returns the domain name or reports that the domain name was not found. The search uses the **files** method (check the local **/etc/hosts** file) only if the server is not available.

COMPAT METHOD: ± IN passwd, group, AND shadow FILES

You can put special codes in the **/etc/passwd**, **/etc/group**, and **/etc/shadow** files that cause the system, when you specify the **compat** method in **nsswitch.conf**, to combine and modify entries in the local files and the NIS maps.

A plus sign (**+**) at the beginning of a line in one of these files adds NIS information; a minus sign (**–**) removes information. For example, to use these codes in the **passwd** file, specify **passwd: compat** in **nsswitch.conf**. The system then goes through the **passwd** file in order, adding or removing the appropriate NIS entries when it reaches each line that starts with a **+** or **–**.

Although you can put a plus sign at the end of the **passwd** file, specify **passwd: compat** in **nsswitch.conf** to search the local **passwd** file, and then go through the NIS map, it is more efficient to put **passwd: file nis** in **nsswitch.conf** and not modify the **passwd** file.

20

sendmail: Setting Up Mail Clients, Servers, and More

In This Chapter

Sending and receiving email require three pieces of software. At each end, there is a client, called an MUA (Mail User Agent), which is a bridge between a user and the mail system. Common MUAs are mutt, KMail, Thunderbird, and Outlook. When you send an email, the MUA hands it to an MTA (a Mail Transfer Agent such as **sendmail**), which transfers it to the destination server. At the destination, an MDA (a Mail Delivery Agent such as **procmail**) puts the mail in the recipient's mailbox file. On Linux systems, the MUA on the receiving system either reads the mailbox file or retrieves mail from a remote MUA or MTA, such as an ISP's SMTP (mail) server, using POP (Post Office Protocol) or IMAP (Internet Message Access Protocol).

Most Linux MUAs expect a local copy of **sendmail** to deliver outgoing email. On some systems, including those with a dial-up connection to the Internet, **sendmail** relays email to an ISP's mail server. Because **sendmail** uses SMTP (Simple Mail Transfer Protocol) to deliver email, **sendmail** is often referred to as an SMTP server.

In the default Red Hat Linux setup, the **sendmail** MTA uses **procmail** as the local MDA. In turn, **procmail** writes email to the end of the recipient's mailbox file. You can also use **procmail** to sort email according to a set of rules, either on a per-user basis or globally. The global filtering function is useful for systemwide filtering to detect spam and for other tasks, but the per-user feature is largely superfluous on a modern system. Traditional UNIX MUAs were simple programs that could not filter mail and thus delegated this function to MDAs such as **procmail**. Modern MUAs, by contrast, incorporate this functionality.

You do not need to set up sendmail to send and receive email

tip Most MUAs can use POP or IMAP for receiving email. These protocols do not require an MTA such as **sendmail**. As a consequence, you do not need to install or configure **sendmail** (or another MTA) to receive email. You still need SMTP to send email. However, the SMTP server can be at a remote location, such as your ISP, so you do not need to concern yourself with it.

INTRODUCTION

When the network that was to evolve into the Internet was first set up, it connected a few computers, each serving a large number of users and running several services. Each computer was capable of sending and receiving email and had a unique hostname, which was used as a destination for email.

Today the Internet has a large number of transient clients. Because these clients do not have fixed IP addresses or hostnames, they cannot receive email directly. Users on these systems usually maintain an account on an email server run by their employer or an ISP, and they collect email from this account using POP or IMAP. Unless you own a domain that you want to receive email at, you will not need to set up **sendmail** as an incoming SMTP server.

You can set up **sendmail** on a client system so that it simply relays outbound mail to an SMTP server. This configuration is required by organizations that use firewalls to prevent email from being sent out on the Internet from any system other than the company's official mail servers. As a partial defense against spreading viruses, some ISPs block outbound port 25 to prevent their customers from sending email directly to a remote computer. This configuration is required by these ISPs.

You can also set up **sendmail** as an outbound server that does not use an ISP as a relay. In this configuration, **sendmail** connects directly to the SMTP servers for the domains receiving the email. An ISP set up as a relay is configured this way.

You can set up **sendmail** to accept email for a registered domain name as specified in the domain's DNS MX record (page 726). However, most mail clients (MUAs) do not interact directly with **sendmail** to receive email. Instead, they use POP or IMAP—protocols that include features for managing mail folders, leaving messages on the server, and reading only the subject of an email without downloading the entire message. If you want to collect your email from a system other than the one running the incoming mail server, you may need to set up a POP or IMAP server, as discussed on page 647.

PREREQUISITES

Install the following packages:

- **sendmail** (required)
- **sendmail-cf** (required to configure **sendmail**)
- **squirrelmail** (optional; provides Webmail, page 644)
- **spamassassin** (optional; provides spam filtering, page 640)
- **mailman** (optional; provides mailing list support, page 646)
- **dovecot** (optional; provides IMAP and POP incoming mail server daemons)

Run chkconfig to cause **sendmail** to start when the system goes multiuser (by default, **sendmail** does not run in single-user mode):

```
# /sbin/chkconfig sendmail on
```

Start **sendmail**. Because **sendmail** is normally running, you need to restart it to cause **sendmail** to reread its configuration files. The following restart command works even when **sendmail** is not running—it just fails to shut down **sendmail**:

```
# /sbin/service sendmail restart
Shutting down sendmail:                                    [  OK  ]
Shutting down sm-client:                                   [  OK  ]
Starting sendmail:                                         [  OK  ]
Starting sm-client:                                        [  OK  ]
```

Run chkconfig to cause the SpamAssassin daemon, **spamd**, to start when the system enters multiuser mode (SpamAssassin is normally installed in this configuration):

```
# /sbin/chkconfig spamassassin on
```

As with **sendmail**, SpamAssassin is normally running. Restart it to cause **spamd** to reread its configuration files:

```
# /sbin/service spamassassin restart
Stopping spamd:                                            [  OK  ]
Starting spamd:                                            [  OK  ]
```

The IMAP and POP protocols are implemented as several daemons. See page 647 for information on these daemons and how to start them.

NOTES

Firewall An SMTP server normally uses TCP port 25. If the SMTP server system is running a firewall, you need to open this port. Using the Red Hat graphical firewall tool (page 768), select **Mail (SMTP)** from the Trusted Services frame to open this port. For more general information see Chapter 25, which details iptables.

cyrus This chapter covers the IMAP and POP3 servers included in the **dovecot** package. Red Hat Linux also provides IMAP and POP3 servers in the **cyrus-imapd** package.

