Chapter 5

Managing Local Linux Users and Groups

Goal

To manage local Linux users and groups and administer local password policies.

Objectives

- Explain the role of users and groups on a Linux system and how they are understood by the computer.
- Run commands as the superuser to administer a Linux system.
- Create, modify, lock, and delete locally defined user accounts.
- Create, modify, and delete locally defined group accounts.

Sections

- Users and Groups (and Quiz)
- Gaining Superuser Access (and Guided Exercise)
- Managing Local User Accounts (and Guided Exercise)
- Managing Local Group Accounts (and Guided Exercise)

Lab

Managing Local Linux Users and Groups

Users and Groups

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**:

```
❶user01:❷x:❸1000:❹1000:❺User One:❻/home/user01:❼/bin/bash
```

❶ Username for this user (**user01**).
❷ 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**.
❸ The UID number for this user account (**1000**).
❹ The GID number for this user account's primary group (**1000**). Groups will be discussed later in this section.
❺ The real name for this user (**User One**).
❻ 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.
❼ 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**:

```
1group01:2x:310000:4user01,user02,user03
```

1. Group name for this group (**group01**).
2. Obsolete group password field. This field should always be **x**.
3. The GID number for this group (**10000**).
4. 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

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?

a. primary user
b. UID
c. GID
d. username

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

a. primary shell
b. home directory
c. login shell
d. command name

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

a. home directory
b. **`/etc/passwd`**
c. **`/etc/GID`**
d. **`/etc/group`**

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

a. home directory
b. login shell
c. **`/etc/passwd`**
d. **`/etc/group`**

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

a. primary group
b. UID
c. GID
d. groupid

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

a. home directory
b. **`/etc/passwd`**
c. **`/etc/UID`**
d. **`/etc/group`**

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

a. home directory
b. UID
c. login shell
d. primary group

Solution

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?

a. primary user
b. UID
c. GID
d. username

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

a. primary shell
b. home directory
c. login shell
d. command name

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

a. home directory
b. `/etc/passwd`
c. `/etc/GID`
d. **`/etc/group`**

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

a. home directory
b. login shell
c. `/etc/passwd`
d. `/etc/group`

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

a. primary group
b. UID
c. GID
d. groupid

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

a. home directory
b. **`/etc/passwd`**
c. `/etc/UID`
d. `/etc/group`

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

a. home directory
b. UID
c. login shell
d. 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

Running Commands as root

In this exercise, you will practice running commands as **root**.

Outcomes

- Use the **sudo** command to run other commands as **root**.
- Use **sudo** to run **su** to get an interactive shell as **root** when the superuser does not have a valid password.
- Explain how **su** and **su -** can affect the shell environment through not running or running login scripts.

Before You Begin

Start your Amazon EC2 instance and use **ssh** to log in as the user **ec2-user**.

It is assumed that the AMI that you are using is pre-configured to allow the **ec2-user** to run any command as any user without a password using **sudo**.

Steps

1. Explore the characteristics of the **ec2-user** shell environment.

    1.1. View the current user and group information and display the current working directory.

```
[ec2-user@ip-192-0-2-1 ~]$ id
uid=1000(ec2-user) gid=1000(ec2-user) groups=1000(ec2-user),4(adm),190(systemd-
journal) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[ec2-user@ip-192-0-2-1 ~]$ pwd
/home/ec2-user
```

    1.2. View the environment variables which specify the user's home directory and the locations searched for executable files.

```
[ec2-user@ip-192-0-2-1 ~]$ echo $HOME
/home/ec2-user
[ec2-user@ip-192-0-2-1 ~]$ echo $PATH
/home/ec2-user/.local/bin:/home/ec2-user/bin:/usr/local/bin:/usr/bin:/usr/local/
sbin:/usr/sbin
```

2. Switch to **root** by using **sudo** to run **su** without the dash, and explore the characteristics of the new shell environment.

    2.1. Become the **root** user at the shell prompt.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo su
[root@ip-192-0-2-1 ec2-user]#
```

2.2. View the current user and group information and display the current working directory. Note that the user identity changed, but not the current working directory.

```
[root@ip-192-0-2-1 ec2-user]# id
uid=0(root) gid=0(root) groups=0(root)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@ip-192-0-2-1 ec2-user]# pwd
/home/ec2-user
```

2.3. View the environment variables which specify the home directory and the locations searched for executable files. Look for references to the **ec2-user** and **root** accounts.

