Chapter 10

Administering Users and Groups

✓ **Objective 2.2 Given a scenario, implement identity management.**

If you want to buy a famous and expensive piece of art, you should make sure it isn't a fake. In other words, you want to make sure it is authentic. The same is true for allowing users access to a computer system. You want to make sure they are authentic users who have been previously given authorization to access the system. This process, called *authentication*, is defined as determining whether a person or program is who they claim to be. This chapter covers administering the access controls Linux uses to check a user's credentials and permit or deny access to the system.

Besides user authentication, you need to know how to audit users, manage group memberships, configure user environments, and, if needed, set up disk space usage limits for the accounts. We'll cover those topics as well.

Managing User Accounts

Adding and modifying user account credentials, which includes usernames, account information, and passwords, is an important (but tedious) part of system administration. In addition, you need to know how to delete these credentials when warranted. Managing user accounts and looking at the underlying credential framework is covered in the following sections.

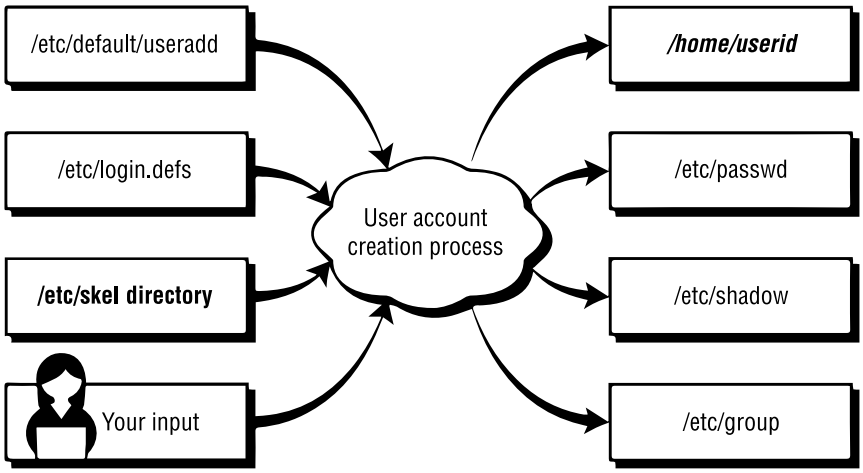
Adding Accounts

To add a new user account on the system, the `useradd` utility is typically used. However, the process actually involves several players besides the `useradd` command. A depiction of the process is illustrated in Figure 10.1.

You can see in Figure 10.1 that there are several team players involved in the account creation process. Notice that the `/etc/skel` directory is bolded. This is because, depending on the other configuration files, it may not be used in the process. The same goes for the `/home/`*`userid`* directory. It may not be created or it may have an alternative name, depending on the system's account creation configuration. You'll learn more about these directories shortly.

Before we jump into the `useradd` utility details, let's look at the two files and the directory involved in the creation side of the process.

FIGURE 10.1 Adding a user account



The */etc/login.defs* File

This configuration file is typically installed by default on most Linux distributions. It contains directives for use in various shadow password suite commands. *Shadow password suite* is an umbrella term for commands dealing with account credentials, such as the `useradd`, `userdel`, and `passwd` commands.

The directives in this configuration file control password length, how long until the user is required to change the account's password, whether or not a home directory is created by default, and so on. The file is typically filled with comments and commented-out directives (which make the directives inactive). Listing 10.1 shows only the active directives within the `/etc/login.defs` file, after stripping out blank and comment lines on a Rocky Linux distribution.

Listing 10.1: Active directives in the `/etc/login.defs` configuration file

```
$ grep -v ^$ /etc/login.defs | grep -v ^\#
MAIL_DIR    /var/spool/mail
UMASK    022
HOME_MODE    0700
PASS_MAX_DAYS  99999
PASS_MIN_DAYS  0
PASS_MIN_LEN  5
PASS_WARN_AGE  7
UID_MIN                  1000
UID_MAX                 60000
```

```
SYS_UID_MIN               201
SYS_UID_MAX               999
GID_MIN                  1000
GID_MAX                 60000
SYS_GID_MIN               201
SYS_GID_MAX               999
CREATE_HOME     yes
USERGROUPS_ENAB  yes
ENCRYPT_METHOD  SHA512
$
```

Notice the `UID_MIN` directive in Listing 10.1. A *User Identification Number (UID)* is the number used by Linux to identify user accounts. A *user account*, sometimes called a *normal account*, is any account an authorized human has been given to access the system and perform daily tasks, such as open desktop applications or run scripts. While humans use account names, Linux uses UIDs. The `UID_MIN` indicates the lowest UID allowed for user accounts. On the system in Listing 10.1, `UID_MIN` is set to 1000. This is typical, though some systems set it at 500.

System accounts are accounts that provide services (daemons) or that perform special tasks, such as the root user account. A system account's minimum UID is set by the `SYS_UID_MIN` directive, and its maximum is set by the `SYS_UID_MAX` directive. The settings in this file are typical.

Keep in mind that these settings are for accounts created *after* the initial Linux distribution installation. For example, the root user account always has a UID of 0, which is below the `SYS_UID_MIN`, as shown snipped in Listing 10.2.

Listing 10.2: The root user account's UID

```
$ gawk -F: '{print $3, $1}' /etc/passwd | sort -n
0 root
1 bin
2 daemon
3 adm
[...]
```

Some additional directives critical to common user account creation are covered briefly in Table 10.1.

TABLE 10.1 A few vital `/etc/login.defs` directives

Name	Description
`PASS_MAX_DAYS`	Number of days until a password change is required. This is the password's expiration date.
`PASS_MIN_DAYS`	Number of days after a password is changed until the password may be changed again.

Name	Description
PASS_MIN_LENGTH	Minimum number of characters required in password.
PASS_WARN_AGE	Number of days a warning is issued to the user prior to a password's expiration.
CREATE_HOME	Default is no. If set to yes, a user account home directory is created.
ENCRYPT_METHOD	The method used to hash account passwords.

The /etc/login.defs file is only one of the configuration files used for the user account process's creation side. The other file is covered next.

The */etc/default/useradd* File

The /etc/default/useradd file is another configuration file that directs the process of creating accounts. It typically is a much shorter file than the /etc/login.defs file. An example from a Rocky Linux distribution is shown in Listing 10.3.

Listing 10.3: The /etc/default/useradd configuration file

```
$ cat /etc/default/useradd
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
$
$ useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
$
```

Notice in Listing 10.3 that there are two different ways to display the active directives in this file. You can use the `cat` command or invoke the `useradd -D` command. Both are equally simple to use. One cool fact about the `useradd -D` command is that you can use it to modify the directives within the `/etc/default/useradd` file.

In Listing 10.3, notice the `HOME` directive. It is currently set to `/home`, which means that any newly created user accounts will have their account directories located within the `/home` directory. Keep in mind that if `CREATE_HOME` is not set or set to `no` in the `/etc/login.defs` file, a home directory is *not* created by default.

Some additional directives critical to common user account creation are covered briefly in Table 10.2.

TABLE 10.2 A few vital `/etc/default/useradd` directives

Name	Description
`HOME`	Base directory for user account directories.
`INACTIVE`	Number of days after a password has expired and has not been changed until the account will be deactivated. See `PASS_MAX_DAYS` in Table 10.1.
`SKEL`	The skeleton directory.
`SHELL`	User account default shell program.

The `SHELL` directive needs a little more explanation. Typically it is set to `/bin/bash`, which means when you access the command line, your user process is running the `/bin/bash` shell program. This program provides you with the prompt at the command line and handles any commands you enter there.

