Chapter 6. Managing Local Users and Groups

[**Describing User and Group Concepts**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Quiz: Describing User and Group Concepts**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s02/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Gaining Superuser Access**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s03/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Guided Exercise: Gaining Superuser Access**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s04/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Managing Local User Accounts**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s05/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Guided Exercise: Managing Local User Accounts**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s06/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Managing Local Group Accounts**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s07/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Guided Exercise: Managing Local Group Accounts**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s08/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Managing User Passwords**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s09/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Guided Exercise: Managing User Passwords**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s10/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Lab: Managing Local Users and Groups**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s11/9a2ef70f-4e72-42df-a498-b694b274af27)

[**Summary**](https://rha.ole.redhat.com/rha/app/courses/rh124-8.2/pages/ch06s12/9a2ef70f-4e72-42df-a498-b694b274af27)

**Abstract**

|  |  |
| --- | --- |
| **Goal** | Create, manage, and delete local users and groups and administer local password policies. |
| **Objectives** | * Describe the purpose of users and groups on a Linux system. * Switch to the superuser account to manage a Linux system, and grant other users superuser access using the **sudo** command. * Create, modify, and delete locally defined user accounts. * Create, modify, and delete locally defined group accounts. * Set a password management policy for users, and manually lock and unlock user accounts. |
| **Sections** | * Describing Users and Groups Concepts (and Quiz) * Gaining Superuser Access (and Guided Exercise) * Managing Local User Accounts (and Guided Exercise) * Managing Local Group Accounts (and Guided Exercise) * Managing User Passwords (and Guided Exercise) |
| **Lab** | Managing Local Linux Users and Groups |

Describing User and Group Concepts

Objectives

After completing this section, you should be able to describe the purpose of users and groups on a Linux system.

What is a User?

A *user* account is used to provide security boundaries between different people and programs that can run commands.

Users have *user names* to identify them to human users and make them easier to work with. Internally, the system distinguishes user accounts by the unique identification number assigned to them, the *user ID* or *UID*. If a user account is used by humans, it will generally be assigned a secret *password* that the user will use to prove that they are the actual authorized user when logging in.

User accounts are fundamental to system security. Every process (running program) on the system runs as a particular user. Every file has a particular user as its owner. File ownership helps the system enforce access control for users of the files. The user associated with a running process determines the files and directories accessible to that process.

There are three main types of user account: the *superuser*, *system users*, and *regular users*.

* The *superuser* account is for administration of the system. The name of the superuser is root and the account has UID 0. The superuser has full access to the system.
* The system has *system user* accounts which are used by processes that provide supporting services. These processes, or *daemons*, usually do not need to run as the superuser. They are assiged non-privileged accounts that allow them to secure their files and other resources from each other and from regular users on the system. Users do not interactively log in using a system user account.
* Most users have *regular user* accounts which they use for their day-to-day work. Like system users, regular users have limited access to the system.

You can use the **id** command to show information about the currently logged-in user.

**[user01@host ~]$ id**

uid=1000(user01) gid=1000(user01) groups=1000(user01) context=unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

To view basic information about another user, pass the username to the **id** command as an argument.

**[user01@host]$ id *user02***

uid=1002(user02) gid=1001(user02) groups=1001(user02) context=unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

To view the owner of a file use the **ls -l** command. To view the owner of a directory use the **ls -ld** command. In the following output, the third column shows the username.

**[user01@host ~]$ ls -l *file1***

-rw-rw-r--. 1 **user01** user01 0 Feb 5 11:10 file1

**[user01@host]$ ls -ld *dir1***

drwxrwxr-x. 2 **user01** user01 6 Feb 5 11:10 dir1

To view process information, use the **ps** command. The default is to show only processes in the current shell. Add the **a** option to view all processes with a terminal. To view the user associated with a process, include the u option. In the following output, the first column shows the username.

**[user01@host]$ ps -au**

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

**root** 777 0.0 0.0 225752 1496 tty1 Ss+ 11:03 0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux

**root** 780 0.0 0.1 225392 2064 ttyS0 Ss+ 11:03 0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600

**user01** 1207 0.0 0.2 234044 5104 pts/0 Ss 11:09 0:00 -bash

**user01** 1319 0.0 0.2 266904 3876 pts/0 R+ 11:33 0:00 ps au

The output of the preceding command displays users by name, but internally the operating system uses the UIDs to track users. The mapping of usernames to UIDs is defined in databases of account information. By default, systems use the /etc/passwd file to store information about local users.

Each line in the /etc/passwd file contains information about one user. It is divided up into seven colon-separated fields. Here is an example of a line from /etc/passwd:

user01:x:1000:1000:User One:/home/user01:/bin/bash

|  |  |
| --- | --- |
|  | * Username for this user (user01). |
|  | * The user's password used to be stored here in encrypted format. That has been moved to the /etc/shadow file, which will be covered later. This field should always be x. |
|  | * The UID number for this user account (1000). |
|  | * The GID number for this user account's primary group (1000). Groups will be discussed later in this section. |
|  | * The real name for this user (User One). |
|  | * The home directory for this user (/home/user01). This is the initial working directory when the shell starts and contains the user's data and configuration settings. |
|  | * The default shell program for this user, which runs on login (**/bin/bash**). For a regular user, this is normally the program that provides the user's command-line prompt. A system user might use **/sbin/nologin** if interactive logins are not allowed for that user. |

What is a Group?

A group is a collection of users that need to share access to files and other system resources. Groups can be used to grant access to files to a set of users instead of just a single user.

Like users, groups have *group names* to make them easier to work with. Internally, the system distinguishes groups by the unique identification number assigned to them, the *group ID* or *GID*.

The mapping of group names to GIDs is defined in databases of group account information. By default, systems use the /etc/group file to store information about local groups.

Each line in the /etc/group file contains information about one group. Each group entry is divided into four colon-separated fields. Here is an example of a line from /etc/group:

group01:x:10000:user01,user02,user03

|  |  |
| --- | --- |
|  | Group name for this group (group01). |
|  | Obsolete group password field. This field should always be x. |
|  | The GID number for this group (10000). |
|  | A list of users who are members of this group as a supplementary group (user01, user02, user03). Primary (or default) and supplementary groups are discussed later in this section. |

**Primary Groups and Supplementary Groups**

Every user has exactly one primary group. For local users, this is the group listed by GID number in the /etc/passwd file. By default, this is the group that will own new files created by the user.

Normally, when you create a new regular user, a new group with the same name as that user is created. That group is used as the primary group for the new user, and that user is the only member of this *User Private Group*. It turns out that this helps make management of file permissions simpler, which will be discussed later in this course.

