Chapter 6. Manage Local Users and Groups

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 through the `sudo` command. • Create, manage, and delete local user accounts. • Create, modify, and delete local group accounts. • Set a password management policy for users, and manually lock and unlock user accounts.
Sections	• User and Group Concepts (and Quiz) • Gain Superuser Access (and Guided Exercise) • Manage Local User Accounts (and Guided Exercise) • Manage Local Group Accounts (and Guided Exercise) • Manage User Passwords (and Guided Exercise)
Lab	Manage Local Users and Groups

Describe User and Group Concepts

Objectives

Describe the purpose of users and groups on a Linux system.

What Is a User?

A *user* account provides security boundaries between people and programs that can run commands.

Users have *usernames* to identify them to human users and for ease of working. Internally, the system distinguishes user accounts by the unique identification number, the user ID or *UID*, which is assigned to them. In most scenarios, if a human uses a user account, then the system assigns a secret *password* for the user to prove that they are the authorized user to log 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. With file ownership, the system enforces access control for users of the files. The user that is associated with a running process determines the files and directories that are accessible to that process.

User accounts are of the following main types: the *superuser*, *system users*, and *regular users*.

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

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 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
```

Use the `ls -l` command to view the owner of a file. Use the `ls -ld` command to view the owner of a directory, rather than the contents of that directory. In the following output, the third column shows the username.

```
[user01@host ~]$ ls -l mytextfile.txt

-rw-rw-r--. 1 user01 user01 0 Feb  5 11:10 mytextfile.txt

[user01@host]$ ls -ld Documents

drwxrwxr-x. 2 user01 user01 6 Feb  5 11:10 Documents
```

Use the `ps` command to view process information. The default is to show only processes in the current shell. Use the `ps` command `-a` option to view all processes with a terminal. Use the `ps` command `-u` option to view the user that is associated with a process. 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    1690  0.0  0.0 220984  1052 ttyS0  Ss+  22:43  0:00 /sbin/agetty -o -p -- \u
--keep-baud 1

user01  1769  0.0  0.1 377700  6844 tty2   Ssl+ 22:45  0:00 /usr/libexec/gdm-x-sessio
n --register-

user01  1773  1.3  1.3 528948 78356 tty2   Sl+  22:45  0:03 /usr/libexec/Xorg vt2 -di
splayfd 3 -au

user01  1800  0.0  0.3 521412 19824 tty2   Sl+  22:45  0:00 /usr/libexec/gnome-sessio
n-binary

user01  3072  0.0  0.0 224152  5756 pts/1  Ss   22:48  0:00 -bash

user01  3122  0.0  0.0 225556  3652 pts/1  R+   22:49  0:00 ps -au
```

The output of the preceding command displays users by name, but internally the operating system uses 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. The file is divided into seven colon-separated fields. An example of a line from `/etc/passwd` follows:

```
[user01@host ~]$ cat /etc/passwd
...output omitted...
user01:x:1000:1000:User One:/home/user01:/bin/bash
```

Consider each part of the code block, separated by a colon:

- **`user01`** : The username for this user.
- **`x`** : The user's encrypted password was historically stored here; it is now a placeholder.
- **`1000`** : The UID number for this user account.
- **`1000`** : The GID number for this user account's primary group. Groups are discussed later in this section.
- **`User One`** : A brief comment, description, or the real name for this user.
- **`/home/user01`** : The user's home directory, and the initial working directory when the login shell starts.
- **`/bin/bash`** : The default shell program for this user that runs at login. Some accounts use the `/sbin/nologin` shell to disallow interactive logins with that account.

What Is a Group?

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

Like users, groups have *group names* for easier recognition. Internally, the system distinguishes groups by the unique identification number, the *group ID* or *GID*, which is assigned to them. The mapping of group names to GIDs is defined in identity management 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. An example of a line from `/etc/group` follows:

```
[user01@host ~]$ cat /etc/group
...output omitted...
group01:x:10000:user01,user02,user03
```

Consider each part of the code block, separated by a colon:

- **`group01`** : Name for this group.
- **`x`** : Obsolete group password field; it is now a placeholder.
- **`10000`** : The GID number for this group (`10000`).
- **`user01,user02,user03`** : A list of users that are members of this group as a supplementary group.

Primary Groups and Supplementary Groups

Every user has exactly one primary group. For local users, this group is listed by GID in the `/etc/passwd` file. The primary group owns files that the user creates.

When a regular user is created, a group is created with the same name as the user, to be the primary group for the user. The user is the only member of this *User Private Group*. This group membership design simplifies the management of file permissions, to have user groups separated by default.

Users might also have *supplementary groups*. Membership in supplementary groups is stored in the `/etc/group` file. Users are granted access to files based on whether any of their groups have access, regardless of whether the groups are primary or supplementary. For example, if the `user01` user has a `user01` primary group and `wheel` and `webadmin` supplementary groups, then that user can read files that any of those three groups can read.

The `id` command can show group membership for a user. In the following example, the `user01` user has the `user01` group as their primary group (`gid`). The `groups` item lists all group memberships for this user, and the user also has the `wheel` and `group01` groups as supplementary groups.

```
[user01@host ~]$ id

uid=1001(user01) gid=1003(user01) groups=1003(user01),10(wheel),10000(group01) contex
t=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

References

`id`(1), `passwd`(5), and `group`(5) man pages `info libc` (*GNU C Library Reference Manual*)

- Section 30: Users and Groups

(The `glibc-devel` package must be installed for this `info` node to be available.)

Next

Quiz: Describe User and Group Concepts

Choose the correct answer to the following questions:

1.

2.

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

A	Primary user
B	[✓ UID]
C	GID
D	Username

3. CheckResetShow Solution

4.

5.

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

6. CheckResetShow Solution

7.

8.

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`]

9. CheckResetShow Solution

10.

11.

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`

12. CheckResetShow Solution

13.

14.

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

A	Primary group
B	UID
C	[✓ GID]
D	Groupid

15. CheckResetShow Solution

16.

17.

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`

18. CheckResetShow Solution

19.

20.

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

A	Home directory
B	UID
C	Login shell
D	[✓ Primary group]

21. CheckResetShow Solution

Previous Next

Gain Superuser Access

Objectives

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 a *superuser* that has all power over the system. In Red Hat Enterprise Linux, it is the `root` user. This user has the power to override normal privileges on the file system, and you can use it to manage and administer the system. For tasks such as installing or removing software, and to manage system files and directories, users must escalate their privileges to the `root` user.

Usually, only the `root` user can control most devices, but some exceptions apply. 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 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 account is compromised, then the system is in danger and you might lose administrative control. Red Hat encourages system administrators to log in always as a normal user, and to escalate privileges to `root` only when needed.

The `root` account on Linux is similar 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

Microsoft Windows users might be familiar with the practice of logging in as the local `Administrator` user to perform system administrator duties. Today, this practice is not recommended; users obtain privileges to perform administration by memberships in the `Administrators` group. Similarly in RHEL, Red Hat recommends that system administrators never log in directly as `root`. Instead, system administrators log in as a normal user and use mechanisms (`su`, `sudo`, or `PolicyKit`, for example) to temporarily gain superuser privileges.

When logged in as `root`, the entire desktop environment unnecessarily runs with administrative privileges. A security vulnerability that might normally compromise only a normal user account can potentially compromise the entire system.

Switch User Accounts

With the `su` command, users can switch to a different user account. If you run the `su` command from a regular user account with another user account as a parameter, then you must provide the password of the account to switch to. When the `root` user runs the `su` command, you do not need to enter the user's password.

This example uses the `su` command from the `user01` account to switch to the `user02` account:

```
[user01@host ~]$ su - user02
Password: user02_password
[user02@host ~]$
```

If you omit the username, then the `su` or `su -` command attempts to switch to `root` by default.

```
[user01@host ~]$ su -
Password: root_password
[root@host ~]#
```

The `su` command starts a *non-login shell*, whereas the `su -` command (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 is a new login as that user, whereas `su` starts a shell as that user, but uses the original user's environment settings.

Usually, 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 most frequent use for the `su` command is to get a command-line interface (shell prompt) that runs as another user, typically the `root` user. However, you can use the `su` command `-c` option to run an arbitrary program as another user. This behavior is similar to the Windows `runas` utility. Run `info su` to view more details.

Run Commands with Sudo

For security reasons, in some cases system administrators configure the `root` user not to have a valid password. Thus, users cannot log in to the system as `root` directly with a password. Moreover, you cannot use `su` to get an interactive shell. In this case, you can use the `sudo` command to get `root` access.

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

The next table summarizes the differences between the `su`, `su -`, and `sudo` commands:

	su	**su -**	**sudo**
Become new user	Yes	Yes	Per escalated command
Environment	Current user's	New user's	Current user's
Password required	New user's	New user's	Current user's
Privileges	Same as new user	Same as new user	Defined by configuration
Activity logged	`su` command only	`su` command only	Per escalated command

Additionally, you can configure the `sudo` command to allow specific users to run any command as some other user, or only some commands as that user. For example, if you configure the `sudo` command to allow the `user01` user to run the `usermod` command as `root`, then you can run the following command to lock or unlock a user account:

```
[user01@host ~]$ sudo usermod -L user02
[sudo] password for user01: user01_password
[user01@host ~]$ su - user02
Password: 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, then bash blocks the command, logs the attempt, and sends by default an email to the `root` user.

```
[user02@host ~]$ sudo tail /var/log/secure
[sudo] password for user02: user02_password
user02 is not in the sudoers file.  This incident will be reported.
[user02@host ~]$
```

Another benefit of `sudo` is to log by default all the executed commands to `/var/log/secure`.

```
[user01@host ~]$ sudo tail /var/log/secure
...output omitted...
Mar  9 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 later versions, all members of the `wheel` group can use `sudo` to run commands as any user, including `root`, by using their own password.

Warning

Historically, UNIX systems use membership of the `wheel` group to grant or control superuser access. RHEL 6 and earlier versions do not grant the `wheel` group any special privileges by default. System administrators who previously used this group for a non-standard purpose must update a previous configuration, to prevent unexpected and unauthorized users from obtaining administrative access on RHEL 7 and later systems.

Get an Interactive Root Shell with Sudo

To access the `root` account with `sudo`, use the `sudo -i` command. This command switches to the `root` account and runs that user's default shell (usually `bash`) and associated interactive login scripts. To run the shell without the interactive scripts, use the `sudo -s` command.

For example, an administrator can get an interactive shell as `root` on an AWS Elastic Cloud Computing (EC2) instance by using SSH public-key authentication to log in as the `ec2-user` normal user. Then run the `sudo -i` command to access the `root` user's shell.

```
[ec2-user@host ~]$ sudo -i
[sudo] password for ec2-user: ec2-user_password
[root@host ~]#
```

Configure sudo

The `/etc/sudoers` file is the main configuration file for the `sudo` command. To avoid problems if multiple administrators try to edit the file at the same time, you can edit it only with the special `visudo` command. The `visudo` editor also validates the file, to ensure no syntax errors.

For example, the following line from the `/etc/sudoers` file enables `sudo` access for `wheel` group members:

```
%wheel        ALL=(ALL:ALL)       ALL
```

- The `%wheel` string is the user or group that the rule applies to. The `%` symbol before the `wheel` word specifies a group.
- The `ALL=(ALL:ALL)` command specifies that on any host with this file (the first `ALL`), users in the `wheel` group can run commands as any other user (the second `ALL`) and as any other group (the third `ALL`) on the system.
- The final `ALL` command specifies that users in the `wheel` group can run any command.

By default, the `/etc/sudoers` file also includes the contents of any files in the `/etc/sudoers.d` directory as part of the configuration file. With this hierarchy, you can add `sudo` access for a user by putting an appropriate file in that directory.

Note

You can enable or disable `sudo` access by copying a file into the directory or removing it from the directory.

In this course, you 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 `user01` user, you can create the `/etc/sudoers.d/user01` file with the following content:

```
user01        ALL=(ALL)       ALL
```

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

```
%group01        ALL=(ALL)       ALL
```

To enable users in the `games` group to run the `id` command as the `operator` user, you can create the `/etc/sudoers.d/games` file with the following content:

```
%games ALL=(operator) /bin/id
```

You can also set up `sudo` to allow a user to run commands as another user without entering their password, by using the `NOPASSWD: ALL` command:

```
ansible        ALL=(ALL)       NOPASSWD: ALL
```

Although obvious security risks apply to granting this level of access to a user or group, system administrators often use this approach with cloud instances, virtual machines, and provisioning systems for configuring servers. You must protect the account with this access and require SSH public-key authentication for a user on a remote system to access it at all.

For example, the official Amazon Machine Image (AMI) for Red Hat Enterprise Linux in the Amazon Web Services Marketplace ships with the `root` and the `ec2-user` passwords locked. The `ec2-user` account is set up to allow remote interactive access through SSH public-key authentication. The `ec2-user` 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
```

You can re-enable the requirement to enter a password for `sudo`, or introduce other changes to tighten security as part of the system configuration.

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

(The `glibc-doc` package must be installed for this `info` node to be available.)

Previous Next

Guided Exercise: Gain Superuser Access

In this exercise, you practice switcdhing to the `root` account and running commands as `root`.

Outcomes

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

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start users-superuser
```

Instructions

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

```
[student@workstation ~]$ ssh student@servera
...output omitted...
```

```
[student@servera ~]$
```

4. Explore the shell environment of the `student` user. 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 executable files.
    1. Run `id` to view the current user and group information.

    ```
    2. [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
    ```

    3. Run `pwd` to display the current working directory.

    ```
    4. [student@servera ~]$ pwd
    ```

    ```
    /home/student
    ```

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

    ```
    6. [student@servera ~]$ echo $HOME
    7. /home/student
    8. [student@servera ~]$ echo $PATH
    ```

    ```
    /home/student/.local/bin:/home/student/bin:/usr/local/bin:/usr/bin:/usr/lo
    cal/sbin:/usr/sbin
    ```

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

    ```
    2. [student@servera ~]$ sudo su
    3. [sudo] password for student: student
    ```

    ```
    [root@servera student]#
    ```

    4. Run `id` to view the current user and group information.

    ```
    5. [root@servera student]# id
    ```

```
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:u
nconfined_t:s0-s0:c0.c1023
```

6. Run `pwd` to display the current working directory.

```
7. [root@servera student]# pwd
```

```
/home/student
```

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

```
9.  [root@servera student]# echo $HOME
10. /root
11. [root@servera student]# echo $PATH
```

```
/root/.local/bin:/root/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/
usr/local/bin
```

When you use the `su` command to become the `root` user, you do not keep the current path of the `student` user. As you can see in the next step, the path is not the `root` user path either.

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

12. Exit the `root` user's shell to return to the `student` user's shell.

```
13. [root@servera student]# exit
14. exit
```

```
[student@servera ~]$
```

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

The `sudo` command might or might 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 authenticated to `sudo` within the last five minutes, then the `sudo` command does not prompt you for the password. If more than five minutes elapsed since you authenticated to `sudo`, then you must enter `student` as the password for authentication to `sudo`.

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

[root@servera ~]#
```

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

2. Run `id` to view the current user and group information.

```
3. [root@servera ~]# id
```

```
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:u
nconfined_t:s0-s0:c0.c1023
```

4. Run `pwd` to display the current working directory.

```
5. [root@servera ~]# pwd
```

```
/root
```

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

```
7. [root@servera ~]# echo $HOME

8. /root

9. [root@servera ~]# echo $PATH
```

```
/root/.local/bin:/root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
in
```

As in the preceding step, after the `sudo` command resets the `PATH` variable from the settings in the `student` user's shell environment, the `su -` command runs the shell login scripts for `root` and sets the `PATH` variable to yet another value. The `su` command without the dash (-) option does not have the same behavior.

10. Exit the `root` user's shell to return to the `student` user's shell.

```
[root@servera ~]# exit
logout
```

```
[student@servera ~]$
```

7. Verify that the `operator1` user can run any command as any user by using the `sudo` command.

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

```
operator1 ALL=(ALL) ALL
```

9. Become the `operator1` user and view the contents of the `/var/log/messages` file. Copy the `/etc/motd` file to `/etc/motdOLD`. Remove the `/etc/motdOLD` file. As these operations require administrative rights, use the `sudo` command to run those commands as the superuser. Do not switch to root by using `sudo su` or `sudo su -`. Use `redhat` as the password of the `operator1` user.
   1. Switch to the `operator1` user.

```
[student@servera ~]$ su - operator1
Password: redhat
```

```
[operator1@servera ~]$
```

   4. Try to view the last five lines of `/var/log/messages` without using `sudo`. It should fail.

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

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

   6. Try to view the last five lines of `/var/log/messages` by using `sudo`. It should succeed.

```
[operator1@servera ~]$ sudo tail -5 /var/log/messages
[sudo] password for operator1: redhat
Mar 9 15:53:36 servera su[2304]: FAILED SU (to operator1) student on pts/1
Mar 9 15:53:51 servera su[2307]: FAILED SU (to operator1) student on pts/1
Mar 9 15:53:58 servera su[2310]: FAILED SU (to operator1) student on pts/1
```

```
12. Mar 9 15:54:12 servera su[2322]: (to operator1) student on pts/1
```

```
Mar 9 15:54:25 servera su[2353]: (to operator1) student on pts/1
```

Note

The preceding output might differ on your system.

13. Try to copy `/etc/motd` as `/etc/motdOLD` without using `sudo`. It should fail.

```
14. [operator1@servera ~]$ cp /etc/motd /etc/motdOLD
```

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

15. Try to copy `/etc/motd` as `/etc/motdOLD` by using `sudo`. It should succeed.

```
16. [operator1@servera ~]$ sudo cp /etc/motd /etc/motdOLD
```

```
[operator1@servera ~]$
```

17. Try to delete `/etc/motdOLD` without using `sudo`. It should fail.

```
18. [operator1@servera ~]$ rm /etc/motdOLD
19. rm: remove write-protected regular empty file '/etc/motdOLD'? y
20. rm: cannot remove '/etc/motdOLD': Permission denied
```

```
[operator1@servera ~]$
```

21. Try to delete `/etc/motdOLD` by using `sudo`. It should succeed.

```
22. [operator1@servera ~]$ sudo rm /etc/motdOLD
```

```
[operator1@servera ~]$
```

23. Return to the `workstation` system as the `student` user.

```
24. [operator1@servera ~]$ exit
25. logout
26. [student@servera ~]$ exit
27. logout
28. Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish users-superuser
```

This concludes the section.

Previous Next

Manage Local User Accounts

Objectives

Create, modify, and delete local user accounts.

Manage Local Users

You can use command-line tools to manage local user accounts. This section reviews some important tools.

Create Users from the Command Line

The `useradd` *`username`* command creates a user called `username`. It sets up the user's home directory and account information, and creates a private group for the user called `username`. At this point, a valid password is not set for the account, and the user cannot log in until a password is set.

The `useradd --help` command displays the basic options to override the defaults. Usually, you can use the same options with the `usermod` command to modify an existing user.

The `/etc/login.defs` file sets some default options for user accounts, such as the range of valid UID numbers and default password aging rules. The values in this file affect only newly created user accounts. A change to this file does not affect existing users.

In Red Hat Enterprise Linux 9, the `useradd` command assigns new users the first free UID that is greater than or equal to 1000, unless you explicitly specify a UID by using the `-u` option.

Modify Existing Users from the Command Line

The `usermod --help` command displays the options to modify an account. Some common options are as follows:

`usermod` options:	**Usage**
`-a, --append`	Use it 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.
`-c, --comment COMMENT`	Add the `COMMENT` text to the comment field.
`-d, --home HOME_DIR`	Specify a home directory for the user account.
`-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.
`-L, --lock`	Lock the user account.
`-m, --move-home`	Move the user's home directory to a new location. You must use it with the `-d` option.
`-s, --shell SHELL`	Specify a particular login shell for the user account.
`-U, --unlock`	Unlock the user account.

Delete Users from the Command Line

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

Warning

When you remove a user without specifying the `userdel -r` option, an unassigned UID now owns the user's files. If you create a user and that user is assigned the deleted user's UID, then the new account owns those files, which is a security risk. Typically, organization security policies disallow deleting user accounts, and instead lock them from being used, to avoid this scenario.

The following example demonstrates how this scenario can lead to information leakage:

```
[root@host ~]# useradd user01
[root@host ~]# ls -l /home
drwx------. 3 user01  user01    74 Mar  4 15:22 user01
[root@host ~]# userdel user01
[root@host ~]# ls -l /home
drwx------. 3    1000    1000   74 Mar  4 15:22 user01
[root@host ~]# useradd -u 1000 user02
[root@host ~]# ls -l /home
drwx------. 3 user02     user02       74 Mar  4 15:23 user02
drwx------. 3 user02     user02       74 Mar  4 15:22 user01
```

Notice that `user02` now owns all files that `user01` previously owned. The `root` user can use the `find / -nouser -o -nogroup` command to find all unowned files and directories.

Set Passwords from the Command Line

The `passwd` *`username`* command sets the initial password or changes the existing password for the `username` user. The `root` user can set a password to any value. The terminal displays a message if the password does not meet the minimum recommended criteria, but then you can retype the new password and the `passwd` command updates it successfully.

```
[root@host ~]# passwd user01
Changing password for user user01.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully.
```

```
[root@host ~]#
```

A regular user must choose a password at least eight characters long. Do not use a dictionary word, the username, or the previous password.

UID Ranges

Red Hat Enterprise Linux uses specific UID numbers and ranges of numbers for specific purposes.

- **UID 0** : The superuser (`root`) account UID.
- **UID 1-200** : System account UIDs that are statically assigned to system processes.
- **UID 201-999** : UIDs that are assigned to system processes that do not own files on this system. Software that requires an unprivileged UID is dynamically assigned a UID from this available pool.
- **UID 1000+** : The UID range to assign to regular, unprivileged users.

Note

RHEL 6 and earlier versions use UIDs in the range 1-499 for system users and UIDs higher than 500 for regular users. You can change the `useradd` and `groupadd` default ranges in the `/etc/login.defs` file.

References

`useradd`(8), `usermod`(8), and `userdel`(8) man pages

Previous Next

Guided Exercise: Manage Local User Accounts

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

Outcomes

- Configure a Linux system with additional user accounts.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start users-user
```

Instructions

1. From `workstation`, open an SSH session to `servera` as the `student` user, and switch to the `root` user.

```
2. [student@workstation ~]$ ssh student@servera
3. ...output omitted...
4. [student@servera ~]$ sudo -i
5. [sudo] password for student: student
```

```
[root@servera ~]#
```

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

```
7. [root@servera ~]# useradd operator1
8. [root@servera ~]# tail /etc/passwd
9. ...output omitted...
```

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

10. Set the password for `operator1` to `redhat`.

```
11.[root@servera ~]# passwd operator1
12.Changing password for user operator1.
13.New password: redhat
14.BAD PASSWORD: The password is shorter than 8 characters
15.Retype new password: redhat
```

```
passwd: all authentication tokens updated successfully.
```

16. Create the additional `operator2` and `operator3` users. 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.
```

    8. 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.
```

17. Update the `operator1` and `operator2` user accounts to include the `Operator One` and `Operator Two` comments, respectively. Verify that the comments exist for the user accounts.
    1. Run the `usermod -c` command to update the comments of the `operator1` user account.

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

    2. Run the `usermod -c` command to update the comments of the `operator2` user account.

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

    3. View the `/etc/passwd` file to confirm that the comments for each of the `operator1` and `operator2` users exist.

```
[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
```

18. Delete the `operator3` user along with any personal data of the user. Confirm that the `operator3` does not exist.
    1. Remove the `operator3` user from the system.

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

    2. Confirm that the `operator3` user does not exist.

    ```
    [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`.

    6. Confirm that the `operator3` user home directory does not exist.

    ```
    [root@servera ~]# ls -l /home
    total 0
    drwx------. 4 devops    devops    90 Mar  3 09:59 devops
    drwx------. 2 operator1 operator1 62 Mar  9 10:19 operator1
    drwx------. 2 operator2 operator2 62 Mar  9 10:19 operator2
    ```

    ```
    drwx------. 3 student   student   95 Mar  3 09:49 student
    ```

    12. Exit the `root` user's shell to return to the `student` user's shell.

    ```
    [root@servera ~]# exit
    logout
    ```

```
[student@servera ~]$
```

15. Log off from the `servera` machine.

```
16. [student@servera ~]$ exit
17. logout
18. Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish users-user
```

This concludes the section.

Previous Next

Manage Local Group Accounts

Objectives

Create, modify, and delete local group accounts.

Manage Local Groups

Several command-line tools enable group management. Although you can use the **Users** GUI utility to manage groups, Red Hat recommends that you use command-line tools.

Create Groups from the Command Line

The `groupadd` command creates groups. Without options, the `groupadd` command uses the next available GID from the range that the `GID_MIN` and `GID_MAX` variables specify in the `/etc/login.defs` file. By default, the command assigns a GID value that is greater than any other existing GIDs, even if a lower value becomes available.

The `groupadd` command `-g` option specifies a GID for the group to use.

```
[root@host ~]# groupadd -g 10000 group01
[root@host ~]# tail /etc/group
...output omitted...
group01:x:10000:
```

Note

Because of the automatic creation of user private groups (GID 1000+), some administrators set aside a separate range of GIDs for creating supplementary groups for other purposes. However, this extra management is unnecessary, because a user's UID and primary GID do not need to be the same number.

The `groupadd` command `-r` option creates system groups. As with normal groups, system groups use a GID from the range of listed valid system GIDs in the `/etc/login.defs` file. The `SYS_GID_MIN` and `SYS_GID_MAX` configuration items in the `/etc/login.defs` file define the range of system GIDs.

```
[root@host ~]# groupadd -r group02
[root@host ~]# tail /etc/group
...output omitted...
group01:x:10000:
group02:x:988:
```

Modify Existing Groups from the Command Line

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

```
[root@host ~]# groupmod -n group0022 group02
[root@host ~]# tail /etc/group
...output omitted...
group0022:x:988:
```

Notice that the group name updates to `group0022` from `group02`.
The `groupmod` command `-g` option specifies a new GID.

```
[root@host ~]# groupmod -g 20000 group0022
[root@host ~]# tail /etc/group
...output omitted...
group0022:x:20000:
```

Notice that the GID changes to `20000` from `988`.

Delete Groups from the Command Line

The `groupdel` command removes groups.

```
[root@host ~]# groupdel group0022
```

Note

You cannot remove a group if it is the primary group of an existing user. Similar to using the `userdel` command, ensure first that you locate files that the group owns.

Change 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.

```
[root@host ~]# id user02
uid=1006(user02) gid=1008(user02) groups=1008(user02)
[root@host ~]# usermod -g group01 user02
[root@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.

```
[root@host ~]# id user03
uid=1007(user03) gid=1009(user03) groups=1009(user03)
[root@host ~]# usermod -aG group01 user03
[root@host ~]# id user03
uid=1007(user03) gid=1009(user03) groups=1009(user03),10000(group01)
```

Important

The `usermod` command `-a` option enables the *append* mode. Without the `-a` option, the command removes the user from any of their current supplementary groups that are not included in the `-G` option's list.

Compare Primary and Supplementary Group Membership

A user's primary group is the group that is viewed on the user's account in the `/etc/passwd` file. A user can belong to only one primary group at a time.

A user's supplementary groups are the additional groups that are configured for the user and viewed on the user's entry in the `/etc/group` file. A user can belong to as many supplementary groups as is necessary to implement file access and permissions effectively.

For configuring group-based file permissions, no difference exists between a user's primary and supplementary groups. If the user belongs to any group that is assigned access to specific files, then that user has access to those files.

The only distinction between a user's primary and supplementary memberships is when a user creates a file. New files must have a user owner and a group owner, which is assigned when the file is created. The user's primary group is used for the new file's group ownership, unless command options override it.

Temporarily Change Your Primary Group

Only a user's primary group is used for new file creation attributes. However, you can temporarily switch your primary group to a supplementary group that you already belong to. You might switch if you are about to create files, manually or scripted, and want to assign a different group as owner when they are being created.

Use the `newgrp` command to switch your primary group, in this shell session. You can switch between any primary or supplementary group that you belong to, but only one group at a time can be primary. Your primary group returns to the default if you log out and log in again. In this example, the `group01` group temporarily becomes this user's primary group.

```
[user03@host ~]# id
uid=1007(user03) gid=1009(user03) groups=1009(user03),10000(group01)
[user03@host ~]$ newgrp group01
```

```
[user03@host ~]# id
uid=1007(user03) gid=10000(group01) groups=1009(user03),10000(group01)
```

References

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

Previous Next

Guided Exercise: Manage Local Group Accounts

In this exercise, you 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

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

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates the necessary user accounts to set up the environment correctly.

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

Instructions

1. From `workstation`, open an SSH session to `servera` as the `student` user and switch to the `root` user.

```
2. [student@workstation ~]$ ssh student@servera
3. ...output omitted...
4. [student@servera ~]$ sudo -i
5. [sudo] password for student: student
```

```
[root@servera ~]#
```

6. Create the `operators` supplementary group with a GID of 30000.

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

7. Create the `admin` supplementary group without specifying a GID.

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

8. Verify that both the `operators` and `admin` supplementary groups exist.

```
9.  [root@servera ~]# tail /etc/group
10. ...output omitted...
11. operators:x:30000:
```

```
admin:x:30001:
```

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

    ```
    2.  [root@servera ~]# usermod -aG operators operator1
    3.  [root@servera ~]# usermod -aG operators operator2
    ```

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

    4. Confirm that the users are in the group.

    ```
    5.  [root@servera ~]# id operator1
    6.  uid=1002(operator1) gid=1002(operator1) groups=1002(operator1),30000(ope
        rators)
    7.  [root@servera ~]# id operator2
    8.  uid=1003(operator2) gid=1003(operator2) groups=1003(operator2),30000(ope
        rators)
    9.  [root@servera ~]# id operator3
    ```

    ```
    uid=1004(operator3) gid=1004(operator3) groups=1004(operator3),30000(ope
    rators)
    ```

13. Ensure that the `sysadmin1`, `sysadmin2`, and `sysadmin3` users belong to the `admin` group. Enable administrative rights for all the `admin` group

members. Verify that any member of the `admin` group can run administrative commands.

1. Add the `sysadmin1`, `sysadmin2`, and `sysadmin3` users to the `admin` group.

```
2. [root@servera ~]# usermod -aG admin sysadmin1
3. [root@servera ~]# usermod -aG admin sysadmin2
```

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

4. Confirm that the users are in the group.

```
5. [root@servera ~]# id sysadmin1
6. uid=1005(sysadmin1) gid=1005(sysadmin1) groups=1005(sysadmin1),30001(adm
   in)
7. [root@servera ~]# id sysadmin2
8. uid=1006(sysadmin2) gid=1006(sysadmin2) groups=1006(sysadmin2),30001(adm
   in)
9. [root@servera ~]# id sysadmin3
```

```
uid=1007(sysadmin3) gid=1007(sysadmin3) groups=1007(sysadmin3),30001(adm
in)
```

10. Examine the `/etc/group` file to verify the supplementary group memberships.

```
11. [root@servera ~]# tail /etc/group
12. ...output omitted...
13. operators:x:30000:operator1,operator2,operator3
```

```
admin:x:30001:sysadmin1,sysadmin2,sysadmin3
```

14. Create the `/etc/sudoers.d/admin` file so that the members of the `admin` group have full administrative privileges.

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

15. Switch to the `sysadmin1` user (a member of the `admin` group) and verify that you can run a `sudo` command.

```
16. [root@servera ~]# su - sysadmin1
17. [sysadmin1@servera ~]$ sudo cat /etc/sudoers.d/admin
```

```
18. [sudo] password for sysadmin1: redhat
```

```
%admin ALL=(ALL) ALL
```

19. Return to the `workstation` machine as the `student` user.

```
20. [sysadmin1@servera ~]$ exit
21. logout
22. [root@servera ~]# exit
23. logout
24. [student@servera ~]$ exit
25. logout
26. Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

This concludes the section.

Previous Next

Manage User Passwords

Objectives

Set a password management policy for users, and manually lock and unlock user accounts.

Shadow Passwords and Password Policy

Originally, encrypted passwords were stored in the world-readable `/etc/passwd` file. These passwords were considered adequate until dictionary attacks on encrypted passwords became common. The cryptographically hashed passwords were moved to the `/etc/shadow` file, which only the `root` user can read.

Like the `/etc/passwd` file, each user has an entry with in the `/etc/shadow` file. An example entry from the `/etc/shadow` file has nine colon-separated fields:

```
[root@host ~]# cat /etc/shadow
...output omitted...
user03:$6$CSsXsd3rwghsdfarf:17933:0:99999:7:2:18113:
```

Each field of this code block is separated by a colon:

- **`user03`** : Name of the user account.
- **`$6$CSsXsd3rwghsdfarf`** : The cryptographically hashed password of the user.
- **`17933`** : The days from the epoch when the password was last changed, where the epoch is `1970-01-01` in the UTC time zone.
- **`0`** : The minimum days since the last password change before the user can change it again.
- **`99999`** : The maximum days without a password change before the password expires. An empty field means that the password never expires.
- **`7`** : The number of days ahead to warn the user that their password will expire.
- **`2`** : The number of days without activity, starting with the day that the password expired, before the account is automatically locked.
- **`18113`** : The day when the account expires in days since the epoch. An empty field means that the account never expires.
- The last field is typically empty and is reserved for future use.

Format of an Cryptographically Hashed Password

The cryptographically hashed password field stores three pieces of information: the hashing algorithm in use, the *salt*, and the cryptographical hash. Salt adds random data to the cryptographical hash, for creating a unique hash to strengthen the cryptographically hashed password. Each piece of information is delimited by the dollar (`$`) character.