Be aware that some distributions, such as Ubuntu, set the `SHELL` directive by default to `/bin/sh`, which is a symbolic link to another shell. On Ubuntu this links to the Dash shell instead of the Bash shell.

The */etc/skel* Directory

The `/etc/skel` directory, or the *skeleton directory* (see Table 10.2) as it is commonly called, holds files. If a home directory is created for a user, these files are to be copied to the user account's home directory when the account is created. Listing 10.4 shows the files within the `/etc/skel` directory on a Fedora Workstation distribution.

Listing 10.4: Files in the `/etc/skel` directory

```
$ ls -a /etc/skel
.  ..  .bash_logout  .bash_profile  .bashrc  .mozilla
$
```

In Listing 10.4, the `ls` command was employed with the `-a` option so that hidden files (files starting with a dot) are displayed. Recall that hidden files do not normally display without the `-a` option on the `ls` command. These files are account environment files as well as a Mozilla Firefox web browser configuration file directory. We'll cover environment files later in this chapter. You can modify any of these files or add new files and directories, if needed.

The `/etc/skel` files are copied to user account home directories only when the account is created. Therefore, if you make changes to the files later, you'll have to migrate those changed files to current user accounts either by hand or by shell scripts.

Now that we've covered the files in the creation side of the user account creation process, let's look at the files and directories that are built or modified as part of the process. Go back and look at Figure 10.1, if necessary, to refresh your memory of the various file and directory names.

The */etc/passwd* File

Account information is stored in the `/etc/passwd` file. Each account's data occupies a single line in the file. When an account is created, a new record for that account is added to the `/etc/passwd` file. A snipped example is shown in Listing 10.5.

Listing 10.5: Account records in the `/etc/passwd` file

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
[...]
tcpdump:x:72:72::/:/sbin/nologin
user1:x:1000:1000:User One:/home/user1:/bin/bash
Christine:x:1001:1001:Christine B:/home/Christine:/bin/bash
[...]
$
```

The `/etc/passwd` file records contain several fields. Each field in a record is delimited by a colon (`:`). There are seven fields in total, as described in Table 10.3.

TABLE 10.3 The `/etc/passwd` file's record fields

Field No.	Description
1	User account's username.
2	Password field. Typically this file is no longer used to store passwords. An `x` in this field indicates passwords are stored in the `/etc/shadow` file.

TABLE 10.3 The /etc/passwd file's record fields *(continued)*

Field No.	Description
3	User account's user identification number (UID).
4	User account's group identification number (GID).
5	Comment field. This field is optional. Traditionally it contains the user's full name.
6	User account's home directory.
7	User account's default shell. If set to /sbin/nologin or /bin/false, then the user cannot interactively log into the system.

You would think that the password file would hold passwords, but due to its file permissions, the password file can be compromised. Therefore, passwords are stored in the more locked-down /etc/shadow file.

You may find yourself working for an organization that has passwords stored in the /etc/passwd file. If so, politely suggest that the passwords be migrated to the /etc/shadow file via the pwconv command. If the organization refuses, walk, or even better run, to the door and go find a job at a different company.

You may have noticed that in a /etc/password record, field #7 may contain either the /sbin/nologin or /bin/false default shell. This is to prevent an account from interactively logging into the system. The /sbin/nologin is typically set for system service account records. System services (daemons) do need to have system accounts, but they do *not* interactively log in. Instead, they run in the background under their own account name. If a malicious person attempted to interactively log in using the account (and they made it past other blockades, which you'll learn about shortly), they are politely kicked off the system. Basically, /sbin/nologin displays a brief message and logs you off before you reach a command prompt. If desired, you can modify the message shown by creating the file /etc/nologin.txt and adding the desired text.

The /bin/false shell is a little more brutal. If this is set as a user account's default shell, there are no messages shown, and the user is just logged out of the system.

The */etc/shadow* File

Another file that is updated when an account is created is the /etc/shadow file. It contains information regarding the account's password, even if you have not yet provided a password

for the account. Like the /etc/passwd file, each account's data occupies a single file line. A snipped example is shown in Listing 10.6.

Listing 10.6: Account records in the /etc/shadow file

```
$ sudo cat /etc/shadow
root:!::0:99999:7:::
bin:*:17589:0:99999:7:::
daemon:*:17589:0:99999:7:::
[...]
user1:$6$bvqdqU[...]:17738:0:99999:7:::
Christine:Wb8I8Iw$6[...]:17751:0:99999:7:::
[...]
$
```

The /etc/shadow records contain several fields. Each field in a record is delimited by a colon (:). There are nine fields in total, described in Table 10.4.

TABLE 10.4 The /etc/shadow file's record fields

Field No.	Description
1	User account's username.
2	Password field. The password is a salted and hashed password. A !! or ! indicates a password has not been set for the account. A ! or * indicates the account cannot use a password to log in. A ! in front of a password indicates the account has been locked.
3	Date of last password change in Unix Epoch time (days) format.
4	Number of days after a password is changed until the password may be changed again.
5	Number of days until a password change is required. This is the password's expiration date.
6	Number of days a warning is issued to the user prior to a password's expiration. (See field #5).
7	Number of days after a password has expired (see field #5) and has not been changed until the account will be deactivated.
8	Date of account's expiration in Unix Epoch time (days) format.
9	Called the *special flag*. It is a field for a special future use, is currently not used, and is blank.

Notice that field #1 is the account's username. This is the only field shared with the `/etc/passwd` file.

Unix Epoch time, which is also called *POSIX time*, is the number of seconds since January 1, 1970, although the `/etc/shadow` file expresses it in days. It has a long history with Unix and Linux systems and will potentially cause problems in the year 2038. You don't have to drag out your calculator to determine what a field's date is using the Epoch. Instead, the `chage` utility, covered later in this chapter, does that for you.

It's vital to understand the different possible expirations. When password expiration has occurred, there is a grace period. The user will have a certain number of days (designated in field #7) to log into the account using the old password but must change the password immediately. However, if password expiration has occurred and the user does *not* log in to the system in time, the user is effectively locked out of the system.

With account expiration, there is no grace period. After the account expires, the user cannot log into the account with its password.

You may have noticed that we have not yet covered the `/etc/group` file. It does get modified as part of the account creation process. However, that discussion is saved for the section "Managing Groups" later in this chapter.

The Account Creation Process

Distributions tend to vary greatly in their configuration when it comes to user accounts. Therefore, before you launch into creating accounts with the `useradd` utility, it's wise to review some directives within each distro's user account configuration files (see Tables 10.1 and 10.2). In Listing 10.7, the `CREATE_HOME` and `SHELL` directives are checked on a Rocky Linux distribution.

Listing 10.7: Checking user account directives on Fedora Workstation

```
$ grep CREATE_HOME /etc/login.defs
CREATE_HOME     yes
$
$ useradd -D | grep SHELL
SHELL=/bin/bash
$
```

You can see on this Rocky Linux distribution that the home directory will be created by default because `CREATE_HOME` is set to `yes`. The `SHELL` directive is pointing to the Bash shell, `/bin/bash`, which is the typical shell for most interactive user accounts.

The `useradd` command, as mentioned earlier, is the primary tool for creating user accounts on most distributions. Creating an account on a Rocky Linux distribution with the `useradd` utility is shown in Listing 10.8.

Listing 10.8: Creating a user account on a Rocky Linux Workstation