Users may also have *supplementary groups*. Membership in supplementary groups is determined by the /etc/group file. Users are granted access to files based on whether any of their groups have access. It doesn't matter if the group or groups that have access are primary or supplementary for the user.

For example, if the user user01 has a primary group user01 and supplementary groups wheel and webadmin, then that user can read files readable by any of those three groups.

The **id** command can also be used to find out about group membership for a user.

**[user03@host ~]$ id**

uid=1003(user03) gid=1003(user03) groups=1003(user03),10(wheel),10000(group01) context=unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

In the preceding example, user03 has the group user03 as their primary group (gid). The groups item lists all groups for this user, and other than the primary group user03, the user has groups wheel and group01 as supplementary groups.

References

**id**(1), **passwd**(5), and **group**(5) man pages

**info libc** (*GNU C Library Reference Manual*)

* Section 30: Users and groups

(Note that the glibc-devel package must be installed for this info node to be available.)

**Quiz: Describing User and Group Concepts**

Choose the correct answer to the following questions:

1. Which item represents a number that Identifies the user at the most fundamental level?

* primary user
* UID
* GID
* username

2. Which item represents the program that provides the user's command-line prompt?

* primary shell
* home directory
* login shell
* command name

3. Which item or file represents the location of the local group information?

* home directory
* /etc/passwd
* /etc/GID
* /etc/group

4. Which item or file represents the location of the user's personal files?

* home directory
* login shell
* /etc/passwd
* /etc/group

5. Which item represents a number that identifies the group at the most fundamental level?

* primary group
* UID
* GID
* groupid

6. Which item or file represents the location of the local user account information?

* home directory
* /etc/passwd
* /etc/UID
* /etc/group

7. What is the fourth field of the /etc/passwd file?

* home directory
* UID
* login shell
* primary group

Gaining Superuser Access

Objectives

After completing this section, you will be able to switch to the superuser account to manage a Linux system, and grant other users superuser access through the **sudo** command.

The Superuser

Most operating systems have some sort of *superuser*, a user that has all power over the system. In Red Hat Enterprise Linux this is the root user. This user has the power to override normal privileges on the file system, and is used to manage and administer the system. To perform tasks such as installing or removing software and to manage system files and directories, users must escalate their privileges to the root user.

The root user only among normal users can control most devices, but there are a few exceptions. For example, normal users can control removable devices, such as USB devices. Thus, normal users can add and remove files and otherwise manage a removable device, but only root can manage "fixed" hard drives by default.

This unlimited privilege, however, comes with responsibility. The root user has unlimited power to damage the system: remove files and directories, remove user accounts, add back doors, and so on. If the root user's account is compromised, someone else would have administrative control of the system. Throughout this course, administrators are encouraged to log in as a normal user and escalate privileges to root only when needed.

The root account on Linux is roughly equivalent to the local Administrator account on Microsoft Windows. In Linux, most system administrators log in to the system as an unprivileged user and use various tools to temporarily gain root privileges.

Warning

One common practice on Microsoft Windows in the past was for the local Administrator user to log in directly to perform system administrator duties. Although this is possible on Linux, Red Hat recommends that system administrators do not log in directly as root. Instead, system administrators should log in as a normal user and use other mechanisms (**su**, **sudo**, or PolicyKit, for example) to temporarily gain superuser privileges.

By logging in as the superuser, the entire desktop environment unnecessarily runs with administrative privileges. In that situation, any security vulnerability which would normally only compromise the user account has the potential to compromise the entire system.

Switching Users

The **su** command allows users to switch to a different user account. If you run **su** from a regular user account, you will be prompted for the password of the account to which you want to switch. When root runs **su**, you do not need to enter the user's password.

**[user01@host ~]$ su - *user02***

**Password:**

**[user02@host ~]$**

If you omit the user name, the **su** or **su -** command attempts to switch to root by default.

**[user01@host ~]$ su -**

**Password:**

**[root@host ~]#**

The command **su** starts a *non-login shell*, while the command **su -** (with the dash option) starts a *login shell*. The main distinction between the two commands is that **su -** sets up the shell environment as if it were a new login as that user, while **su** just starts a shell as that user, but uses the original user's environment settings.

In most cases, administrators should run **su -** to get a shell with the target user's normal environment settings. For more information, see the **bash**(1) man page.

Note

The **su** command is most frequently used to get a command-line interface (shell prompt) which is running as another user, typically root. However, with the -c option, it can be used like the Windows utility **runas** to run an arbitrary program as another user. Run **info su** to view more details.

Running Commands with Sudo

In some cases, the root user's account may not have a valid password at all for security reasons. In this case, users cannot log in to the system as root directly with a password, and **su** cannot be used to get an interactive shell. One tool that can be used to get root access in this case is **sudo**.

Unlike **su**, **sudo** normally requires users to enter their own password for authentication, not the password of the user account they are trying to access. That is, users who use **sudo** to run commands as root do not need to know the root password. Instead, they use their own passwords to authenticate access.

Additionally, **sudo** can be configured to allow specific users to run any command as some other user, or only some commands as that user.

For example, when **sudo** is configured to allow the user01 user to run the command **usermod** as root, user01 could run the following command to lock or unlock a user account:

**[user01@host ~]$ sudo usermod -L user02**

**[sudo] password for user01:**

**[user01@host ~]$ su - user02**

**Password:**

su: Authentication failure

**[user01@host ~]$**

If a user tries to run a command as another user, and the **sudo** configuration does not permit it, the command will be blocked, the attempt will be logged, and by default an email will be sent to the root user.

**[user02@host ~]$ sudo tail /var/log/secure**

**[sudo] password for user02:**

user02 is not in the sudoers file. This incident will be reported.

**[user02@host ~]$**

One additional benefit to using **sudo** is that all commands executed are logged by default to /var/log/secure.

[user01@host ~]$ **sudo tail /var/log/secure**

*...output omitted...*

Feb 6 20:45:46 host sudo[2577]: user01 : TTY=pts/0 ; PWD=/home/user01 ; USER=root ; COMMAND=/sbin/usermod -L user02

*...output omitted...*

In Red Hat Enterprise Linux 7 and Red Hat Enterprise Linux 8, all members of the wheel group can use **sudo** to run commands as any user, including root. The user is prompted for their own password. This is a change from Red Hat Enterprise Linux 6 and earlier, where users who were members of the wheel group did not get this administrative access by default.

