THE DEBIAN PACKAGE MANAGEMENT SYSTEM

The Debian Package Management System (DPMS) is the foundation for managing software on Debian and Debian-like systems. As is expected of any software management system, DPMS provides for easy installation and removal of software packages. Debian packages end with the .deb extension.

At the core of the DPMS is the dpkg (Debian Package) application. dpkg works in the back-end of the system, and several other command-line tools and graphical user interface (GUI) tools have been written to interact with it. Packages in Debian are fondly called ".deb" files. dpkg can directly manipulate .deb files. Various other wrapper tools have been developed to interact with dpkg, either directly or indirectly.

APT

APT is a highly regarded and sophisticated toolset. It is an example of a wrapper tool that interacts directly with dpkg. APT is actually a library of programming functions that are used by other middle-ground tools, like **`apt-get`** and **`apt-cache`**, to manipulate software on Debian-like systems. Several user-land applications have been developed that rely on APT. (*User-land* refers to non-kernel programs and tools.) Examples of such applications are synaptic, aptitude, and dselect. The user-land tools are generally more user-friendly than their command-line counterparts. APT has also been successfully ported to other operating systems.

One fine difference between APT and dpkg is that APT does not directly deal with .deb packages; instead, it manages software via the locations (repositories) specified in a configuration file. This file is the sources.list file. APT utilities use the sources.list file to locate archives (or repositories) of the package distribution system in use on the system.

It should be noted that any of the components of the DPMS (dpkg, apt, or the GUI tools) can be used to directly manage software on Debian-like systems. The tool of choice depends on the user's level of comfort and familiarity with the tool in question.

Figure 3-1 shows what can be described as the DPMS triangle. The tool at the apex of the triangle (dpkg) is the most difficult to use and the most powerful, followed by the next easiest to use (APT), and then followed finally by the user-friendly user-land tools.

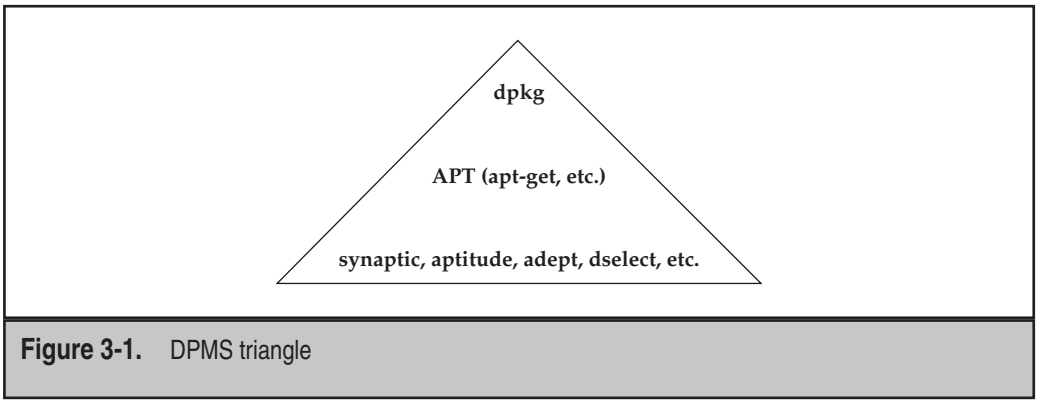

Figure 3-1. DPMS triangle

This will return a long list of matches. You can then look through the list and pick the package you want.

NOTE By default, Yum tries to access repositories that are located somewhere on the Internet. Therefore, your system needs to be able to access the Internet to use Yum in its default state. You can also create your own local software repository on the local file system or on your local area network (LAN) and Yumify it. Simply copy the entire contents of the distribution media (DVD/CD) somewhere and run the **`yum-arch`** command against the directory location.

SOFTWARE MANAGEMENT IN UBUNTU

As we mentioned earlier, software management in the Debian-like distros such as Ubuntu is done using DPMS and all the attendant applications built around it, such as APT and dpkg. In this section we will look at how to perform basic software management tasks on Debian-like distros.

Querying for Information

On your Ubuntu server, the equivalent command to list all currently installed software is

```
yyang@ubuntu-server:~$ dpkg -l
```

The command to get basic information about an installed package is

```
yyang@ubuntu-server:~$ dpkg -l bash
```

The command to get more detailed information about the bash package is

```
yyang@ubuntu-server:~$ dpkg --print-avail bash
```

To view the list of files that comes with the bash package, type

```
yyang@ubuntu-server:~$ dpkg-query -L bash
```

The querying capabilities of dpkg are extensive. You can use DPMS to query for specific information about a package. For example, to find out the size of the installed bash package, you can type

```
yyang@ubuntu-server:~$ dpkg-query -W --showformat='${Package} ${Installed-Size}
\n' bash
```

Installing Software in Ubuntu

There are several ways to get software installed on Ubuntu systems. You can use dpkg to directly install a .deb (pronounced dot deb) file, or you may choose to use **apt-get** to install any software available in the Ubuntu repositories on the Internet or locally (CD/DVD ROM, file system, etc).

NOTE Installing software and uninstalling software on a system is considered an administrative or privileged function. This is why you will notice that any commands that require superuser privileges are preceded with the **sudo** command. The **sudo** command can be used to execute commands in the context of a privileged user (or another user). On the other hand, querying the software database is not considered a privileged function. To use dpkg to install a .deb package named lynx_2.8.6-2ubuntu2_i386.deb , type

```
yyang@ubuntu-serverA:~$ sudo dpkg --install lynx_2.8.6-2ubuntu2_i386.deb
```

Using **apt-get** to install software is a little easier, because APT will usually take care of any dependency issues for you. The only caveat is that the repositories configured in the sources.list file (/etc/apt/sources.list) have to be reachable either over the Internet or locally. The other advantage to using APT to install software is that you only need to know a part of the name of the software; you don't need to know the exact version number. You also don't need to manually download the software before installing.

To use **apt-get** to install a package called lynx, type

```
yyang@ubuntu-server:~$ sudo apt-get install lynx
```

Removing Software in Ubuntu

Uninstalling software in Ubuntu using dpkg is as easy as typing

```
yyang@ubuntu-server:~$ sudo dpkg --remove lynx
```

You can also use **apt-get** to remove software by using the **remove** option. To remove the lynx package using **apt-get**, type

```
yyang@ubuntu-server:~$ sudo apt-get remove lynx
```

A less commonly used method for uninstalling software with APT is by using the install switch, but appending a minus sign to the package name to be removed. This may be useful when you want to install and remove a package in one shot. To remove the Lynx package using this method, type

```
yyang@ubuntu-server:~$ sudo apt-get install lynx-
```

APT makes it easy to completely remove software and any attendant configuration file(s) from a system. This allows you to truly start from scratch by getting rid of any

customized configuration files. Assuming we completely want to remove the lynx application from the system, we would type

```
yyang@ubuntu-server:~$ sudo apt-get --purge remove lynx
```

GUI RPM Package Managers

For those who like a good GUI tool to help simplify their lives, several package managers with GUI front-ends are available. Doing all the dirty work behind these pretty GUI front-ends is RPM. The GUI tools allow you to do quite a few things without forcing you to remember command-line parameters. Some of the more popular ones with each distribution or desktop environment are listed in the sections that follow.