```
$6$CSsXcYG1L/4ZfHr/$2W6evvJahUfzfHpc9X.45Jc6H30E
```

- **`6`** : The hashing algorithm in use for this password. A `6` indicates a SHA-512 hash, the RHEL 9 default; a `1` indicates MD5; and a `5` indicates SHA-256.
- **`CSsXcYG1L/4ZfHr/`** : The salt in use to cryptographically hash the password; it is originally chosen at random.
- **`2W6evvJahUfzfHpc9X.45Jc6H30E`** : The cryptographical hash of the user's password; combining the salt and the plain text password and then cryptographically hashing to generate the password hash.

The primary reason to combine a salt with the password is to defend against attacks that use pre-computed lists of password hashes. Adding salts changes the resulting hashes, which makes the pre-computed list useless. If an attacker obtains a copy of an `/etc/shadow` file that uses salts, then they need to guess passwords with brute force, which requires more time and effort.

Password Verification

When a user tries to log in, the system looks up the entry for the user in the `/etc/shadow` file, and combines the salt for the user with the plain text typed password. The system then cryptographically hashes the combination of the salt and plain text password with the specified hashing algorithm. If the result matches the cryptographical hash, then the user typed the right password. If the result does not match the cryptographical hash, then the user typed the wrong password and the login attempt fails. This method enables the system to determine whether the user typed the correct password without storing that password in a usable form for logging in.

Configure Password Aging

The following diagram shows the relevant password aging parameters, which can be adjusted by using the `chage` command to implement a password aging policy. Notice that the command name is `chage`, which stands for "change age". Do not confuse the command with the word "change".

Figure 6.1: Password aging parameters

The following example demonstrates the `chage` command to change the password policy of the `sysadmin05` user. The command defines a minimum age (`-m`) of zero days, a maximum age (`-M`) of 90 days, a warning period (`-W`) of 7 days, and an inactivity period (`-I`) of 14 days.

```
[root@host ~]# chage -m 0 -M 90 -W 7 -I 14 sysadmin05
```

Assume that you manage the user password policies on a Red Hat server. The `cloudadmin10` user is new in the system, and you want to set a custom password aging policy. You want to set the account expiration 30 days from today, so you use the following commands:

```
[root@host ~]# date +%F
2022-03-10

[root@host ~]# date -d "+30 days" +%F
2022-04-09

[root@host ~]# chage -E $(date -d "+30 days" +%F) cloudadmin10

[root@host ~]# chage -l cloudadmin10 | grep "Account expires"
Account expires                                         : Apr 09, 2022
```

Use the `date` command to get the current date.

Use the `date` command to get the date 30 days from now.

Use the `chage` command `-E` option to change the expiration date for the `cloudadmin10` user.

Use the `chage` command `-l` option to display the password aging policy for the `cloudadmin10` user.

After a few days, you notice in the `/var/log/secure` log file that the `cloudadmin10` user has a strange behavior. The user tried to use `sudo` to interact with files that belong to other users. You suspect that the user might have left an `ssh` session open when working in another machine. You want the `cloudadmin10` user to change the password on the next login, so you use the following command:

```
[root@host ~]# chage -d 0 cloudadmin10
```

The next time that the `cloudadmin10` user logs in, the user is prompted to change the password.

Note

The `date` command can calculate a future date. The `-u` option reports the time in UTC.

```
[user01@host ~]$ date -d "+45 days" -u
Thu May 23 17:01:20 UTC 2019
```

You can change the default password aging configuration in the `/etc/login.defs` file. The `PASS_MAX_DAYS` and `PASS_MIN_DAYS` options set the default maximum and minimum age of the password respectively. The `PASS_WARN_AGE` sets the default warning period of the password. Any change in the default password aging policies affects users that are created after the change. The existing users continue to use the earlier password aging settings rather than the later ones. For more information about the `/etc/login.defs` file, refer to the *Red Hat Security: Linux in Physical, Virtual, and Cloud* (RH415) course and the `login.defs(5)` man page.

Restrict Access

You can use the `usermod` command to modify account expiration for a user. For example, the `usermod` command `-L` option locks a user account and the user cannot log in to the system.

```
[root@host ~]# usermod -L sysadmin03
[user01@host ~]$ su - sysadmin03
Password: redhat
su: Authentication failure
```

If a user leaves the company on a certain date, then you can lock and expire the account with a single `usermod` command. The date must be the number of days since `1970-01-01`, or else use the *YYYY-MM-DD* format. In the following example, the `usermod` command locks and expires the `cloudadmin10` user at `2022-08-14`.

```
[root@host ~]# usermod -L -e 2022-08-14 cloudadmin10
```

When you lock an account, you prevent the user from authenticating with a password to the system. This method is recommended to prevent access to an account by a former employee of the company. Use the `usermod` command -`U` option to enable the access to the account again.

The nologin Shell

The `nologin` shell acts as a replacement shell for the user accounts that are not intended to log in interactively to the system. It is good security practice to disable an account from logging in to the system when the account does not require it. For example, a mail server might require an account to store mail and a password for the user to authenticate with a mail client to retrieve mail. That user does not need to log in directly to 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, then the `nologin` shell closes the connection.

```
[root@host ~]# usermod -s /sbin/nologin newapp
[root@host ~]# su - newapp
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), and `login.defs`(5) man pages.

Previous Next

Guided Exercise: Manage User Passwords

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

Outcomes

- 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.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start users-password
```

Instructions

1. From `workstation`, open an SSH session as the `student` user to the `servera` machine.

```
2. [student@workstation ~]$ ssh student@servera
```

```
[student@servera ~]$
```

3. On `servera`, use the `usermod` command to lock and unlock the `operator1` user.
    1. As the `student` user, use administrative rights to lock the `operator1` account.

    ```
    2. [student@servera ~]$ sudo usermod -L operator1
    ```

    ```
    [sudo] password for student: student
    ```

    3. Try to log in as `operator1`. This command should fail.

    ```
    4. [student@servera ~]$ su - operator1
    5. Password: redhat
    ```

```
su: Authentication failure
```

6. Unlock the `operator1` account.

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

7. Try to log in as `operator1` again. This time, the command should succeed.

```
8. [student@servera ~]$ su - operator1
9. Password: redhat
10. ...output omitted...
```

```
[operator1@servera ~]$
```

11. Log out of the `operator1` user shell to return to the `student` user shell.

```
12. [operator1@servera ~]$ exit
```

```
logout
```

4. Change the password policy for the `operator1` user to require a new password every 90 days. Confirm that the password age is successfully set.
    1. Switch to the `root` user.

```
2. [student@servera ~]$ sudo -i
3. [sudo] password for student: student
```

```
[root@servera ~]#
```

4. Set the maximum age of the `operator1` user's password to 90 days.

```
[root@servera ~]# chage -M 90 operator1
```

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

```
6. [root@servera ~]# chage -l operator1
7. Last password change      : Mar 10, 2022
8. Password expires          : Jun 10, 2022
9. 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
```

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

```
[root@servera ~]# chage -d 0 operator1
```

6. Exit as the `root` user from the `servera` machine.

```
[root@servera ~]# exit
logout
```

```
[student@servera ~]$
```

9. 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 enfo
   rced)
   Current password: redhat
   New password: forsooth123
   Retype new password: forsooth123
   ...output omitted...
   ```

   ```
   [operator1@servera ~]$
   ```

   9. Exit the `operator1` user's shell to return to the `student` user and then switch to the `root` user.

   ```
   [operator1@servera ~]$ exit
   logout
   [student@servera ~]$ sudo -i
   ```

```
[sudo] password for student: student
```

```
[root@servera ~]#
```

10. Set the `operator1` account to expire 180 days from the current day.
    1. Determine a date 180 days in the future. Use the format `%F` with the `date` command to get the exact value. This returned date is an example; use the value on your system for the steps after this one.

```
[root@servera ~]# date -d "+180 days" +%F
```

```
2022-09-06
```

    3. Set the account to expire on the date that is displayed in the preceding step. For example:

```
[root@servera ~]# chage -E 2022-09-06 operator1
```

    4. Verify that the account expiry date is successfully set.

```
[root@servera ~]# chage -l operator1
Last password change      : Mar 10, 2022
Password expires          : Jun 10, 2022
Password inactive         : never
Account expires           : Sep 06, 2022
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
```

11. 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 you open the file with the text editor. You can use the `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_WARN_AGE   7
```

```
...output omitted...
```

Important

The default password and account expiry settings apply to new users but not to existing users.

16. Return to the `workstation` system as the `student` user.

```
[root@servera ~]# exit
logout
[student@servera ~]$ exit
logout
Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish users-password
```

This concludes the section.

Previous Next

Summary

- The user account types in Linux are the superuser, system users, and regular users.
- A user has a primary group and might be a member of supplementary groups.
- The `/etc/passwd`, `/etc/group`, and `/etc/shadow` critical files contain user and group information.
- You can run commands as the superuser with the `su` and `sudo` commands.
- The `useradd`, `usermod`, and `userdel` commands manage users.
- The `groupadd`, `groupmod`, and `groupdel` commands manage groups.
- The `passwd` command manages passwords for users.
- The `chage` command displays and configures password expiration settings for users.

Previous Next

Chapter 7. Control Access to Files

Abstract

Goal	Set Linux file-system permissions on files and interpret the security effects of different permission settings.

Objectives	• List file-system permissions on files and directories, and interpret the effects of those permissions on access by users and groups. • Change the permissions and ownership of files with command-line tools. • Control the default permissions of user-created files, explain the effects of special permissions, and use special and default permissions to set the group owner of files that are created in a directory.
Sections	• Interpret Linux File System Permissions (and Quiz) • Manage File System Permissions from the Command Line (and Guided Exercise) • Manage Default Permissions and File Access (and Guided Exercise)
Lab	Control Access to Files

Interpret Linux File System Permissions

Objectives

List file system permissions on files and directories, and interpret the effect of those permissions on access by users and groups.

Linux File-system Permissions

File permissions control access to files. Linux file permissions are flexible and can handle most normal permission cases.

Files have three user categories that permissions apply to. The file is owned by a user, normally the file creator. The file is also owned by a single group, which is usually the primary group of the user who created the file, but you can change it.

You can set different permissions for the owning user (*user* permissions), the owning group (*group* permissions), and for all other users on the system that are not the user or a member of the owning group (*other* permissions).

The most specific permissions take precedence. User permissions override group permissions, which override other permissions.

As an example of how group membership enables collaboration between users, imagine that your system has two users: `alice` and `bob`. `alice` is a member of the `alice` and `web` groups, and `bob` is a member of the `bob`, `wheel`, and `web` groups.

When `alice` and `bob` work together, the files should be associated with the `web` group, and group permissions should allow both users to have access to the files.

Three permission categories apply: read, write, and execute. The following table explains how these permissions affect access to files and directories.

Table 7.1. Effects of Permissions on Files and Directories

Permission	Effect on files	Effect on directories
`r` (read)	File contents can be read.	Contents of the directory (the file names) can be listed.
`w` (write)	File contents can be changed.	Any file in the directory can be created or deleted.
`x` (execute)	Files can be executed as commands.	The directory can become the current working directory. You can run the `cd` command to it, but it also requires read permission to list files there.

Users normally have both read and execute permissions on read-only directories so that they can list the directory and have read-only access to all of its contents. If a user has only read access on a directory, then they can list the names of the files in it. However, the user cannot access other information, such as permissions or time stamps. If a user has only execute access on a directory, then they cannot list file names in the directory. If they know the name of a file that they have permission to read, then they can access the contents of that file from outside the directory by explicitly specifying the relative file name.

Anyone who owns or has write permissions to a directory can remove files on it, regardless of the ownership or permissions on the file itself. You can override this behavior by using the *sticky bit* permission, to be discussed later in this chapter.

Note

Linux file permissions work differently from the Microsoft NTFS file-system permissions. On Linux, permissions apply only to the file or directory on which they are set. The subdirectories within a directory do not automatically inherit the parent directory's permissions. However, directory permissions can block access to the directory contents, if they are set restrictively.

The `read` permission on a Linux directory is similar to **List folder contents** in Windows. The `write` permission on a Linux directory is similar to **Modify** in Windows; it implies the ability to delete files and subdirectories. In Linux, with `write` permissions and the **sticky bit** on a directory, then only the user or group owner can delete files, which is similar to the Windows **Write** behavior.

The Linux `root` user has the equivalent of the Windows **Full Control** permission on all files. However, SELinux policy can use process and file security contexts to restrict access even to the `root` user. SELinux is discussed in the *Red Hat System Administration II* (RH134) course.

View File and Directory Permissions and Ownership

The `ls` command `-l` option shows detailed information about permissions and ownership:

```
[user@host ~]$ ls -l test
-rw-rw-r--. 1 student student 0 Mar  8 17:36 test
```

Use the `ls` command `-d` option to show detailed information about a directory itself, and not its contents.

```
[user@host ~]$ ls -ld /home
drwxr-xr-x. 5 root root 4096 Feb 31 22:00 /home
```

The first character of the long listing is the file type, and is interpreted as follows:

- **`-`** is a regular file.
- **`d`** is a directory.
- **`l`** is a symbolic link.
- **`c`** is a character device file.
- **`b`** is a block device file.
- **`p`** is a named pipe file.
- **`s`** is a local socket file.

The next nine characters represent the file permissions. These characters are interpreted as three sets of three characters: The first set are permissions that apply to the file owner. The second set are for the file's group owner. The last set applies to all other (world) users. If a set is an `rwx` string, then that set has all three permissions: read, write, and execute. If a letter is replaced by a `-` dash character, then that set does not have that permission.

After the second column (the link count), the first name specifies the file owner, and the second name is the file's group owner.

In the first example, the permissions for the `student` user are the first set of three characters. The `student` user has read and write permissions on the `test` file, but not execute permission. The second set of three characters are the permissions for the `student` group: read and write permissions on `test`, but not execute permission. The third set of three characters are the permissions for all other users: only read permission on `test`.

The most specific set of permissions applies. So if the `student` user has different permissions from the `student` group, and the `student` user is also a member of that group, then only the user owner permissions apply. This permission allows setting a more restrictive set of permissions on a user than their group membership provides, when it might not be practical to remove the user from the group.

Examples of Permission Effects

The following examples illustrate how file permissions interact. For these examples, your system has four users with the following group memberships:

User	**Group Memberships**
`operator1`	operator1, consultant1
`database1`	database1, consultant1
`database2`	database2, operator2
`contractor1`	contractor1, operator2

Those users work with files in the `dir` directory. A long listing of the files in that directory is as follows:

```
[database1@host dir]$ ls -la
total 24
drwxrwxr-x.  2 database1 consultant1   4096 Mar  4 10:23 .
drwxr-xr-x. 10 root      root          4096 Mar  1 17:34 ..
-rw-rw-r--.  1 operator1 operator1     1024 Mar  4 11:02 app1.log
-rw-r--rw-.  1 operator1 consultant1   3144 Mar  4 11:02 app2.log
-rw-rw-r--.  1 database1 consultant1  10234 Mar  4 10:14 db1.conf
-rw-r-----.  1 database1 consultant1   2048 Mar  4 10:18 db2.conf
```

The `ls` command `-a` option shows the permissions of hidden files, including the special files to represent the directory and its parent. In this example, the `.` special directory reflects the permissions of `dir` itself, and the `..` special directory reflects the permissions of its parent directory.

For the `db1.conf` file, the user that owns the file (`database1`) has read and write permissions but not execute permission. The group that owns the file (`consultant1`) has read and write permissions but not execute permission. All other users have read permission but not write or execute permissions.

The following table explores some effects of this set of permissions for these users:

Effect	**Why is this effect true?**
The `operator1` user can change the contents of the `db1.conf` file.	The `operator1` user is a member of the `consultant1` group, and that group has both read and write permissions on the `db1.conf` file.
The `database1` user can view and modify the contents of the `db2.conf` file.	The `database1` user owns the `db2.conf` file and has both read and write access.
The `operator1` user can view but not modify the contents of the `db2.conf` file.	The `operator1` user is a member of the `consultant1` group, and that group has only read access to the `db2.conf` file.
The `database2` and `contractor1` users do not have any access to the contents of the `db2.conf` file.	The `other` permissions apply to the `database2` and `contractor1` users, and those permissions do not include the read or write permission.

Effect	**Why is this effect true?**
The `operator1` user is the only user who can change the contents of the `app1.log` file.	The `operator1` user and members of the `operator1` group have write permission on the file, whereas other users do not. However, the only member of the `operator1` group is the `operator1` user.
The `database2` user can change the contents of the `app2.log` file.	The `database2` user is not the owner of the `app2.log` file and is not in the `consultant1` group, and so the `other` permissions apply. The `other` permissions grant write permission to the file.
The `database1` user can view the contents of the `app2.log` file, but cannot modify the contents of the `app2.log` file.	The `database1` user is a member of the `consultant1` group, and that group has only read permissions on the `app2.log` file. Even though the `other` permissions include write permission, the group permissions take precedence.
The `database1` user can delete the `app1.log` and `app2.log` files.	The `database1` user has write permissions on the `dir` directory, which the `.` special directory shows, and therefore can delete any file in that directory. This operation is possible even if the `database1` user does not have write permission on the files directly.

References

`ls`(1) man page

`info coreutils` (*GNU Coreutils*)

- Section 13: Changing File Attributes

Next

Quiz: Interpret Linux File System Permissions

Review the following information and use it to answer the quiz questions.

The system has four users that are assigned to the following groups:

- The `consultant1` user is a member of the `consultant1` and `database1` groups.
- The `operator1` user is a member of the `operator1` and `database1` groups.
- The `contractor1` user is a member of the `contractor1` and `contractor3` groups.
- The `operator2` user is a member of the `operator2` and `contractor3` groups.

The `.` special directory contains four files with the following permissions:

```
drwxrwxr-x.   operator1     database1     .
-rw-rw-r--.   consultant1   consultant1   app1.log
-rw-r--rw-.   consultant1   database1     app2.log
-rw-rw-r--.   operator1     database1     db1.conf
-rw-r-----.   operator1     database1     db2.conf
```

Choose the correct answers to the following questions:

1.

2.

1. Which regular file does the `operator1` user own and all users can read?

A `app1.log`

B `app2.log`

[✓ C `db1.conf`]

D	`db2.conf`

3. CheckResetShow Solution

4.

5.

2. Which file can the `contractor1` user modify?

A	`app1.log`
B	[✓ `app2.log`]
C	`db1.conf`
D	`db2.conf`

6. CheckResetShow Solution

7.

8.

3. Which file can the `operator2` user not read?

A	`app1.log`
B	`app2.log`
C	`db1.conf`

D [✓ `db2.conf`]

9. CheckResetShow Solution

10.

11.

4. Which file does the `consultant1` group own?

A [✓ `app1.log`]

B `app2.log`

C `db1.conf`

D `db2.conf`

12. CheckResetShow Solution

13.

14.

5. Which files can the `operator1` user delete?

A Only `db1.conf`

B Only `db2.conf`

C [✓ All the files in the directory]

D	None of the files

15. CheckResetShow Solution

16.

17.

6.	Which files can the `operator2` user delete?
A	Only `app1.log`
B	Only `app2.log`
C	Both `app1.log` and `app2.log`
D	[✓ None of the files]

18. CheckResetShow Solution

Previous Next

Manage File System Permissions from the Command Line

Objectives

Change the permissions and ownership of files with command-line tools.

Change File and Directory Permissions

The `chmod` command has the following characteristics: It changes file and directory permissions from the command line. It can be interpreted as "change mode", because the *mode* of a file is another name for file permissions. It takes a permission instruction followed by a list of files or directories to change. You can set the permission instruction either symbolically or in octal (numeric) notation.

Change Permissions with the Symbolic Method

Use the `chmod` command to modify file and directory permissions.

```
chmod Who/What/Which file|directory
```

Who is the class of user, as in the following table. If you do not provide a class of user, then the `chmod` command uses the `all` group as the default.

Who	**Set**	**Description**
u	*user*	The file owner.
g	*group*	Member of the file's group.
o	*other*	Users who are not the file owner nor members of the file's group.
a	*all*	All the three previous groups.

What is the operator that modifies the *Which*, as in the following table.

What	**Operation**	**Description**
+	*add*	Adds the permissions to the file.
-	*remove*	Removes the permissions to the file.
=	*set exactly*	Sets exactly the provided permissions to the file.

Which is the mode, and specifies the permissions to the files or directories, as in the following table.

Which	**Mode**	**Description**
r	*read*	Read access to the file. Listing access to the directory.
w	*write*	Write permissions to the file or directory.

Which	Mode	Description
x	*execute*	Execute permissions to the file. Allows entering the directory, and accessing files and subdirectories inside the directory.
X	*special execute*	Execute permissions to a directory, or execute permissions to a file if at least one of the execute bits is set.

The *symbolic* method of changing file permissions uses letters to represent the permission groups: `u` for user, `g` for group, `o` for other, and `a` for all.

With the symbolic method, you do not need to set a complete new group of permissions. Instead, you can change one or more of the existing permissions. Use the plus (`+`) or the minus (`-`) characters to add or remove permissions, respectively, or use the equal (`=`) character to replace the entire set for a group of permissions.

A single letter represents the permissions themselves: `r` for read, `w` for write, and `x` for execute. You can use an uppercase `X` as the permission flag to add execute permissions only if the file is a directory or if execute is already set for user, group, or other.

The following list shows some examples for changing permissions with the symbolic method:

Remove read and write permission for group and other on the `document.pdf` file:

```
[user@host ~]$ chmod go-rw document.pdf
```

Add execute permission for everyone on the `myscript.sh` file:

```
[user@host ~]$ chmod a+x myscript.sh
```

You can use the `chmod` command `-R` option to recursively set permissions on the files in an entire directory tree. For example, the next command recursively adds read, write, and execute permissions for the members of the group that own the `myfolder` directory and the files and directories inside it.

```
[user@host ~]$ chmod -R g+rwx /home/user/myfolder
```

You can also use the `chmod` command `-R` option with the `-X` option to set permissions symbolically. With the `chmod` command X option, you can set the execute (search) permission on directories so that their contents can be accessed, without changing permissions on most files. However, be cautious with the X option, because if any execute permission is set on a file, then the X option sets the specified execute permission on that file as well.

For example, the following command recursively sets read and write access on the `demodir` directory and all its children for their group owner, but applies group execute permissions only to directories and files where execute permission is already set for user, group, or other.

```
[root@host opt]# chmod -R g+rwX demodir
```

Change Permissions with the Octal Method

You can use the `chmod` command to change file permissions with the octal method instead of the symbolic method. In the following example, the **#** character represents a digit.

```
chmod ### file|directory
```

With the octal method, you can represent permissions as a 3-digit (or 4-digit, when setting advanced permissions) *octal* number. A single octal digit can represent any single value from 0-7.

In the 3-digit octal representation of permissions, each digit stands for one access level, from left to right: user, group, and other. To determine each digit:

- Start with 0.
- To add read permissions for this access level, add 4.
- To add write permissions, add 2.
- To add execute permissions, add 1.

The following diagram illustrates how systems interpret the `644` octal permission value.

Figure 7.1: Visual representation of the octal method

Experienced administrators often use octal permissions to implement on single or matching files, and provide full permission control.

The following list shows some examples for changing permissions with the octal method:

Set read and write permissions for user, and read permission for group and other, on the `sample.txt` file:

```
[user@host ~]$ chmod 644 sample.txt
```

Set read, write, and execute permissions for user, read and execute permissions for group, and no permission for other on the `sampledir` directory:

```
[user@host ~]$ chmod 750 sampledir
```

Change File and Directory User or Group Ownership

The user owns a file that it creates. By default, new files have a group ownership that is the primary group of the user that creates the file. In Red Hat Enterprise Linux, a user's primary group is usually a private group with only that user as a member. To grant access to a file based on group membership, you might need to change the group that owns the file.

Only the `root` user can change the user that owns a file. However, the file's owner and the `root` user can set group ownership. The `root` user can grant file ownership to any group, but only regular users can change the file's group ownership if they are a member of the destination group.

You can change file ownership by using the `chown` (change owner) command. For example, to grant ownership of the `app.conf` file to the `student` user, use the following command:

```
[root@host ~]# chown student app.conf
```

The `chown` command `-R` option recursively changes the ownership of an entire directory tree. The following command grants ownership of the `Pictures` directory and all files and subdirectories within it to the `student` user:

```
[root@host ~]# chown -R student Pictures
```

You can also use the `chown` command to change group ownership of a file by preceding the group name with a colon (`:`). For example, the following command changes the group ownership of the `Pictures` directory to `admins`:

```
[root@host ~]# chown :admins Pictures
```

You can use the `chown` command to change both owner and group at the same time by using the *owner*:*group* syntax. For example, to change the ownership of the `Pictures` directory to the `visitor` user and the group to `guests`, use the following command:

```
[root@host ~]# chown visitor:guests Pictures
```

Instead of using the `chown` command, some users change the group ownership by using the `chgrp` command. This command works similarly to `chown`, except that you can use it only to change group ownership, and the colon (`:`) before the group name is not required.

Important

You might encounter alternative `chown` syntax that separates owner and group with a period character instead of a colon:

```
[root@host ~]# chown owner.group filename
```

Red Hat recommends not using this syntax, and always using a colon. Because a period is a valid character in a username, a `chown` command might misinterpret your intent. The command might interpret the user and group as a file name. Instead, only use a colon character when setting the user and group at the same time.

References

`ls`(1), `chmod`(1), `chown`(1), and `chgrp`(1) man pages

Previous Next

Guided Exercise: Manage File System Permissions from the Command Line

In this exercise, you use file system permissions to create a directory in which all members of a particular group can add and delete files.

Outcomes

- Create a collaborative directory that all members of a group can access.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start perms-cli
```

Instructions

1. From `workstation`, log in to `servera` as the `student` user and switch to the `root` user.

```
2. [student@workstation ~]$ ssh student@servera
3. ...output omitted...
4. [student@servera ~]$ sudo -i
5. [sudo] password for student: student
```

```
[root@servera ~]#
```

6. Create the `/home/consultants` directory.

```
[root@servera ~]# mkdir /home/consultants
```

7. Change the group ownership of the `consultants` directory to `consultants`.

```
[root@servera ~]# chown :consultants /home/consultants
```

8. Modify the permissions of the `consultants` group to allow its group members to create files in, and delete files from, the `/home/consultants` directory.

   The current permissions forbid others from accessing the files. You must set the appropriate permissions.

   1. Verify that the permissions of the `consultants` group allow its group members to create files in, and delete files from, the `/home/consultants` directory.

      Note that the `consultants` group currently does not have write permission.

```
[root@servera ~]# ls -ld /home/consultants
drwxr-xr-x.  2 root    consultants       6 Mar  1 12:08 /home/consultant
s
```

   2. Add write permission to the `consultants` group. Use the symbolic method for setting the appropriate permissions.

```
[root@servera ~]# chmod g+w /home/consultants
[root@servera ~]# ls -ld /home/consultants
```

```
drwxrwxr-x. 2 root consultants 6 Mar  1 13:21 /home/consultants
```

   5. Forbid others from accessing files in the `/home/consultants` directory. Use the octal method for setting the appropriate permissions.

```
[root@servera ~]# chmod 770 /home/consultants
[root@servera ~]# ls -ld /home/consultants
```

```
drwxrwx---. 2 root consultants 6 Mar  1 12:08 /home/consultants/
```

9. Exit the `root` shell and switch to the `consultant1` user. The password is `redhat`.

```
[root@servera ~]# exit
logout
[student@servera ~]$ su - consultant1
Password: redhat
```

```
[consultant1@servera ~]$
```

14. Navigate to the `/home/consultants` directory and create a file called `consultant1.txt`.
    1. Change to the `/home/consultants` directory.

       ```
       [consultant1@servera ~]$ cd /home/consultants
       ```

    2. Create an empty file called `consultant1.txt`.

       ```
       [consultant1@servera consultants]$ touch consultant1.txt
       ```

15. List the default user and group ownership of the new file and its permissions.

```
16. [consultant1@servera consultants]$ ls -l consultant1.txt
```

```
-rw-rw-r--. 1 consultant1 consultant1 0 Mar  1 12:53 consultant1.txt
```

17. Ensure that all members of the `consultants` group can edit the `consultant1.txt` file. Change the group ownership of the `consultant1.txt` file to `consultants`.
    1. Use the `chown` command to change the group ownership of the `consultant1.txt` file to `consultants`.

       ```
       [consultant1@servera consultants]$ chown :consultants consultant1.txt
       ```

    2. List the new ownership of the `consultant1.txt` file.

       ```
       3. [consultant1@servera consultants]$ ls -l consultant1.txt
       ```

       ```
       -rw-rw-r--. 1 consultant1 consultants 0 Mar  1 12:53 consultant1.txt
       ```

18. Exit the shell and switch to the `consultant2` user. The password is `redhat`.

```
19. [consultant1@servera consultants]$ exit
20. logout
21. [student@servera ~]$ su - consultant2
22. Password: redhat
```

```
[consultant2@servera ~]$
```

23. Navigate to the `/home/consultants` directory. Ensure that the `consultant2` user can add content to the `consultant1.txt` file.

1. Change to the `/home/consultants` directory. Add `text` to the `consultant1.txt` file.

```
2. [consultant2@servera ~]$ cd /home/consultants/
```

```
[consultant2@servera consultants]$ echo "text" >> consultant1.txt
```

3. Verify that the text is present in the `consultant1.txt` file.

```
4. [consultant2@servera consultants]$ cat consultant1.txt
```

```
text
```

5. Return to the `workstation` system as the `student` user.

```
6.  [consultant2@servera consultants]$ exit
7.  logout
8.  [student@servera ~]$ exit
9.  logout
10. Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish perms-cli
```

This concludes the section.

Previous Next

Manage Default Permissions and File Access

Objectives

Control the default permissions of user-created files, explain the effect of special permissions, and use special and default permissions to set the group owner of files that are created in a directory.

Special Permissions

Special permissions are a fourth permission type in addition to the user, group, and other types. As the name implies, special permissions provide additional access-related features beyond what the basic permission types allow. This section describes the impact of special permissions, which are summarized in the following table.

Table 7.2. Effects of Special Permissions on Files and Directories

Permission	Effect on files	Effect on directories
u+s (suid)	File executes as the user that owns the file, not as the user that ran the file.	No effect.
g+s (sgid)	File executes as the group that owns the file.	Files that are created in the directory have a group owner to match the group owner of the directory.
o+t (sticky)	No effect.	Users with write access to the directory can remove only files that they own; they cannot remove or force saves to files that other users own.

The *setuid* permission on an executable file means that commands run as the user that owns that file, rather than as the user that ran the command. One example is the `passwd` command:

```
[user@host ~]$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 35504 Jul 16  2010 /usr/bin/passwd
```

In a long listing, you can identify the `setuid` permissions by a lowercase `s` character in the place where you would normally expect the `x` character (owner execute permissions). If the owner does not have execute permissions, then this character is replaced by an uppercase `S` character.

The *setgid* special permission on a directory means that created files in the directory inherit their group ownership from the directory, rather than inheriting group ownership from the creating user. This feature is commonly used on group collaborative directories to automatically change a file from the default private group to the shared group, or if a specific group should always own files in a directory. An example of this behavior is the `/run/log/journal` directory:

```
[user@host ~]$ ls -ld /run/log/journal
drwxr-sr-x. 3 root systemd-journal 60 May 18 09:15 /run/log/journal
```

If `setgid` is set on an executable file, then commands run as the group that owns that file, rather than as the user that ran the command. This condition is similar to the way that `setuid` works. One example is the `locate` command:

```
[user@host ~]$ ls -ld /usr/bin/locate
-rwx--s--x. 1 root slocate 47128 Aug 12 17:17 /usr/bin/locate
```

In a long listing, you can identify the `setgid` permissions by a lowercase `s` character in the place where you would normally expect the `x` character (group execute permissions). If the group does not have execute permissions, then this character is replaced by an uppercase `S` character.

Finally, the *sticky bit* for a directory sets a special restriction on deletion of files. Only the owner of the file (and the `root` user) can delete files within the directory. An example is the `/tmp` directory:

```
[user@host ~]$ ls -ld /tmp
drwxrwxrwt. 39 root root 4096 Feb  8 20:52 /tmp
```

In a long listing, you can identify the sticky permissions by a lowercase `t` character in the place where you would normally expect the `x` character (other execute permissions). If other does not have execute permissions, then this character is replaced by an uppercase `T` character.

Setting Special Permissions

- **Symbolic** : `setuid` = `u+s`; `setgid` = `g+s`; sticky = `o+t`
- **Octal** : In the added fourth preceding digit; `setuid` = 4; `setgid` = 2; sticky = 1

Examples of Special Permissions

Add the `setgid` bit on the `example` directory by using the symbolic method:

```
[user@host ~]# chmod g+s example
```

Remove the `setuid` bit on the `example` directory by using the symbolic method:

```
[user@host ~]# chmod u-s example
```

Set the `setgid` bit and add read, write, and execute permissions for user and group, with no access for others, on the `example` directory by using the octal method:

```
[user@host ~]# chmod 2770 example
```

Remove the `setgid` bit and add read, write, and execute permissions for user and group, with no access for others, on the `example` directory by using the octal method. Note that you need to add an extra `0` at the beginning of the permissions value when removing special permissions by using the octal method:

```
[user@host ~]# chmod 0770 example
```

Default File Permissions

On creation, a file is assigned initial permissions. Two factors affect these initial permissions. The first is whether you are creating a regular file or a directory. The second is the current *umask*, which stands for user file-creation mask.