Warning

RHEL 6 did not grant the wheel group any special privileges by default. Sites that have been using this group for a non-standard purpose might be surprised when RHEL 7 and RHEL 8 automatically grants all members of wheel full **sudo** privileges. This could lead to unauthorized users getting administrative access to RHEL 7 and RHEL 8 systems.

Historically, UNIX-like systems use membership in the wheel group to grant or control superuser access.

**Getting an Interactive Root Shell with Sudo**

If there is a nonadministrative user account on the system that can use **sudo** to run the **su** command, you can run **sudo su -** from that account to get an interactive root user shell. This works because **sudo** will run **su -** as root, and root does not need to enter a password to use **su**.

Another way to access the root account with **sudo** is to use the **sudo -i** command. This will switch to the root account and run that user's default shell (usually **bash**) and associated shell login scripts. If you just want to run the shell, you can use the **sudo -s** command.

For example, an administrator might get an interactive shell as root on an AWS EC2 instance by using SSH public-key authentication to log in as the normal user ec2-user, and then by running **sudo -i** to get the root user's shell.

**[ec2-user@host ~]$ sudo -i**

**[sudo] password for ec2-user:**

**[root@host ~]#**

The **sudo su -** command and **sudo -i** do not behave exactly the same. This will be discussed briefly at the end of the section.

**Configuring Sudo**

The main configuration file for **sudo** is /etc/sudoers. To avoid problems if multiple administrators try to edit it at the same time, it should only be edited with the special **visudo** command.

For example, the following line from the /etc/sudoers file enables **sudo** access for members of group wheel.

%wheel ALL=(ALL) ALL

In this line, %wheel is the user or group to whom the rule applies. A % specifies that this is a group, group wheel. The ALL=(ALL) specifies that on any host that might have this file, wheel can run any command. The final ALL specifies that wheel can run those commands as any user on the system.

By default, /etc/sudoers also includes the contents of any files in the /etc/sudoers.d directory as part of the configuration file. This allows an administrator to add **sudo** access for a user simply by putting an appropriate file in that directory.

Note

Using supplementary files under the /etc/sudoers.d directory is convenient and simple. You can enable or disable **sudo** access simply by copying a file into the directory or removing it from the directory.

In this course, you will create and remove files in the /etc/sudoers.d directory to configure **sudo** access for users and groups.

To enable full **sudo** access for the user user01, you could create /etc/sudoers.d/user01 with the following content:

user01 ALL=(ALL) ALL

To enable full **sudo** access for the group group01, you could create /etc/sudoers.d/group01 with the following content:

%group01 ALL=(ALL) ALL

It is also possible to set up **sudo** to allow a user to run commands as another user without entering their password:

ansible ALL=(ALL) NOPASSWD:ALL

While there are obvious security risks to granting this level of access to a user or group, it is frequently used with cloud instances, virtual machines, and provisioning systems to help configure servers. The account with this access must be carefully protected and might require SSH public-key authentication in order for a user on a remote system to access it at all.

For example, the official AMI for Red Hat Enterprise Linux in the Amazon Web Services Marketplace ships with the root and the ec2-user users' passwords locked. The ec2-user user account is set up to allow remote interactive access through SSH public-key authentication. The user ec2-user can also run any command as root without a password because the last line of the AMI's /etc/sudoers file is set up as follows:

ec2-user ALL=(ALL) NOPASSWD: ALL

The requirement to enter a password for **sudo** can be re-enabled or other changes may be made to tighten security as part of the process of configuring the system.

Note

In this course, you may see **sudo su -** used instead of **sudo -i**. Both commands work, but there are some subtle differences between them.

The **sudo su -** command sets up the root environment exactly like a normal login because the **su -** command ignores the settings made by **sudo** and sets up the environment from scratch.

The default configuration of the **sudo -i** command actually sets up some details of the root user's environment differently than a normal login. For example, it sets the PATH environment variable slightly differently. This affects where the shell will look to find commands.

You can make **sudo -i** behave more like **su -** by editing /etc/sudoers with **visudo**. Find the line

Defaults secure\_path = /sbin:/bin:/usr/sbin:/usr/bin

and replace it with the following two lines:

Defaults secure\_path = /usr/local/bin:/usr/bin

Defaults>root secure\_path = /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin

For most purposes, this is not a major difference. However, for consistency of PATH settings on systems with the default /etc/sudoers file, the authors of this course use **sudo -i** in examples and exercises.

References

**su**(1), **sudo**(8), **visudo**(8) and **sudoers**(5) man pages

**info libc persona** (*GNU C Library Reference Manual*)

* Section 30.2: The Persona of a Process

(Note that the glibc-devel package must be installed for this info node to be available.)

**Guided Exercise: Gaining Superuser Access**

In this exercise, you will practice switching to the root account and running commands as root.

**Outcomes**

You should be able to:

* Use **sudo** to switch to root and access the interactive shell as root without knowing the password of the superuser.
* Explain how **su** and **su -** can affect the shell environment through running or not running the login scripts.
* Use **sudo** to run other commands as root.

Log in to workstation as student using student as the password.

On workstation, run **lab users-sudo start** to start the exercise. This script creates the necessary user accounts and files to set up the environment correctly.

**[student@workstation ~]$ lab users-sudo start**

1. From workstation, open an SSH session to servera as student.

**[student@workstation ~]$ ssh student@servera**

*...output omitted...*

**[student@servera ~]$**

1. Explore the shell environment of student. View the current user and group information and display the current working directory. Also view the environment variables that specify the user's home directory and the locations of the user's executables.
   1. Run **id** to view the current user and group information.

**[student@servera ~]$ id**

uid=1000(student) gid=1000(student) groups=1000(student),10(wheel) context=unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

* 1. Run **pwd** to display the current working directory.

**[student@servera ~]$ pwd**

/home/student

* 1. Print the values of the HOME and PATH variables to determine the home directory and user executables' path, respectively.

**[student@servera ~]$ echo $HOME**

/home/student

**[student@servera ~]$ echo $PATH**

/home/student/.local/bin:/home/student/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin

1. Switch to root in a non-login shell and explore the new shell environment.
   1. Run **sudo su** at the shell prompt to become the root user.

**[student@servera ~]$ sudo su**

**[sudo] password for student: student**

**[root@servera student]#**

* 1. Run **id** to view the current user and group information.

**[root@servera student]# id**

uid=0(root) gid=0(root) groups=0(root) context=unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

* 1. Run **pwd** to display the current working directory.

**[root@servera student]# pwd**

/home/student

* 1. Print the values of the HOME and PATH variables to determine the home directory and user executables' path, respectively.