```
$ sudo useradd DAdams
[sudo] password for Christine:
$
$ grep ^DAdams /etc/passwd
DAdams:x:1002:1002::/home/DAdams:/bin/bash
$
$ sudo grep ^DAdams /etc/shadow
DAdams:!!:17806:0:99999:7:::
$
$ sudo ls -a /home/DAdams/
.  ..  .bash_logout  .bash_profile  .bashrc  .mozilla
$
```

Because the Rocky Linux distribution we are using in Listing 10.8 has the `CREATE_HOME` directive set to `yes` and `SHELL` set to `/bin/bash`, there is no need to employ any `useradd` command options. The only argument needed is the user account name, which is `DAdams`. After the utility is used to create the account in Listing 10.8, notice that records now exist for the new user account in both the `/etc/passwd` and `/etc/shadow` files. Also, a new directory was created, `/home/DAdams`, which contains files from the `/etc/skel` directory. Keep in mind at this point that no password has been added to the `DAdams` account yet, and thus its record in the `/etc/shadow` file shows `!!` in the password field.

Now let's take a look at creating an account on a different Linux distribution. The Ubuntu Desktop distro does things a little differently. In Listing 10.9, you can see that `CREATE_HOME` is *not* set, so it will default to `no`.

Listing 10.9: Checking user account directives on Ubuntu Desktop

```
$ grep CREATE_HOME /etc/login.defs
$
$ useradd -D | grep SHELL
SHELL=/bin/sh
$
```

Also in Listing 10.9, notice that the `SHELL` directive is set to `/bin/sh` instead of the Bash shell. This means that when you create an interactive user account, you will need to specify Bash shell, if desired.

Therefore, when creating a user account on this Ubuntu distribution, if you want the account to have a home directory and use the Bash shell, you will need to employ additional `useradd` command options. The `useradd` utility has many useful options for various needs, and the most typical ones are listed in Table 10.5.

TABLE 10.5 The useradd command's commonly used options

Short	Long	Description
-c	--comment	Comment field contents. Traditionally it contains the user's full name. Optional.
-d	--home or --home-dir	User's home directory specification. Default action is set by the HOME and CREATE_HOME directives.
-D	--defaults	Display /etc/default/useradd directives.
-e	--expiredate	Date of account's expiration in *YYYY-MM-DD* format. Default action is set by the EXPIRE directive.
-f	--inactive	Number of days after a password has expired and has not been changed until the account will be deactivated. A -1 indicates that the account will never be deactivated. Default action is set by the INACTIVE directive.
-g	--gid	Account's group membership, which is active when user logs into system (default group).
-G	--groups	Account's additional group memberships.
-m	--create-home	If it does not exist, create the user account's home directory. Default action is set by the CREATE_HOME directive.
-M	N/A or --no-create-home	Do *not* create the user account's home directory. Default action is set by the CREATE_HOME directive.
-s	--shell	Account's shell. Default action is set by the SHELL directive.
-u	--uid	Account's User Identification (UID) number.
-r	--system	Create a system account instead of a user account.

We need to employ a few of the options in Table 10.5 to create a user account on the Ubuntu Desktop distribution. An example is shown in Listing 10.10.

Listing 10.10: Creating a user account on Ubuntu Desktop

```
$ sudo useradd -md /home/JKirk -s /bin/bash JKirk
[sudo] password for Christine:
$
$ grep ^JKirk /etc/passwd
```

```
JKirk:x:1002:1002::/home/JKirk:/bin/bash
$
$ sudo grep ^JKirk /etc/shadow
JKirk:!:17806:0:99999:7:::
$
$ sudo ls -a /home/JKirk/
.  ..  .bash_logout  .bashrc  examples.desktop  .profile
$
$ sudo ls -a /etc/skel
.  ..  .bash_logout  .bashrc  examples.desktop  .profile
$
```

Notice in Listing 10.10 that three options are used along with the `useradd` command. Because this system does not have the `CREATE_HOME` directive set, the `-m` option is needed to force `useradd` to make a home directory for the account. The `-d` switch designates that the directory name should be `/home/JKirk`. Because the `SHELL` directive is set to `/bin/sh` on this system, the `-s` option is needed to set the account's default shell to `/bin/bash`.

After the utility is used to create the account in Listing 10.10, notice that records now exist for the new user account in the `/etc/passwd` and `/etc/shadow` files. Also, a new directory was created, `/home/JKirk`, which contains files from this distro's `/etc/skel` directory. Keep in mind at this point that no password has been added to the `JKirk` account yet, and thus its record in the `/etc/shadow` file shows `!` in the password field.

The Ubuntu and Debian distributions promote the use of the `adduser` program instead of the `useradd` utility. Their man pages refer to the `useradd` command as a "low-level utility." Some other distros include a symbolic link to `useradd` named `adduser`, which may help (or not). The `adduser` configuration information is typically stored in the `/etc/adduser.conf` file.

Another way to view account records in the `/etc/passwd` and `/etc/shadow` files is via the `getent` utility. For this program you pass only the filename followed by the account name whose record you wish to view. The command is employed in Listing 10.11 to view the account that was created in Listing 10.10.

Listing 10.11: Using `getent` to view a user account on Ubuntu Desktop

```
$ getent passwd JKirk
JKirk:x:1002:1002::/home/JKirk:/bin/bash
$
$ getent shadow JKirk
$
$ sudo getent shadow JKirk
JKirk:!:17806:0:99999:7:::
$
```

Notice in Listing 10.11 that when super user privileges are not used with `getent` for the `shadow` file, nothing is returned. This is because `getent` honors the security settings on the `/etc/shadow` file.

If you need to modify the `/etc/default/useradd` file's directive settings, instead of using a text editor, you can employ the `useradd -D` command. Just tack on the needed arguments. For example, to modify the `SHELL` directive to point to the Bash shell, use super user privileges and issue the `useradd -D -s /bin/bash` command.

When creating an account, you can create a password via the `crypt` utility and then add it when the account is created via the `-p` option on the `useradd` utility. However, that is not only cumbersome but considered a bad practice. In the next section, we'll cover creating and managing account passwords properly.

Maintaining Passwords

When you first create an interactive account, you should immediately afterward create a password for that account using the `passwd` utility. In Listing 10.12, a password is created for the new `DAdams` account on a Rocky Linux system.

Listing 10.12: Using `passwd` for a new account on Fedora Workstation

```
$ sudo passwd DAdams
Changing password for user DAdams.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
$
```

You can also update a password for a particular user using the `passwd` utility and pass the user's account name as an argument, similar to what is shown in Listing 10.12. If you need to update your own account's password, just enter `passwd` with no additional command arguments.

The `passwd` utility works hand in hand with pluggable authentication modules (PAMs). For example, when you set or change a password via the `passwd` utility, the `pam_cracklib` PAM checks the password to flag easily guessed passwords or passwords that use words found in the dictionary. PAM is covered in more detail in Chapter 16, "Looking at Access and Authentication Methods."

You can do more than set and modify passwords with the `passwd` utility. You can also lock or unlock accounts, set an account's password to expired, delete an account's password, and so on. Table 10.6 shows commonly used `passwd` switches; all of these options require super user privileges.

TABLE 10.6 The passwd command's commonly used options

Short	Long	Description
-d	--delete	Removes the account's password.
-e	--expire	Sets an account's password as expired. User is required to change account password at next login.
-i	--inactive	Sets the number of days after a password has expired and has not been changed until the account will be deactivated.
-l	--lock	Places an exclamation point (!) in front of the account's password within the /etc/shadow file, effectively preventing the user from logging into the system via using the account's password.
-n	--minimum	Sets the number of days after a password is changed until the password may be changed again.
-S	--status	Displays the account's password status.
-u	--unlock	Removes a placed exclamation point (!) from the account's password within the /etc/shadow file.
-w	--warning or --warndays	Sets the number of days a warning is issued to the user prior to a password's expiration.
-x	--maximum or --maxdays	Sets the number of days until a password change is required. This is the password's expiration date.

One option in Table 10.6 needs a little more explanation, and that is the -S option. An example is shown in Listing 10.13.

Listing 10.13: Using passwd -S to view an account's password status