Fedora

You can launch the GUI package management tool (see Figure 3-2) in Fedora by selecting System menu | Administration | Add/Remove Software. You can also launch the Fedora package manager from the command line, simply by typing

```
[root@fedora-serverA ~]# gpk-application
```

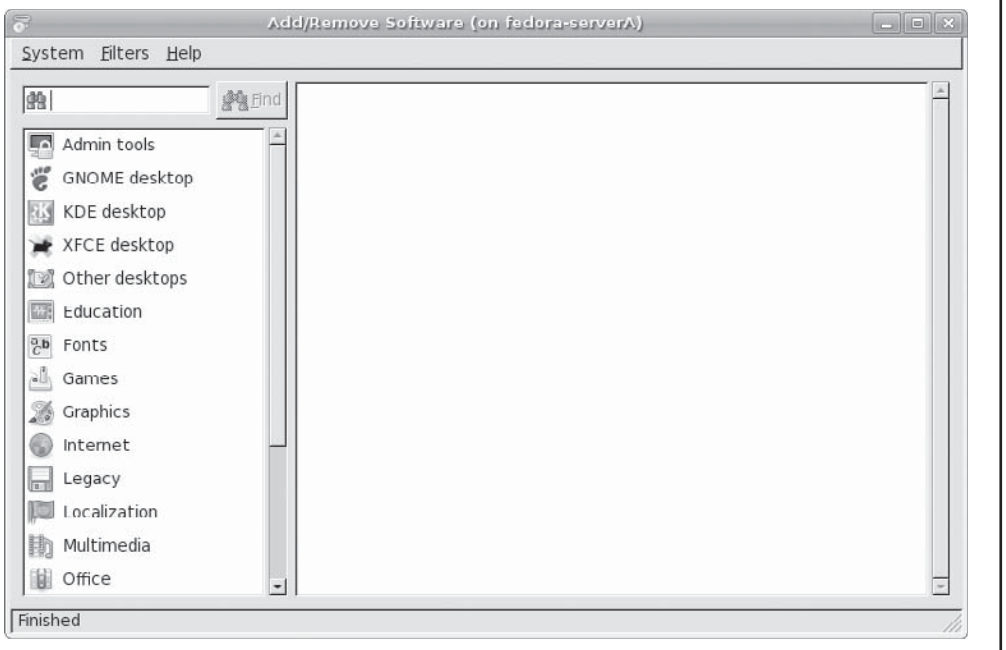

Figure 3-2. Fedora GUI package manager

UNIX/Linux was designed from the ground up to be a multiuser operating system. A multiuser operating system will not be much good without users. And this brings us to the topic of managing users in Linux. Associated with each user is the user's baggage. This baggage might include files, processes, resources, and other information. When dealing with a multiuser system, it is necessary for a system administrator to have a good understanding of what constitutes a user (and all that user's baggage), a group, and how they interact together.

User accounts are used on computer systems to determine who has access to what. The ability of a user to access a system is determined by whether that user exists and has the proper permissions to use the system.

In this chapter, we will examine the technique of managing users on a single host. We'll begin by exploring the actual database files that contain information about users. From there, we'll examine the system tools available to manage the files automatically.

WHAT EXACTLY CONSTITUTES A USER?

Under Linux, every file and program must be owned by a *user.* Each user has a unique identifier called a *user ID (UID).* Each user must also belong to at least one *group*, a collection of users established by the system administrator. Users may belong to multiple groups. Like users, groups also have unique identifiers, called *group IDs (GIDs).*

The accessibility of a file or program is based on its UIDs and GIDs. A running program inherits the rights and permissions of the user who invokes it. (SetUID and SetGID, discussed in "Understanding SetUID and SetGID Programs" later in this chapter, create an exception to this rule.) Each user's rights can be defined in one of two ways: as those of a *normal user* or the *root user.* Normal users can access only what they own or have been given permission to run; permission is granted because the user either belongs to the file's group or because the file is accessible to all users. The root user is allowed to access all files and programs in the system, whether or not root owns them. The root user is often called a *superuser.*

If you are accustomed to Windows, you can draw parallels between that system's user management and Linux's user management. Linux UIDs are comparable to Windows SIDs (system IDs), for example. In contrast to Microsoft Windows, you may find the Linux security model maddeningly simplistic: Either you're root or you're not. Normal users cannot have root privileges in the same way normal users can be granted administrator access under Windows. Although this approach is a little less common, you can also implement finer-grained access control through the use of access control lists (ACLs) in Linux, as you can with Windows. Which system is better? Depends on what you want and whom you ask.

Where User Information Is Kept

If you're already used to Windows 200*x* user management, you're familiar with the Active Directory tool that takes care of the nitty-gritty details of the user database. This tool is convenient, but it makes developing your own administrative tools trickier, since

the only other way to read or manipulate user information is through a series of Lightweight Directory Access Protocol (LDAP), Kerberos, or programmatic system calls.

In contrast, Linux takes the path of traditional UNIX and keeps all user information in straight text files. This is beneficial for the simple reason that it allows you to make changes to user information without the need of any other tool but a text editor such as **vi**. In many instances, larger sites take advantage of these text files by developing their own user administration tools so that they can not only create new accounts, but also automatically make additions to the corporate phone book, web pages, and so on.

However, users and groups working with UNIX style for the first time may prefer to stick with the basic user management tools that come with the Linux distribution. We'll discuss those tools in "User Management Tools" later in this chapter. For now, let's examine the text files that store user and group information in Linux.

The /etc/passwd File

The **/etc/passwd** file stores the user's login, encrypted password entry, UID, default GID, name (sometimes called GECOS), home directory, and login shell. Each line in the file represents information about a user. The lines are made up of various standard fields, with each field delimited by a colon. A sample entry from a **passwd** file with its various fields is illustrated in Figure 4-1.

The fields of the **/etc/passwd** file are discussed in detail in the sections that follow.

Username Field

This field is also referred to as the login field or the account field. It stores the name of the user on the system. The username must be a unique string and uniquely identifies a user to the system. Different sites use different methods for generating user login names. A common method is to use the first letter of the user's first name and append the user's last name. This usually works, because the chances are relatively slim that one would have users with the same first and last names. There are, of course, several variations of this method. For example, for a user whose first name is "Ying" and whose last name is "Yang"—a username of "yyang" can be assigned to that user.

