CHAPTER 13

Understanding Users and Groups

Linux is a multi-user OS, meaning that it provides features to help multiple individuals use the computer. Collectively, these features constitute *accounts*. Previous chapters of this book have referred to accounts in passing but haven't covered them in detail. This chapter changes that; it describes important account principles and a few commands you can use to begin investigating accounts. Related to accounts are *groups*, which are collections of accounts that can be given special permissions on the computer, so this chapter also describes groups. One account, known as `root`, has special privileges on the computer. You use this account to perform most system administration tasks, so you should understand this account before you tackle the administrative tasks described in the last few chapters of this book.

- **Understanding accounts**
- **Using account tools**
- **Working as `root`**

Understanding Accounts

Even a single-user workstation uses multiple accounts. Such a computer may have just one *user account*, but several *system accounts* help keep the computer running.

Accounts enable multiple users to share a single computer without causing each other too much trouble. They also enable system administrators to track who is using system resources and, sometimes, who is doing things they shouldn't be doing. Thus, account features help users use a computer and administrators administer it. Understanding these features is the basis for enabling you to manage accounts.

Some account features help you identify accounts and the files and resources associated with them. Knowing how to use these features will help you track down account-related problems and manage users of a computer.

Understanding Account Features

Certification Objective

Most account features are defined in the /etc/passwd file, which consists of colon-delimited lines, with each line defining a single account. An entry might resemble the following:

```
luke:x:1003:100:Luke Jones:/home/luke:/bin/bash
```

The information contained in the fields of this line includes the following:

Username An account's username is its most salient feature to humans. Most Linux account usernames consist of lowercase letters, and occasionally numbers, as in luke or thx1138. Underscores (_) and dashes (-) are also valid characters in Linux usernames, as are dollar signs ($) at the end.

Password User accounts are normally protected by a password, which is required to log into the computer. Direct login to most system accounts is disabled, so they lack passwords. (The root account is an important exception; it has a password in most distributions.) The password field in the /etc/passwd file usually contains an x, which is a code meaning that the password is stored in /etc/shadow, as described shortly.

Some distributions number user accounts starting at 500 rather than 1,000.

UID In reality, the username is just a label that the computer displays to us numerically challenged humans. The computer uses a user identification (UID) number to track accounts. UID numbers begin with 0 (which refers to the root account) and can range as high as 4,294,967,295 on modern Linux systems. In most distributions, user accounts are numbered 1,000 and above, with lower numbers reserved for system accounts.

File ownership and permissions are described in Chapter 15, "Setting Ownership and Permissions."

GID Accounts are tied to one or more *groups*, which are similar to accounts in many ways; however, a group is a collection of accounts. One of the primary purposes of groups is to enable users to give certain users access to their files, while preventing others from accessing them. Each account is tied directly to a primary group via a group ID (GID) number (100 in the preceding example). Accounts can be tied to other groups by inclusion in the group's definition, as described in Chapter 14, "Creating Users and Groups."

Comment field The *comment field* normally holds the user's full name (Luke Jones in this example), although this field can hold other information instead of or in addition to the user's name.

Home directory User accounts, and some system accounts, have *home directories*. A home directory is an account's "home base." Normally, ownership of an account's home directory belongs to the account. Certain tools and procedures make it easy for users to access their home directories; for instance, the tilde (~) refers to a user's home directory when used at the start of a filename.

Default shell A default text-mode shell is associated with every account. In Linux, this shell is normally Bash (`/bin/bash`), but individual users can change this if they like. Most non-`root` system accounts set the default shell to `/sbin/nologin` as an added security measure—this program displays a message stating that the account is not available. Using `/bin/false` works in a similar way, although without the explanatory message.