```
$ sudo passwd -S DAdams
DAdams PS 2021-10-01 0 99999 7 -1 (Password set, SHA512 crypt.)
$
```

In Listing 10.13, the DAdams account's password status is displayed. The status contains the account password's state, which is either a usable password (P), no password (NP), or a locked password (L). After the password state, the last password change date is shown, followed by the password's minimum, maximum, warning, and inactive settings. Additional status is shown within the parentheses, which includes whether or not the password is set as well as the hash algorithm used to protect it.

You can also use the chage utility to display similar password information, but in a more human-readable format, as shown in Listing 10.14.

Listing 10.14: Using `chage -l` to view an account's password status

```
$ sudo chage -l DAdams
Last password change                                    : Oct 02, 2021
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
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
$
```

The `chage` program can modify password settings as well. You can either employ various command options (see its man pages for details) or use the `chage` utility interactively, as shown in Listing 10.15.

Listing 10.15: Using `chage` to change an account password's settings

```
$ sudo chage DAdams
Changing the aging information for DAdams
Enter the new value, or press ENTER for the default

        Minimum Password Age [0]: 5
        Maximum Password Age [99999]: 30
        Last Password Change (YYYY-MM-DD) [2021-10-02]:
        Password Expiration Warning [7]: 15
        Password Inactive [-1]: 3
        Account Expiration Date (YYYY-MM-DD) [-1]:
$
```

Notice in Listing 10.15 that the password expiration warning is set to 15 days. This is a good setting if your company allows two-week vacations.

Modifying Accounts

The utility employed to modify accounts is the `usermod` program. Similar to the `passwd` command, you can lock and unlock accounts, as shown in Listing 10.16.

Listing 10.16: Using `usermod` to lock an account

```
$ sudo usermod -L  DAdams
$
$ sudo passwd -S DAdams
DAdams LK 2021-10-01 5 30 15 3 (Password locked.)
$
$ sudo getent shadow DAdams
DAdams: !$6$B/zCaNx[...]:17806:5:30:15:3::
```

```
$
$ sudo usermod -U  DAdams
$
$ sudo passwd -S DAdams
DAdams PS 2021-10-01 5 30 15 3 (Password set, SHA512 crypt.)
$
```

Notice in Listing 10.16 that the `usermod -L` command is used to lock the `DAdams` account. The `passwd -S` command shows that the password status is `LK`, indicating it is locked. In Listing 10.16, the snipped `getent` utility output shows that an exclamation point (`!`) was placed in front of the `DAdams` account's password, which is what is causing the account to be locked. The lock is then removed via the `usermod -U` command, and the status is rechecked.

You can make many modifications to user accounts via the `usermod` utility's switches. The commonly used options are shown in Table 10.7.

TABLE 10.7 The `usermod` command's commonly used options

Short	Long	Description
`-c`	`--comment`	Modify the comment field contents.
`-d`	`--home`	Set a new user home directory specification. Use with the `-m` option to move the current directory's files to the new location.
`-e`	`--expiredate`	Modify the account's expiration date. Use *YYYY-MM-DD* format.
`-f`	`--inactive`	Modify the number of days after a password has expired and has not been changed that the account will be deactivated. A `-1` indicates that the account will never be deactivated.
`-g`	`--gid`	Change the account's default group membership.
`-G`	`--groups`	Update the account's additional group memberships. If only specifying new group membership, use the `-a` option to avoid removing the other group memberships.
`-l`	`--login`	Modify the account's username to the specified one. Does not modify the home directory.
`-L`	`--lock`	Lock the account by placing an exclamation point in front of the password within the account's `/etc/shadow` file record.
`-s`	`--shell`	Change the account's shell.
`-u`	`--uid`	Modify the account's User Identification (UID) number.
`-U`	`--unlock`	Unlock the account by removing the exclamation point from the front of the password within the account's `/etc/shadow` file record.

Notice that you can change an account's default group and provide memberships to additional groups. Account groups are covered in detail later in this chapter.

Where `usermod` comes in really handy is in a situation where you've created an account but forgot to check the distribution's account creation configuration settings. Listing 10.17 shows an example of this on an Ubuntu Desktop distribution.

Listing 10.17: Using `usermod` to modify an account

```
$ sudo useradd -md /home/DBowman DBowman
$
$ sudo getent passwd DBowman
DBowman:x:1003:1003::/home/DBowman:/bin/sh
$
$ sudo usermod -s /bin/bash DBowman
$
$ sudo getent passwd DBowman
DBowman:x:1003:1003::/home/DBowman:/bin/bash
$
```

In Listing 10.17, the user account `DBowman` is created, but when the account record is checked using the `getent` utility, it shows that the `/bin/sh` shell is being used instead of the Bash shell. To fix this problem, the `usermod` command is employed with the `-s` option, and the account's shell is modified to the `/bin/bash` shell instead.

Deleting Accounts

Deleting an account on Linux is fairly simple. The `userdel` utility is the key tool in this task. The most common option to use is the `-r` switch. This option will delete the account's home directory tree and any files within it. Listing 10.18 shows an example of deleting an account.

Listing 10.18: Using `userdel` to delete an account

```
$ sudo ls -a /home/DBowman
.  ..  .bash_logout  .bashrc  examples.desktop  .profile
$
$ sudo getent passwd DBowman
DBowman:x:1003:1003::/home/DBowman:/bin/bash
$
$ sudo userdel -r DBowman
userdel: DBowman mail spool (/var/mail/DBowman) not found
$
$ sudo ls -a /home/DBowman
ls: cannot access '/home/DBowman': No such file or directory
```

```
$
$ sudo getent passwd DBowman
$
```

The first two commands in Listing 10.18 show that the `/home/DBowman` directory exists and has files within it and that the account does have a record within the `/etc/passwd` file. The third command includes the `userdel -r` command to delete the account as well as the home directory. Notice that an error message is generated stating that the `/var/mail/DBowman` file could not be found. This is not a problem. It just means that this file was not created when the account was created. Finally, the last two commands show that both the `/home/DBowman` directory and its files were removed and that the `/etc/passwd` file no longer contains a record for the `DBowman` account.

Real World Scenario

Account Deletion Policies

Prior to deleting any accounts on a system, check with your employer's human resources staff and/or legal department or counsel. There may be policies in place concerning file retention for terminated or retired employees as well as those individuals who have left the company to change jobs. You may be required to back up files prior to deleting them from the system and/or perform some other types of documentation. If your employer has no such policy, it is a good idea to suggest that one be developed.

Managing Groups

Groups are organizational structures that are part of Linux's *discretionary access control (DAC)*. DAC is the traditional Linux security control, where access to a file, or any object, is based on the user's identity and current group membership. When a user account is created, it is given membership to a particular group, called the account's *default group*. Though a user account can have lots of group memberships, its process can have only one designated current group at a time. The default group is an account's current group, when the user first logs into the system.

Groups are identified by their name as well as their *group identification number (GID)*. This is similar to how users are identified by UIDs in that the GID is used by Linux to identify a particular group, whereas humans use group names.

If a default group is not designated when a user account is created, then a new group is created. This new group has the same name as the user account's name, and it is assigned a new GID. To see an account's default group, you can use the `getent` command to view

the /etc/passwd record for that account. Recall that the fourth field in the record is the account's GID, which is the default group. Review Table 10.3 if you need a refresher on the various /etc/passwd record fields. Listing 10.19 shows an example of viewing an account's default group information for the DAdams account, which was created on a Rocky Linux distribution.

Listing 10.19: Viewing an account's group memberships