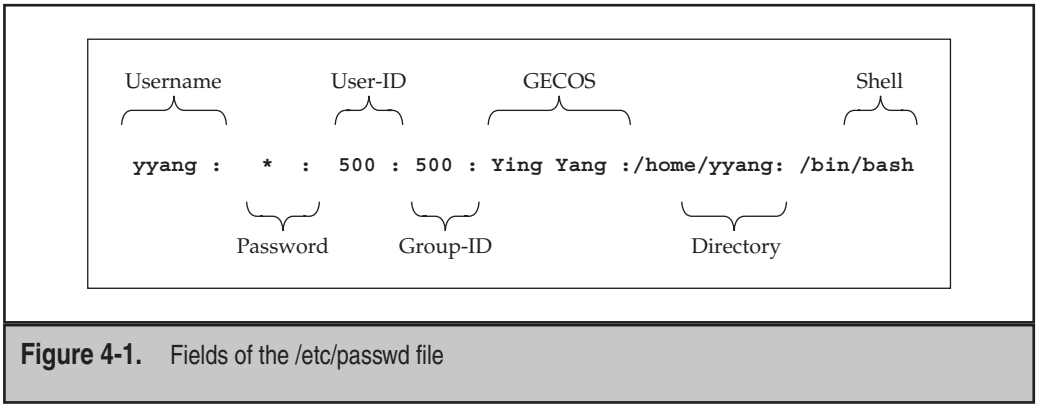

Figure 4-1. Fields of the /etc/passwd file

Password Field

This field contains the encrypted password for the user. On most modern Linux systems, this field contains a letter *x* to indicate that shadow passwords are being used on the system (discussed in detail later). Every user account on the system should have a password or, at the very least, be tagged as impossible to log in. This is crucial to the security of the system—weak passwords make compromising a system just that much simpler.

The original philosophy behind passwords is actually quite interesting, especially since we still rely on a significant part of it today. The idea is simple: Instead of relying on protected files to keep passwords a secret, the system would encrypt the password using an AT&T-developed (and National Security Agency–approved) algorithm called Data Encryption Standard (DES) and leave the encrypted value publicly viewable. What originally made this secure was that the encryption algorithm was computationally difficult to break. The best most folks could do was a brute-force dictionary attack, where automated systems would iterate through a large dictionary and rely on the nature of users to pick English words for their passwords. Many people tried to break DES itself, but since it was an open algorithm that anyone could study, it was made much more bulletproof before it was actually deployed.

When users entered their passwords at a login prompt, the password they entered would be encrypted. The encrypted value would then be compared against the user's password entry. If the two encrypted values matched, the user was allowed to enter the system. The actual algorithm for performing the encryption was computationally cheap enough that a single encryption wouldn't take too long. However, the tens of thousands of encryptions that would be needed for a dictionary attack would take prohibitively long.

But then a problem occurred: Moore's Law on processor speed doubling every 18 months held true, and home computers were becoming powerful and fast enough that programs were able to perform a brute-force dictionary attack within days rather than weeks or months. Dictionaries got bigger, and the software got smarter. The nature of passwords thus needed to be reevaluated. One solution has been to improve the algorithm used to perform the encryption of passwords. Some distributions of Linux have followed the path of the FreeBSD operating system and used the Message-Digest algorithm 5 (MD5) scheme. This has increased the complexity involved in cracking passwords, which, when used in conjunction with shadow passwords (discussed later on), works quite well. (Of course, this is assuming you make your users choose good passwords!)

TIP Choosing good passwords is always a chore. Your users will inevitably ask, "What then, O Almighty System Administrator, makes a good password?" Here's your answer: a non-language word (not English, not Spanish, not German, not a human-language word), preferably with mixed case, numbers, and punctuation—in other words, a string that looks like line noise. Well, this is all nice and wonderful, but if a password is too hard to remember, most people will quickly defeat its purpose by writing it down and keeping it in an easily viewed place. So better make it memorable! A good technique might be to choose a phrase and then pick the first letter of every word in the phrase. Thus, the phrase "coffee is VERY GOOD for you and me" becomes ciVG4yam. The phrase is memorable, even if the resulting password isn't.

User-ID Field (UID)

This field stores a unique number that the operating system and other applications use to identify the user and determine access privileges. It is the numerical equivalent of the Username field. The UID must be unique for every user, with the exception of the UID 0 (zero). Any user who has a UID of 0 has root (administrative) access and thus has the full run of the system. Usually, the only user who has this specific UID has the login root. It is considered bad practice to allow any other users or usernames to have a UID of 0. This is notably different from the Windows NT and 2000 models, in which any number of users can have administrative privileges.

Different Linux distributions sometimes adopt different UID numbering schemes. For example, Fedora and Red Hat Enterprise Linux (RHEL) reserve the UID 99 for the user "nobody," while SuSE and Ubuntu Linux use the UID 65534 for the user "nobody."

Group-ID Field (GID)

The next field in the **/etc/passwd** file is the group-ID entry. It is the numerical equivalent of the primary group that the user belongs to. This field also plays an important role in determining user access privileges. It should be noted that besides a user's primary group, a user can belong to other groups as well (more on this in the section "The /etc/group File").

GECOS

This field can store various pieces of information for a user. It can act as a placeholder for the user description, full name (first name and last name), telephone number, and so on. This field is optional and as result can be left blank. It is also possible to store multiple entries in this field by simply separating the different entries with a comma.

NOTE GECOS is an acronym for General Electric Comprehensive Operating System (now referred to as GCOS) and is a carryover from the early days of computing.

Directory

This is usually the user's home directory, but it can also be any arbitrary location on the system. Every user who actually logs into the system needs a place for configuration files that are unique to the user. This place, called a *home directory*, allows each user to work in a customized environment without having to change the environment customized by another user—even if both users are logged into the system at the same time. In this directory, users are allowed to keep not only their configuration files, but their regular work files as well.

Startup Scripts

Startup scripts are not quite a part of the information stored in the users' database in Linux. But they nonetheless play an important role in determining and controlling a user's environment. In particular, the startup scripts in Linux are usually stored under the user's home directory… and hence the need to mention them while still on the subject of the directory (home directory) field in the **/etc/passwd** file.

Linux/UNIX was built from the get-go as a multiuser environment. Each user is allowed to have his or her own configuration files; thus, the system appears to be customized for each particular user (even if other people are logged in at the same time). The customization of each individual user environment is done through the use of shell scripts, run control files, and the like. These files can contain a series of commands to be executed by the shell that starts when a user logs in. In the case of the bash shell, for example, one of its startup files is the **.bashrc** file. (Yes, there is a period in front of the filename—filenames preceded by periods, also called dot files, are hidden from normal directory listings.) You can think of shell scripts in the same light as batch files, except shell scripts can be much more capable. The **.bashrc** script in particular is similar in nature to **autoexec.bat** in the Windows world.

Various Linux software packages use application-specific and customizable options in directories or files that begin with a dot (.) in each user's home directory. Some examples are **.mozilla** and **.kde**. Here are some common dot (.) files that are present in each user's home directory:

- **.bashrc/.profile** Configuration files for BASH.
- **.tcshrc/.login** Configuration files for tcsh.
- **.xinitrc** This script overrides the default script that gets called when you log into the X Window System.
- **.Xdefaults** This file contains defaults that you can specify for X Window System applications.