Although you might guess by its name that `/etc/passwd` holds password information, this isn't normally the case today, although it was many years ago. For historical reasons, `/etc/passwd` must be readable by all users, so storing passwords there, even in encrypted form, is risky. Thus, passwords today are stored in another file, `/etc/shadow`, that ordinary users can't read. This file associates an encrypted password, as well as other information, with an account. This information can disable an account after a period of time or if the user doesn't change the password within a given period of time. A typical `/etc/shadow` entry looks like this:

Passwords are stored using a *hash*, which is a one-way type of encryption. When a user types a password, it's hashed, and if the hashes match, access is granted.

```
luke:$1$E/moFkeT5UnTQ:15369:0:-1:7:-1:-1:
```

The meaning of each colon-delimited field on this line is as follows:

Username Each line begins with the username. Note that the UID is *not* used in `/etc/shadow`; the username links entries in this file to those in `/etc/passwd`.

Password The password is stored in encrypted form, so it bears no obvious resemblance to the actual password. An asterisk (`*`) or exclamation mark (`!`) denotes an account with no password (that is, the account doesn't accept logins—it's locked). This is common for accounts used by the system itself.

If you've forgotten the `root` password, you can boot with an emergency disc and copy the contents of a password field for an account whose password you do remember.

Last password change The next field (`15369` in this example) is the date of the last password change. This date is stored as the number of days since January 1, 1970.

Days until a change is allowed The next field (`0` in this example) is the number of days before a password change is allowed.

Days before a change is required This field is the number of days after the last password change before another password change is required.

Days of warning before password expiration If your system is configured to expire passwords, you may set it to warn the user when an expiration date is approaching. A value of `7`, as in the preceding example, is typical.

Days between expiration and deactivation Linux allows for a gap between the expiration of an account and its complete deactivation. An expired account either can't be used or requires that the user change the password immediately after logging in. In either case, its password remains intact. A deactivated account's password is erased, and the account can't be used until it's reactivated by the system administrator.

Expiration date This field shows the date on which the account will expire. As with the last password change date, the date is expressed as the number of days since January 1, 1970.

Special flag This field is reserved for future use and normally isn't used or contains a meaningless value. This field is empty in the preceding example.

For fields relating to day counts, a value of `-1` or `99999` typically indicates that the relevant feature has been disabled. The `/etc/shadow` values are generally best left to modification through commands such as `usermod` (described in Chapter 14, "Creating Users and Groups") and `chage`. Understanding the format of the file enables you to review its contents or to replace the `root` password by cutting-and-pasting using an emergency disc, should you forget it.

The `/etc/shadow` file is normally stored with very restrictive permissions, such as `rw-------` (600), with ownership by `root`. (Precise permissions vary from one distribution to another, though.) This fact is critical to the shadow password system's utility because it keeps non-`root` users from reading the file and obtaining the password list, even in an encrypted form. By contrast, `/etc/passwd` must be readable by ordinary users and usually has `rw-r--r--` (644) permissions. If you manually modify `/etc/shadow`, be sure it has the correct permissions when you're done.

Examples of user files stored outside of the user's home directory include e-mail in `/var/spool/mail` and temporary files in `/tmp`.

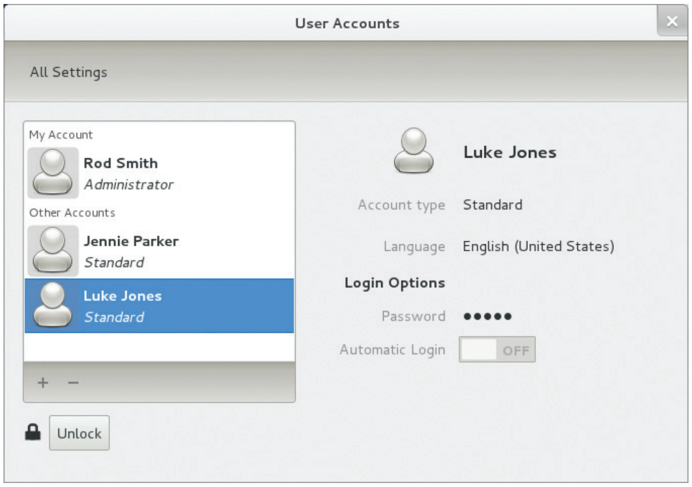
It's important to realize that an account isn't a single entity like a program binary file. Account information is scattered across several configuration files, such as `/etc/passwd`, `/etc/shadow`, `/etc/group`, and possibly in other configuration files that refer to accounts. User files reside in the user's home directory and perhaps elsewhere. Thus, managing accounts can require doing more than just editing a file or two—although in the case of system accounts, that may be enough. For this reason, various utilities exist to help create, manage, and delete accounts, as described in the rest of this chapter and in Chapter 14.