MORE INFORMATION

Web
sendmail www.sendmail.org
IMAP www.imap.org
IMAP and POP3 www.dovecot.org
IMAP and POP3 cyrusimap.web.cmu.edu
SquirrelMail www.squirrelmail.org
Postfix www.postfix.org/docs.html (alternative MTA, page 652)
Qmail qmail.area.com
Mailman www.list.org
procmail www.procmail.org
SpamAssassin spamassassin.org
Spam database razor.sourceforge.net

JUMPSTART I: CONFIGURING sendmail ON A CLIENT

You may not need to configure sendmail to send email

tip With **sendmail** running, give the command described under "Test" on page 631. As long as **sendmail** can connect to port 25 outbound, you should not need to set up **sendmail** to use an SMTP relay as described in this section. If you receive the mail sent by the test, you can skip this section.

This JumpStart configures an outbound **sendmail** server. This server

- Uses a remote SMTP server—typically an ISP—to relay outbound email to its destination (an SMTP relay).
- Sends to the SMTP server email originating from the local system only. It does not forward email originating from other systems.
- Does not handle inbound email. As is frequently the case, you need to use POP or IMAP to receive email.

To set up this server, you must edit **/etc/mail/sendmail.mc** and restart **sendmail**.

Change **sendmail.mc**
The **dnl** at the start of the following line in **sendmail.mc** indicates that this line is a comment:

```
dnl define(`SMART_HOST',`smtp.your.provider')
```

To specify a remote SMTP server, you must open **sendmail.mc** in an editor and change the preceding line, deleting **dnl** from the beginning of the line and replacing **smtp.your.provider** with the FQDN of your ISP's SMTP server (obtain this name from your ISP). Be careful not to alter the back ticks (`` ` ``) and the single quotation marks (') in this line. If your ISP's SMTP server is at **smtp.myisp.com**, you would change the line to

```
define(`SMART_HOST',`smtp.myisp.com')
```

Do not alter the back ticks (`) or the single quotation marks (')

tip Be careful not to alter the back ticks (`) or the single quotation marks (') in any line in **sendmail.mc**. These symbols control the way the m4 preprocessor converts **sendmail.mc** to **sendmail.cf**; **sendmail** will not work properly if you do not preserve these symbols.

Restart sendmail When you restart it, **sendmail** regenerates the **sendmail.cf** file from the **sendmail.mc** file you edited:

```
# /sbin/service sendmail restart
```

Test Test **sendmail** with the following command:

```
$ echo "my sendmail test" | /usr/sbin/sendmail user@remote.host
```

Replace *user@remote.host* with an email address on *another system* where you receive email. You need to send email to a remote system to make sure that **sendmail** is relaying your email.

JUMPSTART II: CONFIGURING sendmail ON A SERVER

If you want to receive inbound email sent to a registered domain that you own, you need to set up **sendmail** as an incoming mail server. This JumpStart describes how to set up such a server. This server

- Accepts outbound email from the local system only.
- Delivers outbound email directly to the recipient's system, without using a relay.
- Accepts inbound email from any system.

This server does not relay outbound email originating on other systems. Refer to "access: Sets Up a Relay Host" on page 638 if you want the local system to act as a relay. For this configuration to work, you must be able to make outbound connections from and receive inbound connections to port 25.

The line in **sendmail.mc** that limits **sendmail** to accepting inbound email from the local system only is

```
DAEMON_OPTIONS(`Port=smtp,Addr=127.0.0.1, Name=MTA')dnl
```

To allow **sendmail** to accept inbound email from other systems, remove the parameter **Addr=127.0.0.1,** from the preceding line:

```
DAEMON_OPTIONS(`Port=smtp, Name=MTA')dnl
```

By default, **sendmail** does not use a remote SMTP server to relay email, so there is nothing to change to cause **sendmail** to send email directly to recipients' systems. (JumpStart I set up a SMART_HOST to relay email.)

Once you have restarted **sendmail**, it will accept mail addressed to the local system, as long as a DNS MX record (page 726) points at the local system. If you are not running a DNS server, you must ask your ISP to set up an MX record.

How sendmail Works

Outbound email

When you send email, the MUA passes the email to **sendmail**, which creates in the **/var/spool/mqueue** (mail queue) directory two files that hold the message while **sendmail** processes it. To create a unique filename for a particular piece of email, **sendmail** generates a random string and uses that string in filenames pertaining to the email. The **sendmail** daemon stores the body of the message in a file named **df** (data file) followed by the generated string. It stores the headers and other information in a file named **qf** (queue file) followed by the generated string.

If a delivery error occurs, **sendmail** creates a temporary copy of the message that it stores in a file whose name starts with **tf** (temporary file) and logs errors in a file whose name starts **xf**. Once an email has been sent successfully, **sendmail** removes all files pertaining to that email from **/var/spool/mqueue**.

Incoming email

By default, the MDA stores incoming messages in users' files in the mail spool directory, **/var/spool/mail**, in **mbox** format. Within this directory, each user has a mail file named with the user's username. Mail remains in these files until it is collected, typically by an MUA. Once an MUA collects the mail from the mail spool, the MUA stores the mail as directed by the user, usually in the user's home directory hierarchy.

mbox versus **maildir**

The **mbox** format stores all messages for a user in a single file. To prevent corruption, the file must be locked while a process is adding messages to or deleting messages from the file; you cannot delete a message at the same time the MTA is adding messages. A competing format, **maildir**, stores each message in a separate file. This format does not use locks, allowing an MUA to read and delete messages at the same time as new mail is delivered. In addition, the **maildir** format is better able to handle larger mailboxes. The downside is that the **maildir** format adds overhead when you are using a protocol such as IMAP to check messages. The **dovecot** package supports both **mbox** and **maildir** formats. Qmail (page 652), a **sendmail** alternative, uses **maildir**-format mailboxes.

Mail logs

The **sendmail** daemon stores log messages in **/var/log/maillog**. Other mail servers, such as the dovecot **imap-login** and **pop3-login** daemons, may also log information to this file. Following is a sample log entry:

/var/log/maillog

```
# cat /var/log/maillog
...
Mar  3 16:25:33 MACHINENAME sendmail[7225]: i23GPXvm007224:
to=<user@localhost.localdomain>, ctladdr=<root@localhost.localdomain>
(0/0), delay=00:00:00, xdelay=00:00:00, mailer=local, pri=30514,
dsn=2.0.0, stat=Sent
```

Each log entry starts with a timestamp, the name of the system sending the email, the name of the mail server (**sendmail**), and a unique identification number. The address of the recipient follows the **to=** label and the address of the sender follows **ctladdr=**. Additional fields provide the name of the mailer and the time it took to send the message. If a message is sent correctly, the **stat=** label is followed by **Sent**.