When you create a user's account, a set of default dot files are also created for the user; this is mostly for convenience, to help get the user started. The user creation tools discussed later on help you do this automatically. The default files are stored under the **/etc/skel** directory.

For the sake of consistency, most sites place home directories at **/home** and name each user's directory by that user's login name. Thus, for example, if your login name were "yyang," your home directory would be **/home/yyang**. The exception to this is for some special system accounts, such as a root user's account or a system service. The

superuser's (root's) home directory in Linux is usually set to **/root** (but for most variants of UNIX, such as Solaris, the home directory is traditionally /). An example of a special system service that might need a specific working directory could be a web server whose web pages are served from the **/var/www/** directory.

In Linux, the decision to place home directories under **/home** is strictly arbitrary, but it does make organizational sense. The system really doesn't care where we place home directories, so long as the location for each user is specified in the password file.

Shell

When users log into the system, they expect an environment that can help them be productive. This first program that users encounter is called a *shell.* If you're used to the Windows side of the world, you might equate this with command.com, Program Manager, or Windows Explorer (not to be confused with Internet Explorer, which is a web browser).

Under UNIX/Linux, most shells are text-based. A popular default user shell in Linux is the Bourne Again Shell, or BASH for short. Linux comes with several shells from which to choose—you can see most of them listed in the **/etc/shells** file. Deciding which shell is right for you is kind of like choosing a favorite beer—what's right for you isn't right for everyone, but still, everyone tends to get defensive about their choice!

What makes Linux so interesting is that you do not have to stick with the list of shells provided in **/etc/shells**. In the strictest of definitions, the password entry for each user doesn't list what shell to run so much as it lists what program to run first for the user. Of course, most users prefer that the first program run be a shell, such as BASH.

The /etc/shadow File

This is the encrypted password file. It stores the encrypted password information for user accounts. In addition to the encrypted password, the **/etc/shadow** file stores optional password aging or expiration information. The introduction of the shadow file came about because of the need to separate encrypted passwords from the **/etc/passwd** file. This was necessary because the ease with which the encrypted passwords could be cracked was growing with the increase in the processing power of commodity computers (home PCs). The idea was to keep the **/etc/passwd** file readable by all users without storing the encrypted passwords in it and then make the **/etc/shadow** file only readable by root or other privileged programs that require access to that information. An example of such a program would be the login program.

One might wonder, "Why not just make the regular **/etc/passwd** file readable by root only or other privileged programs?" Well, it isn't that simple. By having the password file open for so many years, the rest of the system software that grew up around it relied on the fact that the password file was always readable by all users. Changing this would simply cause software to fail.

Just as in the **/etc/passwd** file, each line in the **/etc/shadow** file represents information about a user. The lines are made up of various standard fields, with each field delimited by a colon. The fields are

- Login name
- Encrypted password
- Days since January 1, 1970, that password was last changed
- Days before password may be changed
- Days after which password must be changed
- Days before password is to expire that user is warned
- Days after password expires that account is disabled
- Days since January 1, 1970, that account is disabled
- A reserved field

A sample entry from the **/etc/shadow** file is shown here for the user account mmel:

```
mmel:$1$HEWdPIJ.$qX/RbB.TPGcyerAVDlF4g.:12830:0:99999:7:::
```

The /etc/group File

The **/etc/group** file contains a list of groups, with one group per line. Each group entry in the file has four standard fields, with each field colon-delimited, as in the **/etc/passwd** and **/etc/shadow** files. Each user on the system belongs to at least one group, that being the user's default group. Users may then be assigned to additional groups if needed. You will recall that the **/etc/passwd** file contains each user's default group ID (GID). This GID is mapped to the group's name and other members of the group in the **/etc/group** file. The GID should be unique for each group.

Also, like the **/etc/passwd** file, the group file must be world-readable so that applications can test for associations between users and groups. The fields of each line in the **/etc/group** file are

- **Group name** The name of the group
- **Group password** This is optional, but if set, it allows users who are not part of the group to join
- **Group ID (GID)** The numerical equivalent of the group name
- **Group members** A comma-separated list

A sample group entry in the **/etc/group** file is shown here:

```
bin:x:1:root,bin,daemon
```

This entry is for the "bin" group. The GID for the group is 1, and its members are root, bin, and daemon.

USER MANAGEMENT TOOLS

The wonderful part about having password database files that have a well-defined format in straight text is that it is easy for anyone to write their own management tools. Indeed, many site administrators have already done this in order to integrate their tools along with the rest of their organization's infrastructure. They can start a new user from the same form that lets them update the corporate phone and e-mail directory, LDAP servers, web pages, and so on. Of course, not everyone wants to write their own tools, which is why Linux comes with several existing tools that do the job for you.

In this section, we discuss user management tools that can be used from the command-line interface, as well as graphical user interface (GUI) tools. Of course, learning how to use both is the preferred route, since they both have their advantages and place.

Command-Line User Management

You can choose from among six command-line tools to perform the same actions performed by the GUI tool: **useradd**, **userdel**, **usermod**, **groupadd**, **groupdel**, and **groupmod**. The compelling advantage of using command-line tools for user management, besides speed, is the fact that the tools can usually be incorporated into other automated functions (such as scripts).

NOTE Linux distributions other than Fedora and RHEL may have slightly different parameters from the tools used here. To see how your particular installation is different, read the man page for the particular program in question.

useradd

As the name implies, **useradd** allows you to add a single user to the system. Unlike the GUI tools, this tool has no interactive prompts. Instead, all parameters must be specified on the command line.

Here's how you use this tool:

```
usage: useradd [-u uid [-o]] [-g group] [-G group,...]
               [-d home] [-s shell] [-c comment] [-m [-k template]]
               [-f inactive] [-e expire ] [-p passwd] [-M] [-n] [-r] name
    useradd -D [-g group] [-b base] [-s shell]
               [-f inactive] [-e expire ]
```

Take note that anything in the square brackets in this usage summary is optional. Also, don't be intimidated by this long list of options! They are all quite easy to use and are described in Table 4-1.

Option	Description
-c *comment*	Allows you to set the user's name in the GECOS field. As with any command-line parameter, if the value includes a space, you will need to put quotes around the text. For example, to set the user's name to Ying Yang, you would have to specify **-c "Ying Yang"**.
-d *homedir*	By default, the user's home directory is /home/user_name. When creating a new user, the user's home directory gets created along with the user account. So if you want to change the default to another place, you can specify the new location with this parameter.
-e *expire-date*	It is possible for an account to expire after a certain date. By default, accounts never expire. To specify a date, be sure to place it in *YYYY MM DD* format. For example, use **-e 2009 10 28** for the account to expire on October 28, 2009.
-f *inactive-time*	This option specifies the number of days after a password expires that the account is still usable. A value of **0** (zero) indicates that the account is disabled immediately. A value of **-1** will never allow the account to be disabled, even if the password has expired (for example, **-f 3** will allow an account to exist for three days after a password has expired). The default value is **-1**.
-g *initial-group*	Using this option, you can specify the default group the user has in the password file. You can use a number or name of the group; however, if you use a name of a group, the group must exist in the **/etc/group** file.
-G *group* [,...]	This option allows you to specify additional groups to which the new user will belong. If you use the **-G** option, you must specify at least one additional group. You can, however, specify additional groups by separating the elements of the list with commas. For example, to add a user to the project and admin groups, you should specify **-G project,admin**.