Identifying Accounts

One way to identify user accounts is to use a GUI tool for account management. Such tools vary from one distribution to another, though. One example is the User Accounts tool in the System Settings control panel on a Fedora system. Accessing this option produces a window similar to the one shown in Figure 13.1.

This tool shows only user accounts, not system accounts. It enables changing a few features, such as a user's password, by clicking on them, but its usefulness as an account management tool is limited. Ordinary users, however, can reach it fairly easily and can use it to change their passwords. As an administrator, you can use this tool to quickly verify what accounts are active.

Passwords are normally displayed as dots or asterisks as a security feature.


User Accounts

All Settings

My Account
Rod Smith
Administrator

Other Accounts
Jennie Parker
Standard

Luke Jones
Standard

Luke Jones

Account type Standard

Language English (United States)

Login Options

Password ●●●●●

Automatic Login OFF

FIGURE 13.1 The User Accounts tool in System Settings provides minimal account information.

You can identify all of a computer's accounts by viewing the contents of the `/etc/passwd` file with `cat` or `less` or by loading the file in a text editor. Doing so will reveal all the accounts, including both system and user accounts.

Alternatively, if you're searching for information on a specific account, you can use `grep` to find it in `/etc/passwd`, as in **`grep Jones /etc/passwd`** to find information on any account that's tied to a user with the name Jones. (This specific example assumes that the string `Jones` appears in the `passwd` file, of course.)

An alternative that's similar to perusing `/etc/passwd` is to type **`getent passwd`**. This command retrieves entries from certain administrative databases, including the account database. In most cases, typing **`getent passwd`** produces results that are identical to typing **`cat /etc/passwd`**; however, sometimes the two aren't identical. The `/etc/passwd` file defines only local user accounts. It's possible to configure Linux to use a network account database to define some or all of its accounts. If you use such a configuration, typing **`getent passwd`** returns both local accounts and accounts defined on the network server.

Network Account Databases

Many networks employ network account databases. Such systems include the Network Information System (NIS), an update to this system called NIS+, the Lightweight Directory Access Protocol (LDAP), Kerberos realms, Windows NT 4.0 domains, and Active Directory (AD) domains. All of these systems move account database management onto a single centralized computer (often with one or more backup systems). The advantage of this approach to account maintenance is that users and administrators need not deal with maintaining accounts independently on multiple computers. A single account database can handle accounts on dozens (or even hundreds or thousands) of different computers, greatly simplifying day-to-day administrative tasks and simplifying users' lives. Using such a system, though, means that most user accounts won't appear in `/etc/passwd` and `/etc/shadow`, and groups may not appear in `/etc/group` (described shortly, in "Understanding Groups"). These files will still hold information on local system accounts and groups, though.

Linux can participate in these systems. In fact, some distributions provide options to enable such support at OS installation time. Typically, you must know the name or IP address of the server that hosts the network account database, and you must know what protocol that system uses. You may also need a password or some other protocol-specific information, and the server may need to be configured to accept accesses from the Linux system you're configuring.

Activating use of such network account databases after installing Linux is a complex topic that is not covered in this book. Such systems often alter the behavior of tools such as `passwd` and `usermod` (described in Chapter 14) in subtle or not-so-subtle ways. If you need to use such a system, you'll have to consult documentation specific to the service you intend to use. My book *Linux in a Windows World* (O'Reilly, 2005) covers this topic for Windows NT 4.0 domains, LDAP, and Kerberos; and Mark Minasi and Dan York's *Linux for Windows Administrators* (Sybex, 2002) covers this topic for Windows NT 4.0 domains and NIS.