**[root@servera student]# echo $HOME**

/root

**[root@servera student]# echo $PATH**

/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin

If you already have some experience with Linux and the **su** command, you may have expected that using **su** without the dash (-) option to become root would cause you to keep the current PATH of student. That did not happen. As you will see in the next step, this is not the usual PATH for root either.

What happened? The difference is that you did not run **su** directly. Instead, you ran **su** as root using **sudo** because you did not possess the password of the superuser. The **sudo** command initially overrides the PATH variable from the initial environment for security reasons. Any command that runs after the initial override can still update the PATH variable, as you will see in the following steps.

* 1. Exit the root user's shell to return to the student user's shell.

**[root@servera student]# exit**

exit

**[student@servera ~]$**

1. Switch to root in a login shell and explore the new shell environment.
   1. Run **sudo su -** at the shell prompt to become the root user.

**[student@servera ~]$ sudo su -**

**[root@servera ~]#**

Notice the difference in the shell prompt compared to that of **sudo su** in the preceding step.

**sudo** may or may not prompt you for the student password, depending on the time-out period of **sudo**. The default time-out period is five minutes. If you have authenticated to **sudo** within the last five minutes, **sudo** will not prompt you for the password. If it has been more than five minutes since you authenticated to **sudo**, you need to enter student as the password to get authenticated to **sudo**.

* 1. Run **id** to view the current user and group information.

**[root@servera ~]# id**

uid=0(root) gid=0(root) groups=0(root) context=unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

* 1. Run **pwd** to display the current working directory.

**[root@servera ~]# pwd**

/root

* 1. Print the values of the HOME and PATH variables to determine the home directory and the user executables' path, respectively.

**[root@servera ~]# echo $HOME**

/root

**[root@servera ~]# echo $PATH**

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

As in the preceding step, after **sudo** reset the PATH variable from the settings in the student user's shell environment, the **su -** command ran the shell login scripts for root and set the PATH variable to yet another value. The **su** command without the dash (-) option did not do that.

* 1. Exit the root user's shell to return to the student user's shell.

**[root@servera ~]# exit**

logout

**[student@servera ~]$**

1. Verify that the operator1 user is configured as to run any command as any user using **sudo**.

**[student@servera ~]$ sudo cat /etc/sudoers.d/operator1**

operator1 ALL=(ALL) ALL

1. Become operator1 and view the contents of /var/log/messages. Copy /etc/motd to /etc/motdOLD and remove it (/etc/motdOLD). These operations require administrative rights and so use **sudo** to run those commands as the superuser. Do not switch to root using **sudo su** or **sudo su -**. Use redhat as the password of operator1.
   1. Switch to operator1.

**[student@servera ~]$ su - operator1**

**Password: redhat**

**[operator1@servera ~]$**

* 1. Attempt to view the last five lines of /var/log/messages without using **sudo**. This should fail.

**[operator1@servera ~]$ tail -5 /var/log/messages**

tail: cannot open '/var/log/messages' for reading: Permission denied

* 1. Attempt to view the last five lines of /var/log/messages with **sudo**. This should succeed.

**[operator1@servera ~]$ sudo tail -5 /var/log/messages**

**[sudo] password for operator1: redhat**

Jan 23 15:53:36 servera su[2304]: FAILED SU (to operator1) student on pts/1

Jan 23 15:53:51 servera su[2307]: FAILED SU (to operator1) student on pts/1

Jan 23 15:53:58 servera su[2310]: FAILED SU (to operator1) student on pts/1

Jan 23 15:54:12 servera su[2322]: (to operator1) student on pts/1

Jan 23 15:54:25 servera su[2353]: (to operator1) student on pts/1

Note

The preceding output may differ on your system.

* 1. Attempt to make a copy of /etc/motd as /etc/motdOLD without using **sudo**. This should fail.

**[operator1@servera ~]$ cp /etc/motd /etc/motdOLD**

cp: cannot create regular file '/etc/motdOLD': Permission denied

* 1. Attempt to make a copy of /etc/motd as /etc/motdOLD with **sudo**. This should succeed.

**[operator1@servera ~]$ sudo cp /etc/motd /etc/motdOLD**

**[operator1@servera ~]$**

* 1. Attempt to delete /etc/motdOLD without using **sudo**. This should fail.

**[operator1@servera ~]$ rm /etc/motdOLD**

**rm: remove write-protected regular empty file '/etc/motdOLD'? y**

rm: cannot remove '/etc/motdOLD': Permission denied

**[operator1@servera ~]$**

* 1. Attempt to delete /etc/motdOLD with **sudo**. This should succeed.

**[operator1@servera ~]$ sudo rm /etc/motdOLD**

**[operator1@servera ~]$**

* 1. Exit the operator1 user's shell to return to the student user's shell.

**[operator1@servera ~]$ exit**

logout

**[student@servera ~]$**

* 1. Log off from servera.

**[student@servera ~]$ exit**

logout

Connection to servera closed.

**[student@workstation ~]$**

**Finish**

On workstation, run **lab users-sudo finish** to complete this exercise. This script deletes the user accounts and files created at the start of the exercise to ensure that the environment is clean.

**[student@workstation ~]$ lab users-sudo finish**

This concludes the guided exercise.

Managing Local User Accounts

Objectives

After completing this section, you should be able to create, modify, and delete local user accounts.

Managing Local Users

A number of command-line tools can be used to manage local user accounts.

**Creating Users from the Command Line**

* The **useradd *username*** command creates a new user named username. It sets up the user's home directory and account information, and creates a private group for the user named username. At this point the account does not have a valid password set, and the user cannot log in until a password is set.
* The **useradd --help** command displays the basic options that can be used to override the defaults. In most cases, the same options can be used with the **usermod** command to modify an existing user.
* Some defaults, such as the range of valid UID numbers and default password aging rules, are read from the /etc/login.defs file. Values in this file are only used when creating new users. A change to this file does not affect existing users.

**Modifying Existing Users from the Command Line**

* The **usermod --help** command displays the basic options that can be used to modify an account. Some common options include:

| **usermod options:** | **Usage** |
| --- | --- |
| -c, --comment COMMENT | Add the user's real name to the comment field. |
| -g, --gid GROUP | Specify the primary group for the user account. |
| -G, --groups GROUPS | Specify a comma-separated list of supplementary groups for the user account. |
| -a, --append | Used with the -G option to add the supplementary groups to the user's current set of group memberships instead of replacing the set of supplementary groups with a new set. |
| -d, --home HOME\_DIR | Specify a particular home directory for the user account. |
| -m, --move-home | Move the user's home directory to a new location. Must be used with the **-d** option. |
| -s, --shell SHELL | Specify a particular login shell for the user account. |
| -L, --lock | Lock the user account. |
| -U, --unlock | Unlock the user account. |

**Deleting Users from the Command Line**

* The **userdel *username*** command removes the details of username from /etc/passwd, but leaves the user's home directory intact.
* The **userdel -r *username*** command removes the details of username from /etc/passwd and also deletes the user's home directory.