A message is marked **Sent** when **sendmail** sends it; **Sent** does not indicate that the message has been delivered. If a message is not delivered because an error occurred farther down the line, the sender usually receives an email saying that it was not delivered and giving a reason why.

If you send and receive a lot of email, the **maillog** file can grow quite large. The **syslog** logrotate (page 559) entry is set up to archive and rotate the **maillog** files regularly.

Aliases and Forwarding

Three files can forward email: **.forward** (page 634), **aliases** (discussed next), and **virtusertable** (page 640). Table 20-1 on page 640 compares the three files.

/etc/aliases Most of the time when you send email, it goes to a specific person; the recipient, **user@system**, maps to a specific, real user on the specified system. Sometimes you may want email to go to a class of users and not to a specific recipient. Examples of classes of users include **postmaster**, **webmaster**, **root**, and **tech_support**. Different users may receive this email at different times or the email may be answered by a group of users. You can use the **/etc/aliases** file to map inbound addresses to local users, files, commands, and remote addresses.

Each line in **/etc/aliases** contains the name of a local pseudouser, followed by a colon, whitespace, and a comma-separated list of destinations. The default installation includes a number of aliases that redirect messages for certain pseudousers to **root**. These have the form

```
system:         root
```

Sending messages to the **root** account is a good way of making them easy to review. However, because **root**'s email is rarely checked, you may want to send copies to a real user. The following line forwards mail sent to **abuse** on the local system to **root** and **alex**:

```
abuse:          root, alex
```

You can create simple mailing lists with this type of alias. For example, the following alias sends copies of all email sent to **admin** on the local system to several users, including Zach, who is on a different system:

```
admin:          sam, helen, mark, zach@tcorp.com
```

You can direct email to a file by specifying an absolute pathname in place of a destination address. The following alias, which is quite popular among less conscientious system administrators, redirects email sent to **complaints** to **/dev/null** (page 448), where they disappear:

```
complaints:     /dev/null
```

You can also send email to standard input of a command by preceding the command with a pipe character (|). This technique is commonly used with mailing list software such as Mailman (page 646). For each list it maintains, Mailman has entries, such as the following entry for **mylist**, in the **aliases** file:

```
mylist:             "|/usr/lib/mailman/mail/mailman post mylist"
```

newaliases After you edit **/etc/aliases**, you must either run newaliases as **root** or restart **sendmail** to recreate the **aliases.db** file that **sendmail** reads.

praliases You can use praliases to list aliases currently loaded by **sendmail**:

```
# /usr/sbin/praliases | head -5
postmaster:root
daemon:root
adm:root
lp:root
shutdown:root
```

~/.forward Systemwide aliases are useful in many cases, but non**root** users cannot make or change them. Sometimes you may want to forward your own mail: Maybe you want mail from several systems to go to one address or perhaps you just want to forward your mail while you are working at another office for a week. The **~/.forward** file allows ordinary users to forward their email.

Lines in a **.forward** file are the same as the right column of the **aliases** file explained previously: Destinations are listed one per line and can be a local user, a remote email address, a filename, or a command preceded by a pipe character (|).

Mail that you forward does not go to your local mailbox. If you want to forward mail and keep a copy in your local mailbox, you must specify your local username preceded by a backslash to prevent an infinite loop. The following example sends Sam's email to himself on the local system and on the system at **tcorp.com**:

```
$cat ~sam/.forward
sams@tcorp.com
\sam
```

RELATED PROGRAMS

sendmail The **sendmail** package includes several programs. The primary program, **sendmail**, reads from standard input and sends an email to the recipient specified by its argument. You can use **sendmail** from the command line to check that the mail delivery system is working and to email the output of scripts. See page 631 for an example.

mailq The mailq utility displays the status of the outgoing mail queue and normally reports there are no messages in the queue. Messages in the queue usually indicate a problem with the local or remote **sendmail** configuration or a network problem.

```
# /usr/bin/mailq
/var/spool/mqueue is empty
                    Total requests: 0
```

mailstats The mailstats utility reports on the number and sizes of messages **sendmail** has sent and received since the date it displays on the first line:

```
# /usr/sbin/mailstats
Statistics from Sat Dec 24 16:02:34 2005
M   msgsfr  bytes_from   msgsto    bytes_to  msgsrej msgsdis  Mailer
0        0          0K    17181     103904K        0       0  prog
4   368386    4216614K   136456    1568314K    20616       0  esmtp
9   226151   26101362K   479025   12776528K     4590       0  local
=============================================================
T   594537   30317976K   632662   14448746K    25206       0
C   694638               499700                146185
```

In the preceding output, each mailer is identified by the first column, which displays the mailer number, and by the last column, which displays the name of the mailer. The second through fifth columns display the number and total sizes of messages sent and received by the mailer. The sixth and seventh columns display the number of messages rejected and discarded respectively. The row that starts with **T** lists the column totals, and the row that starts with **C** lists the number of TCP connections.

CONFIGURING sendmail

The **sendmail** configuration files reside in **/etc/mail**, where the primary configuration file is **sendmail.cf**. This directory contains other text configuration files, such as **access**, **mailertable**, and **virtusertable**. The **sendmail** daemon does not read these files but instead reads the corresponding ***.db** files in the same directory.

makemap

You can use makemap or give the command **make** from the **/etc/mail** directory to generate the ***.db** files, although this step is not usually necessary. The **sendmail** init script automatically generates these files when you start or restart **sendmail**:

```
# /sbin/service sendmail restart
```

THE sendmail.mc AND sendmail.cf FILES

This **sendmail.cf** file is not intended to be edited by hand and contains a large warning to this effect:

```
$ cat /etc/mail/sendmail.cf
...
######################################################################
#####
#####   DO NOT EDIT THIS FILE!  Only edit the source .mc file.
#####
######################################################################
...
```

EDITING sendmail.mc AND GENERATING sendmail.cf

The **sendmail.cf** file is generated from **sendmail.mc** using the m4 macro processor. It can be helpful to use a text editor that supports syntax highlighting, such as vim, to edit **sendmail.mc**.

dnl Many of the lines in **sendmail.mc** start with **dnl**, which stands for **delete to new line**; this token causes m4 to delete from the **dnl** to the end of the line (the next NEWLINE character). Because m4 ignores anything on a line after a **dnl** instruction, you can use **dnl** to introduce comments; it works the same way as # does in a shell script.