Understanding Groups

Certification Objective

As noted earlier, groups are collections of accounts that are defined in the `/etc/group` file. Like `/etc/passwd`, `/etc/group` consists of a series of colon-delimited lines, each of which defines a single group. An example looks like this:

```
users:x:100:games,sally
```

The fields in /etc/group are:

Group name The first field, `users` in the preceding example, is the name of the group. You use it with most commands that access or manipulate group data.

Password Groups, like users, can have passwords. A value of `x` means that the password is defined elsewhere, and an empty password field means the group has no password.

The use of group passwords is an advanced topic that's beyond the scope of this book.

GID Linux uses GID values, like UID values, internally. Translation to and from group names is done for the benefit of users and administrators.

User list You can specify users who belong to the group in a comma-delimited list at the end of the /etc/group line.

It's important to recognize that users can be identified as members of a group in either of two ways:

- By specifying the group's GID in users' individual /etc/passwd entries. Because /etc/passwd only has room for one GID value, only one group can be defined in this way. This is the user's primary (or default) group.
- By specifying usernames in the user list in the /etc/group file. A single user can appear multiple times in /etc/group, and a single group can have multiple users associated with it in this way. If a user is associated with a group in this way but not via the user's /etc/passwd entry, this group association is secondary.

When you create new files, those files will be associated with your current group. When you log in, your current group is set to your primary group. If you want to create files that are associated with another group to which you belong, you can use the `newgrp` command, as in:

```
$ newgrp project1
```

This command makes `project1` your current group, so that files you create will be associated with that group. Group ownership of files is important in file security, which is described in more detail in Chapter 15, "Setting Ownership and Permissions."

Using Account Tools

A few commands can help you learn about the users and groups on your computer. Most notably, the `whoami` and `id` utilities can tell you about your own identity, and the `who` and `w` utilities can give you information about who is currently using the computer.

Discovering Your Own Identity

If you've been using su to change your effective user ID, or if you maintain multiple accounts for yourself and you don't recall which one you used to log in, you might become confused about your current status. In such a case, the whoami command can come in handy: It displays your effective user ID:

```
$ whoami
luke
```

Certification Objective

This example reveals that the current account is luke. If you need more information, you can use the id utility:

```
$ id
uid=1003(luke) gid=100(users) groups=100(users),16(cron),18(audio)
```

This example shows information on both users and groups:

- Your user ID and username—uid=1003(luke) in this example
- Your current group—gid=100(users) in this example
- All your group memberships—the entries following groups= in this example

The id command displays both the numeric UID or GID values and the associated names. The current group is the one that's active, either by default or because you used the newgrp command.

You can limit id's output by specifying a number of options, as summarized in Table 13.1. In addition, you can specify a username, as in **id sally**, to obtain information on that user rather than on yourself.

TABLE 13.1 Options for id

Long option	Short option	Effect
--group	-g	Displays only the effective group ID
--groups	-G	Displays all the groups to which you belong
--user	-u	Displays only the user data
--name	-n	Used in conjunction with -g, -G, or -u, displays only the name not the UID or GID
--real	-r	Used in conjunction with -g, -G, or -u, displays only the UID or GID, not the name

Learning Who's Online

Linux permits multiple users to access the computer simultaneously. Most often, this is done by means of remote access servers such as the Secure Shell (SSH); however, you can use Linux's virtual terminal (VT) feature to log in multiple times using a single keyboard and monitor. Sometimes you might want to know who is using the computer. You might do this before shutting down the computer, for instance, to ensure you don't inconvenience another user.