```
$ getent passwd DAdams
DAdams:x:1002:1002::/home/DAdams:/bin/bash
$
$ sudo groups DAdams
DAdams : DAdams
$
$ getent group DAdams
DAdams:x:1002:
$
$ grep 1002 /etc/group
DAdams:x:1002:
$
```

The first command in Listing 10.19 shows that the DAdams account's default group has a GID of 1002, but it does not provide a group name. The groups command does show the group name, which is the same as the user account name, DAdams. This is typical when no default group was designated at account creation time. The third command, another getent command, shows that the group DAdams does indeed map to the 1002 GID. The fourth command confirms this information.

To add a user to a new group or change the account's default group, the group must pre-exist. This task is accomplished via the groupadd utility. The group's GID will be automatically set by the system, but you can override this default behavior with the -g command option. An example of creating a new group is shown in Listing 10.20.

Listing 10.20: Using the groupadd utility to create a new group

```
$ sudo groupadd -g 1042 Project42
$
$ getent group Project42
Project42:x:1042:
$
$ grep Project42 /etc/group
Project42:x:1042:
$
```

Notice in Listing 10.20 that super user privileges are required to create a new group. The getent utility, as well as the grep command, is used to show the new group record in the

/etc/group file. The fields in the /etc/group file are delimited by a colon (:) and are as follows:

- Group name
- Group password: An x indicates that if a group password exists, it is stored in the /etc/gshadow file.
- GID
- Group members: User accounts that belong to the group, separated by a comma.

The Ubuntu and Debian distributions promote the use of the addgroup program instead of the groupadd program. They consider the groupadd command to be a low-level utility.

The new group created did not have a group password created for it. However, the x in the Project42 group record within the /etc/group file does not prove this. To make sure there is no group password, the /etc/gshadow file, where group passwords are stored, is checked in Listing 10.21.

Listing 10.21: Checking for a group password

```
$ sudo getent gshadow Project42
Project42:!::
$
```

The command in Listing 10.21 shows the Project42 group's record within the /etc/gshadow file. The second field contains an exclamation point (!), which indicates that no password has been set for this group.

Group passwords, if set, allow user accounts access to groups to whom they do not belong. If a group password is used, this password is typically shared among the various users who need access to the group. This is a bad security practice. Passwords should never be shared. Each account needs to have its own password, and access to groups should only be allowed via group membership, not group passwords.

Once a new group is created, you can set group membership, which is simply adding user accounts to the group. Listing 10.22 shows an example of doing this with the usermod command.

Listing 10.22: Employing usermod to add an account to a group

```
$ sudo groups DAdams
DAdams : DAdams
$
```

```
$ sudo usermod -aG Project42 DAdams
$
$ sudo groups DAdams
DAdams : DAdams Project42
$
$ getent group Project42
Project42:x:1042:DAdams
$
```

Notice that the `usermod` command in Listing 10.22 uses two options, `-aG`. The `-G` adds the `DAdams` account as a member of the `Project42` group, but the `-a` switch is important because it preserves any previous `DAdams` account group memberships. After the `DAdams` account is added as a `Project42` group member, you can see in the last two command results that the `/etc/group` file record for `Project42` was updated.

If you need to modify a particular group, the `groupmod` command is helpful. A group's GID is modified with the `-g` option, while a group's name is modified with the `-n` switch. In Listing 10.23, the `Project42` group's GID is modified.

Listing 10.23: Using `groupmod` to modify a group

```
$ getent group Project42
Project42:x:1042:DAdams
$
$ sudo groupmod -g 1138 Project42
$
$ getent group Project42
Project42:x:1138:DAdams
$
```

Notice that in Listing 10.23, the `Project42` group's GID is modified to `1138`. The `getent` command confirms that the `/etc/group` file was updated. If the `1138` GID was already in use by another group, the `groupmod` command would have displayed an error message and not changed the group's GID.

To remove a group, the `groupdel` utility is employed. An example is shown in Listing 10.24.

Listing 10.24: Using `groupdel` to delete a group

```
$ sudo groupdel Project42
$
$ getent group Project42
$
```

```
$ sudo groups DAdams
DAdams : DAdams
$
$ sudo find / -gid 1138 2>/dev/null
$
```

Notice in Listing 10.24 after the `Project42` group is deleted that the `getent` command shows that the `Project42` group record has been removed from the `/etc/group` file. What is really nice is that any member of that deleted group has also had their group information updated, as shown in the third command.

Once you have removed a group, it is important to search through the virtual directory system for any files that may have access settings for that group. You can do this audit using the `find` command and the deleted group's GID. An example of this task is shown as the fourth command. If you need help remembering how to use the `find` utility, go back to Chapter 3, "Managing Files, Directories, and Text," where the command was originally covered.

Besides adding, modifying, and deleting user accounts and groups, there are a few other critical tasks involved with administering them. These topics are covered in the next few sections.

Setting Up the Environment

After a user authenticates with the Linux system and before reaching the Bash shell's command-line prompt, the user environment is configured. This environment consists of environment variables, command aliases, and various other settings. For example, the `PATH` environment variable, covered in Chapter 3, is often manipulated in configuring the user's environment.

The user environment configuration is accomplished via environment files. These files contain Bash shell commands to perform the necessary operations and are covered in the following sections along with a few environment variable highlights.

Perusing Bash Parameters

The Bash shell uses a feature called *environment variables* to store information about the shell session and the working environment (thus the name *environment variable*). You can view all the various environment variables set on your system by using the `set`, `env`, and `printenv` commands. However, for the user environment purposes, we're going to focus on only a *few* of these variables. These are listed in Table 10.8.

TABLE 10.8 A few environment variables associated with a user environment

Name	Description
HISTCONTROL	Governs how commands are saved within the history list
HISTSIZE	Controls the maximum number of commands saved within the history list
PATH	Sets the directories in which the shell searches for command programs
PS1	Configures the shell's primary prompt
SHELL	Sets the shell program's absolute directory reference
USER	Contains the current process's user account name

Typically environment variable names are all uppercase. However, you will sometimes see them written with a preceding dollar sign ($), such as $PS1. This is because you can use or display what is stored in the environment variable by adding the $. For example, `echo $HISTSIZE` will display the history list's maximum number of saved commands.

While you can modify these variables on the fly, the focus here is on how these parameters are persistently set or modified for user login processes. When you start a Bash shell by logging in to the Linux system, by default Bash checks several files for the configuration. These files are called *environment files*, which are sometimes called *startup files*. The environment files that Bash processes depend on the method you use to start the Bash shell. You can start a Bash shell in three ways:

- As a default login shell, such as when logging into the system at a tty# terminal
- As an interactive shell that is started by spawning a subshell, such as when opening a terminal emulator in a Linux GUI
- As a noninteractive shell (also called *non-login shell*) that is started, such as when running a shell script

The environment files are actually shell scripts. Shell scripting is covered more thoroughly in Chapter 25, "Deploying Bash Scripts." The following sections take you through the various available environment files.

Understanding User Entries

There are four potential files found in the user's home directory, $HOME, that are environment files. For a default login or interactive shell, the first file found in the following order is run, and the rest are ignored:

- `.bash_profile`

- .bash_login
- .profile

Typically, the fourth file, .bashrc, is run from the file found in the preceding list. However, any time a noninteractive shell is started, the .bashrc file is run.

In Listing 10.25, on a Rocky Linux distribution, the user's directory is checked for all four environment files. Notice that two of them are not found. Therefore, only the .bash_profile and .bashrc files are employed on this system.

Listing 10.25: Reviewing a user account's environment files