Many of the lines in **sendmail.mc** *end* with **dnl**. Because NEWLINEs immediately follow these **dnl**s, these **dnl**s are superfluous; you can remove them if you like.

After you edit **sendmail.mc**, you need to regenerate **sendmail.cf** to make your changes take effect. When you restart **sendmail**, the **sendmail** init script regenerates **sendmail.cf**.

ABOUT sendmail.mc

Lines near the beginning of **sendmail.mc** provide basic configuration information:

```
divert(-1)dnl
include(`/usr/share/sendmail-cf/m4/cf.m4')dnl
VERSIONID(`setup for Red Hat Linux')dnl
OSTYPE(`linux')dnl
```

The line that starts with **divert** tells m4 to discard extraneous output it may generate when processing this file.

The **include** statement tells m4 where to find the macro definition file that it will use to process the rest of this file; it points to the file named **cf.m4**. The **cf.m4** file contains other **include** statements that include parts of the **sendmail** configuration rule sets.

The **VERSIONID** statement defines a string that indicates the version of this configuration. You can change this string to include a brief comment about changes you have made to this file or other information. The value of this string is not significant to **sendmail**.

Do not change the **OSTYPE** statement unless you are migrating a **sendmail.mc** file from another operating system.

Other statements you may want to change are explained in the following sections and in the **sendmail** documentation.

Quoting m4 strings

tip The m4 macro processor, which converts **sendmail.mc** to **sendmail.cf**, requires strings to be preceded by a back tick (`) and closed with a single quotation mark (').

MASQUERADING

Typically you want your email to appear to come from the user and the domain where you receive email; sometimes the outbound server is in a different domain than the inbound server. You can cause **sendmail** to alter outbound messages so that they appear to come from a user and/or domain other than the one they are sent from: In other words, you *masquerade* (page 1042) the message.

Several lines in **sendmail.mc** pertain to this type of masquerading. Each is commented out in the file that Red Hat distributes:

```
dnl MASQUERADE_AS(`mydomain.com')dnl
dnl MASQUERADE_DOMAIN(localhost)dnl
dnl FEATURE(masquerade_entire_domain)dnl
```

The MASQUERADE_AS statement causes email that you send from the local system to appear to come from the specified domain (**mydomain.com** in the commented-out line in the distributed file). Remove the leading **dnl** and change **mydomain.com** to the domain name that you want mail to appear to come from.

The MASQUERADE_DOMAIN statement causes email from the specified system or domain to be masqueraded, just as local email is. That is, email from the system specified in this statement is treated as though it came from the local system: It is changed so that it appears to come from the domain specified in the MASQUERADE_AS statement. Remove the leading **dnl** and change **localhost** to the name of the system or domain that sends the email that you want to masquerade. If the name you specify has a leading period, it specifies a domain. If there is no leading period, the name specifies a system or host. The **sendmail.mc** file can include as many MASQUERADE_DOMAIN statements as necessary.

The **masquerade_entire_domain** feature statement causes **sendmail** also to masquerade subdomains of the domain specified in the MASQUERADE_DOMAIN statement. Remove the leading **dnl** to masquerade entire domains.

Accepting Email from Unknown Hosts

As configured by Red Hat, **sendmail** accepts email from domains that it cannot resolve (and that may not exist). To turn this feature off and cut down the amount of spam you receive, add **dnl** to the beginning of the following line:

```
FEATURE(`accept_unresolvable_domains')dnl
```

When this feature is off, **sendmail** uses DNS to look up the domains of all email it receives. If it cannot resolve the domain, it rejects the email.

Setting Up a Backup Server

You can set up a backup mail server to hold email when the primary mail server experiences problems. For maximum coverage, the backup server should be on a different connection to the Internet from the primary server.

Setting up a backup server is easy. Just remove the leading **dnl** from the following line in the *backup* mail server's **sendmail.mc** file:

```
dnl FEATURE(`relay_based_on_MX')dnl
```

DNS MX records (page 726) specify where email for a domain should be sent. You can have multiple MX records for a domain, each pointing to a different mail server. When a domain has multiple MX records, each record usually has a different

priority; the priority is specified by a two-digit number, where lower numbers specify higher priorities.

When attempting to deliver email, an MTA first tries to deliver email to the highest-priority server. If that delivery attempt fails, it tries to deliver to a lower-priority server. If you activate the **relay_based_on_MX** feature and point a low-priority MX record at a secondary mail server, the mail server will accept email for the domain. The mail server will then forward email to the server identified by the highest-priority MX record for the domain when that server becomes available.

28

Programming the Bourne Again Shell

In This Chapter

Chapter 7 introduced the shells and Chapter 9 went into detail about the Bourne Again Shell. This chapter introduces additional Bourne Again Shell commands, builtins, and concepts that carry shell programming to a point where it can be useful. The first part of this chapter covers programming control structures, which are also known as control flow constructs. These structures allow you to write scripts that can loop over command line arguments, make decisions based on the value of a variable, set up menus, and more. The Bourne Again Shell uses the same constructs found in such high-level programming languages as C.

The next part of this chapter discusses parameters and variables, going into detail about array variables, local versus global variables, special parameters, and positional parameters. The exploration of builtin commands covers type, which displays information about a command, and read, which allows you to accept user input in a shell script. The section on the exec builtin demonstrates how exec provides an efficient way to execute a command by replacing a process and explains how

you can use it to redirect input and output from within a script. The next section covers the trap builtin, which provides a way to detect and respond to operating system signals (such as that which is generated when you press CONTROL-C). The discussion of builtins concludes with a discussion of kill, which can abort a process, and getopts, which makes it easy to parse options for a shell script. (Table 28-6 on page 939 lists some of the more commonly used builtins.)

Next the chapter examines arithmetic and logical expressions and the operators that work with them. The final section walks through the design and implementation of two major shell scripts.

This chapter contains many examples of shell programs. Although they illustrate certain concepts, most use information from earlier examples as well. This overlap not only reinforces your overall knowledge of shell programming but also demonstrates how you can combine commands to solve complex tasks. Running, modifying, and experimenting with the examples in this book is a good way to become comfortable with the underlying concepts.

Do not name a shell script test

tip You can unwittingly create a problem if you give a shell script the name **test** because a Linux utility has the same name. Depending on how the **PATH** variable is set up and how you call the program, you may run your script or the utility, leading to confusing results.