```
[root@ip-192-0-2-1 ec2-user]# echo $HOME
/root
[root@ip-192-0-2-1 ec2-user]# echo $PATH
/sbin:/bin:/usr/sbin:/usr/bin
```

Important

If you already have some experience with Linux and the **su** command, you may have expected that using **su** without the **-** option to become **root** would cause you to keep the current **PATH** of **ec2-user** (as seen in the previous step). That did not happen! But as you'll see in the next step, this isn't the normal **PATH** for **root** either.

What happened? The difference is that you did not run **su** directly (because **root** doesn't have a valid password, and since you are not **root** you need to use a password to use **su**). Instead, you ran **su** as **root** using the **sudo** command so you did not need a **root** password.

It turns out that **sudo** initially overrides the **PATH** from the initial environment for security reasons. That variable can still be updated by the command that was run after that initial override, which is what **su -** will do in a moment.

2.4. Exit the shell to return to the **ec2-user** user.

```
[root@ip-192-0-2-1 ec2-user]# exit
exit
[ec2-user@ip-192-0-2-1 ~]$
```

3. Switch to **root** by using **sudo** to run **su -**, and compare the characteristics of the new shell environment with those of the preceding example.

3.1. Become the **root** user at the shell prompt. Be sure all the login scripts are also executed.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo su -
Last login: Fri Jul 24 17:16:01 EDT 2020 on pts/0
[root@ip-192-0-2-1 ~]#
```

Note the differences: the **`Last login`** message and the difference in the shell prompt.

3.2. View the current user and group information and display the current working directory.

```
[root@ip-192-0-2-1 ~]# id
uid=0(root) gid=0(root) groups=0(root)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@ip-192-0-2-1 ~]# pwd
/root
```

3.3. View the environment variables which specify the home directory and the locations searched for executable files. Look for references to the **`ec2-user`** and **`root`** accounts.

```
[root@ip-192-0-2-1 ~]# echo $HOME
/root
[root@ip-192-0-2-1 ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

Important

In this example, after **`sudo`** reset the **`PATH`** environment variable from the settings in the **`ec2-user`** shell environment, the **`su -`** command ran the shell login scripts for **`root`** and set **`PATH`** again, to yet another value. The **`su`** command without the **`-`** option (in the previous example in this exercise) did not do that.

3.4. Exit the **`root`** shell to return to the **`ec2-user`** shell.

```
[root@ip-192-0-2-1 ~]# exit
logout
[ec2-user@ip-192-0-2-1 ~]$
```

4. Next, run several commands as **`ec2-user`** which require **`root`** access. Then, rather than using **`sudo su -`** to run them as **`root`** from an interactive **`root`** shell, use **`sudo`** to run those commands as **`root`** from the **`ec2-user`** shell as one-off commands.

4.1. View the last 5 lines of the **`/var/log/messages`**. Your output may vary from this example.

```
[ec2-user@ip-192-0-2-1 ~]$ tail -n 5 /var/log/messages
tail: cannot open '/var/log/messages' for reading: Permission denied
[ec2-user@ip-192-0-2-1 ~]$ sudo tail -n 5 /var/log/messages
Jul 24 17:10:10 ip-192-0-2-1 systemd: Started Network Manager Script Dispatcher
 Service.
```

```
Jul 24 17:10:10 ip-192-0-2-1 nm-dispatcher: req:1 'dhcp4-change' [eth0]: new
 request (3 scripts)
Jul 24 17:10:10 ip-192-0-2-1 nm-dispatcher: req:1 'dhcp4-change' [eth0]: start
 running ordered scripts...
Jul 24 17:16:01 ip-192-0-2-1 su: (to root) ec2-user on pts/0
Jul 24 17:18:12 ip-192-0-2-1 su: (to root) ec2-user on pts/0
[ec2-user@ip-192-0-2-1 ~]$
```

4.2. Make a backup of a configuration file in the **/etc** directory.

```
[ec2-user@ip-192-0-2-1 ~]$ cp /etc/issue /etc/issueOLD
cp: cannot create regular file '/etc/issueOLD': Permission denied
[ec2-user@ip-192-0-2-1 ~]$ sudo cp /etc/issue /etc/issueOLD
[ec2-user@ip-192-0-2-1 ~]$
```

4.3. Remove the **/etc/issueOLD** file that was just created.

```
[ec2-user@ip-192-0-2-1 ~]$ rm /etc/issueOLD
rm: remove write-protected regular empty file '/etc/issueOLD'? y
rm: cannot remove '/etc/issueOLD': Permission denied
[ec2-user@ip-192-0-2-1 ~]$ sudo rm /etc/issueOLD
[ec2-user@ip-192-0-2-1 ~]$
```

5. This concludes this exercise. Log out and stop your Amazon EC2 instance.

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.

usermod options:	Usage
-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

Creating Users Using Command-line Tools

In this exercise, you will create a number of users on your Linux system, setting and recording an initial password for each user.

Outcomes

- Be able to configure a Linux system with additional user accounts.

Before You Begin

Start your Amazon EC2 instance and use **ssh** to log in as the user **ec2-user**. It is assumed that **ec2-user** can use **sudo** to run commands as **root**.

Steps

1. Become the **root** user at the shell prompt.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo su -
[root@ip-192-0-2-1 ~]#
```