Warning

When a user is removed with **userdel** without the -r option specified, the system will have files that are owned by an unassigned UID. This can also happen when a file, having a deleted user as its owner, exists outside that user's home directory. This situation can lead to information leakage and other security issues.

In Red Hat Enterprise Linux 7 and Red Hat Enterprise Linux 8, the **useradd** command assigns new users the first free UID greater than or equal to 1000, unless you explicitly specify one using the -u option.

This is how information leakage can occur. If the first free UID had been previously assigned to a user account which has since been removed from the system, the old user's UID will get reassigned to the new user, giving the new user ownership of the old user's remaining files.

The following scenario demonstrates this situation.

**[root@host ~]# useradd user01**

**[root@host ~]# ls -l /home**

drwx------. 3 user01 user01 74 Feb 4 15:22 user01

**[root@host ~]# userdel user01**

**[root@host ~]# ls -l /home**

drwx------. 3 1000 1000 74 Feb 4 15:22 user01

**[root@host ~]# useradd user02**

**[root@host ~]# ls -l /home**

drwx------. 3 user02 user02 74 Feb 4 15:23 user02

drwx------. 3 user02 user02 74 Feb 4 15:22 user01

Notice that user02 now owns all files that user01 previously owned.

Depending on the situation, one solution to this problem is to remove all unowned files from the system when the user that created them is deleted. Another solution is to manually assign the unowned files to a different user. The root user can use the **find / -nouser -o -nogroup** command to find all unowned files and directories.

**Setting Passwords from the Command Line**

* The **passwd *username*** command sets the initial password or changes the existing password of username.
* The root user can set a password to any value. A message is displayed if the password does not meet the minimum recommended criteria, but is followed by a prompt to retype the new password and all tokens are updated successfully.

**[root@host ~]# passwd user01**

Changing password for user user01.

New password: ***redhat***

BAD PASSWORD: The password fails the dictionary check - it is based on a dictionary word

Retype new password: ***redhat***

passwd: all authentication tokens updated successfully.

**[root@host ~]#**

* A regular user must choose a password at least eight characters long and is also not based on a dictionary word, the username, or the previous password.

**UID Ranges**

Specific UID numbers and ranges of numbers are used for specific purposes by Red Hat Enterprise Linux.

* *UID 0* is always assigned to the superuser account, root.
* *UID 1-200* is a range of "system users" assigned statically to system processes by Red Hat.
* *UID 201-999* is a range of "system users" used by system processes that do not own files on the file system. They are typically assigned dynamically from the available pool when the software that needs them is installed. Programs run as these "unprivileged" system users in order to limit their access to only the resources they need to function.
* *UID 1000+* is the range available for assignment to regular users.

Note

Prior to RHEL 7, the convention was that UID 1-499 was used for system users and UID 500+ for regular users. Default ranges used by **useradd** and **groupadd** can be changed in the /etc/login.defs file.

References

**useradd**(8), **usermod**(8), **userdel**(8) man pages

**Guided Exercise: Managing Local User Accounts**

In this exercise, you will create several users on your system and set passwords for those users.

**Outcomes**

You should be able to configure a Linux system with additional user accounts.

Log in to workstation as student using student as the password.

On workstation, run **lab users-manage start** to start the exercise. This script ensures that the environment is set up correctly.

**[student@workstation ~]$ lab users-manage start**

1. From workstation, open an SSH session to servera as student.

**[student@workstation ~]$ ssh student@servera**

*...output omitted...*

**[student@servera ~]$**

1. On servera, switch to root using **sudo**, converting to the root user's shell environment.

**[student@servera ~]$ sudo su -**

**[sudo] password for student: student**

**[root@servera ~]#**

1. Create the operator1 user and confirm that it exists in the system.

**[root@servera ~]# useradd operator1**

**[root@servera ~]# tail /etc/passwd**

*...output omitted...*

operator1:x:1002:1002::/home/operator1:/bin/bash

1. Set the password for operator1 to redhat.

**[root@servera ~]# passwd operator1**

Changing password for user operator1.

**New password: redhat**

BAD PASSWORD: The password is shorter than 8 characters

**Retype new password: redhat**

passwd: all authentication tokens updated successfully.

1. Create the additional users called operator2 and operator3. Set their passwords to redhat.
   1. Add the operator2 user. Set the password for operator2 to redhat.

**[root@servera ~]# useradd operator2**

**[root@servera ~]# passwd operator2**

Changing password for user operator2.

**New password: redhat**

BAD PASSWORD: The password is shorter than 8 characters

**Retype new password: redhat**

passwd: all authentication tokens updated successfully.

* 1. Add the operator3 user. Set the password for operator3 to redhat.

**[root@servera ~]# useradd operator3**

**[root@servera ~]# passwd operator3**

Changing password for user operator3.

**New password: redhat**

BAD PASSWORD: The password is shorter than 8 characters

**Retype new password: redhat**

passwd: all authentication tokens updated successfully.

1. Update the operator1 and operator2 user accounts to include the Operator One and Operator Two comments, respectively. Verify that the comments are successfully added.
   1. Run **usermod -c** to update the comments of the operator1 user account.

**[root@servera ~]# usermod -c "Operator One" operator1**

* 1. Run **usermod -c** to update the comments of the operator2 user account.

**[root@servera ~]# usermod -c "Operator Two" operator2**

* 1. Confirm that the comments for each of the operator1 and operator2 users are reflected in the user records.

**[root@servera ~]# tail /etc/passwd**

*...output omitted...*

operator1:x:1002:1002:Operator One:/home/operator1:/bin/bash

operator2:x:1003:1003:Operator Two:/home/operator2:/bin/bash

operator3:x:1004:1004::/home/operator3:/bin/bash

1. Delete the operator3 user along with any personal data of the user. Confirm that the user is successfully deleted.
   1. Remove the operator3 user from the system.

**[root@servera ~]# userdel -r operator3**

* 1. Confirm that operator3 is successfully deleted.

**[root@servera ~]# tail /etc/passwd**

*...output omitted...*

operator1:x:1002:1002:Operator One:/home/operator1:/bin/bash

operator2:x:1003:1003:Operator Two:/home/operator2:/bin/bash

Notice that the preceding output does not display the user account information of operator3.