Certification Objective

To learn who is online, you can use a command known as who:

```
$ who
luke     :0           2012-03-06 13:27
luke     pts/0        2012-03-06 15:16 (:0.0)
sally    pts/7        2012-03-07 13:00 (callisto)
rod      pts/9        2012-03-07 13:22 (remote.example.org)
```

This example shows four logins—two by luke, one by sally, and one by rod. Information provided in the default output includes:

Username The first column of who's output shows the username.

Terminal identifier The second column of who's output shows a code associated with the terminal. In this example, luke's first login shows :0 as this identifier, which means it's a local GUI (X) login. The remaining logins all have terminal identifiers of the form pts/#, indicating text sessions. A text session can be a terminal launched in X, a text-mode console login, or a remote login via SSH or some other protocol.

Text-mode sessions can also be indicated by codes of the form tty#.

Login date and time who displays the date and time of each login. Thus, you can see that luke's X session began almost one full day before rod logged in.

Remote host The final column of who's output, if present, shows the login source. Console logins (including both text-mode and X-based logins) don't include a source. A source of the form :# or :#.#, as in luke's second session, indicate a terminal opened in X. A hostname or IP address, as in sally's and rod's sessions, indicates remote access from the specified computer.

You can obtain additional information, most of which is obscure or specialized, by passing options to who. One that's more likely than others to be useful is --count (or -q), which produces a more compact summary of the data:

```
$ who -q
luke luke sally rod
# users=4
```

This output includes just the usernames and a line specifying the total number of sessions. (That number counts one user with multiple logins multiple times.) For information on additional who options, consult its man page.

Certification Objective

An alternative to who is w, which is similar to who but produces somewhat more verbose output:

```
$ w
 12:36:40 up 37 days, 45 min,  3 users,  load average: 0.00, 0.02, 0.05
USER   TTY      LOGIN@   IDLE   JCPU   PCPU WHAT
luke   :0     06Mar12 40:23   0.01s  0.01s -bash
luke   pts/0  06Mar12 40.00s  0.01s  0.01s nano
sally  pts/7  13:00   20:37m  0.00s  0.00s -bash
rod    pts/9  13:22   2:50   12:44m  0.00s /bin/sh /etc/xdg/xfce4/xinitrc
```

As you can see, w displays much of the same information as who, including the terminal identifier (TTY) and login time (in a different format). In addition, w displays some further information:

- The session's idle time tells you how long it's been since the user has interacted with this session. This information can help you identify sessions that the user may have abandoned.
- The JCPU column identifies the total amount of CPU time associated with the session. This can be useful debugging information if the computer has become sluggish because of out-of-control processes.
- The PCPU column identifies the total amount of CPU time associated with the current process running in the session. Again, this information can help you track down out-of-control processes.
- The WHAT column tells you what program the session is running.

Some configurations also display a FROM column, which shows a remote hostname. Using the -f option toggles this option on or off. A few other options can eliminate or modify w's output. Consult the program's man page for details.

Working as *root*

Linux is modeled after Unix, which was designed as a multi-user OS. In principle, you can have thousands of accounts on a single Unix (or Linux) computer. One user, though, needs extraordinary power in order to manage the features of the computer as a whole. This is the root user, also known as the *superuser* or the *administrator*. Knowing why root exists, how to do things as root, and how to use root privileges safely is important for managing a Linux system.

Why Work as *root*?

Certification Objective

Most people use computers to do ordinary day-to-day computer tasks—browse the Web, write letters, manage a music collection, and so on. These activities are known collectively as *user tasks*, and they don't require special privileges. As just noted, a Linux computer can have many user accounts, and the users can use the computer from these user accounts (also known as *unprivileged accounts* or *unprivileged users*) to perform such user tasks.

Certification Objective

The `root` account, on the other hand, exists to enable you to perform *administrative tasks*. These tasks include installing new software, preparing a new disk for use in the computer, and managing ordinary user accounts. Such tasks require access to system files that ordinary users need not modify, or sometimes even read.