If you create a directory, then its initial octal permissions are 0777 (`drwxrwxrwx`). If you create a regular file, then its initial octal permissions are 0666 (`-rw-rw-rw-`). You must always explicitly add execute permission to a regular file. This step makes it harder for an attacker to compromise a system, create a malicious file, and run it.

Additionally, the shell session sets a umask to further restrict the initial permissions of a file. The umask is an octal bitmask that clears the permissions of new files and directories that a process creates. If a bit is set in the umask, then the corresponding permission is cleared on new files. For example, the umask 0002 clears the write bit for other users. The leading zeros indicate that the special, user, and group permissions are not cleared. A umask of 0077 clears all the group and other permissions of newly created files.

The `umask` command without arguments displays the current value of the shell's umask:

```
[user@host ~]$ umask
0022
```

Use the `umask` command with a single octal argument to change the umask of the current shell. The argument should be an octal value that corresponds to the new umask value. You can omit any leading zeros in the umask. For example, `umask 077` is the same as `umask 0077`.

The system's default umask values for Bash shell users are defined in the `/etc/login.defs` file, and in the `/etc/bashrc` file. Users can override the system defaults in the `.bash_profile` or `.bashrc` files in their home directories.

Important

In Red Hat Enterprise Linux 8 and earlier, if a user account has a UID of 200 or greater and the username and the account's primary group name are the same, then its default umask is 0002. Otherwise, its default umask is 0022.

Red Hat Enterprise Linux 9 is in the process of changing the default umask so that all accounts have a umask of 0022. In RHEL 9.0 and 9.1, when you start a login shell, your umask is 0022. However, when you start an interactive non-login shell (such as when you start `gnome-terminal` in the graphical UI), your umask is 0002 if your account's UID is 200 or greater and your primary group has the same name as your user account. This default umask is expected to change in future versions of Red Hat Enterprise Linux 9 so that the umask defaults to 0022 in all circumstances.

Bugzilla issue https://bugzilla.redhat.com/show_bug.cgi?id=2062601 is tracking this change in the behavior of RHEL 9.

Effect of umask Utility on Permissions

The following example explains how the umask affects the permissions of files and directories. Look at the default umask permissions for both files and directories in the current shell.

Important

The following examples assume that the umask of the shell is set to 0022.

If you create a regular file, then its initial octal permissions are 0666 (000 110 110 110, in binary representation). Then, the 0022 umask (000 000 010 010) disables the write permission bit for group and others. Thus, the owner has both read and write permission on files, and both group and other are set to read (000 110 100 100).

Figure 7.2: Example of umask calculation on a file

```
[user@host ~]$ umask
0022
[user@host ~]$ touch default.txt
[user@host ~]$ ls -l default.txt
-rw-r--r--. 1 user user 0 May  9 01:54 default.txt
```

If you create a directory, then its initial octal permissions are 0777 (000 111 111 111). Then, the 0022 umask (000 000 010 010) disables the write permission bit for group and other. Thus, the owner has read, write, and execute permissions on directories, and both group and other are set for read and execution (000 111 101 101).

Figure 7.3: Example of umask calculation on a directory

```
[user@host ~]$ umask
0022
[user@host ~]$ mkdir default
[user@host ~]$ ls -ld default
drwxr-xr-x. 2 user user 0 May  9 01:54 default
```

By setting the umask value to 0, the file permissions for other change from read to read and write. The directory permissions for other change from read and execute to read, write, and execute.

```
[user@host ~]$ umask 0
[user@host ~]$ touch zero.txt
[user@host ~]$ ls -l zero.txt
-rw-rw-rw-. 1 user user 0 May  9 01:54 zero.txt
[user@host ~]$ mkdir zero
[user@host ~]$ ls -ld zero
drwxrwxrwx. 2 user user 0 May  9 01:54 zero
```

To mask all file and directory permissions for other, set the umask value to 007.

```
[user@host ~]$ umask 007
[user@host ~]$ touch seven.txt
[user@host ~]$ ls -l seven.txt
-rw-rw----. 1 user user 0 May  9 01:55 seven.txt
[user@host ~]$ mkdir seven
[user@host ~]$ ls -ld seven
drwxrwx---. 2 user user 0 May  9 01:54 seven
```

A umask of 027 ensures that new files have read and write permissions for user and read permission for group. New directories have read and execute permissions for group and no permissions for other.

```
[user@host ~]$ umask 027
[user@host ~]$ touch two-seven.txt
```

```
[user@host ~]$ ls -l two-seven.txt
-rw-r-----. 1 user user 0 May  9 01:55 two-seven.txt
[user@host ~]$ mkdir two-seven
[user@host ~]$ ls -ld two-seven
drwxr-x---. 2 user user 0 May  9 01:54 two-seven
```

Changing Default Permissions

In Red Hat Enterprise Linux 9, the `/etc/login.defs` file sets the default umask for users. By default, its `UMASK` line specifies that the default umask is 0022.

In Red Hat Enterprise Linux 9.0 and 9.1, this file affects only login shells. If you start a new terminal window or start an interactive non-login shell in some other way, then settings in `/etc/bashrc` still apply. For these shells, if the account's UID is 200 or greater and the username and primary group name are the same, then the account is assigned a umask of 0002. Otherwise, the umask is 0022.

The `root` user can change the default umask for interactive non-login shells by adding a `local-umask.sh` shell startup script in the `/etc/profile.d/` directory. The following example shows a `local-umask.sh` file:

```
[root@host ~]# cat /etc/profile.d/local-umask.sh
# Overrides default umask configuration asda sda
if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 007
else
    umask 022
fi
```

The preceding example sets the umask to 0007 for users with a UID greater than 199 and with a username and primary group name that match, and to 0022 for everyone else. (Leading zeros can be omitted.) To set the umask to 0022 for everyone, then create that file with the following content:

```
# Overrides default umask configuration
umask 022
```

The current umask of a shell applies until you log out of the shell and log back in, or until you change it manually with the `umask` command.

References

`bash`(1), `ls`(1), `chmod`(1), and `umask`(1) man pages

Previous Next

Guided Exercise: Manage Default Permissions and File Access

In this exercise, you control the permissions on files that are created in a directory by using umask settings and the `setgid` permission.

Outcomes

- Create a shared directory where the `operators` group automatically owns new files.
- Experiment with various umask settings.
- Adjust default permissions for specific users.
- Verify your change.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start perms-default
```

Instructions

1. Log in to the `servera` system as the `student` user.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
```

```
[student@servera ~]$
```

4. Switch to the `operator1` user with `redhat` as the password.

```
5. [student@servera ~]$ su - operator1
6. Password: redhat
```

```
[operator1@servera ~]$
```

7. List the `operator1` user's default umask value.

```
8. [operator1@servera ~]$ umask
```

```
0022
```

9. Create a `/tmp/shared` directory. In the `/tmp/shared` directory, create a `defaults` file. Look at the default permissions.
    1. Create the `/tmp/shared` directory. List the permissions of the new directory.

    ```
    2. [operator1@servera ~]$ mkdir /tmp/shared
    3. [operator1@servera ~]$ ls -ld /tmp/shared
    ```

    ```
    drwxr-xr-x. 2 operator1 operator1 6 Feb  4 14:06 /tmp/shared
    ```

    4. Create a `defaults` file in the `/tmp/shared` directory.

    ```
    [operator1@servera ~]$ touch /tmp/shared/defaults
    ```

    5. List the permissions of the new file.

    ```
    6. [operator1@servera ~]$ ls -l /tmp/shared/defaults
    ```

    ```
    -rw-r--r--. 1 operator1 operator1 0 Feb  4 14:09 /tmp/shared/defaults
    ```

10. Change the group ownership of the `/tmp/shared` directory to the `operators` group. Confirm the new ownership and permissions.
    1. Change the group ownership of the `/tmp/shared` directory to the `operators` group.

    ```
    [operator1@servera ~]$ chown :operators /tmp/shared
    ```

2. List the permissions of the `/tmp/shared` directory.

```
3. [operator1@servera ~]$ ls -ld /tmp/shared
```

```
drwxr-xr-x. 2 operator1 operators 22 Feb  4 14:09 /tmp/shared
```

4. Create a `group` file in the `/tmp/shared` directory. List the file permissions.

```
5. [operator1@servera ~]$ touch /tmp/shared/group
6. [operator1@servera ~]$ ls -l /tmp/shared/group
```

```
-rw-r--r--. 1 operator1 operator1 0 Feb  4 17:00 /tmp/shared/group
```

Note

The group owner of the `/tmp/shared/group` file is not `operators` but `operator1`.

11. Ensure that the `operators` group owns files that are created in the `/tmp/shared` directory.
    1. Set the group ID to the `operators` group for the `/tmp/shared` directory.

```
[operator1@servera ~]$ chmod g+s /tmp/shared
```

    2. Create a `ops_db.txt` file in the `/tmp/shared` directory.

```
[operator1@servera ~]$ touch /tmp/shared/ops_db.txt
```

    3. Verify that the `operators` group is the group owner for the new file.

```
4. [operator1@servera ~]$ ls -l /tmp/shared/ops_db.txt
```

```
-rw-r--r--. 1 operator1 operators 0 Feb  4 16:11 /tmp/shared/ops_db.txt
```

12. Create an `ops_net.txt` file in the `/tmp/shared` directory. Record the ownership and permissions. Change the umask for the `operator1` user. Create an `ops_prod.txt` file. Record the ownership and permissions of the `ops_prod.txt` file.
    1. Create an `ops_net.txt` file in the `/tmp/shared` directory.

```
[operator1@servera ~]$ touch /tmp/shared/ops_net.txt
```

2. List the permissions of the `ops_net.txt` file.

```
3. [operator1@servera ~]$ ls -l /tmp/shared/ops_net.txt
```

```
-rw-r--r--. 1 operator1 operators 5 Feb  0 15:43 /tmp/shared/ops_net.txt
```

4. Change the umask for the `operator1` user to 027. Confirm the change.

```
5. [operator1@servera ~]$ umask 027
6. [operator1@servera ~]$ umask
```

```
0027
```

7. Create an `ops_prod.txt` file in the `/tmp/shared/` directory. Verify that newly created files have read-only access for the `operators` group and no access for other users.

```
8. [operator1@servera ~]$ touch /tmp/shared/ops_prod.txt
9. [operator1@servera ~]$ ls -l /tmp/shared/ops_prod.txt
```

```
-rw-r-----. 1 operator1 operators 0 Feb  0 15:56 /tmp/shared/ops_prod.tx
t
```

13. Open a new terminal window and log in to `servera` as `operator1`.

```
14. [student@workstation ~]$ ssh operator1@servera
15. ...output omitted...
```

```
[operator1@servera ~]$
```

16. List the umask value for `operator1`.

```
17. [operator1@servera ~]$ umask
```

```
0022
```

18. Change the default umask for the `operator1` user. The new umask prohibits all access for users that are not in their group. Confirm that the umask is changed.
    1. Change the default umask for the `operator1` user to 007.

```
2. [operator1@servera ~]$ echo "umask 007" >> ~/.bashrc
3. [operator1@servera ~]$ cat ~/.bashrc
```

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
...output omitted...
```

```
umask 007
```

11. Log out and log in again as the `operator1` user. Confirm that the change is permanent.

```
[operator1@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation ~]$ ssh operator1@servera
...output omitted...
[operator1@servera ~]$ umask
```

```
0007
```

19. Create an `ops_prod2.txt` file in the `/tmp/shared/` directory. Verify that newly created files have read and write access for the `operators` group and no access for other users, due to the new umask of 007.

```
[operator1@servera ~]$ touch /tmp/shared/ops_prod2.txt
[operator1@servera ~]$ ls -l /tmp/shared/ops_prod2.txt
```

```
-rw-rw----. 1 operator1 operators 0 Feb  0 15:56 /tmp/shared/ops_prod2.txt
```

22. On `servera`, close all `operator1` and `student` user shells. Return to the `workstation` system as the `student` user.

   ### Warning

   Failure to exit from all `operator1` shells causes the finish script to fail.

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

```
logout
Connection to servera closed.
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish perms-default
```

This concludes the section.

Previous Next

Summary

- File ownership permissions have three categories. A file is owned by a user, a single group, and other users. The most specific permission applies. User permissions override group permissions, and group permissions override other permissions.
- The `ls` command `-l` option expands the file listing to include both the file permissions and ownership.
- The `chmod` command changes file permissions from the command line.
- The `chmod` command can use one of two methods to represent permissions: symbolic or octal.
- The `chown` command changes file ownership. The `chown` command `-R` option recursively changes the ownership of a directory tree.
- The `umask` command without arguments displays the current umask value of the shell. Every process on the system has a umask.
- The default umask values for Bash are defined in the `/etc/login.defs` file and might be affected by settings in the `/etc/profile` and `/etc/bashrc` files, files in `/etc/profile.d`, or your account's shell initialization files.
- The `suid`, `sgid`, and `sticky` special permissions provide additional access-related features to files.

Previous Next

Chapter 8. Monitor and Manage Linux Processes

Abstract

Goal	Evaluate and control processes that run on a Red Hat Enterprise Linux system.
Objectives	• Determine status, resource use, and ownership of running programs on a system, to control them. • Use Bash job control to manage multiple processes that were started from the same terminal session. • Use commands to kill and communicate with processes, define the characteristics of a daemon process, and stop user sessions and processes. • Define load average and determine resource-intensive server processes.
Sections	• Process States and Lifecycle (and Quiz) • Control Jobs (and Guided Exercise) • Kill Processes (and Guided Exercise) • Monitor Process Activity (and Guided Exercise)
Lab	Monitor and Manage Linux Processes

Process States and Lifecycle

Objectives

Determine status, resource use, and ownership of running programs on a system, to control them.

Definition of a Process

A *process* is a running instance of a launched, executable program. From the moment that a process is created, it consists of the following items:

- An address space of allocated memory
- Security properties, including ownership credentials and privileges
- One or more execution threads of program code
- A process state

The *environment* of a process is a list of information that includes the following items:

- Local and global variables
- A current scheduling context
- Allocated system resources, such as file descriptors and network ports

An existing *parent* process duplicates its own address space, which is known as a process *fork*, to create a *child* process structure. Every new process is assigned a unique *process ID* (PID) for tracking and security purposes. The PID and the parent's process ID (PPID) are elements of the new process environment. Any process can create a *child* process. All processes are descendants of the first system process, `systemd`, on a Red Hat system.

Figure 8.1: Process lifecycle

Through the *fork* routine, a child process inherits security identities, previous and current file descriptors, port and resource privileges, environment variables, and program code. A child process can then execute its own program code.

Normally, a parent process *sleeps* when the child process runs, and sets a *wait* request to be signaled when the child completes. After the child process exits, it closes or discards its resources and environment, and leaves a *zombie* resource, which is an entry in the process table. The parent, which is signaled to *wake* when the child exits, cleans the process table of the child's entry, and it frees the last resource of the child process. The parent process then continues with its own program code execution.

Describe Process States

In a multitasking operating system, each CPU (or CPU core) can be working on one process at a time. As a process runs, its immediate requirements for CPU time and resource allocation change. Processes are assigned a *state*, which changes as circumstances dictate.

The following diagram and table describe Linux process states in detail.

Figure 8.2: Linux process states

Table 8.1. Linux Process States

Name	**Flag**	**Kernel-defined state name and description**
Running	R	TASK_RUNNING: The process is either executing on a CPU or waiting to run. The process can be executing user routines or kernel routines (system calls), or be queued and ready when in the *Running* (or *Runnable*) state.
Sleeping	S	TASK_INTERRUPTIBLE: The process is waiting for some condition: a hardware request, system resource access, or a signal. When an event or signal satisfies the condition, the process returns to *Running*.
	D	TASK_UNINTERRUPTIBLE: This process is also sleeping, but unlike the S state, does not respond to signals. It is used only when process interruption might cause an unpredictable device state.
	K	TASK_KILLABLE: Same as the uninterruptible D state, but modified to allow a waiting task to respond to the signal to kill it (exit completely). Utilities often display *Killable* processes as the D state.
	I	TASK_REPORT_IDLE: A subset of state D. The kernel does not count these processes when calculating the load average. It is used for kernel threads. The TASK_UNINTERRUPTIBLE and TASK_NOLOAD flags are set. It is similar to TASK_KILLABLE, and is also a subset of state D. It accepts fatal signals.
Stopped	T	TASK_STOPPED: The process is stopped (suspended), usually by being signaled by a user or another process. The process can be continued (resumed) by another signal to return to running.
	T	TASK_TRACED: A process that is being debugged is also temporarily stopped and shares the T state flag.
Zombie	Z	EXIT_ZOMBIE: A child process signals to its parent as it exits. All resources except for the process identity (PID) are released.
	X	EXIT_DEAD: When the parent cleans up (reaps) the remaining child process structure, the process is now released completely. This state cannot be observed in process-listing utilities.

Importance of Process States

When troubleshooting a system, it is important to understand how the kernel communicates with processes, and how processes communicate with each other.

The system assigns a state to every new process. The S column of the `top` command or the `STAT` column of the `ps` command shows the state of each process. On a single CPU system, only one process can run at a time. It is possible to see several processes with an `R` state. However, not all processes are running consecutively; some of them are in *waiting* status.

```
[user@host ~]$ top
PID USER  PR  NI    VIRT    RES    SHR S  %CPU  %MEM   TIME+    COMMAND
2259 root 20   0  270856  40316   8332 S   0.3   0.7   0:00.25  sssd_kcm
   1 root 20   0  171820  16140  10356 S   0.0   0.3   0:01.62  systemd
   2 root 20   0       0      0      0 S   0.0   0.0   0:00.00  kthreadd
...output omitted...
[user@host ~]$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
...output omitted...
root         2  0.0  0.0      0     0 ?        S    11:57   0:00 [kthreadd]
student   3448  0.0  0.2 266904  3836 pts/0    R+   18:07   0:00 ps aux
...output omitted...
```

Use signals to suspend, stop, resume, terminate, or interrupt processes. Processes can catch signals from the kernel, other processes, and other users on the same system. Signals are discussed later in this chapter.

Listing Processes

The `ps` command is used for listing detailed information for current processes.

- User identification (UID), which determines process privileges
- Unique process identification (PID)
- Amount of used CPU and elapsed real time
- Amount of allocated memory
- The process `stdout` location, which is known as the controlling terminal
- The current process state

Important

The Linux version of the `ps` command supports the following option formats:

- UNIX (POSIX) options, which can be grouped and must be preceded by a dash
- BSD options, which can be grouped and must not be used with a dash
- GNU long options, which are preceded by two dashes

For example, the `ps -aux` command is not the same as the `ps aux` command.

The common `ps` command `aux` option displays all processes including processes without a controlling terminal. A long listing (`lax` options) provides more detail, and gives faster results by avoiding username lookups. The similar UNIX syntax uses the `-ef` options to display all processes. In the following examples, scheduled kernel threads are displayed in brackets at the top of the list.

```
[user@host ~]$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.2 171820 16140 ?        Ss   16:47   0:01 /usr/lib/systemd/s
ystemd ...
root           2  0.0  0.0      0     0 ?        S    16:47   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   16:47   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   16:47   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   16:47   0:00 [kworker/0:0H-even
ts_highpri]
...output omitted...
[user@host ~]$ ps lax
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY   TIME COMMAND
4     0       1       0  20   0 171820 16140 -      Ss   ?     0:01 /usr/lib/systemd/s
ystemd ...
1     0       2       0  20   0      0     0 -      S    ?     0:00 [kthreadd]
1     0       3       2   0 -20      0     0 -      I<   ?     0:00 [rcu_gp]
1     0       4       2   0 -20      0     0 -      I<   ?     0:00 [rcu_par_gp]
1     0       6       2   0 -20      0     0 -      I<   ?     0:00 [kworker/0:0H-even
ts_highpri]
...output omitted...
[user@host ~]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 16:47 ?        00:00:01 /usr/lib/systemd/systemd ...
root         2     0  0 16:47 ?        00:00:00 [kthreadd]
root           3       2  0 16:47 ?        00:00:00 [rcu_gp]
root           4       2  0 16:47 ?        00:00:00 [rcu_par_gp]
root           6       2  0 16:47 ?        00:00:00 [kworker/0:0H-events_highpri]
...output omitted...
```

By default, the `ps` command with no options selects all processes with the current user's *effective user ID* (EUID), and selects processes that are associated with the terminal that is running the command. Zombie processes are listed with the `exiting` or `defunct` label.

You can use the `ps` command `--forest` option to display the processes in a tree format so you can view relationships between parent and child processes.

The default output of the `ps` command is sorted by process ID number. At first glance, the output might appear to use chronological order, but the kernel reuses process IDs, so the order is less structured than it appears. Use the `ps` command `-O` or `--sort` options to sort the output. The display order matches that of the system process table, which reuses table rows when processes die and spawn.

References

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

- Section 24: Signal Handling

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

- Section 26: Processes

`ps`(1) and `signal`(7) man pages

Next

Quiz: Process States and Lifecycle

Choose the correct answers to the following questions:

1.

2.

1. Which state represents a process that is stopped or suspended?

A	D
B	R
C	S
D	[✓ T]
E	Z

3. CheckResetShow Solution

4.

5.

2. Which state represents a process that released all of its resources except its PID?

A	D
B	R
C	S
D	T
E	[✓ Z]

6. CheckResetShow Solution

7.

8.

3. Which process does a parent use to duplicate its address space, and creates a child process?

A	exec
B	[✓ fork]

C	zombie
D	syscall
E	reap

9. CheckResetShow Solution

10.

11.

4. Which state represents a process that is sleeping until some condition is met?

A	D
B	R
C	[✓ S]
D	T
E	Z

12. CheckResetShow Solution

Previous Next

Control Jobs

Objectives

Use Bash job control to manage multiple processes that were started from the same terminal session.

Describe Jobs and Sessions

With the *job control* shell feature, a single shell instance can run and manage multiple commands.

A *job* is associated with each *pipeline* that is entered at a shell prompt. All processes in that pipeline are part of the job and are members of the same *process group*. You can consider a minimal pipeline to be only one command that is entered in the shell prompt to create a job with only one member.

Only one job at a time can read input and keyboard-generated signals from a particular terminal window. Processes that are part of that job are *foreground* processes of that *controlling terminal*.

A *background* process of that controlling terminal is any other job that is associated with that terminal. Background processes of a terminal cannot read input or receive keyboard-generated interrupts from the terminal, but can write to the terminal. A background job might be stopped (suspended) or it might be running. If a running background job tries to read from the terminal, then it is automatically suspended.

Each terminal runs in its own *session*, and can have a foreground process and any number of background processes. A job that runs in its own session belongs to its controlling terminal.

The `ps` command shows the device name of the controlling terminal in the `TTY` column. Some processes, such as *system daemons*, are started by the system and not from a controlling terminal. These processes are not members of a job, and cannot be brought to the foreground. The `ps` command displays a question mark (`?`) in the `TTY` column for these processes.

Run Jobs in the Background

Any command or pipeline can be started in the background by appending an ampersand (`&`) character to the command. The `Bash` shell displays a *job number* (unique to the session) and the PID of the new child process. The shell does not wait for the child process to terminate, but instead displays the shell prompt.

```
[user@host ~]$ sleep 10000 &
[1] 5947
[user@host ~]$
```

When a command line with a pipe (`|`) is sent to the background, the PID of the last command in the pipeline is displayed. All pipeline processes are members of that job.

```
[user@host ~]$ example_command | sort | mail -s "Sort output" &
[1] 5998
```

Use the `jobs` command to display the list of jobs for the shell's session.

```
[user@host ~]$ jobs
[1]+ Running    sleep 10000 &
[user@host ~]$
```

Use the `fg` command to bring a background job to the foreground. Use the (`%jobNumber`) format to specify the process to foreground.

```
[user@host ~]$ fg %1
sleep 10000
```

In the preceding example, the `sleep` command is running in the foreground on the controlling terminal. The shell itself is asleep and waiting for this child process to exit.

To send a foreground process to the background, press the keyboard-generated *suspend* request (**Ctrl**+**z**) in the terminal. The job is placed in the background and suspended.

```
[user@host ~]$ sleep 10000
^Z
```

```
[1]+  Stopped                 sleep 10000
[user@host ~]$
```

The `ps j` command displays information about jobs. Use the `ps j` command to find process and session information.

- The PID is the unique process ID of the process.
- The PPID is the PID of the *parent process* of this process, the process that started (forked) it.
- The PGID is the PID of the *process group leader*, normally the first process in the job's pipeline.
- The SID is the PID of the *session leader*, which (for a job) is normally the interactive shell that is running on its controlling terminal.

In the next example, the `sleep` command is currently suspended and the process state is `T`.

```
[user@host ~]$ ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 2764  2768  2768  2768 pts/0     6377 Ss    1000   0:00 /bin/bash
 2768  5947  5947  2768 pts/0     6377 T     1000   0:00 sleep 10000
 2768  6377  6377  2768 pts/0     6377 R+    1000   0:00 ps j
```

Use the `bg` command with the job ID to start running the suspended process.

```
[user@host ~]$ bg %1
[1]+ sleep 10000 &
```

The shell warns a user who attempts to exit a terminal window (session) with suspended jobs. If the user tries again to exit immediately, then the suspended jobs are killed.

Note

In the previous examples, the `+` sign indicates that this job is the current default. If a job-control command is used without the `%jobNumber` argument, then the action is taken on the default job. The `-` sign indicates the previous job that will become the default job when the current default job finishes.

References

`Bash` info page (*The GNU Bash Reference Manual*) https://www.gnu.org/software/bash/manual

- Section 7: Job Control

`bash`(1), `builtins`(1), `ps`(1), and `sleep`(1) man pages

Previous Next

Guided Exercise: Control Jobs

In this exercise, you use job control to start, suspend, and move multiple processes to the background and foreground.

Outcomes

- Use job control to suspend and restart user processes.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start processes-control
```

Instructions

1. On the `workstation` machine, open two terminal windows side by side. In this section, these two terminals are referred to as *left* and *right*. In each terminal, log in to the `servera` machine as the `student` user.

```
2. [student@workstation ~]$ ssh student@servera
```

```
[student@servera ~]$
```

3. In the left terminal shell, create the `/home/student/bin` directory. Create a shell script called `control` in the `/home/student/bin` directory. Change the script permissions to make it executable.
   1. Create the `/home/student/bin` directory.

      ```
      [student@servera ~]$ mkdir /home/student/bin
      ```

   2. Create a script called `control` in the `/home/student/bin` directory. To enter Vim interactive mode, press the **i** key. Use the `:wq` command to save the file and quit.

      ```
      [student@servera ~]$ vim /home/student/bin/control
      #!/bin/bash
      while true; do
        echo -n "$@ " >> ~/control_outfile
        sleep 1
      ```

      ```
      done
      ```

      Note

      The *control* script runs until the process is terminated. The script appends command-line arguments to the `~/control_outfile` file once per second.

   8. Make the `control` file executable.

      ```
      [student@servera ~]$ chmod +x /home/student/bin/control
      ```

4. Execute the `control` script. The script continuously appends the word "technical" and a space to the `~/control_outfile` file at one-second intervals.

   ```
   [student@servera ~]$ control technical
   ```

5. In the right terminal shell, verify that the new process is writing to the `/home/student/control_outfile` file.

   ```
   [student@servera ~]$ tail -f ~/control_outfile
   technical technical technical technical
   ```

```
...output omitted...
```

8. In the left terminal shell, press **Ctrl**+**z** to suspend the running process. The shell returns the job ID in square brackets. In the right terminal shell, confirm that the process output is stopped.

```
9. ^Z
10.[1]+  Stopped                 control technical
```

```
[student@servera ~]$
technical technical technical technical
...no further output...
```

11. In the left terminal shell, view the `jobs` command output. Remember that the `+` sign indicates the default job. Restart the job in the background. In the right terminal shell, verify that the process output is again active.
    1. View the list of jobs.

    ```
    2. [student@servera ~]$ jobs
    ```

    ```
    [1]+  Stopped                 control technical
    ```

    3. Restart the `control` job in the background.

    ```
    4. [student@servera ~]$ bg
    ```

    ```
    [1]+ control technical &
    ```

    5. Verify that the `control` job is running again.

    ```
    6. [student@servera ~]$ jobs
    ```

    ```
    [1]+  Running                 control technical &
    ```

    7. In the right terminal shell, confirm that the `tail` command is producing output.

    ```
    8. ...output omitted...
    ```

    ```
    technical technical technical technical technical technical technical te
    chnical
    ```

12. In the left terminal shell, start two more `control` processes to append to the `~/output` file. Use the ampersand (`&`) special command to start the processes in the background. Replace `technical` with `documents` and then with `database`. Replacing the arguments helps to differentiate between the three processes.

```
[student@servera ~]$ control documents &
[2] 6579
[student@servera ~]$ control database &
```

```
[3] 6654
```

16. In the left terminal shell, use the `jobs` command to view the three running processes. In the right terminal shell, verify that all three processes are appending to the file.

```
[student@servera ~]$ jobs
[1]   Running                 control technical &
[2]-  Running                 control documents &
```

```
[3]+  Running                 control database &
...output omitted...
technical documents database technical documents database technical documents
database technical documents database
...output omitted...
```

20. Suspend the `control technical` process. Confirm that it is suspended. Terminate the `control documents` process and verify that it is terminated.
    1. In the left terminal shell, foreground the `control technical` process. Press **Ctrl**+**z** to suspend the process. Verify that the process is suspended.

```
[student@servera ~]$ fg %1
control technical
^Z
[1]+  Stopped                 control technical
[student@servera ~]$ jobs
[1]+  Stopped                 control technical
[2]   Running                 control documents &
```

```
[3]-  Running                 control database &
```

9. In the right terminal shell, verify that the `control technical` process is no longer sending output.

```
10. database documents  database documents  database
```

```
...no further output...
```

11. In the left terminal shell, run the `control documents` process in the foreground. Press **Ctrl**+**c** to terminate the process. Verify that the process is terminated.

```
12. [student@servera ~]$ fg %2
13. control documents
14. ^C
15. [student@servera ~]$ jobs
16. [1]+  Stopped                 control technical
```

```
[3]-  Running                 control database &
```

17. In the right terminal shell, verify that the `control documents` process is no longer sending output.

```
18. ...output omitted...
19. database database database database database database database database
```

```
...no further output...
```

21. In the left terminal shell, view the remaining jobs. The suspended jobs have a state of `T`. The other background jobs are sleeping and have a state of `S`.

```
22. [student@servera ~]$ ps jT
23.  PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
24.  27277 27278 27278 27278 pts/1    28702 Ss    1000   0:00 -bash
25.  27278 28234 28234 27278 pts/1    28702 T     1000   0:00 /bin/bash /home/stud
    ent/bin/control technical
26.  27278 28251 28251 27278 pts/1    28702 S     1000   0:00 /bin/bash /home/stud
    ent/bin/control database
27.  28234 28316 28234 27278 pts/1    28702 T     1000   0:00 sleep 1
```

```
 28251 28701 28251 27278 pts/1    28702 S     1000   0:00 sleep 1
```

```
 27278 28702 28702 27278 pts/1    28702 R+    1000   0:00 ps jT
```

29. In the left terminal shell, view the current jobs. Terminate the `control database` process and verify that it is terminated.

```
[student@servera ~]$ jobs
[1]+  Stopped                 control technical
```

```
[3]-  Running                 control database &
```

Use the `fg` command with the job ID to run the `control database` process in the foreground. Press **Ctrl**+**c** to terminate the process. Verify that the process is terminated.

```
[student@servera ~]$ fg %3
control database
^C
[student@servera ~]$ jobs
[1]+  Stopped                 control technical
```

32. In the right terminal shell, use the **Ctrl**+**c** command to stop the `tail` command. Delete the `~/control_outfile` file.

```
...output omitted...
^C
```

```
[student@servera ~]$ rm ~/control_outfile
```

35. Close the extra terminal. Return to the `workstation` system as the `student` user.

```
[student@servera ~]$ exit
logout
Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish processes-control
```

This concludes the section.

Previous Next

Kill Processes

Objectives

Use commands to kill and communicate with processes, define the characteristics of a daemon process, and stop user sessions and processes.

Process Control with Signals

A *signal* is a software interrupt that is delivered to a process. Signals report events to an executing program. Events that generate a signal can be an error, an external event (an I/O request or an expired timer), or by the explicit use of a signal-sending command or keyboard sequence.

The following table lists the fundamental signals that system administrators use for routine process management. Refer to signals by their short (HUP) or proper (SIGHUP) name.

Table 8.2. Fundamental process management signals

Signal	Name	Definition
1	HUP	Hangup : Reports termination of the controlling process of a terminal. Also requests process re-initialization (configuration reload) without termination.
2	INT	`Keyboard interrupt` : Causes program termination. It can be blocked or handled. Sent by pressing the INTR (Interrupt) key sequence (**Ctrl**+**c**).
3	QUIT	`Keyboard quit` : Similar to SIGINT; adds a process dump at termination. Sent by pressing the QUIT key sequence (**Ctrl**+****).

Signal	**Name**	**Definition**
9	KILL	`Kill, unblockable` : Causes abrupt program termination. It cannot be blocked, ignored, or handled; consistently fatal.
15 *default*	TERM	`Terminate` : Causes program termination. Unlike SIGKILL, it can be blocked, ignored, or handled. The "clean" way to ask a program to terminate; it allows the program to complete essential operations and self-cleanup before termination.
18	CONT	`Continue` : Sent to a process to resume if stopped. It cannot be blocked. Even if handled, it always resumes the process.
19	STOP	`Stop, unblockable` : Suspends the process. It cannot be blocked or handled.
20	TSTP	`Keyboard stop` : Unlike SIGSTOP, it can be blocked, ignored, or handled. Sent by pressing the suspend key sequence (**Ctrl**+**z**).