Table 4-1. Options for the useradd Command

Option	Description
-m [-k *skel-dir*]	By default, the system automatically creates the user's home directory. This option is the explicit command to create the user's home directory. Part of creating the directory is copying default configuration files into it. These files come from the **/etc/skel** directory by default. You can change this by using the secondary option **-k *skel-dir***. (You must specify **-m** in order to use **-k**.) For example, to specify the **/etc/adminskel** directory, you would use **-m -k /etc/adminskel**.
-M	If you used the **-m** option, you cannot use **-M**, and vice versa. This option tells the command *not* to create the user's home directory.
-n	Red Hat Linux creates a new group with the same name as the new user's login as part of the process of adding a user. You can disable this behavior by using this option.
-s *shell*	A user's login shell is the first program that runs when a user logs into a system. This is usually a command-line environment, unless you are logging in from the X Window System login screen. By default, this is the Bourne Again Shell (**/bin/bash**), though some folks like other shells, such as the Turbo C Shell (**/bin/tcsh**).
-u *uid*	By default, the program will automatically find the next available UID and use it. If, for some reason, you need to force a new user's UID to be a particular value, you can use this option. Remember that UIDs must be unique for all users.
name	Finally, the only parameter that *isn't* optional! You must specify the new user's login name.

Table 4-1. Options for the useradd Command (*cont.*)

usermod

The **usermod** command allows you to modify an existing user in the system. It works in much the same way as **useradd**. Its usage is summarized here:

```
usage: usermod [-u uid [-o]] [-g group] [-G group,...]
               [-d home [-m]] [-s shell] [-c comment] [-l new_name]
               [-f inactive] [-e expire ] [-p passwd] [-L|-U] name
```

Every option you specify when using this command results in that particular parameter being modified for the user. All but one of the parameters listed here are identical to the parameters documented for the **useradd** command. The one exception is **-l**.

The **-l** option allows you to change the user's login name. This and the **-u** option are the only options that require special care. Before changing the user's login or UID, you must make sure the user is not logged into the system or running any processes. Changing this information if the user is logged in or running processes will cause unpredictable results.

userdel

The **userdel** command does the exact opposite of **useradd**—it removes existing users. This straightforward command has only one optional parameter and one required parameter:

```
usage: userdel [-r] username
```

groupadd

The group commands are similar to the user commands; however, instead of working on individual users, they work on groups listed in the **/etc/group** file. Note that changing group information does not cause user information to be automatically changed. For example, if you remove a group whose GID is 100 and a user's default group is specified as 100, the user's default group would not be updated to reflect the fact that the group no longer exists.

The **groupadd** command adds groups to the **/etc/group** file. The command-line options for this program are as follows:

```
usage: groupadd [-g gid [-o]] [-r] [-f] group
```

Table 4-2 describes command options.

groupdel

Even more straightforward than **userdel**, the **groupdel** command removes existing groups specified in the **/etc/group** file. The only usage information needed for this command is

```
usage: groupdel group
```

where ***group*** is the name of the group to remove.

Option	Description
-g *gid*	Specifies the GID for the new group as ***gid***. This value must be unique, unless the **-o** option is used. By default, this value is automatically chosen by finding the first available value greater than or equal to 500.
-r	By default, Fedora and RHEL search for the first GID that is higher than 499. The **-r** options tell **groupadd** that the group being added is a system group and should have the first available GID under 499.
-f	This is the force flag. This will cause **groupadd** to exit without an error when the group about to be added already exists on the system. If that is the case, the group won't be altered (or added again). It is a Fedora- and RHEL-specific option.
group	This option is required. It specifies the name of the group you want to add to be *group*.

Table 4-2. Options for the groupadd Command

groupmod

The **groupmod** command allows you to modify the parameters of an existing group. The options for this command are

```
usage: groupmod [-g gid [-o]] [-n name] group
```

where the **-g** option allows you to change the GID of the group, and the **-n** option allows you to specify a new name of a group. In addition, of course, you need to specify the name of the existing group as the last parameter.

GUI User Managers

The obvious advantage to using the GUI tool is ease of use. It is usually just a point-and-click affair. Many of the Linux distributions come with their own GUI user managers. Fedora comes with a utility called **system-config-users**, RHEL comes with a utility

A GRAND TOUR

The best way to see many of the utilities discussed in this chapter interact with one another is to show them at work. In this section, we take a step-by-step approach to creating, modifying, and removing users and groups. Some new commands that were not mentioned but that are also useful and relevant in managing users on a system are also introduced and used.

Creating Users with useradd

Add new user accounts and assign passwords with the **useradd** and **passwd** commands.

1. Create a new user whose full name is "Ying Yang," with the login name (account name) of **yyang**. Type

   ```
   [root@fedora-serverA ~]# useradd -c "Ying Yang" yyang
   ```

   This command will create a new user account called yyang. The user will be created with the usual Fedora default attributes. The entry in the **/etc/passwd** file will be

   ```
   yyang:x:500:500:Ying Yang:/home/yyang:/bin/bash
   ```

   From this entry, you can tell these things about the Fedora (and RHEL) default new user values:

   - ▼ The UID number is the same as the GID number.
   - ■ The default shell for new users is the bash shell (**/bin/bash**).
   - ▲ A home directory is automatically created for all new users (e.g., **/home/yyang**).

2. Use the **passwd** command to create a new password for the username yyang. Set the password to be **19ang19**, and repeat the same password when prompted. Type

   ```
   [root@fedora-serverA ~]# passwd yyang
   Changing password for user yyang.
   New UNIX password:
   Retype new UNIX password:
   passwd: all authentication tokens updated successfully.
   ```

3. Create another user account called **mmellow** for the user, with a full name of "Mel Mellow," but this time, change the default Fedora behavior of creating a

group with the same name as the username (i.e., this user will instead belong to the general **users** group). Type

```
[root@fedora-serverA ~]# useradd -c "Mel Mellow" -n mmellow
```

4. Use the **id** command to examine the properties of the user mmellow. Type

```
[root@fedora-serverA ~]# id mmellow
```

5. Again, use the **passwd** command to create a new password for the account mmellow. Set the password to be **2owl78**, and repeat the same password when prompted. Type

```
[root@fedora-serverA ~]# passwd mmellow
```

6. Create the final user account, called **bogususer**. But this time, specify the user's shell to be the tcsh shell, and let the user's default primary group be the system "games" group. Type

```
[root@fedora-serverA ~]# useradd -s /bin/tcsh -g games bogususer
```

7. Examine the **/etc/passwd** file for the entry for the bogususer user. Type

```
[root@fedora-serverA ~]# grep bogususer /etc/passwd
bogususer:x:502:20::/home/bogususer:/bin/tcsh
```

From this entry, you can tell that:

- The UID is 502.
- The GID is 20.
- A home directory is also created for the user under the **/home** directory.
- The user's shell is **/bin/tcsh**.