This chapter illustrates concepts with simple examples, which are followed by more complex ones in sections marked "Optional." The more complex scripts illustrate traditional shell programming practices and introduce some Linux utilities often used in scripts. You can skip these sections without loss of continuity the first time you read the chapter. Return to them later when you feel comfortable with the basic concepts.

CONTROL STRUCTURES

The *control flow* commands alter the order of execution of commands within a shell script. Control structures include the **if...then**, **for...in**, **while**, **until**, and **case** statements. In addition, the **break** and **continue** statements work in conjunction with the control structures to alter the order of execution of commands within a script.

if...then

The **if...then** control structure has the following syntax:

```
if test-command
    then
        commands
fi
```

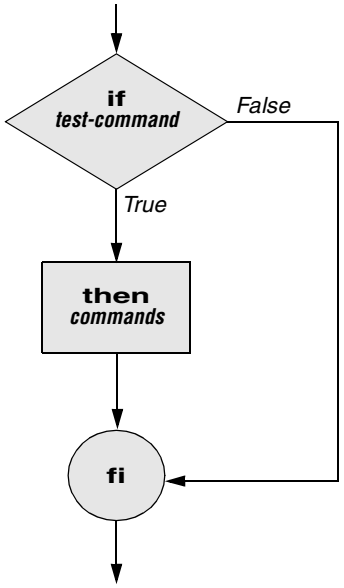

Figure 28-1 An **if...then** flowchart

The ***bold*** words in the syntax description are the items you supply to cause the structure to have the desired effect. The *nonbold* words are the keywords the shell uses to identify the control structure.

test builtin

Figure 28-1 shows that the **if** statement tests the status returned by the ***test-command*** and transfers control based on this status. The end of the **if** structure is marked by a **fi** statement, (*if* spelled backward). The following script prompts for two words, reads them, and then uses an **if** structure to execute commands based on the result returned by the test builtin when it compares the two words. (See the test info page for information on the test utility, which is similar to the test builtin.) The test builtin returns a status of *true* if the two words are the same and *false* if they are not. Double quotation marks around **$word1** and **$word2** make sure that test works properly if you enter a string that contains a SPACE or other special character:

```
$ cat if1
echo -n "word 1: "
read word1
echo -n "word 2: "
read word2

if test "$word1" = "$word2"
    then
        echo "Match"
fi
echo "End of program."
```

```
$ if1
word 1: peach
word 2: peach
Match
End of program.
```

In the preceding example the *test-command* is **test "$word1" = "$word2"**. The test builtin returns a *true* status if its first and third arguments have the relationship specified by its second argument. If this command returns a *true* status (= 0), the shell executes the commands between the **then** and **fi** statements. If the command returns a *false* status (not = 0), the shell passes control to the statement following **fi** without executing the statements between **then** and **fi**. The effect of this **if** statement is to display **Match** if the two words are the same. The script always displays **End of program.**

Builtins In the Bourne Again Shell, test is a builtin—part of the shell. It is also a stand-alone utility kept in **/usr/bin/test**. This chapter discusses and demonstrates many Bourne Again Shell builtins. You usually use the builtin version if it is available and the utility if it is not. Each version of a command may vary slightly from one shell to the next and from the utility to any of the shell builtins. See page 926 for more information on shell builtins.

Checking arguments The next program uses an **if** structure at the beginning of a script to check that you have supplied at least one argument on the command line. The **–eq** test operator compares two integers, where the **$#** special parameter (page 921) takes on the value of the number of command line arguments. This structure displays a message and exits from the script with an exit status of 1 if you do not supply at least one argument:

```
$ cat chkargs
if test $# -eq 0
    then
        echo "You must supply at least one argument."
        exit 1
fi
echo "Program running."
$ chkargs
You must supply at least one argument.
$ chkargs abc
Program running.
```

A test like the one shown in **chkargs** is a key component of any script that requires arguments. To prevent the user from receiving meaningless or confusing information from the script, the script needs to check whether the user has supplied the appropriate arguments. Sometimes the script simply tests whether arguments exist (as in **chkargs**). Other scripts test for a specific number or specific kinds of arguments.

You can use test to ask a question about the status of a file argument or the relationship between two file arguments. After verifying that at least one argument has been given on the command line, the following script tests whether the argument is the

name of an ordinary file (not a directory or other type of file) in the working directory. The test builtin with the **–f** option and the first command line argument (**$1**) check the file:

```
$ cat is_ordfile
if test $# -eq 0
    then
        echo "You must supply at least one argument."
        exit 1
fi
if test -f "$1"
    then
        echo "$1 is an ordinary file in the working directory"
    else
        echo "$1 is NOT an ordinary file in the working directory"
fi
```

You can test many other characteristics of a file with test and various options. Table 28-1 lists some of these options.

Table 28-1 Options to the test builtin

Option	Tests file to see if it
–d	Exists and is a directory file
–e	Exists
–f	Exists and is an ordinary file (not a directory)
–r	Exists and is readable
–s	Exists and has a size greater than 0 bytes
–w	Exists and is writable
–x	Exists and is executable

Other test options provide ways to test relationships between two files, such as whether one file is newer than another. Refer to later examples in this chapter for more detailed information.

Always test the arguments

tip To keep the examples in this book short and focused on specific concepts, the code to verify arguments is often omitted or abbreviated. It is a good practice to test arguments in shell programs that other people will use. Doing so results in scripts that are easier to run and debug.

[] is a synonym for test

The following example—another version of **chkargs**—checks for arguments in a way that is more traditional for Linux shell scripts. The example uses the bracket ([]) synonym for test. Rather than using the word test in scripts, you can surround the arguments to test with brackets. The brackets must be surrounded by whitespace (SPACEs or TABs).

```
$ cat chkargs2
if [ $# -eq 0 ]
    then
        echo "Usage: chkargs2 argument..." 1>&2
        exit 1
fi
echo "Program running."
exit 0
$ chkargs2
Usage: chkargs2 arguments
$ chkargs2 abc
Program running.
```

Usage message

The error message that **chkargs2** displays is called a *usage message* and uses the **1>&2** notation to redirect its output to standard error (page 270). After issuing the usage message, **chkargs2** exits with an exit status of 1, indicating that an error has occurred. The **exit 0** command at the end of the script causes **chkargs2** to exit with a 0 status after the program runs without an error. The Bourne Again Shell returns a 0 status if you omit the status code.