To facilitate performing these tasks, `root` can read and write every file on the computer. Since Linux relies on files to store system settings, this effectively gives `root` the power to change any detail of the OS's operation, which is the point of having a superuser account. If the computer is a workstation that's used by just one individual, you might wonder why the distinction between `root` and the user account is necessary. The explanation is that the power of the `root` account can lead to accidental damage. For instance, take the `rm` command. If you mistype an `rm` command as an ordinary user, you can accidentally delete your own files but not system files. Make the same mistake as `root`, however, and you can delete system files, perhaps making the computer unbootable. Thus, you should be cautious when using the `root` account—a topic I'll come back to shortly, in "Using `root` Privileges Safely."

Acquiring *root* Privileges

When you need to perform a command-line task that requires `root` privileges, you can do so in any of three ways:

Log in as `root` You can log in directly as `root` at a text-mode shell or by using a remote login tool such as SSH. You can even log into GUI mode as `root` on some Linux distributions, although some distributions disallow this by default because it's very dangerous.

Certification Objective

Use `su` The `su` command enables you to change your identity within a shell. Type **`su`** ***`username`*** to change your identity to that of the specified *`username`*. If you omit *`username`*, `root` is assumed, so typing **`su`** enables you to effectively become `root`. You must, however, know the password for the target account for su stands for *switch user* or *substitute user*.

this command to work. After you acquire root privileges in this way, you can type as many commands as root as you like. When you're done, type **exit** to relinquish your superuser status. You can also use su to run a single command as root by using the -c option, as in **su -c *command*** to run *command* as root. If you use a dash (-) within the command, as in **su -** or **su - luke**, the program opens a login session, which runs the target user's login scripts. This can be important because these scripts often set environment variables that can be important for that user, such as $PATH.

Certification Objective

Use sudo The sudo command is similar to su, but it works for just one command at a time, which you type after sudo, similar to using su -c. For instance, typing **sudo cat /etc/shadow** enables you to see the contents of the /etc/shadow file, which is not readable by ordinary users. You must type either your own password or the root password, depending on the sudo configuration, when you use this program. (When using su -c, you must always type the root password.) The next command you type will be executed using your ordinary account privileges. By default, most distributions permit only users who are members of the wheel group to use sudo. Some distributions, such as Ubuntu, rely heavily on sudo and don't permit direct root logins by default.

If you acquire root privileges by logging in directly as root or by using su, your shell prompt will change:

```
[luke@wembleth ~]$ su
Password:
[root@wembleth luke]#
```

To use root commands in Ubuntu, you must either precede them with sudo (as in sudo cat /etc/shadow) or type sudo su to acquire a longer-lasting root shell.

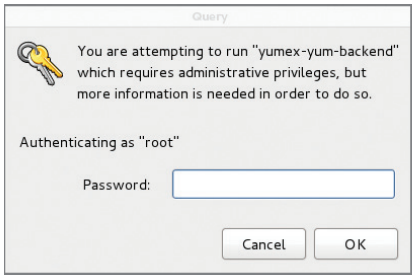
In this example, the username has changed from luke to root; the directory has changed from ~ to luke (since for root, ~ refers to /root, not /home/luke); and the last character of the prompt has changed from a dollar sign ($) to a hash mark (#). Because I use just the last character of the prompt for most examples printed on their own lines in this book, such examples implicitly specify whether a command requires root privileges by the prompt used. For instance, consider accessing the /etc/shadow file mentioned earlier:

```
# cat /etc/shadow
```

The use of the hash mark prompt indicates that you *must* type this command as root.

Some of this book's chapters describe both GUI and text-mode methods of system administration. How, then, can you administer Linux in GUI mode if you must use the text-mode su (or sudo) command to acquire root privileges? One way is to launch a GUI administrative tool from a shell in which you've already used su. This approach works but is a bit inelegant if you're more comfortable with a GUI than with a command line. Most distributions, therefore, provide an alternative: You can launch administrative tools from the computer's desktop menus and the GUI tools will then prompt you for the `root` password, as shown in Figure 13.2. If you type the password correctly, the program will launch. The result is similar to that of launching the program from a shell using `sudo`—it runs as `root`, but without making anything else run as `root`.