```
$ pwd
/home/Christine
$ ls .bash_profile .bash_login .profile .bashrc
ls: cannot access '.bash_login': No such file or directory
ls: cannot access '.profile': No such file or directory
.bash_profile  .bashrc
$
```

When referring to a user account's environment files, often symbols for environment files denoting the user's home directory are employed. For example, you may see the .bashrc environment file referred to as the $HOME/.bashrc or the ~/.bashrc file.

If you want to modify your shell's primary prompt ($PS1) persistently, you can do so by adding the modification to one of your local environment configuration files. Listing 10.26 shows the PS1 variable's modification on a Fedora Workstation distribution.

Listing 10.26: Persistentally setting a user account's shell prompt

```
$ grep PS1 .bash_profile
PS1="To infinity and beyond: "
$
```

Notice in Listing 10.26 that the user's prompt is still set to a $. The new prompt designated in the $HOME/.bash_profile file will not take effect until the file is run, which can be done manually or automatically when the user logs out and back into the shell.

These individual user environment files are typically populated from the /etc/skel directory, depending on your account creation configuration settings. For future accounts, you can make changes to the skeleton environment files. Just keep in mind that any individual user who can access the command line has the ability to modify their own files. Thus, for environment configurations that need to apply to all users, it is better to make a global entry in one of the global environment files, which are covered next.

Grasping Global Entries

Global configuration files modify the working environment and shell sessions for all users starting a Bash shell. As mentioned earlier, the global entries in these files can be modified by the account user by adding user entries to their `$HOME` environment files.

The global environment files consist of the following:

- The `/etc/profile` file
- Files within the `/etc/profile.d/` directory
- The `/etc/bashrc` or the `/etc/bash.bashrc` file

Whether your Linux system has the `/etc/bashrc` or the `/etc/bash.bashrc` file depends on which distribution you are running. Either file is typically called from the user's `$HOME/.bashrc` file.

It is recommended that instead of changing the `/etc/profile` or other files for global environment needs, you create a custom environment file, give it an `.sh` file extension, and place it in the `/etc/profile.d/` directory. All the `.sh` files within the `/etc/profile.d/` directory are run via the `/etc/profile` environment file for logins to the Bash shell.

Now you know how to set up persistent changes to user environments both locally and globally. Next, we'll explore keeping an eye on those users.

Querying Users

Several utilities allow you to audit which users are currently accessing the system as well as users who have accessed it in the past. You can also verify the account name you are using at present and review various information concerning user accounts.

Exploring the *whoami* Utility

The `whoami` command will display what user account you are currently using. While this may seem silly, it is important to know what account you are currently using, especially if your organization has shared accounts or the distribution you're using allows interactive logins into the root account.

It is considered a bad security practice to share user accounts as well as log directly into the root user account. If you need super user account access, it is better to obtain `sudo` privileges. The `sudo` command is covered thoroughly in Chapter 15, "Applying Ownership and Permissions."

The `whoami` command is demonstrated in Listing 10.27. Notice that it only displays the current user account's name.

Listing 10.27: Employing the `whoami` utility

```
$ whoami
Christine
$
```

Understanding the *who* Utility

The `who` command provides a little more data than the `whoami` utility. You can view information concerning your own account or look at every current user on the system. Examples are shown in Listing 10.28.

Listing 10.28: Using the `who` command

```
$ who
user1    tty2         2018-10-03 13:12
Christine pts/0        2018-10-03 14:10 (192.168.0.102)
$
```

Notice in Listing 10.28, when the `who` command is used by itself, it shows all the current system users, the terminal they are using, the date and time they entered the system, and in cases of remote users, their remote IP address.

Though it is a very short command, `w` provides a great deal of useful information. An example is shown in Listing 10.29.

Listing 10.29: Employing the `w` command

```
$ w
 09:58:31 up 49 min,  5 users,  load average: 0.81, 0.37, 0.27
USER     TTY      LOGIN@   IDLE   JCPU   PCPU WHAT
user1    tty2     09:10   49:11  43.89s  0.30s /usr/libexe[...]
Christin pts/1    09:14    2.00s  0.04s  0.01s w
Rich     tty3     09:56    1:35   0.85s  0.81s top
Kevin    tty4     09:57    1:03  16.17s 16.14s ls --color=[...]
Tim      tty5     09:57   38.00s  0.08s  0.03s nano data42[...]
$
```

Notice the `w` command's verbose output in Listing 10.29. The first displayed line shows the following information:

- The current time
- How long the system has been up
- How many users are currently accessing the system
- The CPU load averages for the last 1, 5, and 15 minutes

The next several lines concern current system user information. The columns are as follows:

- `USER`: The account's name
- `TTY`: The account's currently used terminal
- `LOGIN@`: When the user logged into the account
- `IDLE`: How long it has been since the user interacted with the system
- `JCPU`: How much total CPU time the account has used
- `PCPU`: How much CPU time the account's current command (process) has used
- `WHAT`: What command the account is currently running

The `w` utility pulls user information from the `/var/run/utmp` file. It also gathers additional data for display from the `/proc/` directory files.

Identifying with the *id* Program

The `id` utility allows you to pull out various data about the current user process. It also displays information for any account whose identification you pass to `id` as an argument. The `id` command provides a nice one-line summary, as shown in Listing 10.30.

Listing 10.30: Employing the `id` command

```
$ id DAdams
uid=1002(DAdams) gid=1002(DAdams) groups=1002(DAdams)
$
$ id -un 1004
Kevin
$
```

If you don't want all that information the first command provides in Listing 10.30, you can filter the results by employing various `id` utility options, as shown in the second command in Listing 10.30. A few of the more common ones are shown in Table 10.9.

TABLE 10.9 The `id` command's commonly used options

Short	Long	Description
`-g`	`--group`	Displays the account's current group's GID, which is either the account's default group or a group reached by using the `newgrp` command
`-G`	`--groups`	Displays all the account's group memberships via each one's GIDs
`-n`	`--name`	Displays the account's name instead of UID or group name instead of GID by using this switch with the `-g`, `-G`, or `-u` options
`-u`	`--user`	Displays the account's UID

The `id` utility is very useful in shell scripts. In snipped Listing 10.31, you can see how it is used to set the `USER` environment variable in the `/etc/profile` file on a Rocky Linux system.

Listing 10.31: Using the `id` utility within an environment file

```
$ grep USER /etc/profile
    USER="`/usr/bin/id -un`"
[...]
$
```

Displaying Access History with the *last* Utility

The `last` command pulls information from the `/var/log/wtmp` file and displays a list of accounts showing the last time they logged in/out of the system or if they are still logged on. It also shows when system reboots occur and when the `wtmp` file was started. A snipped example is shown in Listing 10.32.

Listing 10.32: Using the `last` command