The usage message is commonly employed to specify the type and number of arguments the script takes. Many Linux utilities provide usage messages similar to the one in **chkargs2**. If you call a utility or other program with the wrong number or kind of arguments, you will often see a usage message. Following is the usage message that cp displays when you call it without any arguments:

```
$ cp
cp: missing file argument
Try 'cp --help' for more information.
```

if...then...else

The introduction of an **else** statement turns the **if** structure into the two-way branch shown in Figure 28-2. The **if...then...else** control structure has the following syntax:

```
if test-command
    then
        commands
    else
        commands
fi
```

Because a semicolon (**;**) ends a command just as a NEWLINE does, you can place **then** on the same line as **if** by preceding it with a semicolon. (Because **if** and **then** are separate builtins, they require a command separator between them; a semicolon and NEWLINE work equally well.) Some people prefer this notation for aesthetic reasons, while others like it because it saves space:

```
if test-command; then
        commands
    else
        commands
fi
```

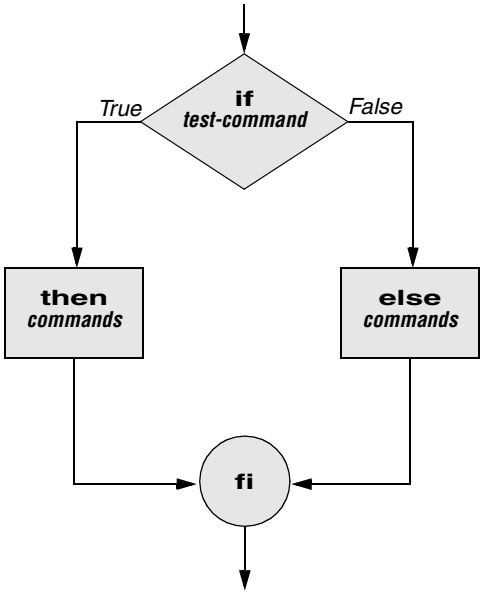

Figure 28-2 An **if...then...else** flowchart

If the ***test-command*** returns a *true* status, the **if** structure executes the commands between the **then** and **else** statements and then diverts control to the statement following **fi**. If the ***test-command*** returns a *false* status, the **if** structure executes the commands following the **else** statement.

When you run the next script, named **out**, with arguments that are filenames, it displays the files on the terminal. If the first argument is **–v** (called an option in this case), **out** uses less (page 128) to display the files one page at a time. After determining that it was called with at least one argument, **out** tests its first argument to see whether it is **–v**. If the result of the test is *true* (if the first argument is **–v**), **out** uses the shift builtin to shift the arguments to get rid of the **–v** and displays the files using less. If the result of the test is *false* (if the first argument is *not* **–v**), the script uses cat to display the files:

```
$ cat out
if [ $# -eq 0 ]
    then
        echo "Usage: out [-v] filenames..." 1>&2
        exit 1
fi
if [ "$1" = "-v" ]
    then
        shift
        less -- "$@"
    else
        cat -- "$@"
fi
```

optional In **out** the **––** argument to cat and less tells these utilities that no more options follow on the command line and not to consider leading hyphens (**–**) in the following list as indicating options. Thus **––** allows you to view a file with a name that starts with a hyphen. Although not common, filenames beginning with a hyphen do occasionally occur. (You can create such a file by using the command **cat > –fname**.) The **––** argument works with all Linux utilities that use the getopts builtin (page 936) to parse their options; it does not work with more and a few other utilities. This argument is particularly useful when used in conjunction with rm to remove a file whose name starts with a hyphen (**rm –– –fname**), including any that you create while experimenting with the **––** argument.

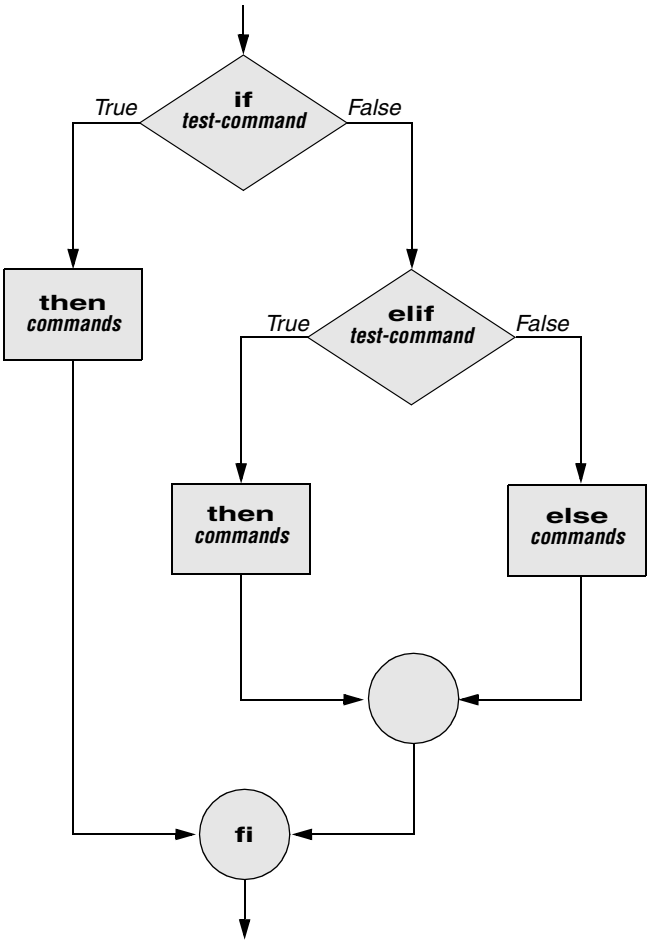

Figure 28-3 An **if...then...elif** flowchart

if...then...elif

The **if...then...elif** control structure (Figure 28-3) has the following syntax:

```
if test-command
    then
        commands
    elif test-command
        then
            commands
...
    else
        commands
fi
```

The **elif** statement combines the **else** statement and the **if** statement and allows you to construct a nested set of **if...then...else** structures (Figure 28-3). The difference between the **else** statement and the **elif** statement is that each **else** statement must be paired with a **fi** statement, whereas multiple nested **elif** statements require only a single closing **fi** statement.