Note

Signal numbers vary between Linux hardware platforms, but signal names and meanings are standard. It is advised to use signal names rather than numbers when signaling. The numbers that are discussed in this section are for x86_64 architecture systems.

Each signal has a *default action*, which is usually one of the following actions:

- **`Term`** : Terminate a program (exit) immediately.
- **`Core`** : Save a program's memory image (core dump), and then terminate.
- **`Stop`** : Stop a running program (suspend) and wait to continue (resume).

Programs react to the expected event signals by implementing handler routines to ignore, replace, or extend a signal's default action.

Send Signals by Explicit Request

You can signal the current foreground process by pressing a keyboard control sequence to suspend (**Ctrl**+**z**), kill (**Ctrl**+**c**), or core dump (**Ctrl**+****) the process. However, you might use signal-sending commands to send signals to background processes in a different session.

You can specify signals either by name (for example, with the `-HUP` or `-SIGHUP` options) or by number (with the related `-1` option). Users can kill their processes, but root privilege is required to kill processes that other users own.

The `kill` command uses a PID number to send a signal to a process. Despite its name, you can use the `kill` command to send any signal, not just those signals for terminating programs. You can use the `kill` command `-l` option to list the names and numbers of all available signals.

```
[user@host ~]$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL    10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP    20) SIGTSTP
...output omitted...
[user@host ~]$ ps aux | grep job
5194 0.0 0.1 222448 2980 pts/1 S  16:39 0:00 /bin/bash /home/user/bin/control job1
5199 0.0 0.1 222448 3132 pts/1 S  16:39 0:00 /bin/bash /home/user/bin/control job2
5205 0.0 0.1 222448 3124 pts/1 S  16:39 0:00 /bin/bash /home/user/bin/control job3
5430 0.0 0.0 221860 1096 pts/1 S+ 16:41 0:00 grep --color=auto job
[user@host ~]$ kill 5194
[user@host ~]$ ps aux | grep job
user   5199  0.0  0.1 222448  3132 pts/1    S    16:39   0:00 /bin/bash /home/user/bi
n/control job2
user   5205  0.0  0.1 222448  3124 pts/1    S    16:39   0:00 /bin/bash /home/user/bi
n/control job3
user   5783  0.0  0.0 221860   964 pts/1    S+   16:43   0:00 grep --color=auto job
[1]   Terminated              control job1
[user@host ~]$ kill -9 5199
[user@host ~]$ ps aux | grep job
user   5205  0.0  0.1 222448  3124 pts/1    S    16:39   0:00 /bin/bash /home/user/bi
n/control job3
user   5930  0.0  0.0 221860  1048 pts/1    S+   16:44   0:00 grep --color=auto job
[2]-  Killed                  control job2
[user@host ~]$ kill -SIGTERM 5205
user   5986  0.0  0.0 221860  1048 pts/1  S+  16:45  0:00 grep --color=auto job
```

```
[3]+  Terminated              control job3
```

Control Specific Processes

Use the `pkill` command to signal one or more processes that match selection criteria. Selection criteria can be a command name, a process that a specific user owns, or all system-wide processes.

Processes and sessions can be individually or collectively signaled. To terminate all processes for one user, use the `pkill` command.

Because the initial process in a login session (*session leader*) is designed to handle session termination requests and to ignore unintended keyboard signals, killing all of a user's processes and login shells requires the SIGKILL signal.

First, use the `pgrep` command to identify the PID numbers to kill. This command operates similarly to the `pkill` command, including most of the same options, except that the `pgrep` command lists processes rather than killing them.

Use the `pgrep` command with the `-l` option to list the process names and IDs. Use either command with the `-u` option to specify the ID of the user who owns the processes.

```
[root@host ~]# pgrep -l -u bob

6964 bash

6998 sleep

6999 sleep

7000 sleep

[root@host ~]# pkill -SIGKILL -u bob

[root@host ~]# pgrep -l -u bob
```

When processes that require attention are in the same login session, killing all of a user's processes might not be needed. Use the `w` command to determine the controlling terminal for the session, and then kill only the processes that reference the same terminal ID.

Unless `SIGKILL` is specified, the session leader (here, the `Bash` login shell) successfully handles and survives the termination request, but terminates all other session processes.

Use the `-t` option to match processes with a specific terminal ID.

```
[root@host ~]# pgrep -l -u bob
7391 bash
7426 sleep
7427 sleep
7428 sleep
[root@host ~]# w -u bob
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
bob      tty3                      18:37    5:04   0.03s  0.03s -bash
[root@host ~]# pkill -t tty3
[root@host ~]# pgrep -l -u bob
7391 bash
[root@host ~]# pkill -SIGKILL -t tty3
[root@host ~]# pgrep -l -u bob
[root@host ~]#
```

Important

Administrators commonly use SIGKILL.

It is always fatal, because the SIGKILL signal cannot be handled or ignored. However, it forces termination without allowing the killed process to run self-cleanup routines. Red Hat recommends sending SIGTERM first, and then trying SIGINT; and only if both fail, trying again with SIGKILL.

You can apply the same selective process termination with parent and child process relationships. Use the `pstree` command to view a process tree for the system or a single user. Use the parent process's PID to kill all children that it created. The parent `Bash` login shell survives this time, because the signal is directed only at its child processes.

```
[root@host ~]# pstree -p bob
bash(8391)─┬─sleep(8425)
           ├─sleep(8426)
           └─sleep(8427)
```

```
[root@host ~]# pkill -P 8391
[root@host ~]# pgrep -l -u bob
bash(8391)
[root@host ~]# pkill -SIGKILL -P 8391
[root@host ~]# pgrep -l -u bob
bash(8391)
```

Signal Multiple Processes

The `killall` command can signal multiple processes, based on their command name.

```
[user@host ~]$ ps aux | grep job
5194  0.0  0.1 222448  2980 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/contr
ol job1
5199  0.0  0.1 222448  3132 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/contr
ol job2
5205  0.0  0.1 222448  3124 pts/1    S    16:39   0:00 /bin/bash /home/user/bin/contr
ol job3
5430  0.0  0.0 221860  1096 pts/1    S+   16:41   0:00 grep --color=auto job
[user@host ~]$ killall control
[1]   Terminated              control job1
[2]-  Terminated              control job2
[3]+  Terminated              control job3
[user@host ~]$
```

Terminate Background Jobs

To terminate background jobs, use the `kill` command and specify the job number.

Use the `jobs` command to find the job number of the process to terminate.

```
[user@host ~]$ jobs
[1]-  Running                 sleep 500 &
[2]+  Running                 sleep 1000 &
[user@host ~]$
```

Terminate a specific job by using the `kill` command. Prefix the job number with a percent sign (%).

```
[user@host ~]$ kill -SIGTERM %1
[user@host ~]$ jobs
[2]+  Running                 sleep 1000 &
```

Administratively Log Out Users

You might need to log out other users for various reasons. Some possible scenarios: the user committed a security violation; the user might have overused resources; the user has an unresponsive system; or the user has improper access to materials. In these cases, you must terminate their session by using signals administratively.

First, to log off a user, identify the login session to be terminated. Use the `w` command to list user logins and currently running processes. Note the `TTY` and `FROM` columns to determine the closing sessions.

All user login sessions are associated with a terminal device (TTY). If the device name is `pts/N`, then it is a *pseudo-terminal* that is associated with a graphical terminal window or a remote login session. If it is `ttyN`, then the user is on a system console, an alternative console, or another directly connected terminal device.

```
[user@host ~]$ w
 12:43:06 up 27 min,  5 users,  load average: 0.03, 0.17, 0.66
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty2                      12:26   14:58   0.04s  0.04s -bash
bob      tty3                      12:28   14:42   0.02s  0.02s -bash
user     pts/1    desktop.example.com 12:41    2.00s  0.03s  0.03s w
[user@host ~]$
```

Discover how long a user has been on the system by viewing the session login time. CPU resources that current jobs consume, including background tasks and child processes, are in the `JCPU` column for each session. Current foreground process CPU consumption is in the `PCPU` column.

References

kill(1), killall(1), pgrep(1), pkill(1), pstree(1), signal(7), and w(1) man pages

For further reading, refer to *Signal Handling* at https://www.gnu.org/software/libc/manual/pdf/libc.pdf#Signal%20Handling

For further reading, refer to *Processes* at https://www.gnu.org/software/libc/manual/pdf/libc.pdf#Processes

Previous Next

Guided Exercise: Kill Processes

In this exercise, you use signals to manage and stop processes.

Outcomes

- Start and stop multiple shell processes.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start processes-kill
```

Instructions

1. On the workstation machine, open two terminal windows side by side. In this section, these terminals are referred to as *left* and *right*. In each terminal, use the ssh command to log in to the servera machine as the student user.

```
2. [student@workstation ~]$ ssh student@servera
```

```
[student@servera ~]$
```

3. In the left terminal shell, create the `/home/student/bin` directory. Create the `instance` shell script in the new directory. Change the script permissions so that it is executable.
    1. Create the `/home/student/bin` directory.

```
[student@servera ~]$ mkdir /home/student/bin
```

    2. Create the `instance` script file in the `/home/student/bin` directory. Press the **i** key to enter Vim interactive mode. The file must have the following content as shown. Use the `:wq` command to save the file.

```
[student@servera ~]$ cd /home/student/bin
[student@servera bin]$ vim /home/student/bin/instance
#!/bin/bash
while true; do
  echo -n "$@ " >> ~/instance_outfile
  sleep 5
```

```
done
```

Note

The `instance` script runs until the process is terminated. It appends command-line arguments to the `~/instance_outfile` file once every 5 seconds.

    9. Make the `instance` script file executable.

```
[student@servera ~]$ chmod +x /home/student/bin/instance
```

4. In the left terminal shell, change into the `/home/student/bin/` directory. Start three processes with the `instance` script file, by passing the `network`, `interface`, and `connection` arguments. Start the processes in the background.

```
[student@servera bin]$ instance network &
[1] 3460
[student@servera bin]$ instance interface &
[2] 3482
[student@servera bin]$ instance connection &
```

```
[3] 3516
```

10. In the right terminal shell, verify that all three processes are appending content to the `/home/student/instance_outfile` file.

```
11. [student@servera ~]$ tail -f ~/instance_outfile
12. network interface network connection interface network connection interface ne
    twork
```

```
...output omitted...
```

13. In the left terminal shell, list existing jobs.

```
14. [student@servera bin]$ jobs
15. [1]   Running                 instance network &
16. [2]-  Running                 instance interface &
```

```
[3]+  Running                 instance connection &
```

17. Use signals to suspend the `instance network` process. Verify that the `instance network` process is set to `Stopped`. Verify that the `network` process is no longer appending content to the `~/instance_output` file.
    1. Stop the `instance network` process. Verify that the process is stopped.

    ```
    2. [student@servera bin]$ kill -SIGSTOP %1
    3. [1]+  Stopped                 instance network
    4. [student@servera bin]$ jobs
    5. [1]+  Stopped                 instance network
    6. [2]   Running                 instance interface &
    ```

    ```
    [3]-  Running                 instance connection &
    ```

    7. In the right terminal shell, view the `tail` command output. Verify that the word `network` is no longer appended to the `~/instance_outfile` file.

    ```
    8. ...output omitted...
    ```

    ```
    interface connection interface connection interface connection interface
    ```

18. In the left terminal shell, terminate the `instance interface` process. Verify that the `instance interface` process disappeared. Verify that the `instance`

`interface` process output is no longer appended to the `~/instance_outfile` file.

1. Terminate the `instance interface` process. Verify that the process is terminated.

```
2. [student@servera bin]$ kill -SIGTERM %2

3. [student@servera bin]$ jobs

4. [1]+  Stopped                 instance network

5. [2]   Terminated              instance interface
```

```
[3]-  Running                 instance connection &
```

6. In the right terminal shell, view the `tail` command output. Verify that the word `interface` is no longer appended to the `~/instance_outfile` file.

```
7. ...output omitted...
```

```
connection connection connection connection connection connection connec
tion connection
```

19. In the left terminal shell, resume the `instance network` process. Verify that the `instance network` process is set to `Running`. Verify that the `instance network` process output is appended to the `~/instance_outfile` file.

    1. Resume the `instance network` process. Verify that the process is in the `Running` state.

```
2. [student@servera bin]$ kill -SIGCONT %1

3. [student@servera bin]$ jobs

4. [1]+  Running                 instance network &
```

```
[3]-  Running                 instance connection &
```

5. In the right terminal shell, view the `tail` command output. Verify that the word `network` is appended to the `~/instance_outfile` file.

```
6. ...output omitted...
```

```
network connection network connection network connection network connect
ion network connection
```

20. In the left terminal shell, terminate the remaining two jobs. Verify that no jobs remain and that output is stopped.
    1. Terminate the `instance network` process. Next, terminate the `instance connection` process.

```
[student@servera bin]$ kill -SIGTERM %1
[student@servera bin]$ kill -SIGTERM %3
[1]+ Terminated              instance network
[student@servera bin]$ jobs
```

```
[3]+ Terminated              instance connection
```

21. In the left terminal shell, list the current running processes in all open terminal shells. Terminate the `tail` processes. Verify that the processes are no longer running.
    1. List all current running processes. Refine the search to view only `tail` lines.

```
[student@servera bin]$ ps -ef | grep tail
student   4581 31358  0 10:02 pts/0    00:00:00 tail -f instance_outfile
```

```
student   4869  2252  0 10:33 pts/1    00:00:00 grep --color=auto tail
```

    4. Terminate the `tail` process. Verify that the process is no longer running.

```
[student@servera bin]$ pkill -SIGTERM tail
[student@servera bin]$ ps -ef | grep tail
```

```
student   4874  2252  0 10:36 pts/1    00:00:00 grep --color=auto tail
```

    7. In the right terminal shell, verify that the `tail` command is no longer running.

```
...output omitted...
network connection network connection network connection Terminated
```

```
[student@servera ~]$
```

22. Close the extra terminal. Return to the `workstation` system as the `student` user.

```
23. [student@servera bin]$ exit
24. logout
25. Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish processes-kill
```

This concludes the section.

Previous Next

Monitor Process Activity

Objectives

Define load average and determine resource-intensive server processes.

Describe Load Average

Load average is a measurement that the Linux kernel provides, to represent the perceived system load for a period of time. It can be used as a rough gauge of how many system resource requests are pending, to determine whether system load increases or decreases.

The kernel collects the current load number every five seconds based on the number of processes in runnable and uninterruptible states. This number is accumulated and reported as an exponential moving average over the most recent 1, 5, and 15 minutes.

Load Average Calculation

The load average represents the perceived system load for a period of time. Linux determines load average by reporting how many processes are ready to run on a CPU and how many processes are waiting for disk or network I/O to complete.

- The load number is a running average of the number of processes that are ready to run (in process state `R`) or are waiting for I/O to complete (in process state `D`).
- Some UNIX systems consider only CPU usage or run queue length to indicate system load. Linux also includes disk or network usage, because the high usage of these resources can significantly impact system performance as CPU load. For high load averages with minimal CPU activity, examine disk and network activity.

Load average is a rough measurement of how many processes are currently waiting for a request to complete before they do anything else. The request might be for CPU time to run the process. Alternatively, the request might be for a critical disk I/O operation to complete, and the process cannot be run on the CPU until the request completes, even though the CPU is idle. Either way, system load is impacted, and the system appears to run more slowly because processes are waiting to run.

Interpret Load Average Values

The `uptime` command is one way to display the current load average. It prints the current time, how long the machine has been up, how many user sessions are running, and the current load average.

```
[user@host ~]$ uptime
 15:29:03 up 14 min,  2 users,  load average: 2.92, 4.48, 5.20
```

The three values for the load average represent the load over the last 1, 5, and 15 minutes. It indicates whether the system load appears to be increasing or decreasing.

If the main contribution to load average is from processes that are waiting for the CPU, then you can calculate the approximate load value per CPU to determine whether the system is experiencing significant waiting.

Use the `lscpu` command to determine the number of CPUs on a system.

In the following example, the system is a dual-core single-socket system with two hyper threads per core. Linux treats this CPU configuration as a four-CPU system for scheduling purposes.

```
[user@host ~]$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  2
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
...output omitted...
```

Imagine that the only contribution to the load number is from processes that need CPU time. Then you can divide the displayed load average values by the number of logical CPUs in the system. A value below 1 indicates adequate resource use and minimal wait times. A value above 1 indicates resource saturation and some processing delay.

```
# From lscpu, the system has four logical CPUs, so divide by 4:
#                               load average: 2.92, 4.48, 5.20
#           divide by number of logical CPUs:    4     4     4
#                                             ----  ----  ----
#                       per-CPU load average: 0.73  1.12  1.30
#
# This system's load average appears to be decreasing.
# With a load average of 2.92 on four CPUs, all CPUs were in use ~73% of the time.
# During the last 5 minutes, the system was overloaded by ~12%.
# During the last 15 minutes, the system was overloaded by ~30%.
```

An idle CPU queue has a load number of 0. Each process that waits for a CPU adds a count of 1 to the load number. If one process is running on a CPU, then the load number is 1, and the resource (the CPU) is in use, but no requests are waiting. If that process runs for an entire minute, then its contribution to the one-minute load average is 1.

However, processes that are uninterruptibly sleeping for critical I/O due to a busy disk or network resource are also included in the count and increase the load average. Although not indicating CPU use, these processes are added to the queue count, because they wait for resources and cannot run on a CPU until they get the resources. This metric is still considered as system load due to resource limitations that cause processes not to run.

Until resource saturation, a load average remains below 1, because tasks are seldom found waiting in the queue. Load average increases only when resource saturation causes requests to remain queued, and the load calculation routine counts them. When resource use approaches 100%, each extra request starts experiencing service wait time.

Real-time Process Monitoring

The `top` command displays a dynamic view of the system's processes and a summary header followed by a process or thread list. Unlike the static `ps` command output, the `top` command continuously refreshes at a configurable interval and provides column reordering, sorting, and highlighting. You can make persistent changes to the `top` settings. The default `top` output columns are as follows:

- The process ID (`PID`).
- The process owner username (`USER`).
- Virtual memory (`VIRT`) is all the memory that the process uses, including the resident set, shared libraries, and any mapped or swapped memory pages (labeled `VSZ` in the `ps` command).
- Resident memory (`RES`) is the physical memory that the process uses, including any resident, shared objects (labeled `RSS` in the `ps` command).
- Process state (`S`) can be one of the following states:
    - `D` = Uninterruptible Sleeping
    - `R` = Running or Runnable
    - `S` = Sleeping
    - `T` = Stopped or Traced
    - `Z` = Zombie
- CPU time (`TIME`) is the total processing time since the process started. It can be toggled to include a cumulative time of all previous children.

- The process command name (`COMMAND`).

Table 8.3. Fundamental Keystrokes in `top` Command

Key	Purpose
? *or* **h**	Help for interactive keystrokes.
l, **t**, **m**	Toggles for load, threads, and memory header lines.
1	Toggle for individual CPUs or a summary for all CPUs in the header.
s	Change the refresh (screen) rate, in decimal seconds (such as 0.5, 1, 5).
b	Toggle reverse highlighting for `Running` processes; the default is bold only.
Shift+**b**	Enables bold use in display, in the header, and for *Running* processes.
Shift+**h**	Toggle threads; show process summary or individual threads.
u, **Shift**+**u**	Filter for any username (effective, real).
Shift+**m**	Sort process listing by memory usage, in descending order.
Shift+**p**	Sort process listing by processor use, in descending order.
k	Kill a process. When prompted, enter `PID`, and then `signal`.
r	Renice a process. When prompted, enter `PID`, and then `nice_value`.
Shift+**w**	Write (save) the current display configuration for use at the next `top` restart.
q	Quit.
f	Manage the columns by enabling or disabling fields. You can also set the sort field for `top`.

Note

The **s**, **k**, and **r** keystrokes are not available when the `top` command is started in a secure mode.

References

`ps`(1), `top`(1), `uptime`(1), and `w`(1) man pages

Previous Next

Guided Exercise: Monitor Process Activity

In this exercise, you use the `top` command to examine running processes and control them dynamically.

Outcomes

- Manage processes in real time.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start processes-monitor
```

Instructions

1. On the `workstation` machine, open two terminal windows side by side. In this section, these terminals are referred to as *left* and *right*. In each terminal, log in to the `servera` machine as the `student` user.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
```

```
[student@servera ~]$
```

4. In the left terminal shell, create the `/home/student/bin` directory. Create a shell script called `monitor` in the new directory to generate an artificial CPU load. Make the `monitor` script file executable.
    1. Create the `/home/student/bin` directory.

    ```
    [student@servera ~]$ mkdir /home/student/bin
    ```

    2. Create the script file in the `/home/student/bin` directory with the content shown.

    ```
    [student@servera ~]$ vim /home/student/bin/monitor
    #!/bin/bash
    ```

```
while true; do
  var=1
  while [[ var -lt 60000 ]]; do
    var=$(($var+1))
  done
  sleep 1
```

```
done
```

Note

The `monitor` script runs until the process is terminated. It generates an artificial CPU load by performing sixty thousand addition calculations. After generating the CPU load, it then sleeps for one second, resets the variable, and repeats.

11. Make the `monitor` file executable.

```
[student@servera ~]$ chmod a+x /home/student/bin/monitor
```

5. In the right terminal shell, run the `top` command. Resize the window to view the contents of the command.

```
[student@servera ~]$ top
top - 12:13:03 up 11 days, 58 min,  3 users,  load average: 0.00, 0.00, 0.00
Tasks: 113 total,   2 running, 111 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.0 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   1829.4 total,   1377.3 free,    193.9 used,    258.2 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   1476.1 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
5861 root      20   0       0      0      0 I   0.3   0.0   0:00.71 kworker/1:3-events
6068 student   20   0  273564   4300   3688 R   0.3   0.2   0:00.01 top
   1 root      20   0  178680  13424   8924 S   0.0   0.7   0:04.03 systemd
   2 root      20   0       0      0      0 S   0.0   0.0   0:00.03 kthreadd
```

```
18.   3 root        0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
```

```
...output omitted...
```

19. In the left terminal shell, verify the number of logical CPUs on this virtual machine.

```
20. [student@servera ~]$ lscpu
21. Architecture:        x86_64
22. CPU op-mode(s):      32-bit, 64-bit
23. Byte Order:          Little Endian
24. CPU(s):              2
```

```
...output omitted...
```

25. In the left terminal shell, run a single instance of the `monitor` script file in the background.

```
26. [student@servera ~]$ monitor &
```

```
[1] 6071
```

27. In the right terminal shell, monitor the `top` command. Use the **l**, **t**, and **m** single keystrokes to toggle the load, threads, and memory header lines. After observing this behavior, ensure that all headers are displayed.
28. Note the process ID (PID) for the `monitor` process. View the CPU percentage for the process, which is expected to hover around 15% to 25%.

```
29. [student@servera ~]$ top
30. PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
31. 071 student   20   0  222448   2964   2716 S  18.7   0.2   0:27.35 monitor
```

```
...output omitted...
```

View the load averages. The one-minute load average value is currently less than 1. The observed value might be affected by resource contention from another virtual machine or from the virtual host.

```
top - 12:23:45 up 11 days,  1:09,  3 users,  load average: 0.21, 0.14, 0.05
```

32. In the left terminal shell, run a second instance of the `monitor` script file in the background.

```
33. [student@servera ~]$ monitor &
```

```
[2] 6498
```

34. In the right terminal shell, note the process ID (PID) for the second `monitor` process. View the CPU percentage for the process, which is also expected to hover between 15% to 25%.

```
35. [student@servera ~]$ top
36.  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
37. 6071 student   20   0  222448   2964   2716 S  19.0   0.2   1:36.53 monitor
38. 6498 student   20   0  222448   2996   2748 R  15.7   0.2   0:16.34 monitor
```

```
...output omitted...
```

Again, view the one-minute load average, which remains less than 1. Wait at least one minute for the calculation to adjust to the new workload.

```
top - 12:27:39 up 11 days,  1:13,  3 users,  load average: 0.36, 0.25, 0.11
```

39. In the left terminal shell, run a third instance of the `monitor` script file in the background.

```
40. [student@servera ~]$ monitor &
```

```
[3] 6881
```

41. In the right terminal shell, note the process ID (PID) for the third `monitor` process. View the CPU percentage for the process, which is again expected to hover between 15% to 25%.

```
42. [student@servera ~]$ top
43.  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
44. 6881 student   20   0  222448   3032   2784 S  18.6   0.2   0:11.48 monitor
45. 6498 student   20   0  222448   2996   2748 S  15.6   0.2   0:47.86 monitor
```

```
6071 student   20   0  222448   2964   2716 S  18.1   0.2   2:07.86 monitor
```

To push the load average above 1, you must start more `monitor` processes. The classroom setup has two CPUs, so only three processes are not enough to stress it. Start three more `monitor` processes in the background. View again the one-minute load average, which is now expected to be above 1. Wait at least one minute for the calculation to adjust to the new workload.

```
[student@servera ~]$ monitor &
[4] 10708
[student@servera ~]$ monitor &
[5] 11122
[student@servera ~]$ monitor &
[6] 11338
top - 12:42:32 up 11 days,  1:28,  3 users,  load average: 1.23, 2.50, 1.54
```

46. When you are finished observing the load average values, terminate each of the `monitor` processes from within the `top` command.
    1. In the right terminal shell, press **k** to observe the prompt below the headers and above the columns.

```
2. ...output omitted...
```

```
PID to signal/kill [default pid = 11338]
```

    3. The prompt chooses the `monitor` processes at the top of the list. Press **Enter** to kill the process.

```
4. ...output omitted...
```

```
Send pid 11338 signal [15/sigterm]
```

    5. Press **Enter** again to confirm the SIGTERM signal 15.

       Verify that the selected process is no longer present in the `top` command. If the PID exists, then repeat these steps to terminate the processes, and substitute SIGKILL signal 9 when prompted.

```
 6498 student   20   0  222448   2996   2748 R  22.9   0.2   5:31.47 mon
itor
```

```
 6881 student   20   0  222448   3032   2784 R  21.3   0.2   4:54.47 mon
itor

11122 student   20   0  222448   2984   2736 R  15.3   0.2   2:32.48 mon
itor

 6071 student   20   0  222448   2964   2716 S  15.0   0.2   6:50.90 mon
itor

10708 student   20   0  222448   3032   2784 S  14.6   0.2   2:53.46 mon
itor
```

47. Repeat the previous step for each remaining `monitor` process. Verify that no `monitor` processes remain in the `top` command.
48. In the right terminal shell, press **q** to exit the `top` command. Close the right terminal.
49. Return to the `workstation` machine as the `student` user.

```
50. [student@servera ~]$ exit
51. logout
52. Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish processes-monitor
```

This concludes the section.

Previous Next

Summary

- A process is a running instance of an executable program. Processes are assigned a state, which can be running, sleeping, stopped, or zombie. The `ps` command lists processes.

- Each terminal has its own session, and can have a foreground process and independent background processes. The `jobs` command displays processes within a terminal session.
- A signal is a software interrupt that reports events to an executing program. The `kill`, `pkill`, and `killall` commands use signals to control processes.
- Load average is an estimate of how busy the system is. To display load average values, you can use the `top`, `uptime`, or `w` commands.

Previous Next

Chapter 9. Control Services and Daemons

Abstract

Goal	Control and monitor network services and system daemons with the `systemd` service.
Objectives	• List system daemons and network services that the `systemd` service and socket units started. • Control system daemons and network services with the `systemctl` command.
Sections	• Identify Automatically Started System Processes (and Guided Exercise) • Control System Services (and Guided Exercise)
Lab	Control Services and Daemons

Identify Automatically Started System Processes

Objectives

List system daemons and network services that were started by the `systemd` service and socket units.

Introduction to the systemd Daemon

The `systemd` daemon manages the startup process for Linux, including service startup and service management in general. The `systemd` daemon activates system resources, server daemons, and other processes, both at boot time and on a running system.

Daemons are processes that either wait or run in the background, to perform various tasks. Generally, daemons start automatically at boot time and continue to run until shutdown or until you manually stop them. By convention, daemon names end with `d`.

A *service* in the `systemd` sense often refers to one or more daemons. However, starting or stopping a service might instead change the state of the system once, without leaving a running daemon process afterward (called `oneshot`).

In Red Hat Enterprise Linux, the first process that starts (PID 1) is the `systemd` daemon, which provides these features:

- Parallelization capabilities (starting multiple services simultaneously), which increase the boot speed of a system.
- On-demand starting of daemons without requiring a separate service.
- Automatic service dependency management, which can prevent long timeouts. For example, a network-dependent service does not try to start until the network is available.
- A method of tracking related processes together by using Linux control groups.

Service Units Description

A `systemd` *unit* is an abstract concept to define objects that the system knows how to manage.

Units are represented by configuration files called *unit files*, which encapsulate information about system services, listening sockets, and other relevant objects for the `systemd` init system.

A unit has a name and a unit type. The name provides a unique identity to the unit. The unit type enables grouping units together with other similar unit types.

The `systemd` daemon uses units to manage different types of objects:

- *Service units* have a `.service` extension and represent system services. You can use service units to start often-accessed daemons, such as a web server.
- *Socket units* have a `.socket` extension and represent inter-process communication (IPC) sockets that `systemd` should monitor. If a client connects to the socket, then the `systemd` manager starts a daemon and passes the connection to it. You can use socket units to delay the start of a service at boot time and to start less often used services on demand.
- *Path units* have a `.path` extension and delay the activation of a service until a specific file-system change occurs. You can use path units for services that use spool directories, such as a printing system.

To manage units, use the `systemctl` command. For example, display available unit types with the `systemctl -t help` command. The `systemctl` command can take abbreviated unit names, process tree entries, and unit descriptions.

List Service Units

Use the `systemctl` command to explore the system's current state. For example, the following command lists and paginates all currently loaded service units.

```
[root@host ~]# systemctl list-units --type=service
UNIT                   LOAD    ACTIVE  SUB      DESCRIPTION
atd.service            loaded  active  running  Job spooling tools
auditd.service         loaded  active  running  Security Auditing Service
chronyd.service        loaded  active  running  NTP client/server
crond.service          loaded  active  running  Command Scheduler
dbus.service           loaded  active  running  D-Bus System Message Bus
...output omitted...
```

In this example, the `--type=service` option limits the type of `systemd` units to service units. The output has the following columns:

UNIT

The service unit name.

LOAD

Whether the systemd daemon correctly parsed the unit's configuration and loaded the unit into memory.

ACTIVE

The high-level activation state of the unit. This information indicates whether the unit started successfully.

SUB

The low-level activation state of the unit. This information indicates more detailed information about the unit. The information varies based on unit type, state, and how the unit is executed.

DESCRIPTION

The short description of the unit.

By default, the `systemctl list-units --type=service` command lists only the service units with `active` activation states. The `systemctl list-units --all` option lists all service units regardless of the activation states. Use the `--state=` option to filter by the values in the `LOAD`, `ACTIVE`, or `SUB` fields.

```
[root@host ~]# systemctl list-units --type=service --all
UNIT                          LOAD      ACTIVE   SUB     DESCRIPTION
  atd.service                 loaded    active   running Job spooling tools
  auditd.service              loaded    active   running Security Auditing ...
  auth-rpcgss-module.service  loaded    inactive dead    Kernel Module ...
  chronyd.service             loaded    active   running NTP client/server
  cpupower.service            loaded    inactive dead    Configure CPU power ...
  crond.service               loaded    active   running Command Scheduler
  dbus.service                loaded    active   running D-Bus System Message Bus
● display-manager.service     not-found inactive dead    display-manager.service
...output omitted...
```

The `systemctl` command without any arguments lists units that are both loaded and active.

```
[root@host ~]# systemctl
UNIT                                 LOAD   ACTIVE SUB       DESCRIPTION
proc-sys-fs-binfmt_misc.automount    loaded active waiting   Arbitrary...
sys-devices-....device               loaded active plugged   Virtio network...
sys-subsystem-net-devices-ens3.deviceloaded active plugged   Virtio network...
...output omitted...
-.mount                              loaded active mounted   Root Mount
boot.mount                           loaded active mounted   /boot
...output omitted...
systemd-ask-password-plymouth.path   loaded active waiting   Forward Password...
systemd-ask-password-wall.path       loaded active waiting   Forward Password...
init.scope                           loaded active running   System and Servi...
session-1.scope                      loaded active running   Session 1 of...
atd.service                          loaded active running   Job spooling tools
auditd.service                       loaded active running   Security Auditing...
chronyd.service                      loaded active running   NTP client/server
crond.service                        loaded active running   Command Scheduler
...output omitted...
```

The `systemctl` command `list-units` option displays units that the `systemd` service attempts to parse and load into memory. This option does not display services that are installed but not enabled. You can use the `systemctl` command `list-unit-files` option to see the state of all the installed unit files:

```
[root@host ~]# systemctl list-unit-files --type=service
UNIT FILE                         STATE       VENDOR PRESET
arp-ethers.service                disabled    disabled
atd.service                       enabled     enabled
auditd.service                    enabled     enabled
auth-rpcgss-module.service        static      -
autovt@.service                   alias       -
blk-availability.service          disabled    disabled
...output omitted...
```

In the output of the `systemctl list-unit-files` command, some common entries for the `STATE` field are `enabled`, `disabled`, `static`, and `masked`. All `STATE` values are listed in the `systemctl` command manual pages.

View Service States

View a unit's status with the `systemctl status` *`name.type`* command. If the unit type is omitted, then the command expects a service unit with that name.