* 1. Exit the root user's shell to return to the student user's shell.

**[root@servera ~]# exit**

logout

**[student@servera ~]$**

* 1. Log off from servera.

**[student@servera ~]$ exit**

logout

Connection to servera closed.

**[student@workstation ~]$**

**Finish**

On workstation, run **lab users-manage finish** to complete this exercise. This script ensures that the environment is clean.

**[student@workstation ~]$ lab users-manage finish**

This concludes the guided exercise.

**Managing Local Group Accounts**

Objectives

After completing this section, students should be able to create, modify, and delete local group accounts.

Managing Local Groups

A group must exist before a user can be added to that group. Several command-line tools are used to manage local group accounts.

**Creating Groups from the Command Line**

* The **groupadd** command creates groups. Without options the **groupadd** command uses the next available GID from the range specified in the /etc/login.defs file while creating the groups.
* The -g option specifies a particular GID for the group to use.

**[user01@host ~]$ sudo groupadd -g *10000* *group01***

**[user01@host ~]$ tail /etc/group**

*...output omitted...*

group01:x:10000:

Note

Given the automatic creation of user private groups (GID 1000+), it is generally recommended to set aside a range of GIDs to be used for supplementary groups. A higher range will avoid a collision with a system group (GID 0-999).

* The -r option creates a system group using a GID from the range of valid system GIDs listed in the /etc/login.defs file. The SYS\_GID\_MIN and SYS\_GID\_MAX configuration items in /etc/login.defs define the range of system GIDs.

**[user01@host ~]$ sudo groupadd -r *group02***

**[user01@host ~]$ tail /etc/group**

*...output omitted...*

group01:x:10000:

**group02:x:988:**

**Modifying Existing Groups from the Command Line**

* The **groupmod** command changes the properties of an existing group. The -n option specifies a new name for the group.

**[user01@host ~]$ sudo groupmod -n *group0022* *group02***

**[user01@host ~]$ tail /etc/group**

*...output omitted...*

group0022:x:988:

Notice that the group name is updated to group0022 from group02.

* The **-g** option specifies a new GID.

**[user01@host ~]$ sudo groupmod -g *20000* *group0022***

**[user01@host ~]$ tail /etc/group**

*...output omitted...*

group0022:x:20000:

Notice that the GID is updated to 20000 from 988.

**Deleting Groups from the Command Line**

* The **groupdel** command removes groups.

**[user01@host ~]$ sudo groupdel *group0022***

Note

You cannot remove a group if it is the primary group of any existing user. As with **userdel**, check all file systems to ensure that no files remain on the system that are owned by the group.

**Changing Group Membership from the Command Line**

* The membership of a group is controlled with user management. Use the **usermod -g** command to change a user's primary group.

**[user01@host ~]$ id *user02***

uid=1006(user02) gid=1008(user02) groups=1008(user02)

**[user01@host ~]$ sudo usermod -g *group01* *user02***

**[user01@host ~]$ id *user02***

uid=1006(user02) gid=10000(group01) groups=10000(group01)

* Use the **usermod -aG** command to add a user to a supplementary group.

**[user01@host ~]$ id *user03***

uid=1007(user03) gid=1009(user03) groups=1009(user03)

**[user01@host ~]$ sudo usermod -aG *group01* *user03***

**[user01@host ~]$ id *user03***

uid=1007(user03) gid=1009(user03) groups=1009(user03),10000(group01)

Important

The use of the -a option makes **usermod** function in *append* mode. Without -a, the user will be removed from any of their current supplementary groups that are not included in the -G option's list.

References

**group**(5), **groupadd**(8),**groupdel**(8), and **usermod**(8) man pages

**Guided Exercise: Managing Local Group Accounts**

In this exercise, you will create groups, use them as supplementary groups for some users without changing those users' primary groups, and configure one of the groups with sudo access to run commands as root.

**Outcomes**

You should be able to:

* Create groups and use them as supplementary groups.
* Configure sudo access for a group.

Log in to workstation as student using student as the password.

On workstation, run **lab users-group-manage start** to start the exercise. This script creates the necessary user accounts to set up the environment correctly.

**[student@workstation ~]$ lab users-group-manage start**

1. From workstation, open an SSH session to servera as student.

**[student@workstation ~]$ ssh student@servera**

*...output omitted...*

**[student@servera ~]$**

1. On servera, switch to root using **sudo**, inheriting the full environment of the root user.

**[student@servera ~]$ sudo su -**

**[sudo] password for student: student**

**[root@servera ~]#**

1. Create the operators supplementary group with the GID of 30000.

**[root@servera ~]# groupadd -g 30000 operators**

1. Create admin as an additional supplementary group.

**[root@servera ~]# groupadd admin**

1. Verify that both the operators and admin supplementary groups exist.

**[root@servera ~]# tail /etc/group**

*...output omitted...*

operators:x:30000:

admin:x:30001:

1. Ensure that the users operator1, operator2 and operator3 belong to the group operators.
   1. Add operator1, operator2, and operator3 to operators.

**[root@servera ~]# usermod -aG operators operator1**

**[root@servera ~]# usermod -aG operators operator2**

**[root@servera ~]# usermod -aG operators operator3**

* 1. Confirm that the users are successfully added to the group.

**[root@servera ~]# id operator1**

uid=1002(operator1) gid=1002(operator1) groups=1002(operator1),30000(operators)

**[root@servera ~]# id operator2**

uid=1003(operator2) gid=1003(operator2) groups=1003(operator2),30000(operators)

**[root@servera ~]# id operator3**

uid=1004(operator3) gid=1004(operator3) groups=1004(operator3),30000(operators)

1. Ensure that the users sysadmin1, sysadmin2 and sysadmin3 belong to the group admin. Enable administrative rights for all the group members of admin. Verify that any member of admin can run administrative commands.
   1. Add sysadmin1, sysadmin2, and sysadmin3 to admin.

**[root@servera ~]# usermod -aG admin sysadmin1**

**[root@servera ~]# usermod -aG admin sysadmin2**

**[root@servera ~]# usermod -aG admin sysadmin3**

* 1. Confirm that the users are successfully added to the group.

**[root@servera ~]# id sysadmin1**

uid=1005(sysadmin1) gid=1005(sysadmin1) groups=1005(sysadmin1),30001(admin)

**[root@servera ~]# id sysadmin2**

uid=1006(sysadmin2) gid=1006(sysadmin2) groups=1006(sysadmin2),30001(admin)

**[root@servera ~]# id sysadmin3**

uid=1007(sysadmin3) gid=1007(sysadmin3) groups=1007(sysadmin3),30001(admin)

* 1. Examine /etc/group to verify the supplemental group memberships.