```
$ last
Tim      tty5                          Thu Oct  4 09:57   still logged in
Kevin    tty4                          Thu Oct  4 09:57   still logged in
Rich     tty3                          Thu Oct  4 09:56   still logged in
Christin pts/1        192.168.0.102    Thu Oct  4 09:14   still logged in
user1    tty2         tty2             Thu Oct  4 09:10   still logged in
reboot   system boot  4.17.12-200.fc28 Thu Oct  4 09:09   still running
Christin pts/0        192.168.0.102    Wed Oct  3 14:10 - 15:32  (01:22)
user1    tty2                          Wed Oct  3 13:12 - 15:33  (02:21)
[...]
wtmp begins Thu Jul 26 16:30:32 2018
$
```

Be aware that the `/var/log/wtmp` file typically gets automatically rotated via the `cron` utility, which is covered in Chapter 26, "Automating Jobs." If you need to gather information from old `wtmp` files, you can use the `-f` switch. For example, you could type **`last -f /var/log/wtmp.1`** to view data from the `/var/log/wtmp.1` file.

The `last` command and the various other utilities covered in these sections are helpful for auditing current users and checking your own account's identity. They are more tools for your Linux administration tool belt.

Managing Disk Space Usage

One way to prevent a filesystem from filling up with files and causing program or entire system issues is to set limits on users' disk space usage. This is accomplished with quotas. Linux can put a cap on the number of files a user may create as well as restrict the total filesystem space consumed by a single user. Not only are these limits available for user accounts, but they also may be set for groups.

The Linux system implements file number quota limits via their inode numbers. Back in Chapter 3 we mentioned file inode (index) numbers. Typically there is one inode number per file, unless a file is hard-linked.

There are essentially four steps for enabling quotas on a particular filesystem. You will need to employ super user privileges to accomplish these steps. They are as follows:

1. Modify the `/etc/fstab` file to enable filesystem quota support.
2. If the filesystem is already mounted, unmount and remount it. If the filesystem was not previously mounted, then just mount it.
3. Create the quota files.
4. Establish user or group quota limits and grace periods.

The necessary `/etc/fstab` file modification is fairly simple. You just edit the file and add either `usrquota` or `grpquota` or both to the filesystem's mount options (fourth field). An example is shown in Listing 10.33.

Listing 10.33: Setting filesystem quotas in the `/etc/fstab` file

```
$ grep /dev/sdb1 /etc/fstab
/dev/sdb1 /home/user1/QuotaFSTest ext4 defaults,usrquota,grpquota 0 0
$
```

Once you have the `/etc/fstab` file modified, if the filesystem is already mounted, you will need to unmount it using the `umount` command. You then mount or remount the system, using the `mount -a` command, which will mount any unmounted filesystems listed in the `/etc/fstab` file. An example is shown in Listing 10.34.

Listing 10.34: Mounting or remounting a quota-enabled filesystem

```
# umount /dev/sdb1
# mount -a
# mount | grep /dev/sdb1
/dev/sdb1 on /home/user1/QuotaFSTest type ext4 (rw,relatime,seclabel,quota,usr
quota,grpquota,data=ordered)
#
```

Notice in Listing 10.34 that you can check if the mount was successful by using the `mount` command and the `grep` utility. Also note that the mounted filesystem has both `usrquota` (user quotas) and `grpquota` (group quotas) enabled.

Once the filesystem has been properly mounted with quota support enabled, you can create the quota files needed to enforce limits. This is done with the `quotacheck` utility. The `-c` switch creates the needed files through a scan of the filesystem, recording any current quota usage. The `-u` option creates the `aquota.user` file, and the `-g` option creates the `aquota.group` file. Therefore, if you are only implementing user and not group quotas, you could leave off the `-g` switch, and vice versa. An example of using `quotacheck` is shown in Listing 10.35.

Listing 10.35: Using `quotacheck` to create user and group quota files

```
# quotacheck -cug /home/user1/QuotaFSTest
#
# ls /home/user1/QuotaFSTest
aquota.group  aquota.user  lost+found
#
```

If you are setting up quotas on more than one filesystem, you can issue the `quotacheck` command one time. Just employ the `-a` option along with the other command switches and it will create the desired quota files for any quota-enabled filesystems designated as currently mounted in the `/etc/mtab` file.

With the quota files created, you can start creating quota limits for user accounts and/or groups by employing the `edquota` utility. To edit user quotas, use the `-u` option (which is the default), and to edit group quotas, use the `-g` switch. A snipped example of editing a user account's quota is shown in Listing 10.36.

Listing 10.36: Employing `edquota` to create user and group quota files

```
# edquota -u user1
Disk quotas for user user1 (uid 1000):
Filesystem    blocks    soft    hard  inodes   soft   hard
/dev/sdb1        212    4096    6144       2      0      0
~
[...]
```

When you enter the `edquota` command, you are put into the `vim` (`vi`) editor for the quota file, unless you have set the `$EDITOR` environment variable to point to another text editor. In the quota file, there are two preset items that you cannot permanently modify: `blocks` (blocks used) and `inodes` (number of current files). That is because this information was obtained when the `quotacheck` command was previously run and it is not set via the `edquota` utility.

You can, however, modify the `soft` and `hard` limits for both `blocks` and `inodes`. When you set a hard block limit, you are setting the maximum number of blocks the user can fill with data. When you set inode hard limits, you are setting the total number of files that the user can create. Once the user hits either of these limits, no more disk space or file creation is available for that account on this particular filesystem. Note that if set to `0`, the limit is disabled. Notice in Listing 10.36 that inode limits are disabled.

Soft limits are a little nicer. Once the user hits a set soft limit, they can go for an extended period past this limit. It is called a *grace period*.

Once you have user (or group) quotas modified, you need to establish the grace period for any soft limits set. To do this, you use the `edquota -t` command. These grace periods are used for all users and groups. An example is shown in Listing 10.37.

Listing 10.37: Employing `edquota -t` to set soft limit grace periods

```
# edquota -t
Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
  Filesystem     Block grace period     Inode grace period
  /dev/sdb1             7days                  7days
~
```

When you issue the `edquota -t` command, you are again thrown into the `vim` editor, unless you have modified the `$EDITOR` environment variable. Grace periods can be set for both blocks and inodes and can be a matter of days, hours, minutes, or even seconds, which doesn't seem very graceful.

If you have some sort of odd problem when enabling filesystem quotas, you can quickly turn them off with the `quotaoff` command, using super user privileges. The –a option will allow you to turn them off for all the system's quota-enabled filesystems. You will need to specify user quotas (–u) and/or group quotas (–g) in the command. Once you have fixed the issues, turn filesystem quotas back on using the `quotaon` command.

When you have modified a user's quota limits and set grace periods, it's a good idea to double-check your modifications. The `quota` command can help here. An example is shown in Listing 10.38.

Listing 10.38: Using `quota` to check a user's quota limits

```
# quota -u user1
Disk quotas for user user1 (uid 1000):
Filesystem  blocks  quota  limit  grace  files  quota  limit  grace
/dev/sdb1     212   4096   6144             2      0      0
#
```

Notice in Listing 10.38 that no information is listed for the user account in the `grace` column. This means that the user has *not* gone over a soft limit and is *not* currently in a grace period.

After all that work, you should do another check. You can audit all your filesystems employing quota limits with the `repquota` command. An example is shown in Listing 10.39.

Listing 10.39: Using `repquota` to check all the filesystems' quotas