Creating Groups with groupadd

Next, create a couple of groups: nonsystem and system.

1. Create a new group called **research**. Type

```
[root@fedora-serverA ~]# groupadd research
```

2. Examine the entry for the research group in the **/etc/group** file. Type

```
[root@fedora-serverA ~]# grep research /etc/group
research:x:501:
```

This output shows that the group ID for the research group is 501.

3. Create another group called **sales**. Type

```
[root@fedora-serverA ~]# groupadd sales
```

4. Create the final group called **bogus**, and in addition, force this group to be a system group (i.e., the GID will be lower than 499). Type

```
[root@fedora-serverA ~]# groupadd -r bogus
```

5. Examine the entry for the bogus group in the **/etc/group** file. Type

```
[root@fedora-serverA ~]# grep bogus /etc/group
bogus:x:497:
```

The output shows that the group ID for the bogus group is 497.

Modifying User Attributes with usermod

Now try using **usermod** to change the user and group IDs for a couple of accounts.

1. Use the **usermod** command to change the user ID (UID) of the bogususer to 600. Type

```
[root@fedora-serverA ~]# usermod -u 600 bogususer
```

2. Use the **id** command to view your changes. Type

```
[root@fedora-serverA ~]# id bogususer
```

The output shows the new UID (600) for the user.

3. Use the **usermod** command to change the primary group ID (GID) of the bogus-user account to that of the bogus group (GID = 101) and to also set an expiry date of 12-12-2010 for the account. Type

```
[root@fedora-serverA ~]# usermod -g 497  -e  2010-12-12  bogususer
```

4. View your changes with the **id** command. Type

```
[root@fedora-serverA ~]# id bogususer
```

5. Use the **chage** command to view the new account expiration information for the user. Type

```
[root@fedora-serverA ~]# chage -l bogususer
Last password change                                    : Sep 23, 2009
Password expires                                        : never
Password inactive                                       : never
```

```
Account expires                                          : Dec 12, 2010
Minimum number of days between password change           : 0
Maximum number of days between password change           : 99999
Number of days of warning before password expires        : 7
```

Modifying Group Attributes with groupmod

Now try using the **groupmod** command.

1. Use the **groupmod** command to rename the bogus group to **bogusgroup**. Type

   ```
   [root@fedora-serverA ~]# groupmod -n bogusgroup bogus
   ```

2. Again use the **groupmod** command to change the group ID (GID) of the **bogusgroup** to 600. Type

   ```
   [root@fedora-serverA ~]# groupmod -g 600 bogusgroup
   ```

3. View your changes to the bogusgroup in the **/etc/group** file. Type

   ```
   [root@fedora-serverA ~]# grep bogusgroup /etc/group
   ```

Deleting Groups and Users with groupdel and userdel

Try using the **groupdel** and **userdel** commands to delete groups and users, respectively.

1. Use the **groupdel** command to delete the bogusgroup group. Type

   ```
   [root@fedora-serverA ~]# groupdel bogusgroup
   ```

   You will notice that the bogusgroup entry in the **/etc/group** file will be removed accordingly.

2. Use the **userdel** command to delete the user bogususer that you created previously. At the shell prompt, type

   ```
   [root@fedora-serverA ~]# userdel -r bogususer
   ```

NOTE When you run the **userdel** command with only the user's login specified on the command line (for example, **userdel bogususer**), all of the entries in the **/etc/passwd** and **/etc/shadow** files, as well as references in the **/etc/group** file, are automatically removed. But if you use the optional **-r** parameter (for example, **userdel -r bogususer**), all of the files owned by the user in that user's home directory are removed as well.

SUMMARY

This chapter documented the nature of users under Linux. Much of what you read here also applies to other variants of UNIX, which makes administering users in heterogeneous environments much easier with the different *NIXs.

The main points covered in this chapter were:

- Each user gets a unique UID.
- Each group gets a unique GID.
- The **/etc/passwd** file maps UIDs to usernames.
- Linux handles encrypted passwords in multiple ways.
- Linux includes tools that help you administer users.
- Should you decide to write your own tools to manage the user databases, you'll now understand the format for doing so.
- PAM, the Pluggable Authentication Modules, is Linux's generic way of handling multiple authentication mechanisms.

These changes are pretty significant for an administrator coming from the Windows XP/Vista/NT/200*x* environment and can be a little tricky at first. Not to worry, though—the Linux/UNIX security model is quite straightforward, so you should quickly get comfortable with how it all works.

If the idea of getting to build your own tools to administer users appeals to you, definitely look into books on the Perl scripting language. It is remarkably well suited for manipulating tabular data (such as the **/etc/passwd** file). Take some time and page through a few Perl programming books at your local bookstore if this is something that interests you.

Disabling swap area

If at any point you want to disable a swap area, you can do so using the `swapoff` command. You might do this, in particular, if the swap area is no longer needed and you want to reclaim the space being consumed by a swap file or remove a USB drive that is providing a swap partition.

First, make sure that no space is being used on the swap device (using the `free` command), then use `swapoff` to turn off the swap area so you can reuse the space. Here is an example:

```
# free -m
             total  used    free  shared  buffers   cached
Mem:          3629  2433    1195       0       99      580
-/+ buffers/cache:  1754    1874
Swap:         4095     0    4095
# swapoff /dev/sdc2
```


Using the fstab file to define mountable file systems

The hard disk partitions on your local computer and the remote filesystems you use every day are probably set up to automatically mount when you boot Linux. The `/etc/fstab` file contains definitions for each partition, along with options describing how the partition is mounted. Here's an example of an `/etc/fstab` file:

```
# /etc/fstab
/dev/mapper/vg_abc-lv_root             /        ext4  defaults  1 1
UUID=78bdae46-9389-438d-bfee-06dd934fae28 /boot ext4  defaults  1 2
/dev/mapper/vg_abc-lv_home       /home   ext4  defaults      1 2
/dev/mapper/vg_abc-lv_swap       swap    swap  defaults      0 0
# Mount entries added later.
/dev/sdb1                        /win    vfat ro             1 2
192.168.0.27:/nfsstuff          /remote  nfs  users,_netdev  0 0
//192.168.0.28/myshare          /share   cifs guest,_netdev  0 0
# special Linux filesystems
tmpfs                            /dev/shm tmpfs defaults      0 0
devpts                         /dev/pts devpts gid=5,mode=620 0 0
sysfs                          /sys     sysfs  defaults       0 0
proc                           /proc    proc   defaults       0 0
```

The `/etc/fstab` file just shown is from a default Red Hat Enterprise Linux 6 server install, with a few lines added.