**[root@servera ~]# tail /etc/group**

*...output omitted...*

operators:x:30000:operator1,operator2,operator3

admin:x:30001:sysadmin1,sysadmin2,sysadmin3

* 1. Create the /etc/sudoers.d/admin file such that the members of admin have full administrative privileges.

**[root@servera ~]# echo "%admin ALL=(ALL) ALL" >> /etc/sudoers.d/admin**

* 1. Switch to sysadmin1 (a member of admin) and verify that you can run a **sudo** command as sysadmin1.

**[root@servera ~]# su - sysadmin1**

**[sysadmin1@servera ~]$ sudo cat /etc/sudoers.d/admin**

**[sudo] password for sysadmin1: redhat**

%admin ALL=(ALL) ALL

* 1. Exit the sysadmin1 user's shell to return to the root user's shell.

**[sysadmin1@servera ~]$ exit**

logout

**[root@servera ~]#**

* 1. Exit the root user's shell to return to the student user's shell.

**[root@servera ~]# exit**

logout

**[student@servera ~]$**

* 1. Log off from servera.

**[student@servera ~]$ exit**

logout

Connection to servera closed.

**[student@workstation ~]$**

**Finish**

On workstation, run **lab users-group-manage finish** to complete this exercise. This script deletes the user accounts created at the start of the exercise.

**[student@workstation ~]$ lab users-group-manage finish**

This concludes the guided exercise.

Managing User Passwords

Objectives

After completing this section, you should be able to set a password management policy for users, and manually lock and unlock user accounts.

Shadow Passwords and Password Policy

At one time, encrypted passwords were stored in the world-readable /etc/passwd file. This was thought to be reasonably secure until dictionary attacks on encrypted passwords became common. At that point, the encrypted passwords were moved to a separate /etc/shadow file which is readable only by root. This new file also allowed password aging and expiration features to be implemented.

Like /etc/passwd, each user has a line in the /etc/shadow file. A sample line from /etc/shadow with its nine colon-separated fields is shown below.

user03:$6$CSsX*...output omitted...*:17933:0:99999:7:2:18113: 

|  |  |
| --- | --- |
|  | 1. Username of the account this password belongs to. |
|  | 1. The *encrypted password* of the user. The format of encrypted passwords is discussed later in this section. |
|  | 1. The day on which the password was last changed. This is set in days since 1970-01-01, and is calculated in the UTC time zone. |
|  | 1. The minimum number of days that have to elapse since the last password change before the user can change it again. |
|  | 1. The maximum number of days that can pass without a password change before the password expires. An empty field means it does not expire based on time since the last change. |
|  | 1. Warning period. The user will be warned about an expiring password when they login for this number of days before the deadline. |
|  | 1. Inactivity period. Once the password has expired, it will still be accepted for login for this many days. After this period has elapsed, the account will be locked. |
|  | 1. The day on which the account expires. This is set in days since 1970-01-01, and is calculated in the UTC time zone. An empty field means it does not expire on a particular date. |
|  | 1. The last field is usually empty and is reserved for future use. |

**Format of an Encrypted Password**

The encrypted password field stores three pieces of information: the *hashing algorithm* used, the *salt*, and the *encrypted hash*. Each piece of information is delimited by the $ sign.

$6$CSsXcYG1L/4ZfHr/$2W6evvJahUfzfHpc9X.45Jc6H30E*...output omitted...*



|  |  |
| --- | --- |
|  | 1. The hashing algorithm used for this password. The number 6 indicates it is a SHA-512 hash, which is the default in Red Hat Enterprise Linux 8. A 1 would indicate MD5, a 5 SHA-256. |
|  | 1. The salt used to encrypt the password. This is originally chosen at random. |
|  | 1. The encrypted hash of the user's password. The salt and the unencrypted password are combined and encrypted to generate the encrypted hash of the password. |

The primary reason to combine a salt with the password is to defend against attacks using pre-computed lists of password hashes. Adding salts changes the resulting hashes, making the pre-computed list useless. If an attacker is able to obtain a copy of an /etc/shadow file that is using salts, they will need to perform brute-force password guessing, requiring more time and effort.

**Password Verification**

When a user tries to log in, the system looks up the entry for the user in /etc/shadow, combines the salt for the user with the unencrypted password that was typed in, and encrypts them using the hashing algorithm specified. If the result matches the encrypted hash, the user typed in the right password. If the result does not match the encrypted hash, the user typed in the wrong password and the login attempt fails. This method allows the system to determine if the user typed in the correct password without storing that password in a form usable for logging in.

Configuring Password Aging

The following diagram relates the relevant password aging parameters, which can be adjusted using the **chage** command to implement a password aging policy.

|  |
| --- |
|  |

**[user01@host ~]$ sudo chage -m *0* -M *90* -W *7* -I *14* *user03***

The preceding **chage** command uses the -m, -M, -W, and -I options to set the minimum age, maximum age, warning period, and inactivity period of the user's password, respectively.

The **chage -d 0 user03** command forces the user03 user to update its password on the next login.

The **chage -l user03** command displays the password aging details of user03.

The **chage -E 2019-08-05 user03** command causes the user03 user's account to expire on 2019-08-05 (in YYYY-MM-DD format).

Note

The **date** command can be used to calculate a date in the future. The -u option reports the time in UTC.

**[user01@host ~]$ date -d "+*45* days" -u**

Thu May 23 17:01:20 UTC 2019

Edit the password aging configuration items in the /etc/login.defs file to set the default password aging policies. The PASS\_MAX\_DAYS sets the default maximum age of the password. The PASS\_MIN\_DAYS sets the default minimum age of the password. The PASS\_WARN\_AGE sets the default warning period of the password. Any change in the default password aging policies will be effective for new users only. The existing users will continue to use the old password aging settings rather than the new ones.

Restricting Access

You can use the **chage** command to set account expiration dates. When that date is reached, the user cannot log in to the system interactively. The **usermod** command can lock an account with the -L option.

**[user01@host ~]$ sudo usermod -L *user03***

**[user01@host ~]$ su - *user03***

**Password: *redhat***

su: Authentication failure

If a user leaves the company, the administrator may lock and expire an account with a single **usermod** command. The date must be given as the number of days since 1970-01-01, or in the *YYYY-MM-DD* format.

**[user01@host ~]$ sudo usermod -L -e *2019-10-05* *user03***

The preceding **usermod** command uses the -e option to set the account expiry date for the given user account. The -L option locks the user's password.