If you're a member of the `wheel` group, most distributions prompt you for your own password rather than the `root` password when using such tools.


Query

You are attempting to run "yumex-yum-backend" which requires administrative privileges, but more information is needed in order to do so.

Authenticating as "root"

Password:

FIGURE 13.2 **When you launch an administrative tool from a GUI, Linux asks you for a password.**

SETTING THE *ROOT* PASSWORD

If you're using your own computer, you may already know or have the `root` password, since most distributions prompt you to enter the `root` password when you install the system. Some distributions, such as openSUSE, set the `root` password to be identical to the first user account's password by default. Ubuntu doesn't set a `root` password at all; instead, Ubuntu's security model favors the use of `sudo` or GUI equivalents.

If you've forgotten the `root` password, you can recover or reset it in several ways, such as:

- You can type **`sudo passwd root`** to reset the `root` password. As described in Chapter 14, `passwd` is the tool to set an account's password, and so this command will reset the `root` password. Used in this way, `sudo` will prompt you for a password (you should type your user password) followed by two prompts for a new `root` password. This method will work, however, only if you're an authorized user of `sudo` on your system.

(Continues)

Setting the *root* Password (Continued)

- Reboot into an emergency system, such as Parted Magic (http://partedmagic.com), and mount your regular installation's root filesystem. You can then locate its /etc/shadow file (it will be located at another mount point, such as /mnt/local/etc/shadow) and load it into a text editor. Locate the entry for an account whose password you know and copy the second colon-delimited entry from that line into the equivalent space for the root account. Passwords are stored in an encrypted form, so you won't recognize the password you copy and paste in this way. After you save your changes, reboot and log into the root account. Once you've logged in, change the root password to something unique.

If you're using a school's or employer's computer, you should either refrain from attempting to perform tasks as root or obtain the root password through authorized channels. If you've been hired as a system administrator, your employer should provide this information, or specify another way to perform maintenance (such as using sudo). If you aren't the computer's official administrator, attempting to perform root tasks on it may be grounds for dismissal. In fact, you might even be sued or imprisoned for breaking into a computer without authorization!

Using *root* Privileges Safely

Note that emergency systems, such as Parted Magic, typically run all your commands as root, so be careful when using them!

As already described, root power is dangerous. For instance, preparing a disk requires creating a filesystem on the disk. This topic is not covered in detail in this book, but you should know that you can create a filesystem with mkfs, which takes a partition identifier as an option. Suppose that you want to create a new filesystem on /dev/sda3, so you quickly type a command:

```
# mkfs /dev/sda2
```

Did you catch the error? The command specifies /dev/sda2 rather than /dev/sda3. If /dev/sda2 held important data (such as your Linux installation or your /home directory), this mistake could be absolutely disastrous.

This is just one example of the trouble that improper use of root can cause. Others include intruders gaining root access to your computer; unintended changes to configuration files; damage to some (even if not all) of the computer's system files; and changes to ownership or permissions on ordinary user files, rendering them inaccessible to their true owners. Thus, I recommend you take the following precautions whenever you need `root` access:

- Ask yourself if you really need `root` access. Sometimes there's a way to achieve a goal without superuser privileges or by using those privileges in a more limited way than you'd originally planned. For instance, you might find that only `root` can write to a removable disk. Such a problem can usually be overcome by adjusting permissions on the disk in one way or another, thus limiting the use of `root`.
- Before pressing the Enter key after typing any command as `root` (or clicking any confirmation button in a GUI program running as `root`), take your hands *off* the keyboard and mouse, look over the command, and verify that it's correct in every respect. A simple typo can cause a world of pain. This step has saved me from disaster more than once!
- Never run a suspicious program as `root`. On multi-user systems, unscrupulous users can try to trick administrators into running programs that will do nasty things or give the attacker `root` privileges. Programs downloaded from random Internet sites could in principle be designed to compromise your security, and such programs are much more dangerous when run as `root`.