For now, you can ignore the `tmpfs`, `devpts`, `sysfs`, and `proc` entries. Those are special devices associated with shared memory, terminal windows, device information, and kernel parameters, respectively.

In general, the first column of `/etc/fstab` shows the device or share (what is mounted), while the second column shows the mount point (where it is mounted). That is followed

by the type of filesystem, any mount options (or defaults), and two numbers (used to tell commands such as `dump` and `fsck` what to do with the filesystem).

The first three entries represent the disk partitions assigned to the root of the filesystem (`/`), the `/boot` directory, and the `/home` directory. All three are `ext4` filesystems. The fourth line is a swap device (used to store data when RAM overflows). Notice that the device names for `/`, `/home`, and `swap` all start with `/dev/mapper`. That's because they are LVM logical volumes that are assigned space from a pool of space called an LVM group (more on LVM in the "Using Logical Volume Management Partitions" section later in this chapter).

The `/boot` partition is on its own physical partition, `/dev/sda1`. Instead of using `/dev/sda1`, however, a unique identifier (UUID) identifies the device. Why use a UUID instead of `/dev/sda1` to identify the device? Say that that you plugged another disk into your computer and booted up. It probably won't happen, but it is possible that the new disk might be identified as `/dev/sda`, causing the system to look for the contents of `/boot` on the first partition of that disk.

To see all the UUIDs assigned to storage devices on your system, type the `blkid` command, as follows:

```
# blkid
/dev/sda1:
  UUID="78bdae46-9389-438d-bfee-06dd934fae28" TYPE="ext4"
/dev/sda2:
  UUID="wlvuIv-UiI2-pNND-f39j-oH0X-9too-AOII7R" TYPE="LVM2_member"
/dev/mapper/vg_abc-lv_root:
  UUID="3e6f49a6-8fec-45e1-90a9-38431284b689" TYPE="ext4"
/dev/mapper/vg_abc-lv_swap:
  UUID="77662950-2cc2-4bd9-a860-34669535619d" TYPE="swap"
/dev/mapper/vg_abc-lv_home:
  UUID="7ffbcff3-36b9-4cbb-871d-091efb179790" TYPE="ext4"
/dev/sdb1:
  SEC_TYPE="msdos" UUID="75E0-96AA" TYPE="vfat"
```

Any of the device names can be replaced by the UUID designation in the left column of an `/etc/fstab` entry.

I added the next three entries in `/etc/fstab` to illustrate some different kinds of entries. I connected a hard drive from an old Microsoft Windows system, and I had it mounted on the `/win` directory. I added the `ro` option so it would mount read-only.

The next two entries represent remote filesystems. On the `/remote` directory, the `/nfsstuff` directory is mounted read/write (rw) from the host at address 192.168.0.27 as an NFS share. On the `/share` directory, the Windows share named `myshare` is mounted from the host at 192.168.0.28. In both cases, I added the `_netdev` option, which tells Linux to wait for the network to come up before trying to mount the shares. (For more information on mounting CIFS and NFS shares, refer to Chapters 19, “Configure a Windows File Sharing (Samba) Server,” and 20, “Configuring an NFS File Server,” respectively.)

Coming from Windows

The "Using the fstab file to define mountable file systems" section shows mounting a hard disk partition from an old VFAT filesystem being used in Windows. Most Windows systems today use the NTFS filesystem. Support for this system, however, is not delivered with every Linux system. NTFS is available from Fedora in the ntfs-3g package. Other NTFS support is available from the Linux-NTFS project (`http://www.linux-ntfs.org/`).

To help you understand the contents of the `/etc/fstab` file, here is what is in each field of that file:

- **Field 1**—The name of the device representing the filesystem. This field can include the `LABEL` or UUID option, with which you can indicate a volume label or universally unique identifier (UUID) instead of a device name. The advantage to this approach is that because the partition is identified by volume name, you can move a volume to a different device name and not have to change the `fstab` file. (See the description of the `mkfs` command later in the "Using the mkfs Command to Create a Filesystem" section of this chapter for information on creating and using labels.)
- **Field 2**—The mount point in the filesystem. The filesystem contains all data from the mount point down the directory tree structure unless another filesystem is mounted at some point beneath it.
- **Field 3**—The filesystem type. Valid filesystem types are described in the "Supported filesystems" section earlier in this chapter (although you can only use filesystem types for which drivers are included for your kernel).
- **Field 4**—Use `defaults` or a comma-separated list of options (no spaces) you want to use when the entry is mounted. See the `mount` command manual page (under the `-o` option) for information on other supported options.

TIP

Typically, only the root user is allowed to mount a filesystem using the `mount` command. However, to allow any user to mount a filesystem (such as a filesystem on a CD), you could add the `user` option to Field 4 of `/etc/fstab`.

- **Field 5**—The number in this field indicates whether the filesystem needs to be dumped (that is, have its data backed up). A `1` means that the filesystem needs to be dumped, and a `0` means that it doesn't. (This field is no longer particularly useful because most Linux administrators use more sophisticated backup options than the `dump` command. Most often, a `0` is used.)
- **Field 6**—The number in this field indicates whether the indicated filesystem should be checked with `fsck` when the time comes for it to be checked: `1` means it needs to be checked first, `2` means to check after all those indicated by `1` have already been checked, and `0` means don't check it.

If you want to find out more about mount options as well as other features of the /etc/fstab file, there are several man pages you can refer to, including man 5 nfs and man 8 mount.

Using the mount command to mount file systems

Linux systems automatically run mount -a (mount all filesystems) each time you boot. For that reason, you generally use the mount command only for special situations. In particular, the average user or administrator uses mount in two ways:

- To display the disks, partitions, and remote filesystems currently mounted
- To temporarily mount a filesystem

Any user can type mount (with no options) to see what filesystems are currently mounted on the local Linux system. The following is an example of the mount command. It shows a single hard disk partition (/dev/sda1) containing the root (/) filesystem, and proc and devpts filesystem types mounted on /proc and /dev, respectively.

```
$ mount
/dev/sda3 on / type ext4 (rw)
/dev/sda2 on /boot type ext4 (rw)
/dev/sda1 on /mnt/win type vfat (rw)
/dev/proc on /proc type proc (rw)
/dev/sys on /sys type sysfs (rw)
/dev/devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/shm on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
/dev/cdrom on /media/MyOwnDVD type iso9660 (ro,nosuid,nodev)
```

Traditionally, the most common devices to mount by hand are removable media, such as DVDs or CDs. However, depending on the type of desktop you are using, CDs and DVDs may be mounted for you automatically when you insert them. (In some cases, applications are launched as well when media is inserted. For example, a CD music player or photo editor may be launched when your inserted medium has music or digital images on it.)