```
[root@host ~]# systemctl status sshd.service
● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: en
abled)
     Active: active (running) since Mon 2022-03-14 05:38:12 EDT; 25min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 1114 (sshd)
      Tasks: 1 (limit: 35578)
     Memory: 5.2M
        CPU: 64ms
     CGroup: /system.slice/sshd.service
             └─1114 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Mar 14 05:38:12 workstation systemd[1]: Starting OpenSSH server daemon...
Mar 14 05:38:12 workstation sshd[1114]: Server listening on 0.0.0.0 port 22.
Mar 14 05:38:12 workstation sshd[1114]: Server listening on :: port 22.
Mar 14 05:38:12 workstation systemd[1]: Started OpenSSH server daemon.
...output omitted...
```

Some fields from the `systemctl` command `status` option output:

Table 9.1. Service Unit Information

Field	Description
Loaded	Whether the service unit is loaded into memory

Field	**Description**
Active	Whether the service unit is running, and if so, for how long
Docs	Where to find more information about the service
Main PID	The main process ID of the service, including the command name
Status	More information about the service
Process	More information about related processes
CGroup	More information about related control groups

Not all these fields are always present in the command output.

Keywords in the status output indicate the state of the service:

Table 9.2. Service States in the Output of systemctl

Keyword	**Description**
loaded	The unit configuration file is processed.
active (running)	The service is running with continuing processes.
active (exited)	The service successfully completed a one-time configuration.
active (waiting)	The service is running but is waiting for an event.
inactive	The service is not running.
enabled	The service starts at boot time.
disabled	The service is not set to start at boot time.
static	The service cannot be enabled, but an enabled unit might start it automatically.

Note

The `systemctl status` *`NAME`* command replaces the `service` *`NAME`* `status` command from Red Hat Enterprise Linux 6 and earlier versions.

Verify the Status of a Service

The `systemctl` command verifies the specific states of a service. For example, use the `systemctl` command `is-active` option to verify whether a service unit is active (running):

```
[root@host ~]# systemctl is-active sshd.service
active
```

The command returns the service unit state, which is usually `active` or `inactive`.

Run the `systemctl` command `is-enabled` option to verify whether a service unit is enabled to start automatically during system boot:

```
[root@host ~]# systemctl is-enabled sshd.service
enabled
```

The command returns whether the service unit is enabled to start at boot time, and the state is usually `enabled` or `disabled`.

To verify whether the unit failed during startup, run the `systemctl` command `is-failed` option:

```
[root@host ~]# systemctl is-failed sshd.service
active
```

The command returns `active` if the service is correctly running, or `failed` if an error occurred during startup. If the unit was stopped, then it returns `unknown` or `inactive`.

To list all the failed units, run the `systemctl --failed --type=service` command.

References

`systemd`(1), `systemd.unit`(5), `systemd.service`(5), `systemd.socket`(5), and `systemctl`(1) man pages

For more information, refer to the *Managing Services with systemd* chapter in the *Red Hat Enterprise Linux 9 Configuring Basic System Settings Guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/configuring_basic_system_settings/managing-services-with-systemd_configuring-basic-system-settings#managing-services-with-systemd_configuring-basic-system-settings

Next

Guided Exercise: Identify Automatically Started System Processes

In this exercise, you list installed service units and identify which services are currently enabled and active on a server.

Outcomes

- List the installed service units.
- Identify active and enabled services on the system.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start services-identify
```

Instructions

1. Use the `ssh` command to log in to the `servera` machine as the `student` user.

```
2. [student@workstation ~]$ ssh student@servera
```

```
[student@servera ~]$
```

3. List all the installed service units on the `servera` machine.

```
4. [student@servera ~]$ systemctl list-units --type=service
```

```
UNIT                   LOAD    ACTIVE  SUB      DESCRIPTION
atd.service            loaded  active  running  Deferred execution scheduler
auditd.service         loaded  active  running  Security Auditing Service
chronyd.service        loaded  active  running  NTP client/server
crond.service          loaded  active  running  Command Scheduler
dbus-broker.service    loaded  active  running  D-Bus System Message Bus
```

```
...output omitted...
```

Press **q** to exit the command.

11. List all socket units, active and inactive, on the `servera` machine.

```
[student@servera ~]$ systemctl list-units --type=socket --all
UNIT                   LOAD    ACTIVE  SUB        DESCRIPTION
dbus.socket            loaded  active  running    D-Bus System Message Bus Socket
dm-event.socket        loaded  active  listening  Device-mapper event daemon FIFOs
lvm2-lvmpolld.socket   loaded  active  listening  LVM2 poll daemon socket
...output omitted...

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.
13 loaded units listed.
```

```
To show all installed unit files use 'systemctl list-unit-files'.
```

23. Explore the status of the `chronyd` service. You can use this service for network time protocol synchronization (NTP).
    1. Display the status of the `chronyd` service. Note the process ID of any active daemon.

    ```
    [student@servera ~]$ systemctl status chronyd
    ● chronyd.service - NTP client/server
         Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled; vendor preset: enabled)
    ```

```
     Active: active (running) since Mon 2022-03-14 05:38:15 EDT; 1h 16min ago
       Docs: man:chronyd(8)
             man:chrony.conf(5)
    Process: 728 ExecStart=/usr/sbin/chronyd $OPTIONS (code=exited, status=0/SUCCESS)
   Main PID: 747 (chronyd)
      Tasks: 1 (limit: 10800)
     Memory: 3.7M
        CPU: 37ms
     CGroup: /system.slice/chronyd.service
             └─747 /usr/sbin/chronyd -F 2

Mar 14 05:38:15 servera.lab.example.com systemd[1]: Starting NTP client/server...
Mar 14 05:38:15 servera.lab.example.com chronyd[747]: chronyd version 4.1 starting (+CMDMON +NTP +REFCLOCK +RTC +PRIVDROP +SCFILTER +SIGND +ASYNCDNS +NTS +SECHASH +IPV6 +DEBUG)
Mar 14 05:38:15 servera.lab.example.com chronyd[747]: commandkey directive is no longer supported
Mar 14 05:38:15 servera.lab.example.com chronyd[747]: generatecommandkey directive is no longer supported
Mar 14 05:38:15 servera.lab.example.com chronyd[747]: Frequency -11.870 +/- 1.025 ppm read from /var/lib/chrony/drift
Mar 14 05:38:15 servera.lab.example.com chronyd[747]: Loaded seccomp filter (level 2)
Mar 14 05:38:15 servera.lab.example.com systemd[1]: Started NTP client/server.
```

```
Mar 14 05:38:23 servera.lab.example.com chronyd[747]: Selected source 172.25.254.254
```

Press **q** to exit the command.

23. Confirm that the `chronyd` daemon is running by using its process ID. In the preceding command, the output of the process ID that is associated with the `chronyd` service is 747. The process ID might differ on your system.

```
[student@servera ~]$ ps -p 747
```

```
  PID TTY          TIME CMD
```

```
  747 ?        00:00:00 chronyd
```

24. Explore the status of the `sshd` service. You can use this service for secure encrypted communication between systems.
    1. Determine whether the `sshd` service is enabled to start at system boot.

```
[student@servera ~]$ **systemctl is-enabled sshd**
```

```
enabled
```

    3. Determine whether the `sshd` service is active without displaying all the status information.

```
[student@servera ~]$ **systemctl is-active sshd**
```

```
active
```

    5. Display the status of the `sshd` service.

```
[student@servera ~]$ **systemctl status sshd**
● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2022-03-14 05:38:16 EDT; 1h 19min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 784 (sshd)
      Tasks: 1 (limit: 10800)
     Memory: 6.7M
        CPU: 82ms
     CGroup: /system.slice/sshd.service
             └─784 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Mar 14 05:38:16 servera.lab.example.com systemd[1]: Starting OpenSSH server daemon...
Mar 14 05:38:16 servera.lab.example.com sshd[784]: Server listening on 0.0.0.0 port 22.
Mar 14 05:38:16 servera.lab.example.com sshd[784]: Server listening on :: port 22.
Mar 14 05:38:16 servera.lab.example.com systemd[1]: Started OpenSSH server daemon.
Mar 14 06:51:36 servera.lab.example.com sshd[1090]: Accepted publickey for student from 172.25.250.9 port 53816 ssh2: RSA SHA256:M8ikhcEDm2tQ95Z0o7ZvufqEixCFCt+wowZLNzNlBT0
```

```
Mar 14 06:51:36 servera.lab.example.com sshd[1090]: pam_unix(sshd:session): session opened for user student(uid=1000) by (uid=0)
```

Press **q** to exit the command.

25. List the enabled or disabled states of all service units.

```
[student@servera ~]$ systemctl list-unit-files --type=service
UNIT FILE                         STATE           VENDOR PRESET
arp-ethers.service                disabled        disabled
atd.service                       enabled         enabled
auditd.service                    enabled         enabled
auth-rpcgss-module.service        static          -
autovt@.service                   alias           -
blk-availability.service          disabled        disabled
bluetooth.service                 enabled         enabled
chrony-wait.service               disabled        disabled
chronyd.service                   enabled         enabled
```

```
...output omitted...
```

Press **q** to exit the command.

37. Return to the `workstation` system as the `student` user.

```
[student@servera ~]$ exit
logout
Connection to servera closed.
```

```
[student@workstation]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish services-identify
```

This concludes the section.

Previous Next

Control System Services

Objectives

Control system daemons and network services with `systemctl`.

Start and Stop Services

You can manually start, stop, or reload services to update the service, update the configuration file, uninstall the service, or manually manage an infrequently used service.

Use the `systemctl status` command to verify the status of a service, if the service is running or stopped.

```
[root@host ~]# systemctl status sshd.service
● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: en
abled)
     Active: active (running) since Wed 2022-03-23 11:58:18 EDT; 2min 56s ago
...output omitted...
```

Use the `systemctl start` command as the `root` user (with the `sudo` command if necessary). If you run the `systemctl start` command with the service name only (without the service type), then the `systemd` service looks for `.service` files.

```
[root@host ~]# systemctl start sshd
```

To stop a running service, use the `systemctl` command `stop` option. The following example shows how to stop the `sshd.service` service:

```
[root@host ~]# systemctl stop sshd.service
```

Restart and Reload Services

When you restart a running service, the service first stops and then starts again. On the service restart, the new process gets a new ID during the startup and thus the process ID changes. To restart a running service, use the `systemctl` command `restart` option. The following example shows how to restart the `sshd` service:

```
[root@host ~]# systemctl restart sshd.service
```

Some services can reload their configuration files without requiring a restart, which is called a *service reload*. Reloading a service does not change the process ID that is associated with various service processes. To reload a running service, use the `systemctl` command `reload` option. The following example shows how to reload the `sshd.service` service after configuration changes:

```
[root@host ~]# systemctl reload sshd.service
```

If you are unsure whether the service has the function to reload the configuration file changes, use the `systemctl` command `reload-or-restart` option. The command reloads the configuration changes if the reloading function is available. Otherwise, the command restarts the service to implement the new configuration changes:

```
[root@host ~]# systemctl reload-or-restart sshd.service
```

List Unit Dependencies

Some services require other services to be running first, which creates dependencies on the other services. Other services start only on demand, rather than at boot time. In both cases, the `systemd` and `systemctl` commands start services as needed, whether to resolve the dependency or to start an infrequently used service. For example, if the printing system (CUPS) service is not running and you place a file into the print spool directory, then the system starts the CUPS-related daemons or commands to satisfy the print request.

```
[root@host ~]# systemctl stop cups.service
Warning: Stopping cups, but it can still be activated by:
  cups.path
  cups.socket
```

However, to stop all the printing services on a system, you must stop all three units. Disabling the service disables the dependencies.

The `systemctl list-dependencies UNIT` command displays a hierarchy mapping of dependencies to start the service unit. To list reverse dependencies (units that depend on the specified unit), use the `--reverse` option with the command.

```
[root@host ~]# systemctl list-dependencies sshd.service
sshd.service
● ├─system.slice
● ├─sshd-keygen.target
● │ ├─sshd-keygen@ecdsa.service
● │ ├─sshd-keygen@ed25519.service
● │ └─sshd-keygen@rsa.service
● └─sysinit.target
...output omitted...
```

Mask and Unmask Services

At times, some installed services on your system might conflict with each other. For example, many ways exist to manage mail servers
(the `postfix` and `sendmail` services). Masking a service prevents an administrator from accidentally starting a service that conflicts with other services. Masking creates a link in the configuration directories to the `/dev/null` file, which prevents the service from starting. To mask a service, use
the `systemctl` command `mask` option.

```
[root@host ~]# systemctl mask sendmail.service
Created symlink /etc/systemd/system/sendmail.service → /dev/null.
```

Then, verify the state of the service by using the `systemctl list-unit-files` command:

```
[root@host ~]# systemctl list-unit-files --type=service
UNIT FILE                                  STATE
...output omitted...
sendmail.service                           masked
...output omitted...
```

Attempting to start a masked service unit fails with the following output:

```
[root@host ~]# systemctl start sendmail.service
Failed to start sendmail.service: Unit sendmail.service is masked.
```

Use the `systemctl unmask` command to unmask the service unit.

```
[root@host ~]# systemctl unmask sendmail
Removed /etc/systemd/system/sendmail.service.
```

Important

You, or another unit file, can manually start a disabled service, but it does not start automatically at boot. A masked service does not start manually or automatically.

Enable Services to Start or Stop at Boot

Starting a service on a running system does not guarantee that the service automatically starts when the system reboots. Similarly, stopping a service on a running system does not prevent it from starting again when the system reboots. Creating links in the `systemd` configuration directories enables the service to start at boot. You can create or remove these links by using the `systemctl` command with the `enable` or `disable` option.

```
[root@root ~]# systemctl enable sshd.service
Created symlink /etc/systemd/system/multi-user.target.wants/sshd.service → /usr/lib/s
ystemd/system/sshd.service.
```

This command creates a symbolic link from the service unit file, usually in the `/usr/lib/systemd/system` directory, to the disk location where the `systemd` command looks for files, in the `/etc/systemd/system/TARGETNAME.target.wants` directory. Enabling a service does not start the service in the current session. To start the service and enable it to start automatically during boot, you can execute both the `systemctl start` and `systemctl enable` commands, or use the equivalent `systemctl enable --now` command.

```
[root@root ~]# systemctl enable --now sshd.service
Created symlink /etc/systemd/system/multi-user.target.wants/sshd.service → /usr/lib/s
ystemd/system/sshd.service.
```

To disable the service from starting automatically, use the `systemctl disable` command, which removes the symbolic link that was created when enabling a service. Disabling a service does not stop the service if it is currently running. To disable and stop a service, you can execute both the `systemctl stop` and `systemctl disable` commands, or use the equivalent `systemctl disable --now` command.

```
[root@host ~]# systemctl disable --now sshd.service
Removed /etc/systemd/system/multi-user.target.wants/sshd.service.
```

To verify whether the service is enabled or disabled, use the `systemctl is-enabled` command.

```
[root@host ~]# systemctl is-enabled sshd.service
enabled
```

Summary of systemctl Commands

You can start and stop services on a running system, and enable or disable them for an automatic start at boot time.

Table 9.3. Useful Service Management Commands

Command	Task
`systemctl status UNIT`	View detailed information about a unit's state.

Command	Task
`systemctl stop UNIT`	Stop a service on a running system.
`systemctl start UNIT`	Start a service on a running system.
`systemctl restart UNIT`	Restart a service on a running system.
`systemctl reload UNIT`	Reload the configuration file of a running service.
`systemctl mask UNIT`	Disable a service from being started, both manually and at boot.
`systemctl unmask UNIT`	Make available a masked service.
`systemctl enable UNIT`	Configure a service to start at boot time. Use the `--now` option to also start the service.
`systemctl disable UNIT`	Disable a service from starting at boot time. Use the `--now` option to also stop the service.

References

`systemd`(1), `systemd.unit`(5), `systemd.service`(5), `systemd.socket`(5), and `systemctl`(1) man pages

For more information, refer to the *Managing System Services with systemctl* chapter in the *Red Hat Enterprise Linux 9 Configuring Basic System Settings Guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/configuring_basic_system_settings/index#managing-system-services-with-systemctl_configuring-basic-system-settings

Previous Next

Guided Exercise: Control System Services

In this exercise, you use the `systemctl` command to stop, start, restart, reload, enable, and disable the `systemd` services.

Outcomes

- Use the `systemctl` command to control the `systemd` services.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start services-control
```

Instructions

1. Log in to the `servera` machine as the `student` user and switch to the `root` user.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera ~]$ sudo -i
[sudo] password for student: student
```

```
[root@servera ~]#
```

6. Restart and reload the `sshd` service, and observe the results.
    1. Display the status of the `sshd` service. Note the process ID of the `sshd` daemon. Press **q** to exit the command.

```
[root@servera ~]# systemctl status sshd
● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2022-05-19 04:04:45 EDT; 16min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 784 (sshd)
      Tasks: 1 (limit: 10799)
```

```
    Memory: 6.6M
```

```
...output omitted...
```

11. Restart the `sshd` service and view the status. In this example, the process ID of the daemon changes from 784 to 1193. Press **q** to exit the command.

```
[root@servera ~]# systemctl restart sshd
[root@servera ~]# systemctl status sshd
● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2022-05-19 04:21:40 EDT; 5s ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 1193 (sshd)
      Tasks: 1 (limit: 10799)
     Memory: 1.7M
```

```
...output omitted...
```

22. Reload the `sshd` service and view the status. The process ID of the daemon does not change. Press **q** to exit the command.

```
[root@servera ~]# systemctl reload sshd
[root@servera ~]# systemctl status sshd
● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2022-05-19 04:21:40 EDT; 52s ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 1201 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)
   Main PID: 1193 (sshd)
      Tasks: 1 (limit: 10799)
     Memory: 1.7M
```

```
...output omitted...
```

7. Verify that the `chronyd` service is running. Press **q** to exit the command.

```
[root@servera ~]# systemctl status chronyd
● chronyd.service - NTP client/server
     Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled; vendor
preset: enabled)
     Active: active (running) since Thu 2022-05-19 04:04:44 EDT; 19min ago
```

```
  ...output omitted...
```

12. Stop the `chronyd` service and view the status. Press **q** to exit the command.

```
[root@servera ~]# systemctl stop chronyd
[root@servera ~]# systemctl status chronyd
○ chronyd.service - NTP client/server
     Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled; vendor
preset: enabled)
     Active: inactive (dead) since Thu 2022-05-19 04:24:59 EDT; 4s ago
...output omitted...
May 19 04:24:59 servera.lab.example.com systemd[1]: Stopping NTP client/server
...
May 19 04:24:59 servera.lab.example.com systemd[1]: chronyd.service: Deactivat
ed successfully.
```

```
May 19 04:24:59 servera.lab.example.com systemd[1]: Stopped NTP client/server.
```

21. Determine whether the `chronyd` service is enabled to start at system boot.

```
[root@servera ~]# systemctl is-enabled chronyd
```

```
enabled
```

23. Reboot the `servera` machine, and then view the status of the `chronyd` service.

```
[root@servera ~]# systemctl reboot
Connection to servera closed by remote host.
Connection to servera closed.
```

```
[student@workstation ~]$
```

Log in as the `student` user on the `servera` machine, and switch to the `root` user. View the status of the `chronyd` service. Press **q** to exit the command.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera ~]$ sudo -i
[sudo] password for student: student
[root@servera ~]# systemctl status chronyd
● chronyd.service - NTP client/server
     Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled; vendor
preset: enabled)
     Active: active (running) since Thu 2022-05-19 04:27:12 EDT; 40s ago
   ...output omitted...
```

27. Disable the `chronyd` service so that it does not start at boot, and then view the status of the service. Press **q** to exit the command.

```
28. [root@servera ~]# systemctl disable chronyd
29. Removed /etc/systemd/system/multi-user.target.wants/chronyd.service.
30. [root@servera ~]# systemctl status chronyd
31. ● chronyd.service - NTP client/server
32.      Loaded: loaded (/usr/lib/systemd/system/chronyd.service; disabled; vendor
    preset: enabled)
33.      Active: active (running) since Thu 2022-05-19 04:27:12 EDT; 2min 48s ago
```

```
   ...output omitted...
```

34. Reboot `servera`, and then view the status of the `chronyd` service.

```
35. [root@servera ~]# systemctl reboot
36. Connection to servera closed by remote host.
37. Connection to servera closed.
```

```
[student@workstation ~]$
```

Log in as the `student` user on `servera`, and view the status of the `chronyd` service. Press **q** to exit the command.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera ~]$ systemctl status chronyd
○ chronyd.service - NTP client/server
     Loaded: loaded (/usr/lib/systemd/system/chronyd.service; disabled; vendor preset: enabled)
     Active: inactive (dead)
       Docs: man:chronyd(8)
             man:chrony.conf(5)
```

38. Return to the `workstation` system as the `student` user.

```
39. [student@servera ~]$ exit
40. logout
41. Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish services-control
```

This concludes the section.

Previous Next

Summary

- The `systemd` utility provides a method for activating system resources, server daemons, and other processes, both at boot time and on a running system.
- Use the `systemctl` utility to start, stop, reload, enable, and disable services.
- Use the `systemd` utility to manage service units, socket units, and path units.
- Use the `systemctl status` command to determine the status of system daemons and network services that the `systemd` utility started.

- The `systemctl list-dependencies` command lists all service units that a specific service unit depends on.
- The `systemd` utility can mask a service unit so that it does not run, even to satisfy dependencies.

Previous Next

Chapter 10. Configure and Secure SSH

Abstract

Goal	Configure secure command-line service on remote systems with OpenSSH.
Objectives	• Log in to a remote system and run commands with `ssh`. • Configure a user account to use key-based authentication to log in to remote systems securely without a password. • Disable direct logins as `root` and password-based authentication for the OpenSSH service.
Sections	• Access the Remote Command Line with SSH (and Guided Exercise) • Configure SSH Key-based Authentication (and Guided Exercise) • Customize OpenSSH Service Configuration (and Guided Exercise)
Lab	Configure and Secure SSH

Access the Remote Command Line with SSH

Objectives

Log in to a remote system and run commands with `ssh`.

Describe Secure Shell

The OpenSSH package provides the Secure Shell, or SSH protocol, in Red Hat Enterprise Linux. With SSH protocol, systems can communicate in an encrypted and secure channel over an insecure network.

Use the `ssh` command to create a secure connection to a remote system, authenticate as a specific user, and obtain an interactive shell session on the remote system. The `ssh` command can run a session on a remote system without running an interactive shell.

Secure Shell Examples

The following `ssh` command logs you in on the `hosta` remote server by using the same username as the current local user. The remote system prompts you to authenticate with the `developer1` user's password in this example.

```
[developer1@host ~]$ ssh hosta
developer1@hosta's password: redhat
...output omitted...
[developer1@hosta ~]$
```

Use the `exit` command to log out of the remote system.

```
[developer1@hosta ~]$ exit
logout
Connection to hosta closed.
[developer1@host ~]$
```

The following `ssh` command logs you in on the `hosta` remote server with the `developer2` username. The remote system prompts you to authenticate with the `developer2` user's password.

```
[developer1@host ~]$ ssh developer2@hosta
developer2@hosta's password: shadowman
...output omitted...
[developer2@hosta ~]$
```

The following `ssh` command runs the `hostname` command on the `hosta` remote system as the `developer2` user without accessing the remote interactive shell.

```
[developer1@host ~]$ ssh developer2@hosta hostname
developer2@hosta's password: shadowman
hosta.lab.example.com
[developer1@host ~]$
```

This command displays the output in the local system's terminal.

Identifying Remote Users

The `w` command displays a list of users that are currently logged in to the system. It also displays the remote system location and commands that the user ran.

```
[developer1@host ~]$ ssh developer1@hosta
developer1@hosta's password: redhat
[developer1@hosta ~]$ w
 16:13:38 up 36 min,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
developer2   pts/0    172.25.250.10    16:13    7:30   0.01s  0.01s -bash
developer1   pts/1    172.25.250.10    16:24    3.00s  0.01s  0.00s w
[developer2@hosta ~]$
```

The output shows that the `developer2` user logged in to the system on the pseudo-terminal `0` at `16:13` today from the host with the `172.25.250.10` IP address, and has been idle at a shell prompt for seven minutes and thirty seconds. The output also shows that the `developer1` user logged in to the system on the pseudo-terminal `1`, and has been idle for three seconds after executing the `w` command.

SSH Host Keys

SSH secures communication through public-key encryption. When an SSH client connects to an SSH server, the server sends a copy of its public key to the client before logging in. This key helps to set up secure encryption for the communication channel and to authenticate the client's system.

When a user runs the `ssh` command for connecting to an SSH server, the command checks for a copy of the server's public key in its local known hosts file. The key might be preconfigured in the `/etc/ssh/ssh_known_hosts` file, or the user might have the `~/.ssh/known_hosts` file that contains the key in their home directory.

If the client has a copy of the key, then the `ssh` command compares the key from the known hosts server files to the key that it received. If the keys do not match, then the client assumes that the network traffic to the server is compromised, and prompts the user to confirm whether to continue with the connection.

Strict Host Key Checking

The `StrictHostKeyChecking` parameter is set in the user-specific `~/.ssh/config` file, or in the system-wide `/etc/ssh/ssh_config` file, or by specifying the `-o StrictHostKeyChecking=` option of the `ssh` command.

- If the `StrictHostKeyChecking` parameter is set to `yes`, then the `ssh` command always aborts the SSH connection if the public keys do not match.
- If the `StrictHostKeyChecking` parameter is set to `no`, then the `ssh` command enables the connection and adds the key of the target host to the `~/.ssh/known_hosts` file.

If the target host SSH key changed since the last time that you connected to it, then the `ssh` command asks for confirmation to log in and accept the new key.

If you accept the new key, then a copy of the public key is saved in the `~/.ssh/known_hosts` file to automatically confirm the server's identity on subsequent connections.

Note

Red Hat recommends to set the `StrictHostKeyChecking` parameter to `yes` in the user-specific `~/.ssh/config` file or in the system-wide `/etc/ssh/ssh_config` file, so that the `ssh` command always aborts the SSH connection if the public keys do not match.

```
[developer1@host ~]$ ssh hostb
The authenticity of host 'hostb (172.25.250.12)' can't be established.
ECDSA key fingerprint is SHA256:qaS0PToLrqlCO2XGklA0iY7CaP7aPKimerDoaUkv720.
Are you sure you want to continue connecting (yes/no)? no
[developer1@host ~]$
```

Verify the fingerprint of the target server's SSH host key by using the `ssh-keygen` command. In this example, the `ssh-keygen` command is run on the `hostb` target server.

The `ssh-keygen` command displays the key fingerprint so that you can match it to the output of the `ssh` command and verify that the key is valid. Use the `-lf` options to list the public key fingerprint in the host's default public key file.

Because you cannot connect over SSH, you must verify the target host's key fingerprint by logging in locally. Use an out-of-band communication method to share public keys, such as a phone call or video conference.

```
[developer1@hostb ~]$ ssh-keygen -lf /etc/ssh/ssh_host_ecdsa_key.pub
256 SHA256:qaS0PToLrqlCO2XGklA0iY7CaP7aPKimerDoaUkv720 root@server (ECDSA)
```

After you verify the key on the target host, you can accept the key and connect to the target host.

```
[developer1@host ~]$ ssh hostb
The authenticity of host 'hostb (172.25.250.12)' can't be established.
ECDSA key fingerprint is SHA256:qaS0PToLrqlCO2XGklA0iY7CaP7aPKimerDoaUkv720.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'hostb,172.25.250.12' (ECDSA) to the list of known hosts.
developer1@hostb's password: redhat
...output omitted...
[developer1@hostb ~]$
```

SSH Known Hosts Key Management

Information about known remote systems and their keys are stored in either of the following places:

- The system-wide `/etc/ssh/ssh_known_hosts` file.
- The `~/.ssh/known_hosts` file in each user's home directory.

The `/etc/ssh/ssh_known_hosts` file is a system-wide file that stores the public keys for hosts that the system knows. You must create and manage this file, either manually or through some automated method such as by using Ansible or a script that uses the `ssh-keyscan` utility.

A server's public key might have changed because the key was lost due to hard drive failure or it was replaced for some legitimate reason. In that case, to successfully log in to that system, the `/etc/ssh/ssh_known_hosts` file must be modified to replace the previous public key entry with the new public key.

If you connect to a remote system and the public key of that system is not in the `/etc/ssh/ssh_known_hosts` file, then the SSH client searches for the key in the `~/.ssh/known_hosts` file.

Each known host key entry consists of one line that contains three fields:

- The first field is the list of hostnames and IP addresses that share the public key.
- The second field is the encryption algorithm that is used for the key.
- The last field is the key itself.

```
[developer1@host ~]$ cat ~/.ssh/known_hosts

server1 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOmiLKMExRnsS1g7OTxMsOmgHuUSGQBUxHhuUGcv1
9uT

server1 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC8WDOooY+rh6NPa9yhLsNQXBqcQknTL/WSd3zPv
HLLd7KaC4IiEUxnwbfLBit8tRcirbQFxO20Am

...output omitted...
```

Troubleshooting Host Key Issues

If the IP address or the public key of the remote system changes, and you try to connect to that system again via SSH, then the SSH client detects that the key entry for that system in the `~/.ssh/known_hosts` file is no longer valid. A warning message states that the remote host identification changed, and that you must modify the key entry.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:hxttxb/qVi1/ycUU2wXF6mfGH++Ya7WYZv0r+tIkg4I.
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/user/.ssh/known_hosts:12
ECDSA host key for server1.example.com has changed and you have requested strict chec
king.
Host key verification failed.
```

If you do not know why the key changed, then verify the new key fingerprint, because this key might be an actual attack on your network. Use an out-of-band method for verification, such as speaking with the system administrator of the target system.

If you know why the key changed, such as an IP address change, then resolve this problem by removing the relevant key entry from the `~/.ssh/known_hosts` file, and then reconnect to the system to receive the new key entry.

The line number of the relevant key entry is specified in the warning message. You can also find and remove the relevant key entry by running the following command:

```
[developer1@host ~]$ ssh-keygen -R remotesystemname -f ~/.ssh/known_hosts
# Host remotesystemname found: line 12
/home/user/.ssh/known_hosts updated.
Original contents retained as /home/user/.ssh/known_hosts.old
```

References

`ssh`(1), `w`(1), and `hostname`(1) man pages

For more information, refer to *Using Secure Communications Between Two Systems with OpenSSH* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/securing_networks/assembly_using-secure-communications-between-two-systems-with-openssh_securing-networks

Next

Guided Exercise: Access the Remote Command Line

In this exercise, you log in to a remote system as different users and execute commands.

Outcomes

- Log in to a remote system.
- Execute commands with the OpenSSH secure shell.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start ssh-access
```

Instructions

1. From `workstation`, open an SSH session to the `servera` machine as the `student` user.

```
2. [student@workstation ~]$ ssh student@servera
```

```
[student@servera ~]$
```

3. Open an SSH session to the `serverb` machine as the `student` user. Accept the host key. Use `student` as the password when prompted for the password of the `student` user on the `serverb` machine.

```
4. [student@servera ~]$ ssh student@serverb
5. The authenticity of host 'serverb (172.25.250.11)' can't be established.
6. ED25519 key fingerprint is SHA256:h/hEJa/anxp6AP7BmB5azIPVbPNqieh0oKi4KWOTK80.
7. This key is not known by any other names
```

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'serverb' (ED25519) to the list of known hosts.
student@serverb's password: student
...output omitted...
```

```
[student@serverb ~]$
```

The `ssh` command records the host key on the `/home/student/.ssh/known_hosts` file in the `servera` machine to identify the `serverb` machine. The `student` user initiated the SSH connection from the `servera` machine. If the `/home/student/.ssh/known_hosts` file does not exist, then it is created along with the new entry in it. The `ssh` command fails to execute correctly if the remote host appears to have a different key from the recorded key.

12. Display the users that are currently logged in to the `serverb` machine. The `student` user is logged in to the system from the host with an IP address of `172.25.250.10`, which is the `servera` machine in the classroom network.

```
[student@serverb ~]$ w --from
03:39:04 up 16 min,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
```

```
student  pts/0    172.25.250.10    20:40    1.00s  0.01s  0.00s w --from
```

16. Exit the `student` user's shell on the `serverb` machine.

```
[student@serverb ~]$ exit
logout
Connection to serverb closed.
```

```
[student@servera ~]$
```

20. Open an SSH session to the `serverb` machine as the `root` user. Use `redhat` as the password of the `root` user. The command did not ask you to accept the host key, because it was found among the known hosts. If the identity of the `serverb` machine changes, then OpenSSH prompts you to challenge the new host key.

```
[student@servera ~]$ ssh root@serverb
```

```
22. root@serverb's password: redhat
23. ...output omitted...
```

```
[root@serverb ~]#
```

24. Run the `w` command to display the users that are currently logged in to the `serverb` machine. The output indicates that the `root` user is logged in to the system from the host with an IP address of `172.25.250.10`, which is the `servera` machine in the classroom network.

```
25. [root@serverb ~]# w --from
26. 03:46:05 up 23 min,  1 user,  load average: 0.00, 0.00, 0.00
27. USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
```

```
root     pts/0    172.25.250.10    20:44    1.00s  0.02s  0.00s w --from
```

28. Exit the `root` user's shell on the `serverb` machine.