If a program asks for your or the `root` password and it's not an administrative program you trust, be suspicious! Research the program before giving it your password!

- Use `root` privileges for as brief a period as possible. If you need to type just one or two commands as `root`, do so and then type **`exit`** in the `root` shell to log out or return to your normal privileges. Alternatively, use `sudo` to run the commands. It's easy to overlook the fact that you're using a `root` shell and therefore type commands as `root` that don't need that privilege. Every command typed as `root` is a risk.
- Never leave a `root` shell accessible to others. If you're performing `root` maintenance tasks and are called away, type **`exit`** in your `root` shell before leaving the computer.
- Be careful with the `root` password. Don't share the password with others, and be cautious about typing it in a public area or when others might be looking over your shoulder. If you're using Linux professionally, your employer may have guidelines concerning who may have `root` access to a computer. Learn and obey those rules. Be sure to select a strong `root` password, too.

Chapter 14 describes how to select a strong password.

Following these rules of thumb can help keep you from damaging your computer or giving somebody else `root` access to the computer.

THE ESSENTIALS AND BEYOND

Accounts are critical to Linux's normal functioning. Ordinarily, most of the tasks you perform on a Linux computer require the privileges of a normal user, so you'll use your own user account to handle these tasks. You can use tools like `whoami`, `id`, who, and w to identify your normal account and to determine who else might be using the computer. Occasionally, you'll need to perform administrative tasks that require additional privileges. To acquire such privileges, you must use the `root` account, which can read and write any ordinary file, access hardware in a low-level way, reconfigure the network, and perform other tasks that ordinary users aren't allowed to do. Because `root` is so powerful, you should use that power sparingly and be extremely careful when you do use it, lest a typo or other accident cause serious problems.

SUGGESTED EXERCISES

- Type **`whoami`** followed by **`id`** to review your ordinary user account status. Chances are the `id` command will reveal that you're a member of a number of groups. Perform a Web search to learn what each one does.
- Read the `/etc/passwd` file or type **`getent passwd`** to review what accounts are defined on the computer. Are there ordinary user accounts (those with UIDs above 500 or 1,000, depending on your distribution) other than your own? Try performing a Web search to learn the purpose of a few of the system accounts (those with UIDs below 500 or 1,000, depending on your distribution).

REVIEW QUESTIONS

1. What is the purpose of the system account with a UID of 0?
   - **A.** It's the system administration account.
   - **B.** It's the account for the first ordinary user.
   - **C.** Nothing; UID 0 is left intentionally undefined.
   - **D.** It varies from one distribution to another.
   - **E.** It's a low-privilege account that's used as a default by some servers.
2. What type of information will you find in `/etc/passwd` for ordinary user accounts? (Select all that apply.)
   - **A.** A user ID (UID) number
   - **B.** A complete listing of every group to which the user belongs
   - **C.** The path to the account's home directory
   - **D.** The path to the account's default GUI desktop environment
   - **E.** The path to the account's default text-mode shell

(Continues)

The Essentials and Beyond *(Continued)*

3. You want to run the command `iptables -L` as `root`, but you're logged in as an ordinary user. Which of the following commands will do the job, assuming the system is configured to give you `root` access via the appropriate command?
   - **A.** **`sudo iptables -L`**
   - **B.** **`root iptables -L`**
   - **C.** **`passwd iptables -L`**
   - **D.** **`su iptables -L`**
   - **E.** **`admin iptables -L`**
4. True or false: `whoami` provides more information than `id`.
5. True or false: Linux stores information on its groups in the `/etc/groups` file.
6. True or false: As a general rule, you should employ extra care when running programs as `root`.
7. The file that associates usernames with UID numbers in Linux is ________________. (Provide the complete path to the file.)
8. To learn who is currently logged into the computer and what programs they're currently running, you can type ______.
9. UIDs below 500 or 1,000 (depending on the distribution) are reserved for use by ___________ accounts.