2. Add the user **juliet**.

```
[root@ip-192-0-2-1 ~]# useradd juliet
```

3. Confirm that **juliet** has been added by examining the **/etc/passwd** file.

```
[root@ip-192-0-2-1 ~]# tail -n 2 /etc/passwd
ec2-user:x:1000:1000:Cloud User:/home/ec2-user:/bin/bash
juliet:x:1001:1001::/home/juliet:/bin/bash
```

4. Use the **passwd** command to initialize **juliet**'s password. For this and any other passwords you set in this section, the values are not important but should be "secure". Be careful, the **root** user can set arbitrarily weak passwords!

   The example below assumes you typed something (the same thing!) for the `New password` and `Retype new password` prompts.

Warning

It is assumed that the AMI used for your Amazon EC2 instance was pre-configured for security reasons to prohibit remote logins over **ssh** which have been authenticated by password, and that key-based authentication is required. Please read the **Warning** box at the end of this exercise for further discussion.

```
[root@ip-192-0-2-1 ~]# passwd juliet
Changing password for user juliet.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@ip-192-0-2-1 ~]#
```

5. Continue adding the remaining users in the steps below and set initial passwords. These users will be used in the next exercise.

   5.1. **romeo**

```
[root@ip-192-0-2-1 ~]# useradd romeo
[root@ip-192-0-2-1 ~]# passwd romeo
Changing password for user romeo.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@ip-192-0-2-1 ~]#
```

   5.2. **hamlet**

```
[root@ip-192-0-2-1 ~]# useradd hamlet
[root@ip-192-0-2-1 ~]# passwd hamlet
```

   5.3. **sunni**

```
[root@ip-192-0-2-1 ~]# useradd sunni
[root@ip-192-0-2-1 ~]# passwd sunni
```

   5.4. **huong**

```
[root@ip-192-0-2-1 ~]# useradd huong
[root@ip-192-0-2-1 ~]# passwd huong
```

   5.5. **jerlene**

```
[root@ip-192-0-2-1 ~]# useradd jerlene
[root@ip-192-0-2-1 ~]# passwd jerlene
```

6. This concludes this exercise. Log out and stop your Amazon EC2 instance.

Warning

In this exercise, you set passwords for various accounts on a cloud instance that is accessible from the public Internet. This is not a recommended practice if users can connect to the instance using **ssh** and authenticate using their password.

Attackers have been known to scan public cloud providers for instances providing remote **ssh** access and configured with weak passwords. The recommended AMI used to develop this course was configured to prohibit password-based **ssh** authentication, and to provide only **ssh** as a public-facing service, so setting those passwords for this exercise should not matter in practice. However, this is not necessarily true of a default Red Hat Enterprise Linux installation.

Additional keys can be installed by users to allow key-based authentication, but the procedure is beyond the scope of this course.

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 Groups Using Command-line Tools

In this exercise, you will add users to newly created supplementary groups.

Outcomes

- The **shakespeare** group consists of users **juliet**, **romeo**, and **hamlet**.
- The **artists** group consists of users **sunni**, **huong**, and **jerlene**.

Before You Begin

Start your Amazon EC2 instance and use **ssh** to log in as the user **ec2-user**. It is assumed that **ec2-user** can use **sudo** to run commands as **root**.

Steps

1. Become the **root** user at the shell prompt.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo su -
```

2. Create a supplementary group called **shakespeare** with a group ID of **30000**.

```
[root@ip-192-0-2-1 ~]# groupadd -g 30000 shakespeare
```

3. Create a supplementary group called **artists**.

```
[root@ip-192-0-2-1 ~]# groupadd artists
```

4. Confirm that **shakespeare** and **artists** have been added by examining the **/etc/group** file.

```
[root@ip-192-0-2-1 ~]# tail -n 5 /etc/group
sunni:x:1004:
huong:x:1005:
jerlene:x:1006:
shakespeare:x:30000:
artists:x:30001:
```

5. Add the **juliet** user to the **shakespeare** group as a supplementary group.

```
[root@ip-192-0-2-1 ~]# usermod -G shakespeare juliet
```

6. Confirm that **juliet** has been added using the **id** command.

```
[root@ip-192-0-2-1 ~]# id juliet
uid=1001(juliet) gid=1001(juliet) groups=1001(juliet),30000(shakespeare)
```

7. Continue adding the remaining users to groups as follows:

    7.1. Add **romeo** and **hamlet** to the **shakespeare** group.

```
[root@ip-192-0-2-1 ~]# usermod -G shakespeare romeo
[root@ip-192-0-2-1 ~]# usermod -G shakespeare hamlet
```

    7.2. Add **sunni**, **huong**, and **jerlene** to the **artists** group.