The following example shows an **if...then...elif** control structure. This shell script compares three words that the user enters. The first **if** statement uses the Boolean operator AND (**–a**) as an argument to test. The test builtin returns a *true* status only if the first and second logical comparisons are *true* (that is, if **word1** matches **word2** and **word2** matches **word3**). If test returns a *true* status, the script executes the command following the next **then** statement, passes control to the statement following **fi**, and terminates:

```
$ cat if3
echo -n "word 1: "
read word1
echo -n "word 2: "
read word2
echo -n "word 3: "
read word3

if [ "$word1" = "$word2" -a "$word2" = "$word3" ]
    then
        echo "Match: words 1, 2, & 3"
    elif [ "$word1" = "$word2" ]
    then
        echo "Match: words 1 & 2"
    elif [ "$word1" = "$word3" ]
    then
        echo "Match: words 1 & 3"
    elif [ "$word2" = "$word3" ]
    then
        echo "Match: words 2 & 3"
    else
        echo "No match"
fi
```

```
$ if3
word 1: apple
word 2: orange
word 3: pear
No match
$ if3
word 1: apple
word 2: orange
word 3: apple
Match: words 1 & 3
$ if3
word 1: apple
word 2: apple
word 3: apple
Match: words 1, 2, & 3
```

If the three words are not the same, the structure passes control to the first **elif**, which begins a series of tests to see if any pair of words is the same. As the nesting continues, if any one of the **if** statements is satisfied, the structure passes control to the next **then** statement and subsequently to the statement following **fi**. Each time an **elif** statement is not satisfied, the structure passes control to the next **elif** statement. The double quotation marks around the arguments to echo that contain ampersands (&) prevent the shell from interpreting the ampersands as special characters.

optional THE lnks SCRIPT

The following script, named **lnks**, demonstrates the **if...then** and **if...then...elif** control structures. This script finds hard links to its first argument, a filename. If you provide the name of a directory as the second argument, **lnks** searches for links in that directory and all subdirectories. If you do not specify a directory, **lnks** searches the working directory and its subdirectories. This script does not locate symbolic links.

```
$ cat lnks
#!/bin/bash
# Identify links to a file
# Usage: lnks file [directory]

if [ $# -eq 0 -o $# -gt 2 ]; then
    echo "Usage: lnks file [directory]" 1>&2
    exit 1
fi
if [ -d "$1" ]; then
    echo "First argument cannot be a directory." 1>&2
    echo "Usage: lnks file [directory]" 1>&2
    exit 1
else
    file="$1"
fi
```

```
if [ $# -eq 1 ]; then
        directory="."
    elif [ -d "$2" ]; then
        directory="$2"
    else
        echo "Optional second argument must be a directory." 1>&2
        echo "Usage: lnks file [directory]" 1>&2
        exit 1
fi

# Check that file exists and is an ordinary file:
if [ ! -f "$file" ]; then
    echo "lnks: $file not found or special file" 1>&2
    exit 1
fi
# Check link count on file
set -- $(ls -l "$file")
linkcnt=$2
if [ "$linkcnt" -eq 1 ]; then
    echo "lnks: no other hard links to $file" 1>&2
    exit 0
fi

# Get the inode of the given file
set $(ls -i "$file")

inode=$1

# Find and print the files with that inode number
echo "lnks: using find to search for links..." 1>&2
find "$directory" -xdev -inum $inode -print
```

Alex has a file named **letter** in his home directory. He wants to find links to this file in his and other users' home directory file trees. In the following example, Alex calls **lnks** from his home directory to perform the search. The second argument to **lnks**, **/home**, is the pathname of the directory he wants to start the search in. The **lnks** script reports that **/home/alex/letter** and **/home/jenny/draft** are links to the same file:

```
$ lnks letter /home
lnks: using find to search for links...
/home/alex/letter
/home/jenny/draft
```

In addition to the **if...then...elif** control structure, **lnks** introduces other features that are commonly used in shell programs. The following discussion describes **lnks** section by section.

Specify the shell

The first line of the **lnks** script uses **#!** (page 274) to specify the shell that will execute the script:

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

In this chapter the **#!** notation appears only in more complex examples. It ensures that the proper shell executes the script, even when the user is running a different shell or the script is called from another shell script.

Comments

The second and third lines of **lnks** are comments; the shell ignores the text that follows a pound sign up to the next NEWLINE character. These comments in **lnks** briefly identify what the file does and how to use it:

```
# Identify links to a file
# Usage: lnks file [directory]
```

Usage messages

The first **if** statement tests whether **lnks** was called with zero arguments or more than two arguments:

```
if [ $# -eq 0 -o $# -gt 2 ]; then
    echo "Usage: lnks file [directory]" 1>&2
    exit 1
fi
```

If either of these conditions is *true,* **lnks** sends a usage message to standard error and exits with a status of 1. The double quotation marks around the usage message prevent the shell from interpreting the brackets as special characters. The brackets in the usage message indicate that the **directory** argument is optional.

The second **if** statement tests whether the first command line argument (**$1**) is a directory (the **–d** argument to test returns a *true* value if the file exists and is a directory):

```
if [ -d "$1" ]; then
    echo "First argument cannot be a directory." 1>&2
    echo "Usage: lnks file [directory]" 1>&2
    exit 1
else
    file="$1"
fi
```

If the first argument is a directory, **lnks** displays a usage message and exits. If it is not a directory, **lnks** saves the value of **$1** in the **file** variable because later in the script set resets the command line arguments. If the value of **$1** is not saved before the **set** command is issued, its value will be lost.

Test the arguments

The next section of **lnks** is an **if...then...elif** statement:

```
if [ $# -eq 1 ]; then
        directory="."
    elif [ -d "$2" ]; then
        directory="$2"
    else
        echo "Optional second argument must be a directory." 1>&2
        echo "Usage: lnks file [directory]" 1>&2
        exit 1
fi
```

The first ***test-command*** determines whether the user specified a single argument on the command line. If the ***test-command*** returns 0 (*true*), the user-created variable named **directory** is assigned the value of the working directory (**.**). If the ***test-command*** returns *false*, the **elif** statement tests whether the second argument is a directory. If it is a directory, the **directory** variable is set equal to the second command line argument, **$2**. If **$2** is not a directory, **lnks** sends a usage message to standard error and exits with a status of 1.

The next **if** statement in **lnks** tests whether **$file** does not exist. This test keeps **lnks** from wasting time looking for links to a nonexistent file.

The test builtin with the three arguments **!**, **–f**, and **$file** evaluates to *true* if the file **$file** does *not* exist:

```
[ ! -f "$file" ]
```

The **!** operator preceding the **–f** argument to test negates its result, yielding *false* if the file **$file** *does* exist and is an ordinary file.