```
29. [root@serverb ~]# exit
30. logout
31. Connection to serverb closed.
```

```
[student@servera ~]$
```

32. Remove the `/home/student/.ssh/known_hosts` file from the `servera` machine. This operation causes `ssh` to lose the recorded identities of the remote systems.

```
[student@servera ~]$ rm /home/student/.ssh/known_hosts
```

Host keys can change for legitimate reasons: perhaps the remote machine was replaced because of a hardware failure, or the remote machine was reinstalled. Usually, it is advisable to remove the key entry only for the particular host in the `known_hosts` file. Because this particular `known_hosts` file has only one entry, you can remove the entire file.

33. Open an SSH session to the `serverb` machine as the `student` user. If asked, accept the host key. Use `student` when prompted for the password of the `student` user on the `serverb` machine.

```
34. [student@servera ~]$ ssh student@serverb
```

```
The authenticity of host 'serverb (172.25.250.11)' can't be established.
ED25519 key fingerprint is SHA256:h/hEJa/anxp6AP7BmB5azIPVbPNqieh0oKi4KWOTK80.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'serverb' (ED25519) to the list of known hosts.
student@serverb's password: student
...output omitted...
```

```
[student@serverb ~]$
```

The `ssh` command asked for your confirmation to accept or reject the host key, because it could not find one for the remote host.

42. Exit the `student` user's shell on the `serverb` machine, and confirm that a new instance of the `known_hosts` file exists on the `servera` machine.

```
[student@serverb ~]$ exit
logout
Connection to serverb closed.
[student@servera ~]$ ls -l /home/student/.ssh/known_hosts
```

```
-rw-------. 1 student student 819 Mar 24 03:47 /home/student/.ssh/known_hosts
```

47. Confirm that the new instance of the `known_hosts` file has the host key of the `serverb` machine. The following command output is an example; your `workstation` output might differ.

```
[student@servera ~]$ cat /home/student/.ssh/known_hosts
...output omitted...
serverb ecdsa-sha2-nistp256 AAAAB3NzaC1yc2EAAAADAQ...
```

```
...output omitted...
```

51. Run the `hostname` command remotely on the `serverb` machine without accessing the interactive shell.

```
[student@servera ~]$ ssh student@serverb hostname
student@serverb's password: student
```

```
serverb.lab.example.com
```

54. Return to the `workstation` system as the `student` user.

```
55. [student@servera ~]$ exit
56. logout
```

```
Connection to servera closed.
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish ssh-access
```

This concludes the section.

Previous Next

Configure SSH Key-based Authentication

Objectives

Configure a user account to use key-based authentication to log in to remote systems securely without a password.

SSH Key-based Authentication

You can configure your account for passwordless access to SSH servers that enabled key-based authentication, which is based on public key encryption (PKI).

To prepare your account, generate a cryptographically related pair of key files. One key is private and held only by you. The second key is your related public key, which is not secret. The private key acts as your authentication credential, and it must be stored securely. The public key is copied to your account on servers that you will remotely access, and verifies your use of your private key.

When you log in to your account on a remote server, the remote server uses your public key to encrypt a challenge message and send it to your SSH client. Your SSH client must then prove that it can decrypt this message, which demonstrates that you have the associated private key. If this verification succeeds, then your request is trusted, and you are granted access without giving a password.

Passwords can be easily learned or stolen, but securely stored private keys are harder to compromise.

SSH Keys Generation

Use the `ssh-keygen` command to create a key pair. By default, the `ssh-keygen` command saves your private and public keys in the `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub` files, but you can specify a different name.

```
[user@host ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): Enter
Created directory '/home/user/.ssh'.
Enter passphrase (empty for no passphrase): Enter
Enter same passphrase again: Enter
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:vxutUNPio3QDCyvkYm1 user@host.lab.example.com
The key's randomart image is:
+---[RSA 2048]----+
|                 |
|   .     .       |
|  o o     o      |
| . = o   o .     |
|  o + = S E .    |
| ..O o + * +     |
|.+% O . + B .    |
|=*oO . . + *     |
|++.     . +.     |
+----[SHA256]-----+
```

You can choose to provide a passphrase to `ssh-keygen`, which is used to encrypt your private key. Using a passphrase is recommended, so that your private key cannot be used by someone to access it. If you set a passphrase, then you must enter the passphrase each time that you use the private key. The passphrase is used locally to decrypt your private key before use, unlike your password, which must be sent in clear text across the network for use.

You can use the `ssh-agent` key manager locally, which caches your passphrase on first use in a login session, and then provides the passphrase for all subsequent private key use in the same login session. The `ssh-agent` command is discussed later in this section.

In the following example, a passphrase-protected private key is created with the public key.

```
[user@host ~]$ ssh-keygen -f .ssh/key-with-pass
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): your_passphrase
Enter same passphrase again: your_passphrase
Your identification has been saved in .ssh/key-with-pass.
Your public key has been saved in .ssh/key-with-pass.pub.
The key fingerprint is:
SHA256:w3GGB7EyHUry4aOcNPKmhNKS7dl1YsMVLvFZJ77VxAo user@host.lab.example.com
The key's randomart image is:
+---[RSA 2048]----+
|    . + =.o ...  |
|     = B XEo o.  |
|  . o O X =....  |
| = = = B = o.    |
|= + * * S .      |
|.+ = o + .       |
|  + .            |
|                 |
|                 |
+----[SHA256]-----+
```

The `ssh-keygen` command `-f` option specifies the files to save the keys in. In the preceding example, the `ssh-keygen` command saved the key pair in the `/home/user/.ssh/key-with-pass` and `/home/user/.ssh/key-with-pass.pub` files.

Warning

During new `ssh-keygen` command use, if you specify the name of an existing pair of key files, including the default `id_rsa` pair, you overwrite that existing key pair, which can be restored only if you have a backup for those files. Overwriting a key pair loses the original private key that is required to access accounts that you configured with the corresponding public key on remote servers.

If you cannot restore your local private key, then you lose access to remote servers until you distribute your new public key to replace the previous public key on each server. Always create backups of your keys, if they are overwritten or lost.

Generated SSH keys are stored by default in the `.ssh` subdirectory of your home directory. To function correctly, the private key must be readable and writable only by the user that it belongs to (octal permissions 600). The public key is not secure, and anyone on the system might also be able to read it (octal permissions 644).

Share the Public Key

To configure your remote account for access, copy your public key to the remote system. The `ssh-copy-id` command copies the public key of the SSH key pair to the remote system. You can specify a specific public key with the `ssh-copy-id` command, or use the default `~/.ssh/id_rsa.pub` file.

```
[user@host ~]$ ssh-copy-id -i .ssh/key-with-pass.pub user@remotehost

/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/user/.ssh/id_rsa
.pub"

/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out a
ny that are already installed

/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted no
w it is to install the new keys

user@remotehost's password: redhat

Number of key(s) added: 1
```

```
Now try logging into the machine, with:   "ssh 'user@remotehost'"
and check to make sure that only the key(s) you wanted were added.
```

After you place the public key, test the remote access, with the corresponding private key. If the configuration is correct, you access your account on the remote system without being asked for your account password. If you do not specify a private key file, then the `ssh` command uses the default `~/.ssh/id_rsa` file if it exists.

Important

If you configured a passphrase to protect your private key, then SSH requests the passphrase on first use. However, if the key authentication succeeds, then you are not asked for your account password.

```
[user@host ~]$ ssh -i .ssh/key-with-pass user@remotehost
Enter passphrase for key '.ssh/key-with-pass': your_passphrase
...output omitted...
[user@remotehost ~]$
```

Non-interactive Authentication with the Key Manager

If you encrypt your private key with a passphrase, then you must enter the passphrase each time that you use the private key for authentication. However, you can configure the `ssh-agent` key manager to cache passphrases. Then, each time you use SSH, the `ssh-agent` key manager provides the passphrase for you. Using a key manager is convenient and can improve security by providing fewer opportunities for other people to observe your passphrase.

The `ssh-agent` key manager can be configured to start automatically when you log in. The GNOME graphical desktop environment can automatically start and configure the `ssh-agent` key manager. If you log in to a text environment, then you must start the `ssh-agent` program manually for each session. Start the `ssh-agent` program with the following command:

```
[user@host ~]$ eval $(ssh-agent)
Agent pid 10155
```

When you manually start the `ssh-agent` command, it runs additional shell commands to set environment variables that are needed for use with the `ssh-`

`add` command. You can manually load your private key passphrase to the key manager by using the `ssh-add` command.

The following example `ssh-add` commands add the private keys from the default `~/.ssh/id_rsa` file and then from a `~/.ssh/key-with-pass` file:

```
[user@host ~]$ ssh-add
Identity added: /home/user/.ssh/id_rsa (user@host.lab.example.com)
[user@host ~]$ ssh-add .ssh/key-with-pass
Enter passphrase for .ssh/key-with-pass: your_passphrase
Identity added: .ssh/key-with-pass (user@host.lab.example.com)
```

The following `ssh` command uses the default private key file to access your account on a remote SSH server:

```
[user@host ~]$ ssh user@remotehost
Last login: Mon Mar 14 06:51:36 2022 from host.example.com
[user@remotehost ~]$
```

The following `ssh` command uses the `~/.ssh/key-with-pass` private key to access your account on the remote server. The private key in this example was previously decrypted and added to the `ssh-agent` key manager; therefore the `ssh` command does not prompt you for the passphrase to decrypt the private key.

```
[user@host ~]$ ssh -i .ssh/key-with-pass user@remotehost
Last login: Mon Mar 14 06:58:43 2022 from host.example.com
[user@remotehost ~]$
```

When you log out of a session that used an `ssh-agent` key manager, all cached passphrases are cleared from memory.

Basic SSH Connection Troubleshooting

SSH can appear complex when remote access with key pair authentication is not succeeding. The `ssh` command provides three verbosity levels with the `-v`, `-vv`, and `-vvv` options, which respectively provide increasing debugging information during `ssh` command use.

The next example demonstrates the information that is provided when using the lowest verbosity option:

```
[user@host ~]$ ssh -v user@remotehost

OpenSSH_8.7p1, OpenSSL 3.0.1 14 Dec 2021

debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Reading configuration data /etc/ssh/ssh_config.d/01-training.conf
debug1: /etc/ssh/ssh_config.d/01-training.conf line 1: Applying options for *
debug1: Reading configuration data /etc/ssh/ssh_config.d/50-redhat.conf
...output omitted...

debug1: Connecting to remotehost [192.168.1.10] port 22.
debug1: Connection established.
...output omitted...

debug1: Authenticating to remotehost:22 as 'user'
...output omitted...
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,pas
sword
...output omitted...

debug1: Next authentication method: publickey
debug1: Offering public key: /home/user/.ssh/id_rsa RSA SHA256:hDVJjD7xrUjXGZVRJQixxF
V6NF/ssMjS6AuQ1+VqUc4
debug1: Server accepts key: /home/user/.ssh/id_rsa RSA SHA256:hDVJjD7xrUjXGZVRJQixxFV
6NF/ssMjS6AuQ1+VqUc4
Authenticated to remotehost ([192.168.1.10]:22) using "publickey".
...output omitted...
[user@remotehost ~]$
```

OpenSSH and OpenSSL versions.

OpenSSH configuration files.

Connection to the remote host.

Trying to authenticate the user on the remote host.

Authentication methods that the remote host allows.

Trying to authenticate the user by using the SSH key.

Using the `/home/user/.ssh/id_rsa` key file to authenticate.

The remote hosts accepts the SSH key.

If an attempted authentication method fails, then a remote SSH server falls back to other allowed authentication methods, until all the available methods are tried. The next example demonstrates a remote access with an SSH key that fails, but then the SSH server offers password authentication that succeeds.

```
[user@host ~]$ ssh -v user@remotehost
...output omitted...
debug1: Next authentication method: publickey
debug1: Offering public key: /home/user/.ssh/id_rsa RSA SHA256:bsB6l5R184zvxNlrcRMmYd
32oBkU1LgQj09dUBZ+Z/k
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,pas
sword
...output omitted...
debug1: Next authentication method: password
user@remotehost's password: password
Authenticated to remotehost ([172.25.250.10]:22) using "password".
...output omitted...
[user@remotehost ~]$
```

SSH Client Configuration

You can create the `~/.ssh/config` file to preconfigure SSH connections. Within the configuration file, you can specify connection parameters such as users, keys, and ports for specific hosts. This file eliminates the need to manually specify command parameters each time that you connect to a host. Consider the following `~/.ssh/config` file, which preconfigures two host connections with different users and keys:

```
[user@host ~]$ cat ~/.ssh/config
host servera
     HostName                      servera.example.com
     User                          usera
     IdentityFile                  ~/.ssh/id_rsa_servera

host serverb
     HostName                      serverb.example.com
     User                          userb
     IdentityFile                  ~/.ssh/id_rsa_serverb
```

References

`ssh-keygen`(1), `ssh-copy-id`(1), `ssh-agent`(1), and `ssh-add`(1) man pages

Previous Next

Guided Exercise: Configure SSH Key-based Authentication

In this exercise, you configure a user to use key-based authentication for SSH.

Outcomes

- Generate an SSH key pair without passphrase protection.
- Generate an SSH key pair with passphrase protection.
- Authenticate with both passphrase-less and passphrase-protected SSH keys.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start ssh-configure
```

Instructions

1. Log in to the `serverb` machine as the `student` user.

```
[student@workstation ~]$ ssh student@serverb
...output omitted...
```

```
[student@serverb ~]$
```

4. Switch to the `operator1` user on the `serverb` machine. Use `redhat` as the password.

```
[student@serverb ~]$ su - operator1
Password: redhat
```

```
[operator1@serverb ~]$
```

7. Generate a set of SSH keys. Do not enter a passphrase.

```
[operator1@serverb ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/operator1/.ssh/id_rsa): Enter
Created directory '/home/operator1/.ssh'.
Enter passphrase (empty for no passphrase): Enter
Enter same passphrase again: Enter
Your identification has been saved in /home/operator1/.ssh/id_rsa.
Your public key has been saved in /home/operator1/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:JainiQdnRosC+xXh operator1@serverb.lab.example.com
The key's randomart image is:
+---[RSA 3072]----+
|E+*+ooo .        |
|.= o.o o .       |
|o.. = . . o      |
|+. + * . o .     |
|+ = X . S +      |
| + @ +   = .     |
|. + =   o        |
|.o . . . .       |
|o     o..        |
```

```
+----[SHA256]-----+
```

29. Send the public key of the SSH key pair to the `operator1` user on the `servera` machine, with `redhat` as the password.

```
[operator1@serverb ~]$ ssh-copy-id operator1@servera
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/operator1
/.ssh/id_rsa.pub"
The authenticity of host 'servera (172.25.250.10)' can't be established.
ED25519 key fingerprint is SHA256:h/hEJa/anxp6AP7BmB5azIPVbPNqieh0oKi4KWOTK80.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filte
r out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prom
pted now it is to install the new keys
operator1@servera's password: redhat

Number of key(s) added: 1

Now try logging in to the machine, with:   "ssh 'operator1@servera'"
```

```
and check to make sure that only the key(s) you wanted were added.
```

43. Execute the `hostname` command on the `servera` machine remotely by using the `ssh` command without accessing the remote interactive shell.

```
[operator1@serverb ~]$ ssh operator1@servera hostname
```

```
servera.lab.example.com
```

The preceding `ssh` command does not prompt you for a password, because it uses the passphrase-less private key against the exported public key to authenticate as the `operator1` user on the `servera` machine.

This approach is not secure, because anyone who has access to the private key file can log in to the `servera` machine as the `operator1` user.

In a following step in this exercise, you make your private key more secure by encrypting it and protecting access to it by adding a passphrase.

45. Generate another set of SSH keys with the default name and without a passphrase, and overwrite the previously generated SSH key files. Try to connect to the `servera` machine by using the new SSH keys.
    The `ssh` command asks for a password, because it cannot authenticate with the SSH key. Run again the `ssh` command with the `-v` (verbose) option to verify it.

    Send the new public key of the SSH key pair to the `operator1` user on the `servera` machine, to replace the previous public key. Use `redhat` as the password for the `operator1` user on the `servera` machine. Execute the `hostname` command on the `servera` machine remotely by using the `ssh` command without accessing the remote interactive shell, to verify that it works again.

    1. Again, generate another set of SSH keys with the default name and without a passphrase, and overwrite the previously generated SSH key files.

    ```
    [operator1@serverb ~]$ ssh-keygen
    Generating public/private rsa key pair.
    Enter file in which to save the key (/home/operator1/.ssh/id_rsa): Enter
    /home/operator1/.ssh/id_rsa already exists.
    Overwrite (y/n)? y
    Enter passphrase (empty for no passphrase): Enter
    Enter same passphrase again: Enter
    Your identification has been saved in /home/operator1/.ssh/id_rsa
    Your public key has been saved in /home/operator1/.ssh/id_rsa.pub
    ```

    ```
    ...output omitted...
    ```

    11. Try to connect to the `servera` machine by using the new SSH keys. The `ssh` command asks for a password, because it cannot authenticate with the SSH key. Press **Ctrl**+**c** to exit from the `ssh` command when it prompts for a password. Run again the `ssh` command with the `-v` (verbose) option to verify it. Press again **Ctrl**+**c** to exit from the `ssh` command when it prompts for a password.

```
[operator1@serverb ~]$ ssh operator1@servera hostname
operator1@servera's password: ^C
[operator1@serverb ~]$ ssh -v operator1@servera hostname
OpenSSH_8.7p1, OpenSSL 3.0.1 14 Dec 2021
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Reading configuration data /etc/ssh/ssh_config.d/01-training.conf
...output omitted...
debug1: Next authentication method: publickey
debug1: Offering public key: /home/operator1/.ssh/id_rsa RSA SHA256:ad597Zf64xckV26xht8bjQbzqSPuOXQPXksGEWVsP80
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,password
debug1: Trying private key: /home/operator1/.ssh/id_dsa
debug1: Trying private key: /home/operator1/.ssh/id_ecdsa
debug1: Trying private key: /home/operator1/.ssh/id_ecdsa_sk
debug1: Trying private key: /home/operator1/.ssh/id_ed25519
debug1: Trying private key: /home/operator1/.ssh/id_ed25519_sk
debug1: Trying private key: /home/operator1/.ssh/id_xmss
debug1: Next authentication method: password
```

```
operator1@servera's password: ^C
```

29. Send the new public key of the SSH key pair to the `operator1` user on the `servera` machine, to replace the previous public key. Use `redhat` as the password for the `operator1` user on the `servera` machine. Execute the `hostname` command on the `servera` machine remotely by using the `ssh` command without accessing the remote interactive shell, to verify that it works again.

```
[operator1@serverb ~]$ ssh-copy-id operator1@servera
...output omitted...
operator1@servera's password: redhat

Number of key(s) added: 1

Now try logging in to the machine, with:   "ssh 'operator1@servera'"
```

```
and check to make sure that only the key(s) you wanted were added.
[operator1@serverb ~]$ ssh operator1@servera hostname
```

```
servera.lab.example.com
```

46. Generate another set of SSH keys with passphrase-protection. Save the key as `/home/operator1/.ssh/key2`. Use `redhatpass` as the passphrase of the private key.

```
[operator1@serverb ~]$ ssh-keygen -f .ssh/key2
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): redhatpass
Enter same passphrase again: redhatpass
Your identification has been saved in .ssh/key2.
Your public key has been saved in .ssh/key2.pub.
The key fingerprint is:
SHA256:OCtCjfPm5QrbPBgqb operator1@serverb.lab.example.com
The key's randomart image is:
+---[RSA 3072]----+
|O=X*             |
|OB=.             |
|E*o.             |
|Booo   .         |
|..= . o S        |
| +.o   o         |
|+.oo+ o          |
|+o.O.+           |
|+ . =o.          |
```

```
+----[SHA256]-----+
```

66. Send the public key of the passphrase-protected key pair to the `operator1` user on the `servera` machine. The command does not prompt you for a password, because it uses the public key of the passphrase-less private key that you exported to the `servera` machine in the preceding step.

```
[operator1@serverb ~]$ ssh-copy-id -i .ssh/key2.pub operator1@servera
```

```
68./usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ".ssh/key2.pub"

69./usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filte
   r out any that are already installed

70./usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prom
   pted now it is to install the new keys

71.

72.Number of key(s) added: 1

73.

74.Now try logging in to the machine, with:   "ssh 'operator1@servera'"
```

```
and check to make sure that only the key(s) you wanted were added.
```

75. Execute the `hostname` command on the `servera` machine remotely by using the `ssh` command. Use the `/home/operator1/.ssh/key2` key as the identity file. Specify `redhatpass` as the passphrase, which you set for the private key in the preceding step.

    The command prompts you for the passphrase that you used to protect the private key of the SSH key pair. If an attacker gains access to the private key, then the attacker cannot use it to access other systems, because a passphrase protects the private key itself. The `ssh` command uses a different passphrase from the `operator1` user on the `servera` machine, and so users must know both passphrases.

```
[operator1@serverb ~]$ ssh -i .ssh/key2 operator1@servera hostname

Enter passphrase for key '.ssh/key2': redhatpass

servera.lab.example.com
```

    Use the `ssh-agent` program, as in the following step, to avoid interactively typing the passphrase when logging in with SSH. Using the `ssh-agent` program is both more convenient and more secure when the administrators log in to remote systems regularly.

76. Run the `ssh-agent` program in your `Bash` shell, and add the passphrase-protected private key (`/home/operator1/.ssh/key2`) of the SSH key pair to the shell session.

The command starts the `ssh-agent` program and configures the shell session to use it. Then, you use the `ssh-add` command to provide the unlocked private key to the `ssh-agent` program.

```
[operator1@serverb ~]$ eval $(ssh-agent)
Agent pid 1729
[operator1@serverb ~]$ ssh-add .ssh/key2
Enter passphrase for .ssh/key2: redhatpass
Identity added: .ssh/key2 (operator1@serverb.lab.example.com)
```

77. Execute the `hostname` command on the `servera` machine remotely without accessing a remote interactive shell. Use the `/home/operator1/.ssh/key2` key as the identity file.

    The command does not prompt you to enter the passphrase interactively.

```
[operator1@serverb ~]$ ssh -i .ssh/key2 operator1@servera hostname
servera.lab.example.com
```

78. Open another terminal on the `workstation` machine and log in to the `serverb` machine as the `student` user.

```
79. [student@workstation ~]$ ssh student@serverb
80. ...output omitted...
```

```
[student@serverb ~]$
```

81. On the `serverb` machine, switch to the `operator1` user and remotely log in to the `servera` machine. Use the `/home/operator1/.ssh/key2` key as the identity file to authenticate by using the SSH keys.
    1. Use the `su` command to switch to the `operator1` user. Use `redhat` as the password for the `operator1` user.

    ```
    2. [student@serverb ~]$ su - operator1
    3. Password: redhat
    ```

    ```
    [operator1@serverb ~]$
    ```

    4. Log in to the `servera` machine as the `operator1` user.

The command prompts you to enter the passphrase interactively, because you do not invoke the SSH connection from the same shell where you started the `ssh-agent` program.

```
[operator1@serverb ~]$ ssh -i .ssh/key2 operator1@servera
Enter passphrase for key '.ssh/key2': redhatpass
...output omitted...
[operator1@servera ~]$
```

82. Exit and close all extra terminals, and return to the `workstation` machine.
    1. Exit and close extra terminal windows. The `exit` commands leave the `operator1` user's shell; terminate the shell session where `ssh-agent` is active; and return to the `student` user's shell on the `serverb` machine.

```
2. [operator1@servera ~]$ exit
3. logout
4. Connection to servera closed.
```

```
[operator1@serverb ~]$
```

5. Return to the `workstation` system as the `student` user.

```
6. [operator1@serverb ~]$ exit
7. logout
8. [student@serverb ~]$ exit
9. logout
10. Connection to serverb closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish ssh-configure
```

This concludes the section.

Previous Next

Customize OpenSSH Service Configuration

Objectives

Disable direct logins as `root` and password-based authentication for the OpenSSH service.

Configure the OpenSSH Server

The `sshd` daemon provides the OpenSSH service. You can configure the service by editing the `/etc/ssh/sshd_config` file.

The default configuration of the OpenSSH server works well for many use cases. However, you might want to make some changes to strengthen the security of your system. You might want to prohibit direct remote login to the `root` account, and you might want to prohibit password-based authentication (in favor of SSH private key authentication).

Prohibit the Superuser from Logging In

It is a good practice to prohibit direct login to the `root` user account from remote systems. Some risks of allowing direct login as the `root` user include the following cases:

- The `root` username exists on every Linux system by default, so a potential attacker needs only to guess the password, instead of a valid username and password combination. This scenario reduces complexity for an attacker.
- The `root` user has unrestricted privileges, so its compromise can lead to maximum damage to the system.
- From an auditing perspective, it can be hard to track which authorized user logged in as the `root` user and made changes. If users must log in as a regular user and switch to the `root` account, then you can view a log event for accountability.

Important

Starting in Red Hat Enterprise Linux 9, the `PermitRootLogin` parameter is set to the `prohibit-password` value by default. This value enforces the use of key-based authentication instead of passwords for logging in as the `root` user, and reduces the risk of brute-force attacks.

The OpenSSH server uses the `PermitRootLogin` configuration setting in the `/etc/ssh/sshd_config` file to allow or prohibit users to log in to the system as the `root` user, as in the following example:

```
PermitRootLogin yes
```

If the `PermitRootLogin` parameter is set to the `yes` value, then anyone can log in as the `root` user remotely. To prevent this situation, set the value to `no`. Alternatively, to prevent password-based authentication but to allow private key-based authentication for `root`, set the `PermitRootLogin` parameter to `without-password`.

The SSH server (`sshd`) must be reloaded to apply any changes.

```
[root@host ~]# systemctl reload sshd
```

Prohibit Password-based Authentication for SSH

Allowing only private key-based logins to the remote command line has advantages:

- Attackers cannot use password-guessing attacks to remotely break into known accounts on the system.
- With passphrase-protected private keys, an attacker needs both the passphrase and a copy of the private key. With passwords, an attacker needs only the password.
- By using passphrase-protected private keys with `ssh-agent`, the passphrase is entered and exposed less often, and logging in is more convenient for the user.

The OpenSSH server uses the `PasswordAuthentication` parameter in the `/etc/ssh/sshd_config` file to control whether users can use password-based authentication to log in to the system.

```
PasswordAuthentication yes
```

With the default value of `yes` for the `PasswordAuthentication` parameter in the `/etc/ssh/sshd_config` file, the SSH server allows users to use password-based authentication when logging in. The value of `no` for `PasswordAuthentication` prevents users from using password-based authentication.

Whenever you change the `/etc/ssh/sshd_config` file, you must reload the `sshd` service to apply the changes.

Important

If you turn off password-based authentication for `ssh`, then you must ensure that the user's `~/.ssh/authorized_keys` file on the remote server is populated with their public key, so that they can log in.

References

`ssh`(1), `sshd_config`(5) man pages

Previous Next

Guided Exercise: Customize OpenSSH Service Configuration

In this exercise, you disable direct logins as `root` and disable password-based authentication for the OpenSSH service on one of your servers.

Outcomes

- Disable direct logins as `root` over `ssh`.
- Disable password-based authentication for remote users to connect over SSH.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that all required resources are available.

```
[student@workstation ~]$ lab start ssh-customize
```

Instructions

1. From `workstation`, open an SSH session to the `serverb` machine as the `student` user.

```
[student@workstation ~]$ ssh student@serverb
```

```
[student@serverb ~]$
```

3. Use the `su` command to switch to the `operator2` user on the `serverb` machine. Use `redhat` as the password for the `operator2` user.

```
[student@serverb ~]$ su - operator2
Password: redhat
```

```
[operator2@serverb ~]$
```

6. Use the `ssh-keygen` command to generate SSH keys. Do not enter any passphrase for the keys.

```
[operator2@serverb ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/operator2/.ssh/id_rsa): Enter
Created directory '/home/operator2/.ssh'.
Enter passphrase (empty for no passphrase): Enter
Enter same passphrase again: Enter
Your identification has been saved in /home/operator2/.ssh/id_rsa.
Your public key has been saved in /home/operator2/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:LN5x1irX0OWxgyd/qhATNgZWOtLUj16EZkM1JHkCR+I operator2@serverb.lab.examp
le.com
The key's randomart image is:
+---[RSA 3072]----+
|        *=+    |
|       = =O.o.   |
|      . Eo=B  o  |
|       o +.=o+ o |
|      . S..= =  |
|     . o +. + . |
|      . o + . . .|
|         o .   o |
|            ...  |
```

```
+----[SHA256]-----+
```

28. Use the `ssh-copy-id` command to send the public key of the SSH key pair to the `operator2` user on the `servera` machine. Use `redhat` as the password for the `operator2` user on `servera`.

```
[operator2@serverb ~]$ ssh-copy-id operator2@servera
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/operator2
/.ssh/id_rsa.pub"
The authenticity of host 'servera (172.25.250.10)' can't be established.
ED25519 key fingerprint is SHA256:h/hEJa/anxp6AP7BmB5azIPVbPNqieh0oKi4KWOTK80.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filte
r out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prom
pted now it is to install the new keys
operator2@servera's password: redhat
Number of key(s) added: 1
Now try logging into the machine, with:   "ssh 'operator2@servera'"
```

```
and check to make sure that only the key(s) you wanted were added.
```

39. Confirm that you can successfully log in to the `servera` machine as the `operator2` user with the SSH keys.
    1. Open an SSH session to the `servera` machine as the `operator2` user.

```
[operator2@serverb ~]$ ssh operator2@servera
...output omitted...
```

```
[operator2@servera ~]$
```

The preceding `ssh` command used SSH keys for authentication.

4. Log out of the `servera` machine.

```
[operator2@servera ~]$ exit
logout
```

```
Connection to servera closed.
```

40. Confirm that you can successfully log in to the `servera` machine as the `root` user with `redhat` as the password.
    1. Open an SSH session to the `servera` machine as the `root` user with `redhat` as the password.

```
[operator2@serverb ~]$ ssh root@servera
root@servera's password: redhat
...output omitted...
```

```
[root@servera ~]#
```

The preceding `ssh` command used the password of the superuser for authentication, because SSH keys do not exist for the superuser.

5. Log out of the `servera` machine.

```
[root@servera ~]# exit
logout
Connection to servera closed.
```

```
[operator2@serverb ~]$
```

41. Confirm that you can successfully log in to the `servera` machine as the `operator3` user with `redhat` as the password.
    1. Open an SSH session to the `servera` machine as the `operator3` user with `redhat` as the password.

```
[operator2@serverb ~]$ ssh operator3@servera
operator3@servera's password: redhat
...output omitted...
```

```
[operator3@servera ~]$
```

The preceding `ssh` command used the password of the `operator3` user for authentication, because SSH keys do not exist for the `operator3` user.

5. Log out of the `servera` machine.

```
[operator3@servera ~]$ exit
logout
Connection to servera closed.
```

```
[operator2@serverb ~]$
```

42. Configure the `sshd` service on the `servera` machine to prevent users from logging in as the `root` user. Use `redhat` as the password of the superuser when required.

1. Open an SSH session to the `servera` machine as the `operator2` user with the SSH keys.

```
[operator2@serverb ~]$ ssh operator2@servera
...output omitted...
```

```
[operator2@servera ~]$
```

4. On the `servera` machine, switch to the `root` user. Use `redhat` as the password for the `root` user.

```
[operator2@servera ~]$ su -
Password: redhat
```

```
[root@servera ~]#
```

7. Set `PermitRootLogin` to `no` in the `/etc/ssh/sshd_config` file and reload the `sshd` service. You can use the `vim /etc/ssh/sshd_config` command to edit the configuration file of the `sshd` service.

```
...output omitted...
PermitRootLogin no
...output omitted...
```

```
[root@servera ~]# systemctl reload sshd
```

11. Open another terminal on `workstation`, and open an SSH session to the `serverb` machine as the `operator2` user. From the `serverb` machine, try to log in to the `servera` machine as the `root` user. This command should fail, because you disabled the `root` user login over SSH in the preceding step.

    Note

    For your convenience, password-less login is already configured between `workstation` and `serverb` in the classroom environment.

```
[student@workstation ~]$ ssh operator2@serverb
...output omitted...
[operator2@serverb ~]$ ssh root@servera
root@servera's password: redhat
Permission denied, please try again.
root@servera's password: redhat
Permission denied, please try again.
root@servera's password: redhat
root@servera: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,
password).
```

    By default, the `ssh` command attempts to authenticate with key-based authentication first, and if that method fails, then with password-based authentication.

43. Configure the `sshd` service on the `servera` machine to allow users to authenticate with SSH keys only, rather than with their passwords.
    1. Return to the first terminal with the `root` user's active shell on the `servera` machine. Set the `PasswordAuthentication` parameter to `no` in the `/etc/ssh/sshd_config` file and reload the `sshd` service. You can use the `vim /etc/ssh/sshd_config` command to edit the configuration file of the `sshd` service.

```
2. ...output omitted...
3. PasswordAuthentication no
4. ...output omitted...
```

```
[root@servera ~]# systemctl reload sshd
```

5. Go to the second terminal with the `operator2` user's active shell on the `serverb` machine, and try to log in to the `servera` machine as the `operator3` user. This command should fail, because SSH keys are not configured for the `operator3` user, and the `sshd` service on the `servera` machine does not allow the use of passwords for authentication.

```
6. [operator2@serverb ~]$ ssh operator3@servera
```

```
operator3@servera: Permission denied (publickey,gssapi-keyex,gssapi-with
-mic).
```

Note

For more granularity, you can use the explicit `-o PubkeyAuthentication=no` and `-o PasswordAuthentication=yes` options with the `ssh` command. You can then override the `ssh` command's defaults and confidently determine that the preceding command fails based on the settings that you adjusted in the `/etc/ssh/sshd_config` file in the preceding step.