Occasionally, however, you may find it useful to mount a filesystem manually. For example, you want to look at the contents of an old hard disk, so you install it as a second disk on your computer. If the partitions on the disk did not automount, you could mount partitions from that disk manually. For example, to mount read-only a disk partition sdb1 that has an older ext3 filesystem, you could type this:

```
# mkdir /mnt/temp
# mount -t ext3 -o ro /dev/sdb1 /mnt/tmp
```

Another reason to use the mount command is to remount a partition to change its mount options. Say that you want to remount /dev/sdb1 as read/write, but you do not want to unmount it (maybe someone is using it). You could use the remount option as follows:

```
# mount -t ext3 -o remount,rw /dev/sdb1
```

Mounting a disk image in loopback

Another valuable way to use the `mount` command has to do with disk images. If you download a CD or floppy disk image from the Internet and you want to see what it contains, you can do so without burning it to CD or floppy. With the image on your hard disk, create a mount point and use the `-o loop` option to mount it locally. Here's an example:

```
# mkdir /mnt/mycdimage
# mount -o loop whatever-i686-disc1.iso /mnt/mycdimage
```

In this example, the `/mnt/mycdimage` directory is created, and then the disk image file (`whatever-i686-disc1.iso`) residing in the current directory is mounted on it. You can now `cd` to that directory, view the contents of it, and copy or use any of its contents. This is useful for downloaded CD images from which you want to install software without having to burn the image to CD. You could also share that mount point over NFS, so you could install the software from another computer. When you are done, just type **`umount /mnt/mycdimage`** to unmount it.

Other options to `mount` are available only for specific filesystem types. See the `mount` manual page for those and other useful options.

Using the umount command

When you are done using a temporary filesystem, or you want to unmount a permanent filesystem temporarily, use the `umount` command. This command detaches the filesystem from its mount point in your Linux filesystem. To use `umount`, you can give it either a directory name or a device name. For example:

```
# umount /mnt/test
```

This unmounts the device from the mount point `/mnt/test`. You can also unmount using the form

```
# umount /dev/sdb1
```

In general, it's better to use the directory name (`/mnt/test`) because the `umount` command will fail if the device is mounted in more than one location. (Device names all begin with `/dev`.)

If you get the message `device is busy`, the `umount` request has failed because either an application has a file open on the device or you have a shell open with a directory on the device as a current directory. Stop the processes or change to a directory outside the device you are trying to unmount for the `umount` request to succeed.

An alternative for unmounting a busy device is the `-l` option. With `umount -l` (a lazy unmount), the unmount happens as soon as the device is no longer busy. To unmount a remote NFS filesystem that's no longer available (for example, the server went down), you can use the `umount -f` option to forcibly unmount the NFS filesystem.

TIP

A really useful tool for discovering what's holding open a device you want to unmount is the `lsof` command. Type **`lsof`** with the name of the partition you want to unmount (such as `lsof /mnt/test`). The output shows you what commands are holding files open on that partition. The `fuser-v /mnt test` command can be used in the same way.

Using the mkfs Command to Create a Filesystem

You can create a filesystem for any supported filesystem type on a disk or partition that you choose. You do so with the `mkfs` command. Although this is most useful for creating filesystems on hard-disk partitions, you can create filesystems on USB flash drives, floppy disks, or rewritable CDs as well.

Before you create a new filesystem, make sure of the following:

- You have partitioned the disk as you want (using the `fdisk` command).
- You get the device name correct, or you may end up overwriting your hard disk by mistake. For example, the first partition on the second SCSI or USB flash drive on your system is `/dev/sdb1` and the third disk is `/dev/sdc1`.
- To unmount the partition if it's mounted before creating the filesystem.

The following is an example of using `mkfs` to create a filesystem on the first (and only) partition on a 2GB USB flash drive located as the third SCSI disk (`/dev/sdc1`):

```
# mkfs -t ext3 /dev/sdc1
mke2fs 1.40.8 (13-Mar-2008)
Warning: 256-byte inodes not usable on older systems
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
122160 inodes, 487699 blocks
24384 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=503316480
15 block groups
32768 blocks per group, 32768 fragments per group
8144 inodes per group
Superblock backups stored on blocks:
       32768, 98304, 163840, 229376, 294912
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 39 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

You can see the statistics that are output with the formatting done by the `mkfs` command. The number of inodes and blocks created are output, as are the number of blocks per group and fragments per group. An inode, which hold metadata such as ownership and timestamps for each file, will be consumed for every file and directory in the filesystem. So the number of inodes shown here limits the total number of files you can create in that filesystem.

You can now mount this filesystem (`mkdir /mnt/myusb ; mount /dev/sdc1 /mnt /myusb`), change to `/mnt/usb` as your current directory (`cd /mnt/myusb`), and create files on it as you please.

Summary

Managing filesystems is a critical part of administering a Linux system. Using commands such as `fdisk`, you can view and change disk partitions. Filesystems can be added to partitions using the `mkfs` command. Once created, filesystems can be mounted and unmounted using the mount and umount commands, respectively.

Logical Volume Management (LVM) offers a more powerful and flexible way of managing disk partitions. With LVM, you create pools of storage, called volumes, that can allow you to grow and shrink logical volumes, as well as extend the size of your volume groups by adding more physical volumes.

With most of the basics needed to become a system administrator covered at this point in the book, Chapter 13 introduces concepts for extending those skills to manage network servers. Topics in that chapter include information on how to install, manage, and secure servers.

Exercises

Use these exercises to test your knowledge of creating disk partitions, logical volume management, and working with filesystems. You will need a USB flash drive that is at least 1GB that you can erase for these exercises.

These tasks assume you are running a Fedora or Red Hat Enterprise Linux system (although some tasks will work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Run a command as root to watch the `/var/log/messages` file and insert your USB flash drive. Determine the device name of the USB flash drive.
2. Run a command to list the partition table for the USB flash drive.

3. Delete all the partitions on your USB flash drive, save the changes, and make sure that the changes were made both on the disk's partition table and in the Linux kernel.
4. Add three partitions to the USB flash drive: 100MB Linux partition, 200MB swap partition, and 500MB LVM partition. Save the changes.
5. Put an `ext3` filesystem on the Linux partition.
6. Create a mount point called `/mnt/mypart` and mount the Linux partition on it.
7. Enable the swap partition and turn it on so that there is additional swap space immediately available.
8. Create a volume group called `abc` from the LVM partition, create a 200MB logical volume from that group called `data`, add a VFAT partition, and then temporarily mount the logical volume on a new directory named `/mnt/test`. Check that it was successfully mounted.
9. Grow the logical volume from 200MB to 300MB.
10. Do what you need to do to safely remove the USB flash drive from the computer: unmount the Linux partition, turn off the swap partition, unmount the logical volume, and delete the volume group from the USB flash drive.