Next **lnks** uses set and **ls –l** to check the number of links **$file** has:

```
# Check link count on file
set -- $(ls -l "$file")
linkcnt=$2
if [ "$linkcnt" -eq 1 ]; then
    echo "lnks: no other hard links to $file" 1>&2
    exit 0
fi
```

The set builtin uses command substitution (page 334) to set the positional parameters to the output of **ls –l**. The second field in this output is the link count, so the user-created variable **linkcnt** is set equal to **$2**. The **––** used with set prevents set from interpreting as an option the first argument produced by **ls –l** (the first argument is the access permissions for the file and typically begins with **–**). The **if** statement checks whether **$linkcnt** is equal to 1; if it is, **lnks** displays a message and exits. Although this message is not truly an error message, it is redirected to standard error. The way **lnks** has been written, all informational messages are sent to standard error. Only the final product of **lnks**—the pathnames of links to the specified file—is sent to standard output, so you can redirect the output as you please.

If the link count is greater than one, **lnks** goes on to identify the *inode* (page 1037) for **$file**. As explained on page 193, comparing the inodes associated with filenames is a good way to determine whether the filenames are links to the same file. The **lnks** script uses set to set the positional parameters to the output of **ls –i**. The first argument to set is the inode number for the file, so the user-created variable named **inode** is assigned the value of **$1**:

```
# Get the inode of the given file
set $(ls -i "$file")

inode=$1
```

Finally **lnks** uses the find utility to search for files having inode numbers that match **$inode**:

```
# Find and print the files with that inode number
echo "lnks: using find to search for links..." 1>&2
find "$directory" -xdev -inum $inode -print
```

The find utility searches for files that meet the criteria specified by its arguments, beginning its search with the directory specified by its first argument (**$directory**) and searching all subdirectories. The remaining arguments specify that the filenames of files having inodes matching **$inode** should be sent to standard output. Because files in different filesystems can have the same inode number and not be linked, find must search only directories in the same filesystem as **$directory**. The **–xdev** argument prevents find from searching directories on other filesystems. Refer to page 190 for more information about filesystems and links.

The echo command preceding the find command in **lnks**, which tells the user that find is running, is included because find frequently takes a long time to run. Because **lnks** does not include a final exit statement, the exit status of **lnks** is that of the last command it runs, find.

DEBUGGING SHELL SCRIPTS

When you are writing a script such as **lnks**, it is easy to make mistakes. You can use the shell's **–x** option to help debug a script. This option causes the shell to display each command before it runs the command. Tracing a script's execution in this way can give you information about where a problem lies.

You can run **lnks** as in the previous example and cause the shell to display each command before it is executed. Either set the **–x** option for the current shell (**set –x**) so that all scripts display commands as they are run or use the **–x** option to affect only the shell that is running the script called by the command line.

```
$ bash -x lnks letter /home
+ '[' 2 -eq 0 -o 2 -gt 2 ']'
+ '[' -d letter ']'
+ file=letter
+ '[' 2 -eq 1 ']'
+ '[' -d /home ']'
+ directory=/home
+ '[' '!' -f letter ']'
...
```

PS4 Each command that the script executes is preceded by the value of the **PS4** variable—a plus sign (**+**) by default, so you can distinguish debugging output from script-produced output. You must export **PS4** if you set it in the shell that calls the script. The next command sets **PS4** to **>>>>** followed by a SPACE and exports it:

```
$ export PS4='>>>> '
```

You can also set the **–x** option of the shell running the script by putting the following set command at the top of the script:

```
set -x
```

Put **set –x** anywhere in the script you want to turn debugging on. Turn the debugging option off with a plus sign.

```
set +x
```

The **set –o xtrace** and **set +o xtrace** commands do the same things as **set –x** and **set +x**, respectively.

for...in

The **for...in** control structure has the following syntax:

for ***loop-index*** *in* ***argument-list***
do
    commands
done

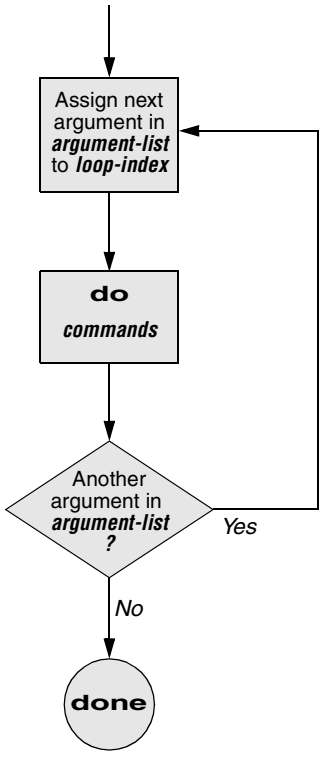

Figure 28-4 A **for...in** flowchart

The **for...in** structure (Figure 28-4) assigns the value of the first argument in the ***argument-list*** to the ***loop-index*** and executes the ***commands*** between the **do** and **done** statements. The **do** and **done** statements mark the beginning and end of the **for** loop.

After it passes control to the **done** statement, the structure assigns the value of the second argument in the ***argument-list*** to the ***loop-index*** and repeats the ***commands***. The structure repeats the ***commands*** between the **do** and **done** statements one time for each argument in the ***argument-list***. When the structure exhausts the ***argument-list***, it passes control to the statement following **done**.

The following **for...in** structure assigns **apples** to the user-created variable **fruit** and then displays the value of **fruit**, which is **apples**. Next the structure assigns **oranges** to **fruit** and repeats the process. When it exhausts the argument list, the structure transfers control to the statement following **done**, which displays a message.

```
$ cat fruit
for fruit in apples oranges pears bananas
do
    echo "$fruit"
done
echo "Task complete."

$ fruit
apples
oranges
pears
bananas
Task complete.
```

The next script lists the names of the directory files in the working directory by looping over all the files, using test to determine which files are directories:

```
$ cat dirfiles
for i in *
do
    if [ -d "$i" ]
        then
            echo "$i"
    fi
done
```

The ambiguous file reference character * matches the names of all files (except hidden files) in the working directory. Prior to executing the **for** loop, the shell expands the * and uses the resulting list to assign successive values to the index variable **i**.

for

The **for** control structure has the following syntax:

for ***loop-index***
do
    commands
done

In the **for** structure the *loop-index* takes on the value of each of the command line arguments, one at a time. It is the same as the **for...in** structure (Figure 28-4) except for where it gets values for the *loop-index*. The **for** structure performs a sequence of commands, usually involving each argument in turn.

The following shell script shows a **for** structure displaying each command line argument. The first line of the script, **for arg**, implies **for arg in "$@"**, where the shell expands **"$@"** into a list of quoted command line arguments **"$1" "$2" "$3"** and so on. The balance of the script corresponds to the **for...in** structure.

```
$ cat for_test
for arg
do
    echo "$arg"
done
$ for_test candy gum chocolate
candy
gum
chocolate
```