7. Return to the first terminal with the `root` user's active shell on the `servera` machine. Verify that `PubkeyAuthentication` is enabled in the `/etc/ssh/sshd_config` file. You can use the `vim /etc/ssh/sshd_config` command to view the configuration file of the `sshd` service.

```
8. ...output omitted...
9. #PubkeyAuthentication yes
```

```
...output omitted...
```

The `PubkeyAuthentication` line is commented. Commented lines indicate the default values of a parameter. The public key authentication of SSH is active by default, as the commented line indicates.

10. Return to the second terminal with the `operator2` user's active shell on the `serverb` machine, and try to log in to the `servera` machine as

the `operator2` user. This command should succeed, because the SSH keys are configured for the `operator2` user to log in to the `servera` machine from the `serverb` machine.

```
11. [operator2@serverb ~]$ ssh operator2@servera
12. ...output omitted...
```

```
[operator2@servera ~]$
```

13. From the second terminal, exit the `operator2` user's shell on both the `servera` and `serverb` machines.

```
14. [operator2@servera ~]$ exit
15. logout
16. Connection to servera closed.
17. [operator2@serverb ~]$ exit
18. logout
19. Connection to serverb closed.
```

```
[student@workstation ~]$
```

20. Close the second terminal on the `workstation` machine.

```
[student@workstation ~]$ exit
```

21. From the first terminal, exit the `root` user's shell on the `servera` machine.

```
22. [root@servera ~]# exit
```

```
logout
```

23. From the first terminal, exit the `operator2` user's shell on both the `servera` and `serverb` machines.

```
24. [operator2@servera ~]$ exit
25. logout
26. Connection to servera closed.
27. [operator2@serverb ~]$ exit
28. logout
```

```
[student@serverb ~]$
```

29. Log out of `serverb`, and return to the `student` user's shell on `workstation`.

```
30. [student@serverb ~]$ exit
31. logout
32. Connection to serverb closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish ssh-customize
```

This concludes the section.

Previous Next

Summary

- With the `ssh` command, users can access remote systems securely with the SSH protocol.
- A client system stores the identities of remote servers in the `~/.ssh/known_hosts` and `/etc/ssh/ssh_known_hosts` files.
- SSH supports both password-based and key-based authentication.
- The `ssh-keygen` command generates an SSH key pair for authentication. The `ssh-copy-id` command exports the public key to remote systems.
- The `sshd` service implements the SSH protocol on Red Hat Enterprise Linux systems.
- Configure advanced SSH settings in the `/etc/ssh/sshd_config` configuration file.
- It is a recommended practice to configure `sshd` to disable remote logins as `root` and to require public key authentication rather than password-based authentication.

Chapter 11. Manage Networking

Abstract

Goal	Configure network interfaces and settings on Red Hat Enterprise Linux servers.
Objectives	• Fundamental concepts of network addressing and routing for a server. • Test and inspect the current network configuration with command-line utilities. • Manage network settings and devices with the `nmcli` command. • Modify network configuration by editing configuration files. • Configure a server's static hostname and its name resolution and test the results.
Sections	• Describe Networking Concepts (and Quiz) • Validate Network Configuration (and Guided Exercise) • Configure Networking from the Command Line (and Guided Exercise) • Edit Network Configuration Files (and Guided Exercise) • Configure Hostnames and Name Resolution (and Guided Exercise)

Lab	Manage Networking

Describe Networking Concepts

Objectives

Describe fundamental concepts of network addressing and routing for a server.

TCP/IP Network Model

The TCP/IP network model is a four-layered set of communication protocols that describes how data communications are packetized, addressed, transmitted, routed, and received between computers over a network.

The protocol is specified by RFC 1122, *Requirements for Internet Hosts - Communication Layers*.

Following are the four layers of the TCP/IP network model:

- Application

  Each application has specifications for communication so that clients and servers can communicate across platforms. Common protocols include SSH, HTTPS (secure web), FTP (file sharing), and SMTP (electronic mail delivery).

- Transport

  TCP and UDP are transport protocols. TCP is a reliable connection-oriented communication, whereas UDP is a connectionless datagram protocol. Application protocols can use either TCP or UDP ports. A list of well-known and registered ports is in the `/etc/services` file.

  When a packet is sent on the network, the combination of the service port and IP address forms a socket. Each packet has a source socket and a destination socket. This information can be used when monitoring and filtering network traffic.

- Internet

The internet layer, or network layer, carries data from the source host to the destination host. The IPv4 and IPv6 protocols are internet layer protocols. Each host has an IP address and a prefix to determine network addresses. Routers are used to connect networks.

- Link

  The link layer, or media access layer, provides the connection to physical media. The most common types of networks are wired Ethernet (802.3) and wireless Wi-Fi (802.11). Each physical device has a *Media Access Control (MAC)* address, which is also known as a hardware address, to identify the destination of packets on the local network segment.

Figure 11.1: Comparison of the TCP/IP and OSI network models

Describe Network Interface Names

Each network port on a system has a name, which you use to configure and identify it.

Earlier versions of Red Hat Enterprise Linux used names such as `eth0`, `eth1`, and `eth2` for each network interface. The `eth0` interface name was the first network port that the operating system detected; `eth1` was the second interface, and so on. However, as devices were added and removed, the mechanism that detected and named devices could change which interface was assigned to which name. Furthermore, the PCIe standard does not guarantee the order in which PCIe devices are detected on boot, which could change device naming unexpectedly due to variations during device or system startup.

In Red Hat Enterprise Linux 7 and later, the default naming system generates names that are consistent across reboots. Instead of being based on the detection order, the assignment of network interface names is based on information from the firmware, the PCI bus topology, and the type of network device.

Network interface names start with the type of interface:

- Ethernet interfaces begin with *en*
- WLAN interfaces begin with *wl*
- WWAN interfaces begin with *ww*

The rest of the interface name after the type is based on information from the server's firmware, or is determined by the location of the device in the PCI topology.

- `o`*`N`* indicates an on-board device with unique index *N* from the server's firmware. The `eno1` name is on-board Ethernet device 1.
- `s`*`N`* indicates a device in PCI hotplug slot *N*. For example, `ens3` is an Ethernet card in PCI hotplug slot 3.
- `p`*`M`*`s`*`N`* indicates a PCI device on bus *M* in slot *N*. A `wlp4s0` interface is a WLAN card on PCI bus 4 in slot 0. If the card is a multi-function device (such as an Ethernet card with multiple ports, or a device with both Ethernet and another function), then you might see `f`*`N`* in the device name. An `enp0s1f0` interface is function 0 of the Ethernet card on bus 0 in slot 1. A second card interface would be named `enp0s1f1`, which is function 1 of that same device.

Persistent naming means that when the name is set for a network interface on the system, the name of the interface does not change, even if you add or remove hardware. A behavior of persistent naming is that a system with a single interface generates a device name by using a hardware information scheme, and is not expected to use the *eth0* kernel naming scheme.

IPv4 Networks

Although IPv4 remains the most common addressing scheme in enterprise networks today, IPv6 has surpassed IPv4 usage in cellular networks. You need a basic understanding of IPv4 networking to manage networking on your servers.

IPv4 Addresses

An *IPv4 address* is a 32-bit number, which is expressed as four 8-bit octets in a decimal format that ranges in value from 0 to 255, separated by single dots. The address is divided into two parts: the network prefix and the host number. The network prefix identifies a unique physical or virtual subnet. The host number identifies a specific host on the subnet. All hosts on the same subnet have the same network prefix and can talk to each other directly without a router. A *network gateway* connects different networks, and a network router commonly operates as the gateway for a subnet.

Note

A *subnet* is a segment of a larger network, and the use of the term depends on the context. An IP network is partitioned into multiple, smaller network segments. Typically, *segment* refers to the physical or virtual link layer, whereas *subnet* refers to the logical, network-layer addressing for the corresponding segment.

Additionally, *subnetting* an assigned large network address subdivides it into multiple, smaller network segments. This IPv4 section introduces network addresses that are implemented as single subnets. The upcoming IPv6 section includes another context, where large networks are *subnetted* into multiple subnets.

In the original IPv4 specification, the allowed network prefixes were one of three fixed sizes for *unicast* packets that have a single source and destination. The network prefix might be 8 bits (*class A*), 16 bits (*class B*), or 24 bits (*class C*). Today, the number of bits in the network prefix is variable, which means that the prefix can be any number in the supported range, and this later specification is called *Classless Inter-Domain Routing (CIDR)*. Although fixed-address classes are no longer in use, many network professionals still refer to networks with 8-bit, 16-bit, or 24-bit network prefixes by using the original class A, B, or C designation.

A *network mask* (*netmask*) is a binary mask whose length indicates how many bits belong to the network prefix that identifies the subnet. Because an IPv4 address is always 32 bits long, a subnet with a longer network mask has fewer available bits to identify hosts, which means fewer possible hosts. A subnet with a shorter network mask has more available bits to identify hosts, which means more possible hosts and a larger subnet.

Network masks are expressed in one of two forms, which are both used routinely. The first form, which is known as *CIDR notation*, appends a forward slash (`/`) and an integer up to 32 that indicates the number of bits in the binary mask. The second notation displays the number of bits in the binary mask as four 8-bit octets in decimal format.

IPv4 Subnets and Netmasks

The number of available host addresses in a subnet depends on the network prefix size. For example, a network prefix of `/24` leaves 8 bits, or 255 possible host addresses in the subnet. A network prefix of `/16` leaves 16 bits, or 65536 possible host addresses in the subnet.

- The *network address* for a subnet is the lowest possible address on a subnet, where the host number is all binary zeros.
- The *broadcast address* for a subnet is the highest possible address on a subnet, where the host number is all binary ones, and is a special address for broadcasting packets to all subnet hosts.
- The *gateway address* for a subnet can be any unique host number in the subnet, but is commonly set to the first available host number, which is a binary number of all zeroes except for a '1' in the last bit. This gateway numbering convention is not mandatory, and subnets that do not need external communication do not set a network gateway.

The following figures illustrate the use of an IP address and a netmask to calculate the network prefix and the host number for a subnet. Perform a *binary AND* calculation where each bit in the IP address and netmask is binary, and compare each bit to its corresponding bit in the IP address and netmask through the prefix length. In an AND calculation, both bits must be a '1' for the result to be a

'1', and all other combinations result in `0`. Perform a *binary OR* calculation on the remaining bits in the host number, where either bit can be a '1' for the result to be a '1'. In a binary OR calculation, only two '0' bits result in a '0'.

Figure 11.2: IPv4 netmask calculation for a small network

Figure 11.3: IPv4 netmask calculation for a larger network

Example Network Calculations

In the following example, identify the netmask first, and then perform the binary calculations. A netmask of `/24` means that the leading 24 bits of the address define the network address (`192.168.1.0`). In this scenario, 8 bits, or 254 addresses, are available for host addressing.

Table 11.1. IPv4 address of `192.168.1.107/24`

Network prefix	/24 or 255.255.255.0	`11111111.11111111.11111111.00000000`
Host address	192.168.1.107	`11000000.10101000.00000001.01101011`
Network address	192.168.1.0	`11000000.10101000.00000001.00000000`
Address range for hosts on subnet	192.168.1.1 - 192.168.1.254	`11000000.10101000.00000001.00000001` to `11000000.10101000.00000001.11111110`
Broadcast address	192.168.1.255	`11000000.10101000.00000001.11111111`

In the following example, a /19 netmask is a valid network prefix that uses only a partial octet. Variable length netmasks allow subnets with a different quantity of hosts than the full-octet netmasks. The remaining 13 bits, or 8190 addresses, are available for host addressing.

Table 11.2. IPv4 address of `172.16.181.23/19`

Network prefix	/19 or 255.255.224.0	`11111111.11111111.11100000.00000000`

Host address	172.16.181.23	`10101100.00010000.10110101.00010111`
Network address	172.16.160.0	`10101100.00010000.10100000.00000000`
Address range for hosts on subnet	172.16.160.1 - 172.16.191.254	`10101100.00010000.10100000.00000001` to `10101100.00010000.10111111.11111110`
Broadcast address	172.16.191.255	`10101100.00010000.10111111.11111111`

In the following example, the `/8` indicates a large network. Only the first octet is used for the network prefix (`10.0.0.0`). The remaining 24 bits, or 16,777,214 addresses, are available for host addressing. The `10.255.255.255` broadcast address is the last address of the network.

Table 11.3. IPv4 address of `10.1.1.18/8`

Network prefix	/8 or 255.0.0.0	`11111111.00000000.00000000.00000000`
Host address	10.1.1.18	`00001010.00000001.00000001.00010010`
Network address	10.0.0.0	`00001010.00000000.00000000.00000000`
Address range for hosts on subnet	10.0.0.1 - 10.255.255.254	`00001010.00000000.00000000.00000001` to `00001010.11111111.11111111.11111110`

Broadcast address	10.255.255.255	`00001010.11111111.11111111.11111111`

IPv4 Routes

Network packets move from host to host on a subnet and through routers from network to network. Each host has a routing table, which determines which network interface is correct for sending packets to particular networks. A routing table entry lists the destination network, which network interface to use, and the IP address of the router to forward the packet to the final destination. The routing table entry that matches the network prefix of the destination address is used to route the packet. If multiple entries are valid for the destination address, then the entry with the longer prefix is used.

If the destination network does not match a more specific entry, then the packet is routed by using the `0.0.0.0/0` default entry. This default route points to the gateway router on a local subnet that the host can reach.

When a router receives packets that are not addressed to itself, the router forwards the traffic based on its own routing table. Forwarding might send the packet directly to the destination host if this router is on the destination's subnet, or might forward the packet again to another router network. Packet forwarding on routers continues until the packet reaches the requested destination network and host.

Figure 11.4: Example network topology

Table 11.4. Example routing table for the `hostb` machine

Destination	**Interface**	**Router (if needed)**
192.168.5.0/24	`enp0s1f0`	
192.168.6.0/24	`enp0s2f0`	
172.17.0.0/16	`enp0s1f0`	192.168.5.1
0.0.0.0/0 (*default*)	`enp0s1f0`	192.168.5.1

Consider the preceding network diagram and network routing table.

- Network traffic from the `hostb` machine to any host in the `192.168.6.0/24` network is transmitted directly via the `enp0s2f0` interface.
  - This traffic is because the `hostb` machine has an interface attached to that network, and is the closest match to the route entry.
- Network traffic from the `hostb` machine to a host with the `172.17.50.120` IP address uses the `enp0s1f0` interface, because the traffic matches the third entry in the routing table.
  - The `hostb` machine does not have an interface that is directly attached to this network, so this traffic is sent to the next hop router with the address of `192.168.5.1`, which is reachable via the `enp0s1f0` interface. The traffic is then forwarded to its destination.
  - Because the `hostb` machine does not have an interface that is directly connected to the `172.17.0.0/16` network, someone with knowledge of the network topology must add this route entry to the routing table.
- Network traffic with a destination that does not match any entry in the routing table is sent to the default route. The default route, which is designated with `0.0.0.0/0`, is shown in the fourth entry.
  - For example, all traffic from the `hostb` machine to the internet is forwarded to the next hop router with the address of `192.168.5.1`, which is reachable via the `enp0s1f0` interface. The traffic is then forwarded to its destination.

IPv4 Address and Route Configuration

A server can automatically configure its IPv4 network settings by communicating with a DHCP server. A local DHCP client queries the subnet by using a link layer protocol to locate a DHCP server or proxy, and negotiates to use a unique address and other settings for a specific lease period. The client must periodically request lease renewal to maintain use of the assigned network configuration.

As an alternative, you can configure a server to use a static network configuration. Static network settings are read from local configuration files. The settings that you use must be appropriate for your subnet. Coordinate with your network administrator to avoid conflicts with other servers in the same subnets.

IPv6 Networks

IPv6 is designed to greatly expand the number of total available device addresses. IPv6 is used in both enterprise networks and for mobile communications. Most if not all *Internet Service Providers (ISPs)* use IPv6 extensively for assigning to internal equipment and for dynamic assignment for customer devices.

IPv6 can also be used in parallel with IPv4 in a dual-stack mode. A network interface can have both IPv6 and IPv4 addresses. Red Hat Enterprise Linux operates in a dual-stack mode by default.

IPv6 Addresses

An IPv6 address is a 128-bit number, which is normally expressed as eight colon-separated groups of four hexadecimal *nibbles* (half-bytes). Each nibble represents four bits of the IPv6 address, so each group represents 16 bits of the IPv6 address.

```
2001:0db8:0000:0010:0000:0000:0000:0001
```

To simplify writing IPv6 addresses, leading zeros in a colon-separated group are not needed. However, at least one hexadecimal digit must be written in each colon-separated group.

```
2001:db8:0:10:0:0:0:1
```

Because addresses with long strings of zeros are common, one or more consecutive groups of zeros only can be combined with *exactly one* block of two colon `::` characters.

```
2001:db8:0:10::1
```

The `2001:db8::0010:0:0:0:1` IPv6 address, though a valid representation, is a less convenient way to write the example address. This different representation can confuse administrators who are new to IPv6. The following list shows tips for writing consistently readable addresses:

- Suppress leading zeros in a group.
- Use a two-colon `::` block to shorten the address as much as possible.
- If an address contains two consecutive groups of zeros, which are equal in length, then shorten the leftmost groups of zeros to `::` and the rightmost groups to `:0:` for each group.
- Although it is allowed, do not use `::` to shorten a single group of zeros. Use `:0:` instead, and save `::` for consecutive groups of zeros.
- Always use lowercase letters for `a` through `f` hexadecimal characters.

Important

When including a TCP or UDP network port after an IPv6 address, always enclose the IPv6 address in square brackets so that the port does not appear to be part of the address.

```
[2001:db8:0:10::1]:80
```

IPv6 Subnets

A normal IPv6 unicast address is divided into two parts: the *network prefix* and *interface ID*. The network prefix identifies the subnet. Two network interfaces on the same subnet cannot have the same interface ID; the interface ID identifies a particular interface on the subnet.

Unlike IPv4, IPv6 has a standard subnet mask, `/64`, which is used for almost all normal addresses. In this case, half of the 128 bit address is the network prefix and the other half is the interface ID. With 64 bits for host addresses, a single subnet could theoretically contain 2^64 hosts.

Typically, the network provider allocates a shorter prefix to an organization, such as a `/48`. This prefix leaves the rest of the network part for assigning subnets (up to the maximum `/64` length) from that allocated prefix. For example, when a `/48` allocation prefix is assigned, 16 bits are available for up to 65536 subnets.

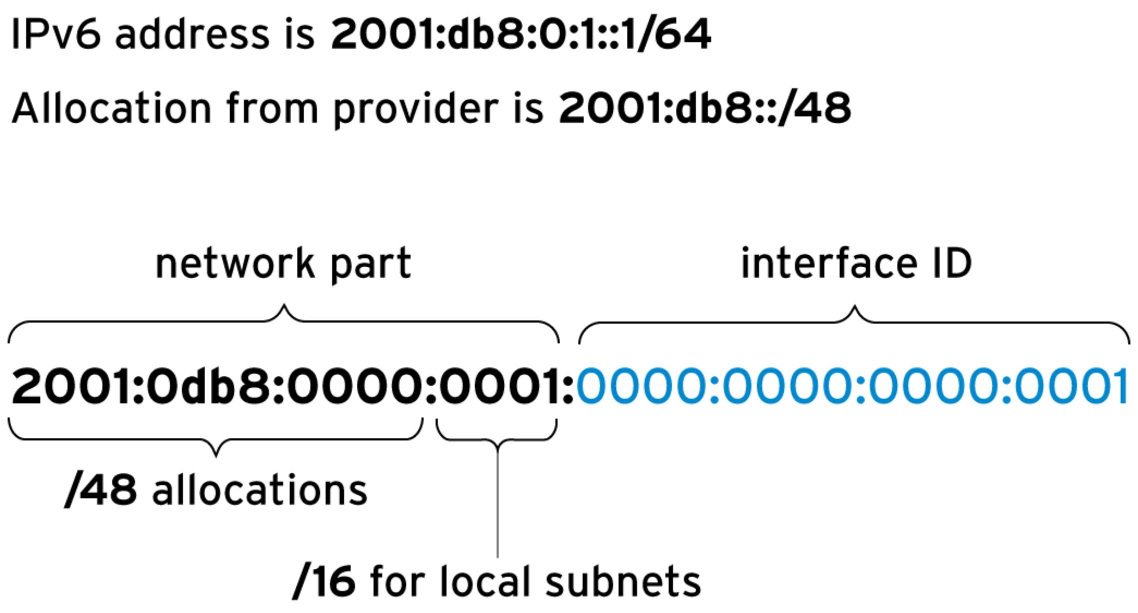
IPv6 address is 2001:db8:0:1::1/64
Allocation from provider is 2001:db8::/48
network part
interface ID
2001:0db8:0000:0001:0000:0000:0000:0001
/48 allocations
/16 for local subnets

Figure 11.5: IPv6 address parts and subnetting

Table 11.5. Common IPv6 Addresses and Networks

IPv6 address or network	Purpose	Description
`::1/128`	localhost	The IPv6 equivalent to the `127.0.0.1/8` address, which is set on the loopback interface.
`::`	The unspecified address	The IPv6 equivalent to `0.0.0.0`. For a network service, it might indicate that it is listening on all configured IP addresses.
`::/0`	The default route (the IPv6 internet)	The IPv6 equivalent to the `0.0.0.0/0` address. The default route in the routing table matches this network; the router for this network is where all traffic is sent in the absence of a better route.
`2000::/3`	Global unicast addresses	The *Internet Assigned Numbers Authority (IANA)* currently allocates "normal" IPv6 addresses from this space. The addresses include all the networks that range from `2000::/16` through `3fff::/16`.
`fd00::/8`	Unique local addresses (RFC 4193)	IPv6 has no direct equivalent of the RFC 1918 private address space, although this network range is close. A site can use these networks to self-allocate a private routable IP address space inside the organization. However, these networks cannot be used on the global internet. The site

IPv6 address or network	Purpose	Description
		must randomly select a `/48` from this space, but it can subnet the allocation into `/64` networks normally.
`fe80::/10`	Link-local addresses	Every IPv6 interface automatically configures a link-local unicast address that works only on the local link on the `fe80::/64` network. However, the entire `fe80::/10` range is reserved for future use by the local link. This topic is discussed in more detail later.
`ff00::/8`	Multicast	The IPv6 equivalent to the `224.0.0.0/4` address. Multicast is used to transmit to multiple hosts at the same time, and is particularly important in IPv6 because it has no broadcast addresses.

Important

The previous table lists network address *allocations* that are reserved for specific purposes. These allocations might consist of many networks. IPv6 networks are allocated from the global unicast and link-local unicast address spaces that have a standard `/64` network mask.

A link-local address in IPv6 is an unroutable address that the system uses only to talk to other systems on the same network link. To ensure that the IP address is unique, the system uses a specific method to compute the interface ID of the link-local address.

Note

Originally, the interface ID for the IPv6 link-local address was constructed from the MAC address of the network device. Exposing the MAC address as part of the IPv6 address might cause some security and privacy issues, because it becomes possible to identify and follow a computer on the network.

By default in Red Hat Enterprise Linux 9, NetworkManager generates a random but stable interface ID for the interface, according to the algorithm in RFC 7217. This algorithm is controlled by the `ipv6.addr-gen-mode` connection setting, which defaults to `stable-privacy`.

IPv6 Privacy Extensions (RFC 4941) are a different solution to the same concern and are controlled by different settings, which are disabled by default.

Use the `ip addr show` command to retrieve the link-local IPv6 address, as in the following example. Add the `-br` option for a brief listing of addresses only.

```
[user@host ~]$ ip addr show dev eth0
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group def
ault qlen 1000
    link/ether 52:54:00:01:fa:0a brd ff:ff:ff:ff:ff:ff
    inet 10.42.0.1/16 brd 10.42.255.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::7418:cf98:c742:3681/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Add the `-br` option for a brief listing of IPv4 and IPv6 addresses only.

```
[user@host ~]$ ip -br addr show dev eth0
eth0             UP             10.42.0.1/16 fe80::7418:cf98:c742:3681/64
```

To operate correctly, IPv6 relies on the link-local address. The interface always keeps that address, even when you assign a routable IPv6 address manually or with an automated method.

With multicast, one system can send traffic to a special IP address that multiple systems receive. Multicast differs from broadcast, because broadcast packets are not routable and reach only local subnet hosts. Conversely, multicast packets are routed to specific hosts that announced a request for the uniquely addressed multicast packets. Multicast packets can be routed to other subnets, if all intermediary routers support forwarding multicast requests and routing.

Multicast plays a larger role in IPv6 than in IPv4, because IPv6 has no broadcast address. The `ff02::1` IPv6 address is a key multicast address that is used as the `all-nodes` link-local address, and behaves like a broadcast address. You can ping this address to send traffic to all nodes on the link. Link-scope multicast addresses (which start with `ff02::/8`) must be specified with a scope identifier, as for a link-local address.

```
[user@host ~]$ ping6 ff02::1%ens3
PING ff02::1%ens3(ff02::1) 56 data bytes
64 bytes from fe80::211:22ff:feaa:bbcc: icmp_seq=1 ttl=64 time=0.072 ms
```

```
64 bytes from fe80::200:aaff:fe33:2211: icmp_seq=1 ttl=64 time=102 ms (DUP!)
64 bytes from fe80::bcd:efff:fea1:b2c3: icmp_seq=1 ttl=64 time=103 ms (DUP!)
64 bytes from fe80::211:22ff:feaa:bbcc: icmp_seq=2 ttl=64 time=0.079 ms
...output omitted...
```

IPv6 Address Configuration

IPv4 has two ways to configure addresses on network interfaces. The administrator can manually configure network addresses on interfaces, or addresses can be configured dynamically with DHCP. IPv6 supports manual configuration, and two methods of dynamic configuration, one of which is DHCPv6.

You can select interface IDs for static IPv6 addresses, similar to IPv4. In IPv4, two addresses on a network cannot be used: the lowest address, which is the network address; and the highest address, which is the broadcast address. In IPv6, two interface IDs are reserved, and cannot be used as normal host interface addresses:

- The all-zeros identifier `0000:0000:0000:0000` ("subnet router anycast") that all routers on the link use. For example, on the `2001:db8::/64` network, the anycast address is `2001:db8::`.
- The identifiers `fdff:ffff:ffff:ff80` through `fdff:ffff:ffff:ffff`.

DHCPv6 lease negotiations work differently from IPv4 DHCP, because DHCPv6 has no broadcast address. A host sends a DHCPv6 request from a link-local address to port 547/UDP on the dedicated `ff02::1:2` `all-dhcp-servers` link-local multicast group. A listening DHCPv6 server can choose to reply with appropriate information to port 546/UDP on the client's provided link-local address.

The `dhcp` package in Red Hat Enterprise Linux 9 provides support for a DHCPv6 server.

In addition to DHCPv6, IPv6 also supports another dynamic configuration method, which is called *Stateless Address Autoconfiguration (SLAAC)*. To use SLAAC, a host configures its interface with a link-local `fe80::/64` address, and sends a "router solicitation" to the `ff02::2` `all-routers` link-local multicast group. An IPv6 router on the local link responds to the host's link-local address with the subnet's previously configured network prefix and other relevant information. The host uses the provided network prefix with an interface ID that is constructed the same as for link-local addresses. The router periodically sends multicast updates (*router advertisements*) to confirm or to update the provided network information.

With the `radvd` package in Red Hat Enterprise Linux 9, a Red Hat Enterprise Linux based IPv6 router can provide SLAAC through router advertisements.

Important

A typical Red Hat Enterprise Linux 9 system that is configured for dynamic IPv4 addresses with DHCP is typically configured for dynamic IPv6 by using SLAAC. Hosts with a dynamic IPv6 configuration might unexpectedly obtain additional IPv6 addresses when a new IPv6 router is added to the network.

Some IPv6 deployments combine SLAAC and DHCPv6, and use SLAAC to provide the network address information, where DHCPv6 provides more network options, such as DNS servers and search domains.

Hostnames and IP Addresses

IP addresses are not human-friendly in daily use. Users generally prefer to work with hostnames rather than with number strings. Linux has *name resolution* mechanisms to map a hostname to an IP address.

One method is to create static entries for each hostname in each system's `/etc/hosts` file. With this method, you must manually update each server's copy of the `hosts` file.

When configured, you can look up the address for a hostname (or a hostname from an address) by using the *Domain Name System (DNS)* network service. DNS is a distributed network of servers that provides name resolution mappings. For name resolution to work, a host must be configured to know where to contact a *nameserver*. The nameserver does not need to be on the same subnet, but the host must be able to reach it. A nameserver configuration is typically obtained through DHCP or by creating static address settings in the `/etc/resolv.conf` file. Later sections of this chapter discuss how to configure name resolution.

References

`services`(5), `ping`(8), `biosdevname`(1), and `udev`(7) man pages

For more information, refer to the *Configuring and Managing Networking Guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/configuring_and_managing_networking/index/

Understanding systemd's Predictable Network Device Names

Selected IETF RFC references:

RFC2460: Internet Protocol, Version 6 (IPv6) Specification

RFC4291 IP Version 6 Addressing Architecture

RFC5952: A Recommendation for IPv6 Address Text Representation

RFC4862: IPv6 Stateless Address Autoconfiguration

RFC3315: Dynamic Host Configuration Protocol for IPv6 (DHCPv6)

RFC3736: Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6

RFC4193: Unique Local IPv6 Unicast Addresses

RFC7217: A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)

RFC8415: Dynamic Host Configuration Protocol for IPv6 (DHCPv6)

Next

Quiz: Describe Networking Concepts

Choose the correct answer to the following questions:

1.

2.

1. What is the size,

in bits,
of an
IPv4
address?

A	4
B	8
C	16
D	[✓ 32]
E	64
F	128

3. CheckResetShow Solution

4.

5.

2. Which term determines how many leading bits in the IP address contribute to its network address?

A	netscope

B	[✓ netmask]
C	subnet
D	multicast
E	netaddr
F	network

6. CheckResetShow Solution

7.

8.

3. Which address represents a valid IPv4 *host* address on the 192.168.1.0/24 network?

A	[✓ 192.168.1.188]
B	192.168.1.0
C	192.168.1.255
D	192.168.1.256

9. CheckResetShow Solution

10.

11.

4. Which number is the size, in bits, of an IPv6 address?

A	4
B	8
C	16
D	32
E	64
F	[✓ 128]

12. CheckResetShow Solution

13.

14.

5. Which address does not represent a valid IPv6 address?

A	2000:0000:0000:0000:0000:0000:0000:0001

B	2::1
C	::
D	ff02::1:0:0
E	2001:3::7:0:2
F	[✓ 2001:db8::7::2]
G	2000::1

15. CheckResetShow Solution

16.

17.

6. Which term refers to the ability of one system to send traffic to a special IP address that multiple systems receive?

A	netscope

B	netmask
C	subnet
D	[✓ multicast]
E	netaddr
F	network

18. CheckResetShow Solution

Previous Next

Validate Network Configuration

Objectives

Test and inspect the current network configuration with command-line utilities.

Gather Network Interface Information

The `ip link` command lists all available network interfaces on your system. In the following example, the server has three network interfaces: `lo`, which is the loopback device that is connected to the server itself, and two Ethernet interfaces, `ens3` and `ens4`.

```
[user@host ~]$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT grou
p default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT gr
oup default qlen 1000
    link/ether 52:54:00:00:00:0a brd ff:ff:ff:ff:ff:ff
```

```
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT gr
oup default qlen 1000

    link/ether 52:54:00:00:00:1e brd ff:ff:ff:ff:ff:ff
```

To configure a network interface correctly, you must know which interface is connected to which network. Often, you can obtain the MAC address of the interface that is connected to each network, either because it is physically printed on the card or server, or because it is a virtual machine and you know how it is configured. The MAC address of the device is listed after `link/ether` for each interface. So you know that the network card with the MAC address `52:54:00:00:00:0a` is the network interface `ens3`.

Display IP Addresses

Use the `ip` command to view device and address information. A single network interface can have multiple IPv4 or IPv6 addresses.

```
[user@host ~]$ ip addr show ens3
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 10

00


    link/ether 52:54:00:00:00:0b brd ff:ff:ff:ff:ff:ff


    inet 192.0.2.2/24 brd 192.0.2.255 scope global ens3
       valid_lft forever preferred_lft forever


    inet6 2001:db8:0:1:5054:ff:fe00:b/64 scope global
       valid_lft forever preferred_lft forever


    inet6 fe80::5054:ff:fe00:b/64 scope link
       valid_lft forever preferred_lft forever
```

An active interface is `UP`.

The `link/ether` string specifies the hardware (MAC) address of the device.

The `inet` string shows an IPv4 address, its network prefix length, and scope.

The `inet6` string shows an IPv6 address, its network prefix length, and scope. This address is of *global* scope and is normally used.

This `inet6` string shows that the interface has an IPv6 address of *link* scope that can be used only for communication on the local Ethernet link.

Display Performance Statistics

The `ip` command can also show statistics about network performance. Counters for each network interface can identify the presence of network issues. The counters record statistics, such as for the number of received (RX) and transmitted (TX) packets, packet errors, and dropped packets.