```
# repquota -a
*** Report for user quotas on device /dev/sdb1
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --      12       0       0              1     0     0
user1     --     212    4096    6144              2     0     0
#
```

This should keep your filesystems humming along. However, be aware that it is a good idea to set up a periodic check of your filesystems' quotas using the `quotacheck` utility. You can automate this by setting up a `cron` job to do so, as covered in Chapter 26.

Summary

Managing the user account and group memberships for a Linux system involves many critical pieces. You need to understand the account creation process as well as the files used. You must grasp the entire mechanism for times when troubleshooting authentication issues for a particular user is necessary. In addition, being able to use various utilities for identifying various users can assist.

User accounts may be gathered together into various groups, which provide additional access. These group memberships are part of the authorization in which users can gain entry into various files and directories. Knowing the key areas of group administration is critical for proper Linux system management.

Besides protecting your system through properly authenticated and authorization user and group mechanisms, you can also shield your filesystems from overuse. In particular, setting up filesystem user and group quotas will provide an additional layer of protection.

Exam Essentials

Describe the players in managing user accounts. The `/etc/login.defs` and `/etc/default/useradd` files configure various settings for the `useradd` command's default behavior. Because the directive settings within these files vary from distribution to distribution, it is wise to peruse them prior to employing the `useradd` utility to create accounts. When an account is created, the `/etc/passwd`, `/etc/shadow`, and `/etc/group` files are all modified. Depending on the user account creation configuration, a user home directory may be created and files copied to it from the `/etc/skel` directory.

Summarize managing groups. The commands involved in creating, modifying, and deleting groups are the `groupadd`, `groupmod`, and `groupdel` commands. These commands cause modifications to the `/etc/group` file. If you need to add a user to a group, you need to employ the `usermod` utility. A user can easily switch from the account's default group to another group in which the account is a member by using the `newgrp` program. Account group membership can be audited via the `groups` and `getent` commands as well as by viewing the `/etc/group` file.

Outline the environment files. The Bash shell uses environment variables to store information about the shell session and the working environment. These variables are set using environment files. Which environment files are run depends on how a user is logging into a system as well as the distribution the account is on. User environment files are hidden files in that they begin with a dot (.) and are potentially the `.bash_profile`, `.bash_login`, `.profile`, and `.bashrc` files. Global files may include `/etc/bashrc`, `/etc/bash.bashrc`, `/etc/profile`, and files within the `/etc/profile.d/` directory.

Explain the various methods to query user account information. There are several utilities you can employ to determine user account information for users who are currently logged into their accounts as well as those who are not. The "who" commands have three variations, which are the `whoami`, `who`, and `w` utilities. The `id` program is useful for matching UID and GID numbers to particular user accounts. The `last` command is helpful for viewing not only when a system has rebooted but also whether or not a user is currently logged into the system or when the last time the account was accessed.

Describe how to manage filesystem usage quotas. Prior to setting user account or group quota limits on a system, you must enable quotas on the filesystem using the `usrquota` and `grpquota` options in the `/etc/fstab` file. Once the filesystem is unmounted and then remounted, you can create the needed user and/or group files with the `quotacheck` utility. After that is accomplished, user or group limits are set with the `edquota` command. You can also view and/or verify quotas using the `repquota` program.

Review Questions

1. Which of the following are fields within an `/etc/passwd` file record? (Choose all that apply.)
   - **A.** User account's username
   - **B.** Password
   - **C.** Password change date
   - **D.** Special flag
   - **E.** UID
2. Which of the following are fields in an `/etc/shadow` file record? (Choose all that apply.)
   - **A.** Password expiration date
   - **B.** Account expiration date
   - **C.** Password
   - **D.** Comment
   - **E.** Default shell
3. Which field contains the same data for both an `/etc/passwd` and an `/etc/shadow` file record?
   - **A.** Password
   - **B.** Account expiration date
   - **C.** UID
   - **D.** GID
   - **E.** User account's username
4. Which of the following commands will allow you to view the `NUhura` account's record data in the `/etc/passwd` file? (Choose all that apply.)
   - **A.** `getent NUhura passwd`
   - **B.** `cat /etc/passwd`
   - **C.** `passwd NUhura`
   - **D.** `grep NUhura /etc/passwd`
   - **E.** `getent passwd NUhura`
5. You use the `useradd -D` command to view account creation configuration directives. What file does this command pull its displayed information from?
   - **A.** The `/etc/passwd` file
   - **B.** The `/etc/shadow` file
   - **C.** The `/etc/group` file
   - **D.** The `/etc/login.defs` file
   - **E.** The `/etc/default/useradd` file

6. You create an account using the appropriate utility, except for some reason the account's home directory was not created. Which of the following most likely caused this to occur?
   A. The `HOME` directive is set to no.
   B. You did not employ super user privileges.
   C. The `CREATE_HOME` directive is not set.
   D. The `INACTIVE` directive is set to `-1`.
   E. The `EXPIRE` date is set and it is before today.

7. Your boss has asked you to remove `KSingh`'s account and all his home directory files from the system immediately. Which command should you use?
   A. `usermod -r KSingh`
   B. `rm -r /home/KSingh`
   C. `userdel Ksingh`
   D. `userdel -r KSingh`
   E. `usermod -d KSingh`

8. Which of the following will allow you to change an account's `/etc/shadow` file record data? (Choose all that apply.)
   A. The `passwd` command
   B. The `usermod` command
   C. The `userdel` command
   D. The `getent` command
   E. The `chage` command

9. Which of the following commands will allow you to switch temporarily from your account's default group to another group you are a member of?
   A. The `usermod` command
   B. The `newgrp` command
   C. The `groups` command
   D. The `groupadd` command
   E. The `groupmod` command

10. Which of the following commands is the best one to add `JKirk` as a member to a new group called the `NCC-1701` group and not remove any of the account's previous group memberships?
    A. `usermod -g NCC-1701 JKirk`
    B. `usermod -G NCC-1701 JKirk`
    C. `usermod -aG NCC-1701 JKirk`
    D. `groupadd NCC-1701`
    E. `groupmod NCC-1701 JKirk`

11. Which of the following commands could be used to view the members of the `NCC-1701` group? (Choose all that apply.)
    A. `groups NCC-1701`
    B. `getent group NCC-1701`
    C. `getent groups NCC-1701`
    D. `grep NCC-1701 /etc/group`
    E. `grep NCC-1701 /etc/groups`

12. User environment files typically come from where?
    A. `/etc/skel`
    B. `/home/`*`userid`*
    C. `$HOME`
    D. `~`
    E. `/etc/`

13. A user has logged into the tty3 terminal. Which of the following user environment files is executed first if found in the user's home directory?
    A. The `.bash_login` file
    B. The `.bashrc` file
    C. The `.profile` file
    D. The `.bash.bashrc` file
    E. The `.bash_profile` file

14. Which of the following files and directories may be involved in setting up the environment for all system users? (Choose all that apply.)
    A. `/etc/bash_profile/`
    B. `/etc/profile`
    C. `/etc/profile.d/`
    D. `/etc/bashrc`
    E. `/etc/bash.bashrc`

15. Which of the following commands displays information about the account issuing the command? (Choose all that apply.)
    A. `whoami`
    B. `who am i`
    C. `cat $HOME/.bashrc`
    D. `cat $HOME/.profile`
    E. `id`

16. Which of the following commands will display CPU load data along with information concerning users who are currently logged into the system?
    A. The `who` command
    B. The `id` command
    C. The `whoami` command
    D. The `w` command
    E. The `last` command

17. The `last` command, by default, pulls its data from what file?
    A. The `/var/run/utmp` file
    B. The `/var/log/wtmp` file
    C. The `/var/log/wtmp.1` file
    D. The `/etc/shadow` file
    E. The `/etc/passwd` file

18. Which of the following are options used in the `/etc/fstab` file to designate a filesystem as one that uses quotas? (Choose all that apply.)
    A. `usrquota`
    B. `quotaon`
    C. `grpquota`
    D. `quotacheck`
    E. `aquota.user`

19. A problem has occurred concerning group quotas on three filesystems. You need to quickly remove all filesystems' quota limits to temporarily resolve this issue. What is the best command to employ?
    A. `vi /etc/fstab`
    B. `quotaoff -a`
    C. `quotacheck -cg`
    D. `quotacheck -cu`
    E. `umount`

20. You need to edit quota grace periods. Which of the following commands should you use?
    A. `edquota -u`
    B. `edquota -g`
    C. `edquota -t`
    D. `edquota -G`
    E. `edquota --grace`