```
[root@ip-192-0-2-1 ~]# usermod -G artists sunni
[root@ip-192-0-2-1 ~]# usermod -G artists huong
[root@ip-192-0-2-1 ~]# usermod -G artists jerlene
```

    7.3. Verify the supplemental group memberships by examining the **/etc/group** file.

```
[root@ip-192-0-2-1 ~]# tail -n 5 /etc/group
sunni:x:1004:
huong:x:1005:
jerlene:x:1006:
shakespeare:x:30000:juliet,romeo,hamlet
artists:x:30001:sunni,huong,jerlene
```

8. This concludes this exercise. Log out and stop your Amazon EC2 instance.

Lab

Managing Local Linux Users and Groups

In this lab, you will create three new users and a group, and make those users members of that group as a supplementary group.

Outcomes

- Be able to create three new user acccounts for Dinesh Jonasen (**`djonasen`**), Sumiko Alves (**`salves`**), and Denver Tyson (**`dtyson`**).
- Be able to create a new group called **`consultants`**, including the three new user accounts for Dinesh Jonasen, Sumiko Alves, and Denver Tyson as members.

Before You Begin

Start your Amazon EC2 instance and use **`ssh`** to log in as the user **`ec2-user`**. It is assumed that **`ec2-user`** can use **`sudo`** to run commands as **`root`**.

Steps

1. Create a new group named **`consultants`** with a GID of 40000.
2. Create three new users: **`djonasen`**, **`salves`**, and **`dtyson`**, set a password for each, and add each to the supplementary group **`consultants`**. Each user's primary group should be a user private group with the same name as the user.

Warning

It's assumed that you set a "secure" password for each of these users, as discussed in the **Warning** in a previous exercise in this chapter.

3. Check each user to confirm the account exists and has the correct group memberships.
4. This concludes this lab. Log out and stop your Amazon EC2 instance.

Solution

Managing Local Linux Users and Groups

In this lab, you will create three new users and a group, and make those users members of that group as a supplementary group.

Outcomes

- Be able to create three new user acccounts for Dinesh Jonasen (**djonasen**), Sumiko Alves (**salves**), and Denver Tyson (**dtyson**).
- Be able to create a new group called **consultants**, including the three new user accounts for Dinesh Jonasen, Sumiko Alves, and Denver Tyson as members.

Before You Begin

Start your Amazon EC2 instance and use **ssh** to log in as the user **ec2-user**. It is assumed that **ec2-user** can use **sudo** to run commands as **root**.

Steps

1. Create a new group named **consultants** with a GID of 40000.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo groupadd -g 40000 consultants
[ec2-user@ip-192-0-2-1 ~]$ tail -n 5 /etc/group
huong:x:1005:
jerlene:x:1006:
shakespeare:x:30000:juliet,romeo,hamlet
artists:x:30001:sunni,huong,jerlene
consultants:x:40000:
```

2. Create three new users: **djonasen**, **salves**, and **dtyson**, set a password for each, and add each to the supplementary group **consultants**. Each user's primary group should be a user private group with the same name as the user.

Warning

It's assumed that you set a "secure" password for each of these users, as discussed in the **Warning** in a previous exercise in this chapter.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo useradd -G consultants djonasen
[ec2-user@ip-192-0-2-1 ~]$ sudo useradd -G consultants salves
[ec2-user@ip-192-0-2-1 ~]$ sudo useradd -G consultants dtyson
[ec2-user@ip-192-0-2-1 ~]$ tail -n 5 /etc/group
artists:x:30001:sunni,huong,jerlene
consultants:x:40000:djonasen,salves,dtyson
djonasen:x:1007:
salves:x:1008:
dtyson:x:1009:
```

```
[ec2-user@ip-192-0-2-1 ~]$ sudo passwd djonasen
Changing password for user djonasen.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[ec2-user@ip-192-0-2-1 ~]$ sudo passwd salves
Changing password for user salves.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[ec2-user@ip-192-0-2-1 ~]$ sudo passwd dtyson
Changing password for user dtyson.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

3. Check each user to confirm the account exists and has the correct group memberships.

   The exact UID numbers may differ for your solution, but the names of the users and groups should be correct, and **`consultants`** should have GID 40000.

```
[ec2-user@ip-192-0-2-1 ~]$ id djonasen
uid=1007(djonasen) gid=1007(djonasen) groups=1007(djonasen),40000(consultants)
[ec2-user@ip-192-0-2-1 ~]$ id salves
uid=1008(salves) gid=1008(salves) groups=1008(salves),40000(consultants)
[ec2-user@ip-192-0-2-1 ~]$ id dtyson
uid=1009(dtyson) gid=1009(dtyson) groups=1009(dtyson),40000(consultants)
```

4. This concludes this lab. Log out and stop your Amazon EC2 instance.