Locking the account prevents the user from authenticating with a password to the system. It is the recommended method of preventing access to an account by an employee who has left the company. If the employee returns, the account can later be unlocked with **usermod -U**. If the account was also expired, be sure to also change the expiration date.

**The nologin Shell**

The **nologin** shell acts as a replacement shell for the user accounts not intended to interactively log into the system. It is wise from a security standpoint to disable an account from logging into the system, when the account does not require it. For example, a mail server may require an account to store mail and a password for the user to authenticate with a mail client used to retrieve mail. That user does not need to log directly into the system.

A common solution to this situation is to set the user's login shell to /sbin/nologin. If the user attempts to log in to the system directly, the **nologin** shell closes the connection.

**[user01@host ~]$ usermod -s /sbin/nologin *user03***

**[user01@host ~]$ su - *user03***

Last login: Wed Feb 6 17:03:06 IST 2019 on pts/0

This account is currently not available.

Important

The **nologin** shell prevents interactive use of the system, but does not prevent all access. Users might be able to authenticate and upload or retrieve files through applications such as web applications, file transfer programs, or mail readers if they use the user's password for authentication.

References

**chage**(1), **usermod**(8), **shadow**(5), **crypt**(3) man pages

Guided Exercise: Managing User Passwords

In this exercise, you will set password policies for several users.

**Outcomes**

You should be able to:

* Force a password change when the user logs in to the system for the first time.
* Force a password change every 90 days.
* Set the account to expire 180 days from the current day.

Log in to workstation as student using student as the password.

On workstation, run **lab users-pw-manage start** to start the exercise. This script creates the necessary user accounts and files to ensure that the environment is set up correctly.

**[student@workstation ~]$ lab users-pw-manage start**

1. From workstation, open an SSH session to servera as student.

**[student@workstation ~]$ ssh student@servera**

*...output omitted...*

**[student@servera ~]$**

1. On servera, explore locking and unlocking user accounts as student.
   1. As student, lock the operator1 account using administrative rights.

**[student@servera ~]$ sudo usermod -L operator1**

**[sudo] password for student: student**

* 1. Attempt to log in as operator1. This should fail.

**[student@servera ~]$ su - operator1**

**Password: redhat**

su: Authentication failure

* 1. Unlock the operator1 account.

**[student@servera ~]$ sudo usermod -U operator1**

* 1. Attempt to log in as operator1 again. This should succeed.

**[student@servera ~]$ su - operator1**

**Password: redhat**

*...output omitted...*

**[operator1@servera ~]$**

* 1. Exit out of the operator1 user's shell to return to the student user's shell.

**[operator1@servera ~]$ exit**

logout

1. Change the password policy for operator1 to require a new password every 90 days. Confirm that the password age is successfully set.
   1. Set the maximum age of the operator1 user's password to 90 days.

**[student@servera ~]$ sudo chage -M 90 operator1**

* 1. Verify that the operator1 user's password expires 90 days after it is changed.

**[student@servera ~]$ sudo chage -l operator1**

Last password change : Jan 25, 2019

Password expires : Apr 25, 2019

Password inactive : never

Account expires : never

Minimum number of days between password change : 0

**Maximum number of days between password change : 90**

Number of days of warning before password expires : 7

1. Force a password change on the first login for the operator1 account.

**[student@servera ~]$ sudo chage -d 0 operator1**

1. Log in as operator1 and change the password to forsooth123. After setting the password, return to the student user's shell.
   1. Log in as operator1 and change the password to forsooth123 when prompted.

**[student@servera ~]$ su - operator1**

**Password: redhat**

You are required to change your password immediately (administrator enforced)

**Current password: redhat**

**New password: forsooth123**

**Retype new password: forsooth123**

*...output omitted...*

**[operator1@servera ~]$**

* 1. Exit the operator1 user's shell to return to the student user's shell.

**[operator1@servera ~]$ exit**

logout

1. Set the operator1 account to expire 180 days from the current day. Hint: The **date -d "+180 days"** gives you the date and time 180 days from the current date and time.
   1. Determine a date 180 days in the future. Use the format %F with the **date** command to get the exact value.

**[student@servera ~]$ date -d "+180 days" +%F**

2019-07-24

You may get a different value to use in the following step based on the current date and time in your system.

* 1. Set the account to expire on the date displayed in the preceding step.

**[student@servera ~]$ sudo chage -E *2019-07-24* operator1**

* 1. Verify that the account expiry date is successfully set.

**[student@servera ~]$ sudo chage -l operator1**

Last password change : Jan 25, 2019

Password expires : Apr 25, 2019

Password inactive : never

**Account expires : Jul 24, 2019**

Minimum number of days between password change : 0

Maximum number of days between password change : 90

Number of days of warning before password expires : 7

1. Set the passwords to expire 180 days from the current date for all users. Use administrative rights to edit the configuration file.
   1. Set PASS\_MAX\_DAYS to 180 in /etc/login.defs. Use administrative rights when opening the file with the text editor. You can use the **sudo vim /etc/login.defs** command to perform this step.

*...output omitted...*

# Password aging controls:

#

# PASS\_MAX\_DAYS Maximum number of days a password may be

# used.

# PASS\_MIN\_DAYS Minimum number of days allowed between

# password changes.

# PASS\_MIN\_LEN Minimum acceptable password length.

# PASS\_WARN\_AGE Number of days warning given before a

# password expires.

#

**PASS\_MAX\_DAYS 180**

PASS\_MIN\_DAYS 0

PASS\_MIN\_LEN 5

PASS\_WARN\_AGE 7

*...output omitted...*

Important

The default password and account expiry settings will be effective for new users but not for existing users.

* 1. Log off from servera.

**[student@servera ~]$ exit**

logout

Connection to servera closed.

**[student@workstation ~]$**

**Finish**

On workstation, run **lab users-pw-manage finish** to complete this exercise. This script deletes the user accounts and files created at the start of the exercise to ensure that the environment is clean.

**[student@workstation ~]$ lab users-pw-manage finish**

This concludes the guided exercise.

Summary

In this chapter, you learned:

* There are three main types of user account: the superuser, system users, and regular users.
* A user must have a primary group and may be a member of one or more supplementary groups.
* The three critical files containing user and group information are /etc/passwd, /etc/group, and /etc/shadow.
* The **su** and **sudo** commands can be used to run commands as the superuser.
* The **useradd**, **usermod**, and **userdel** commands can be used to manage users.
* The **groupadd**, **groupmod**, and **groupdel** commands can be used to manage groups.
* The **chage** command can be used to configure and view password expiration settings for users.