```
[user@host ~]$ ip -s link show ens3

2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 10
00

link/ether 52:54:00:00:00:0a brd ff:ff:ff:ff:ff:ff

    RX: bytes  packets  errors  dropped overrun mcast

    269850     2931     0       0       0       0

    TX: bytes  packets  errors  dropped carrier collsns

    300556     3250     0       0       0       0
```

Verify Connectivity Between Hosts

The `ping` command tests connectivity. The command continues to run until **Ctrl+c** is pressed, unless options are given to limit the number of sent packets.

```
[user@host ~]$ ping -c3 192.0.2.254

PING 192.0.2.1 (192.0.2.254) 56(84) bytes of data.

64 bytes from 192.0.2.254: icmp_seq=1 ttl=64 time=4.33 ms

64 bytes from 192.0.2.254: icmp_seq=2 ttl=64 time=3.48 ms

64 bytes from 192.0.2.254: icmp_seq=3 ttl=64 time=6.83 ms


--- 192.0.2.254 ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 2003ms

rtt min/avg/max/mdev = 3.485/4.885/6.837/1.424 ms
```

The `ping6` command is the IPv6 version of the `ping` command in Red Hat Enterprise Linux. The difference between these commands is that the `ping6` command communicates over IPv6 and takes IPv6 addresses.

```
[user@host ~]$ ping6 2001:db8:0:1::1
PING 2001:db8:0:1::1(2001:db8:0:1::1) 56 data bytes
64 bytes from 2001:db8:0:1::1: icmp_seq=1 ttl=64 time=18.4 ms
64 bytes from 2001:db8:0:1::1: icmp_seq=2 ttl=64 time=0.178 ms
64 bytes from 2001:db8:0:1::1: icmp_seq=3 ttl=64 time=0.180 ms
^C
--- 2001:db8:0:1::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.178/6.272/18.458/8.616 ms
[user@host ~]$
```

When you ping the link-local addresses and the link-local all-nodes multicast group (`ff02::1`), the network interface to use must be specified explicitly with a scope zone identifier (such as `ff02::1%ens3`). If this network interface is omitted, then the *connect: Invalid argument* error is displayed.

You can use the `ping6 ff02::1` command to find other IPv6 nodes on the local network.

```
[user@host ~]$ ping6 ff02::1%ens4
PING ff02::1%ens4(ff02::1) 56 data bytes
64 bytes from fe80::78cf:7fff:fed2:f97b: icmp_seq=1 ttl=64 time=22.7 ms
64 bytes from fe80::f482:dbff:fe25:6a9f: icmp_seq=1 ttl=64 time=30.1 ms (DUP!)
64 bytes from fe80::78cf:7fff:fed2:f97b: icmp_seq=2 ttl=64 time=0.183 ms
64 bytes from fe80::f482:dbff:fe25:6a9f: icmp_seq=2 ttl=64 time=0.231 ms (DUP!)
^C
--- ff02::1%ens4 ping statistics ---
2 packets transmitted, 2 received, +2 duplicates, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.183/13.320/30.158/13.374 ms
[user@host ~]$
[user@host ~]$ ping6 -c 1 fe80::f482:dbff:fe25:6a9f%ens4
PING fe80::f482:dbff:fe25:6a9f%ens4(fe80::f482:dbff:fe25:6a9f) 56 data bytes
64 bytes from fe80::f482:dbff:fe25:6a9f: icmp_seq=1 ttl=64 time=22.9 ms

--- fe80::f482:dbff:fe25:6a9f%ens4 ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 22.903/22.903/22.903/0.000 ms
```

Other hosts on the same link can use IPv6 link-local addresses, like normal addresses.

```
[user@host ~]$ ssh fe80::f482:dbff:fe25:6a9f%ens4
user@fe80::f482:dbff:fe25:6a9f%ens4's password:
Last login: Thu Jun  5 15:20:10 2014 from host.example.com
[user@server ~]$
```

Troubleshoot Router Issues

Network routing is complex, and sometimes traffic does not behave as you might expect. You can use different tools to diagnose router issues.

Describe the Routing Table

Use the `ip` command `route` option to show routing information.

```
[user@host ~]$ ip route
default via 192.0.2.254 dev ens3 proto static metric 1024
192.0.2.0/24 dev ens3 proto kernel scope link src 192.0.2.2
10.0.0.0/8 dev ens4 proto kernel scope link src 10.0.0.11
```

All packets that are destined for the `10.0.0.0/8` network are sent directly to the destination through the `ens4` device. All packets that are destined for the `192.0.2.0/24` network are sent directly to the destination through the `ens3` device. All other packets are sent to the default router at `192.0.2.254`, and also through device `ens3`.

Use the `ip` command `-6` option to show the IPv6 routing table.

```
[user@host ~]$ ip -6 route
unreachable ::/96 dev lo  metric 1024  error -101
unreachable ::ffff:0.0.0.0/96 dev lo  metric 1024  error -101
2001:db8:0:1::/64 dev ens3  proto kernel  metric 256
unreachable 2002:a00::/24 dev lo  metric 1024  error -101
unreachable 2002:7f00::/24 dev lo  metric 1024  error -101
unreachable 2002:a9fe::/32 dev lo  metric 1024  error -101
unreachable 2002:ac10::/28 dev lo  metric 1024  error -101
unreachable 2002:c0a8::/32 dev lo  metric 1024  error -101
unreachable 2002:e000::/19 dev lo  metric 1024  error -101
unreachable 3ffe:ffff::/32 dev lo  metric 1024  error -101
fe80::/64 dev ens3  proto kernel  metric 256
default via 2001:db8:0:1::ffff dev ens3  proto static  metric 1024
```

1. The `2001:db8:0:1::/64` network uses the `ens3` interface (which presumably has an address on that network).
2. The `fe80::/64` network uses the `ens3` interface, for the link-local address. On a system with multiple interfaces, a route to the `fe80::/64` network exists in each interface for each link-local address.
3. The default route to all networks on the IPv6 Internet (the `::/0` network) uses the router at the `2001:db8:0:1::ffff` network and it is reachable with the `ens3` device.

Trace Traffic Routes

To trace the network traffic path to reach a remote host through multiple routers, use either the `traceroute` or the `tracepath` command. These commands can identify issues with one of your routers or an intermediate router. Both commands use UDP packets to trace a path by default; however, many networks block UDP and ICMP traffic. The `traceroute` command has options to trace the path with UDP (default), ICMP (`-I`), or TCP (`-T`) packets. Typically, the `traceroute` command is not installed by default.

```
[user@host ~]$ tracepath access.redhat.com
...output omitted...
 4:  71-32-28-145.rcmt.qwest.net                          48.853ms asymm  5
 5:  dcp-brdr-04.inet.qwest.net                          100.732ms asymm  7
 6:  206.111.0.153.ptr.us.xo.net                          96.245ms asymm  7
 7:  207.88.14.162.ptr.us.xo.net                          85.270ms asymm  8
 8:  ae1d0.cir1.atlanta6-ga.us.xo.net                     64.160ms asymm  7
 9:  216.156.108.98.ptr.us.xo.net                        108.652ms
10:  bu-ether13.atlngamq46w-bcr00.tbone.rr.com           107.286ms asymm 12
```

```
...output omitted...
```

Each line in the output of the `tracepath` command represents a router or hop that the packet passes through between the source and the final destination. The command outputs information for each hop as it becomes available, including the *round trip timing (RTT)* and any changes in the *maximum transmission unit (MTU)* size. The `asymm` indication means that the traffic that reached the router returned from that router by different (*asymmetric*) routes. These routers here are for outbound traffic, not for return traffic.

The `tracepath6` and `traceroute -6` commands are the equivalent IPv6 commands to the `tracepath` and `traceroute` commands.

```
[user@host ~]$ tracepath6 2001:db8:0:2::451
 1?: [LOCALHOST]                            0.091ms pmtu 1500
 1:  2001:db8:0:1::ba                       0.214ms
 2:  2001:db8:0:1::1                        0.512ms
 3:  2001:db8:0:2::451                      0.559ms reached
     Resume: pmtu 1500 hops 3 back 3
```

Troubleshoot Port and Service Issues

TCP services use sockets as endpoints for communication, and are composed of an IP address, protocol, and port number. Services typically listen on standard ports, whereas clients use a random available port. Well-known names for standard ports are listed in the `/etc/services` file.

The `ss` command is used to display socket statistics. The `ss` command replaces the earlier `netstat` tool, from the `net-tools` package, which might be more familiar to some system administrators but is not always installed.

```
[user@host ~]$ ss -ta
State      Recv-Q Send-Q      Local Address:Port          Peer Address:Port
LISTEN     0      128                    *:sunrpc                    *:*

LISTEN     0      128                    *:ssh                       *:*

LISTEN     0      100             127.0.0.1:smtp                      :
```

```
LISTEN     0      128                     *:36889                    *:*


ESTAB      0      0           172.25.250.10:ssh         172.25.254.254:59392
LISTEN     0      128                    :::sunrpc                  :::*


LISTEN     0      128                    :::ssh                     :::*


LISTEN     0      100                   ::1:smtp                    :::*
LISTEN     0      128                    :::34946                   :::*
```

`*:ssh` : The port that is used for SSH is listening on all IPv4 addresses. The asterisk (`*`) character represents *all* when referencing IPv4 addresses or ports.

`127.0.0.1:smtp` : The port that is used for SMTP is listening on the `127.0.0.1` IPv4 loopback interface.

`172.25.250.10:ssh` : The established SSH connection is on the 172.25.250.10 interface and originates from a system with an address of `172.25.254.254`.

`:::ssh` : The port that is used for SSH is listening on all IPv6 addresses. The double colon (`::`) syntax represents all IPv6 interfaces.

`::1:smtp` : The port that is used for SMTP is listening on the ::1 IPv6 loopback interface.

Table 11.6. Options for ss and netstat

Option	Description
`-n`	Show numbers instead of names for interfaces and ports.
`-t`	Show TCP sockets.
`-u`	Show UDP sockets.
`-l`	Show only listening sockets.
`-a`	Show all (listening and established) sockets.
`-p`	Show the process that uses the sockets.
`-A inet`	Display active connections (but not listening sockets) for the `inet` address family. That is, ignore local UNIX domain sockets. For the `ss` command, both IPv4 and IPv6 connections are displayed. For the `netstat` command, only IPv4 connections are displayed. (The `netstat -A inet6` command displays IPv6 connections, and the `netstat -46` command displays IPv4 and IPv6 at the same time.)

References

ip-link(8), ip-address(8), ip-route(8), ip(8), ping(8), tracepath(8), traceroute(8), ss(8), and netstat(8) man pages

For more information, refer to the *Configuring and Managing Networking Guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/configuring_and_managing_networking/index

Previous Next

Guided Exercise: Validate Network Configuration

In this exercise, you inspect the network configuration of one of your servers.

Outcomes

- Identify the current network interfaces and network addresses.

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start net-validate
```

Instructions

1. Use the ssh command to log in to servera as the student user. The systems are configured to use SSH keys for authentication and passwordless access to servera.

```
2. [student@workstation ~]$ ssh student@servera
3. ...output omitted...
```

```
[student@servera ~]$
```

4. Locate the network interface name that is associated with the `52:54:00:00:fa:0a` Ethernet address. Record or remember this name and use it to replace the `en`*x* placeholder in subsequent commands.

   ### Important

   Network interface names are determined by their bus type and the detection order of devices during boot. Your network interface names vary according to the course platform and hardware in use.

   On your system, locate the interface name (such as `ens06` or `en1p2`) that is associated with the `52:54:00:00:fa:0a` Ethernet address. Use this interface name to replace the `en`*x* placeholder that is used throughout this exercise.

```
[student@servera ~]$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defa
ult qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enX: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gro
up default qlen 1000
    link/ether 52:54:00:00:fa:0a brd ff:ff:ff:ff:ff:ff
```

5. Display the current IP address and netmask for all interfaces.

```
[student@servera ~]$ ip -br addr
lo          UP          127.0.0.1/8 ::1/128
```

```
enX:        UP          172.25.250.10/24 fe80::3059:5462:198:58b2/64
```

8. Display the statistics for the `en`*x* interface.

```
[student@servera ~]$ ip -s link show enX
2: enX: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
e DEFAULT group default qlen 1000
    link/ether 52:54:00:00:fa:0a brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    89014225   168251   0       154418  0       0
    TX: bytes  packets  errors  dropped carrier collsns
```

```
    608808     6090     0       0       0       0
```

15. Display the route information.

```
16. [student@servera ~]$ ip route
17. default via 172.25.250.254 dev enX proto static metric 100
```

```
172.25.250.0/24 dev enX proto kernel scope link src 172.25.250.10 metric 100
```

18. Verify that the router is accessible.

```
19. [student@servera ~]$ ping -c3 172.25.250.254
20. PING 172.25.250.254 (172.25.250.254) 56(84) bytes of data.
21. 64 bytes from 172.25.250.254: icmp_seq=1 ttl=64 time=0.196 ms
22. 64 bytes from 172.25.250.254: icmp_seq=2 ttl=64 time=0.436 ms
23. 64 bytes from 172.25.250.254: icmp_seq=3 ttl=64 time=0.361 ms
24.
25. --- 172.25.250.254 ping statistics ---
26. 3 packets transmitted, 3 received, 0% packet loss, time 49ms
```

```
rtt min/avg/max/mdev = 0.196/0.331/0.436/0.100 ms
```

27. Show all the hops between the local system and the `classroom.example.com` system.

```
28. [student@servera ~]$ tracepath classroom.example.com
29.  1?: [LOCALHOST]                      pmtu 1500
30.  1:  bastion.lab.example.com                              0.337ms
31.  1:  bastion.lab.example.com                              0.122ms
32.  2:  172.25.254.254                                       0.602ms reached
```

```
     Resume: pmtu 1500 hops 2 back 2
```

33. Display the listening TCP sockets on the local system.

```
34. [student@servera ~]$ ss -lt
35. State      Recv-Q Send-Q      Local Address:Port        Peer Address:Port
36. LISTEN     0      128               0.0.0.0:sunrpc           0.0.0.0:*
37. LISTEN     0      128               0.0.0.0:ssh              0.0.0.0:*
38. LISTEN     0      128                  [::]:sunrpc              [::]:*
```

```
LISTEN     0      128                  [::]:ssh                  [::]:*
```

39. Return to the `workstation` system as the `student` user.

```
40. [student@servera ~]$ exit
41. logout
42. Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish net-validate
```

This concludes the section.

Previous Next

Configure Networking from the Command Line

Objectives

Manage network settings and devices with the `nmcli` command.

Describe the NetworkManager Service

The `NetworkManager` service monitors and manages a system's network settings. In the GNOME graphical environment, a Notification Area applet displays network configuration and status information that is received from the `NetworkManager` daemon. You can interact with the `NetworkManager` service via the command line or with graphical tools. Service configuration files are stored in the `/etc/NetworkManager/system-connections/` directory.

The `NetworkManager` service manages network *devices* and *connections*. A *device* is a physical or virtual network interface that provides for network traffic.

A *connection* has related configuration settings for a single network device. A connection can also be known as a *network profile*. Each connection must have a unique name or ID, which can match the device name that it configures.

A single device can have multiple connection configurations and switch between them, but only one connection can be active per device. For example, a laptop wireless device might configure a fixed IP address for use at a secure work site in a connection, but might configure a second connection with an automated address and a virtual private network (VPN) to access the same company network from home.

Important

Starting in Red Hat Enterprise Linux 8, `ifcfg` format configuration files and the `/etc/sysconfig/network-scripts/` directory are deprecated. NetworkManager now uses an INI-style key file format, which is a key-value pair structure to organize properties. NetworkManager stores network profiles in the `/etc/NetworkManager/system-connections/` directory. For compatibility with earlier versions, `ifcfg` format connections in the `/etc/sysconfig/network-scripts/` directory are still recognized and loaded.

View Network Information

Use the `nmcli` utility to create and edit connection files from the command line. The `nmcli device status` command displays the status of all network devices:

```
[user@host ~]$ nmcli dev status
DEVICE  TYPE      STATE         CONNECTION
eno1    ethernet  connected     eno1
ens3    ethernet  connected     static-ens3
eno2    ethernet  disconnected  --
lo      loopback  unmanaged     --
```

Note

You can abbreviate `nmcli` objects and actions. For example, you can abbreviate `nmcli device disconnect` as `nmcli dev dis`, and abbreviate `nmcli connection modify` as `nmcli con mod`. The abbreviation can be as short as a single letter, provided that it uses enough characters to uniquely identify the object to manage.

The `nmcli connection show` command displays a list of all connections. Use the `--active` option to list only active connections.

```
[user@host ~]$ nmcli con show
NAME         UUID                                  TYPE            DEVICE
eno2         ff9f7d69-db83-4fed-9f32-939f8b5f81cd  802-3-ethernet  --
static-ens3  72ca57a2-f780-40da-b146-99f71c431e2b  802-3-ethernet  ens3
eno1         87b53c56-1f5d-4a29-a869-8a7bdaf56dfa  802-3-ethernet  eno1
[user@host ~]$ nmcli con show --active
NAME         UUID                                  TYPE            DEVICE
static-ens3  72ca57a2-f780-40da-b146-99f71c431e2b  802-3-ethernet  ens3
eno1         87b53c56-1f5d-4a29-a869-8a7bdaf56dfa  802-3-ethernet  eno1
```

Add a Network Connection

Use the `nmcli connection add` command to add network connections. The data for the added network connection is stored in the `/etc/NetworkManager/system-connections/` directory as a file with a `.nmconnection` suffix.

The following example adds an `eno2` connection of the `ethernet` type for the `eno2` network interface:

```
[root@host ~]# nmcli con add con-name eno2 \
type ethernet ifname eno2
Connection 'eno2' (8159b66b-3c36-402f-aa4c-2ea933c7a5ce) successfully added
```

The next example creates an `eno3` connection of the `ethernet` type for the `eno3` network interface with a static IPv4 network setting. This command configures the `192.168.0.5` IP address with a network prefix of `/24` and a network gateway of `192.168.0.254`. The `nmcli connection add` command fails if the connection name that you try to add exists.

```
[root@host ~]# nmcli con add con-name eno3 type ethernet ifname eno3 \
ipv4.method manual ipv4.addresses 192.168.0.5/24 ipv4.gateway 192.168.0.254
```

The next example creates an `eno4` connection for the `eno4` device with static IPv6 and IPv4 addresses. This command configures the `2001:db8:0:1::c000:207` IPv6 address with the `/64` network prefix and the `2001:db8:0:1::1` address as the default gateway. The command also configures the `192.0.2.7` IPv4 address with the `/24` network prefix and the `192.0.2.1` address as the default gateway.

```
[root@host ~]# nmcli con add con-name eno4 type ethernet ifname eno4 \
ipv6.addresses 2001:db8:0:1::c000:207/64 ipv6.gateway 2001:db8:0:1::1 \
ipv6.method manual ipv4.addresses 192.0.2.7/24 ipv4.gateway 192.0.2.1 \
ipv4.method manual
```

Manage Network Connections

The `nmcli connection up` command activates a network connection on the device that it is bound to. Activating a network connection requires the connection name, not the device name.

```
[user@host ~]$ nmcli con show
NAME          UUID                                  TYPE            DEVICE
static-ens3   72ca57a2-f780-40da-b146-99f71c431e2b  802-3-ethernet  --
static-ens5   87b53c56-1f5d-4a29-a869-8a7bdaf56dfa  802-3-ethernet  --
[root@host ~]# nmcli con up static-ens3
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager
/ActiveConnection/2)
```

The `nmcli device disconnect` command disconnects the network device and brings down the connection.

```
[root@host ~]# nmcli dev disconnect ens3
```

Important

Use `nmcli device disconnect` to stop traffic on a network interface and deactivate the connection.

Because most connections enable the `autoconnect` parameter, the `nmcli connection down` command is ineffective for stopping traffic. Although the connection deactivates, autoconnect immediately reactivates the connection if the device is up and available. Autoconnect is a desired behavior, because it maintains connections through temporary network outages.

By disconnecting the device under the connection, the connection is forced to be down until the device is connected again.

Update Network Connection Settings

NetworkManager service connections have two setting types. Static connection properties are configured by the administrator and stored in the `/etc/NetworkManager/system-connections/*.nmconnection` configuration files. Dynamic connection properties are requested from a DHCP server and are not stored persistently.

To list the current settings for a connection, use the `nmcli connection show` command. Settings in lowercase are static properties that the administrator can change. Settings in uppercase are active settings in temporary use for this connection instance.

```
[root@host ~]# nmcli con show static-ens3
connection.id:                          static-ens3
connection.uuid:                        87b53c56-1f5d-4a29-a869-8a7bdaf56dfa
connection.interface-name:              --
connection.type:                        802-3-ethernet
connection.autoconnect:                 yes
connection.timestamp:                   1401803453
connection.read-only:                   no
connection.permissions:
connection.zone:                        --
connection.master:                      --
connection.slave-type:                  --
connection.secondaries:
connection.gateway-ping-timeout:        0
802-3-ethernet.port:                    --
802-3-ethernet.speed:                   0
802-3-ethernet.duplex:                  --
802-3-ethernet.auto-negotiate:          yes
```

```
802-3-ethernet.mac-address:            CA:9D:E9:2A:CE:F0
802-3-ethernet.cloned-mac-address:     --
802-3-ethernet.mac-address-blacklist:
802-3-ethernet.mtu:                    auto
802-3-ethernet.s390-subchannels:
802-3-ethernet.s390-nettype:           --
802-3-ethernet.s390-options:
ipv4.method:                           manual
ipv4.dns:                              192.168.0.254
ipv4.dns-search:                       example.com
ipv4.addresses:                        { ip = 192.168.0.2/24,
                                         gw = 192.168.0.254 }
ipv4.routes:
ipv4.ignore-auto-routes:               no
ipv4.ignore-auto-dns:                  no
ipv4.dhcp-client-id:                   --
ipv4.dhcp-send-hostname:               yes
ipv4.dhcp-hostname:                    --
ipv4.never-default:                    no
ipv4.may-fail:                         yes
ipv6.method:                           manual
ipv6.dns:                              2001:4860:4860::8888
ipv6.dns-search:                       example.com
ipv6.addresses:                        { ip = 2001:db8:0:1::7/64,
                                         gw = 2001:db8:0:1::1 }
ipv6.routes:
ipv6.ignore-auto-routes:               no
ipv6.ignore-auto-dns:                  no
ipv6.never-default:                    no
ipv6.may-fail:                         yes
ipv6.ip6-privacy:                      -1 (unknown)
ipv6.dhcp-hostname:                    --
...output omitted...
```

Use the `nmcli connection modify` command to update connection settings. These changes are saved in the `/etc/NetworkManager/system-connections/*name*.nmconnection` file. Consult the `nm-settings`(5) man page for the available settings.

Use the following command to update the `static-ens3` connection to set the `192.0.2.2/24` IPv4 address and the `192.0.2.254` default gateway. Use the `nmcli` command `connection.autoconnect` parameter to automatically enable or disable the connection at system boot.

```
[root@host ~]# nmcli con mod static-ens3 ipv4.addresses 192.0.2.2/24 \
ipv4.gateway 192.0.2.254 connection.autoconnect yes
```

Use the following command to update the `static-ens3` connection to set the `2001:db8:0:1::a00:1/64` IPv6 address and the `2001:db8:0:1::1` default gateway.

```
[root@host ~]# nmcli con mod static-ens3 ipv6.addresses 2001:db8:0:1::a00:1/64 \
ipv6.gateway 2001:db8:0:1::1
```

Important

To change a DHCP connection configuration to be static, update the `ipv4.method` setting from `auto` or `dhcp` to `manual`. For an IPv6 connection, update the `ipv6.method` setting. If the method is not set correctly, then the connection might hang or be incomplete when activated, or it might obtain an address from DHCP or SLAAC in addition to the configured static address.

Some settings can have multiple values. A specific value can be added to the list or deleted from the connection settings by adding a plus (`+`) or minus (`-`) symbol to the start of the setting name. If a plus or minus is not included, then the specified value replaces the setting's current list. The following example adds the `2.2.2.2` DNS server to the `static-ens3` connection.

```
[root@host ~]# nmcli con mod static-ens3 +ipv4.dns 2.2.2.2
```

You can also modify network profiles by editing the connection's configuration file in `/etc/NetworkManager/system-connections/`. Whereas `nmcli` commands communicate directly with NetworkManager to implement modifications immediately, connection file edits are not implemented until NetworkManager is asked to reload the configuration file. With manual editing, you can create complex configurations in steps, and then load the final configuration when ready. The following example loads all connection profiles.

```
[root@host ~]# nmcli con reload
```

The next example loads only the `eno2` connection profile at `/etc/NetworkManager/system-connections/eno2.nmconnection`.

```
[root@host ~]# nmcli con reload eno2
```

Delete a Network Connection

The `nmcli connection delete` command deletes a connection from the system. This command disconnects the device and removes the connection configuration file.

```
[root@host ~]# nmcli con del static-ens3
```

Permissions to Modify NetworkManager Settings

The `root` user can use the `nmcli` command to change the network configuration.

Non-privileged users that are logged in on the physical or virtual console can also make most network configuration changes. If a person is on the system's console, then the system is likely being used as a workstation or laptop where the user needs to configure, activate, and deactivate connections. Non-privileged users that log in with `ssh` must switch to the `root` user to change network settings.

Use the `nmcli general permissions` command to view your current permissions. The following example lists the `root` user's NetworkManager permissions.

```
[root@host ~]# nmcli gen permissions
PERMISSION                                                       VALUE
org.freedesktop.NetworkManager.checkpoint-rollback               yes
org.freedesktop.NetworkManager.enable-disable-connectivity-check  yes
org.freedesktop.NetworkManager.enable-disable-network            yes
org.freedesktop.NetworkManager.enable-disable-statistics         yes
org.freedesktop.NetworkManager.enable-disable-wifi               yes
org.freedesktop.NetworkManager.enable-disable-wimax              yes
org.freedesktop.NetworkManager.enable-disable-wwan                yes
org.freedesktop.NetworkManager.network-control                    yes
org.freedesktop.NetworkManager.reload                             yes
org.freedesktop.NetworkManager.settings.modify.global-dns         yes
org.freedesktop.NetworkManager.settings.modify.hostname           yes
org.freedesktop.NetworkManager.settings.modify.own                yes
org.freedesktop.NetworkManager.settings.modify.system             yes
org.freedesktop.NetworkManager.sleep-wake                         yes
org.freedesktop.NetworkManager.wifi.scan                          yes
org.freedesktop.NetworkManager.wifi.share.open                    yes
org.freedesktop.NetworkManager.wifi.share.protected               yes
```

The following example lists the user's NetworkManager permissions.

```
[user@host ~]$ nmcli gen permissions
PERMISSION                                                        VALUE
org.freedesktop.NetworkManager.checkpoint-rollback                auth
org.freedesktop.NetworkManager.enable-disable-connectivity-check  no
org.freedesktop.NetworkManager.enable-disable-network             no
org.freedesktop.NetworkManager.enable-disable-statistics          no
org.freedesktop.NetworkManager.enable-disable-wifi                no
org.freedesktop.NetworkManager.enable-disable-wimax               no
org.freedesktop.NetworkManager.enable-disable-wwan                no
org.freedesktop.NetworkManager.network-control                    auth
org.freedesktop.NetworkManager.reload                             auth
org.freedesktop.NetworkManager.settings.modify.global-dns         auth
org.freedesktop.NetworkManager.settings.modify.hostname           auth
org.freedesktop.NetworkManager.settings.modify.own                auth
org.freedesktop.NetworkManager.settings.modify.system             auth
org.freedesktop.NetworkManager.sleep-wake                         no
org.freedesktop.NetworkManager.wifi.scan                          auth
org.freedesktop.NetworkManager.wifi.share.open                    no
org.freedesktop.NetworkManager.wifi.share.protected               no
```

Useful NetworkManager Commands

The following table lists the key `nmcli` commands that are discussed in this section:

Command	**Purpose**
`nmcli dev status`	Show the NetworkManager status of all network interfaces.
`nmcli con show`	List all connections.
`nmcli con show` *`name`*	List the current settings for the connection name.
`nmcli con add con-name` *`name`*	Add and name a new connection profile.
`nmcli con mod` *`name`*	Modify the connection name.
`nmcli con reload`	Reload the configuration files, after manual file editing.
`nmcli con up` *`name`*	Activate the connection name.
`nmcli dev dis` *`dev`*	Disconnect the interface, which also deactivates the current connection.
`nmcli con del` *`name`*	Delete the specified connection and its configuration file.

References

For more information, refer to the *Getting Started with nmcli* chapter at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/configuring_and_managing_networking/index#getting-started-with-nmcli_configuring-and-managing-networking

`NetworkManager`(8), `nmcli`(1), `nmcli-examples`(5), `nm-settings`(5), `hostnamectl`(1), `resolv.conf`(5), `hostname`(5), `ip`(8), and `ip-address`(8) man pages

Previous Next

Guided Exercise: Configure Networking from the Command Line

In this exercise, you use the `nmcli` command to configure network settings.

Outcomes

- Update a network connection setting from DHCP to static.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start net-configure
```

Instructions

1. Use the `ssh` command to log in to the `servera` machine as the `student` user.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera ~]$ sudo -i
[sudo] password for student: student
```

```
[root@servera ~]#
```

6. Display the network interface information.

   ### Important

   Network interface names are determined by their bus type and the detection order of devices during boot. Your network interface names might vary according to the course platform and hardware in use.

   On your system, locate the interface name (such as `eth1`, `ens06`, or `enp0p2`) that is associated with the Ethernet address `52:54:00:00:fa:0a`. Use this interface name to replace the `eth0` placeholder throughout this exercise if different.

Locate the network interface name that is associated with the `52:54:00:00:fa:0a` Ethernet address. Record or remember this name, and use it to replace the `eth0` placeholder in subsequent commands.

```
[root@servera ~]# ip link

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAU
LT group default qlen 1000

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mo
de DEFAULT group default qlen 1000

    link/ether 52:54:00:00:fa:0a brd ff:ff:ff:ff:ff:ff

    altname enp0s3

    altname ens3
```

7. Use the `nmcli` command to view network settings.
    1. Use the `nmcli con show --active` command to display only the active connections.

       Your network interface name should appear under the `DEVICE` column of the output, and the name of the active connection for that device is listed under the `NAME` column. This exercise assumes that the active connection is called `Wired connection 1`. If the name of the active connection is different, then use that name instead of `Wired connection 1` for the rest of this exercise.

```
[root@servera ~]# nmcli con show --active

NAME                UUID                                  TYPE      DEV
ICE

Wired connection 1  ec3a15fb-2e26-3254-9433-90c66981e924  ethernet  eth0
```

    2. Display all configuration settings for the active connection.

```
[root@servera ~]# nmcli con show "Wired connection 1"

connection.id:                          Wired connection 1

connection.uuid:                        ec3a15fb-2e26-3254-9433-90c66981
e924

connection.stable-id:                   --

connection.type:                        802-3-ethernet

connection.interface-name:              eth0
connection.autoconnect:                 yes
...output omitted...
ipv4.method:                            manual
ipv4.dns:                               172.25.250.220
ipv4.dns-search:                        lab.example.com,example.com
ipv4.dns-options:                       --
ipv4.dns-priority:                      0
ipv4.addresses:                         172.25.250.10/24
ipv4.gateway:                           172.25.250.254
...output omitted...
ipv6.method:                            auto
ipv6.dns:                               --
ipv6.dns-search:                        --
ipv6.dns-options:                       --
ipv6.dns-priority:                      0
ipv6.addresses:                         --
ipv6.gateway:                           --
ipv6.routes:                            --
...output omitted...
GENERAL.NAME:                           Wired connection 1
GENERAL.UUID:                           ec3a15fb-2e26-3254-9433-90c66981e924
GENERAL.DEVICES:                        eth0
GENERAL.IP-IFACE:                       eth0
GENERAL.STATE:                          activated
GENERAL.DEFAULT:                        yes
```

```
...output omitted...
```

34. Show the device status.

```
[root@servera ~]# nmcli dev status
DEVICE     TYPE          STATE          CONNECTION
eth0       ethernet      connected      Wired connection 1
```

```
lo          loopback     unmanaged     --
```

38. Display the settings for the `eth0` device.

```
[root@servera ~]# nmcli dev show eth0
GENERAL.DEVICE:                     eth0
GENERAL.TYPE:                       ethernet
GENERAL.HWADDR:                     52:54:00:00:FA:0A
GENERAL.MTU:                        1500
GENERAL.STATE:                      100 (connected)
GENERAL.CONNECTION:                 Wired connection 1
GENERAL.CON-PATH:                   /org/freedesktop/NetworkManager/ActiveConnection/1
WIRED-PROPERTIES.CARRIER:           on
IP4.ADDRESS[1]:                     172.25.250.10/24
IP4.GATEWAY:                        172.25.250.254
IP4.ROUTE[1]:                       dst = 172.25.250.0/24, nh = 0.0.0.0, mt = 100
IP4.ROUTE[2]:                       dst = 0.0.0.0/0, nh = 172.25.250.254, mt = 100
IP4.DNS[1]:                         172.25.250.220
IP4.SEARCHES[1]:                    lab.example.com
IP4.SEARCHES[2]:                    example.com
IP6.ADDRESS[1]:                     fe80::c38a:ac39:36a1:a43c/64
IP6.GATEWAY:                        --
```

```
IP6.ROUTE[1]:                       dst = fe80::/64, nh = ::, mt = 1024
```

8. Create a static connection with the same IPv4 address, network prefix, and default gateway as the active connection. Name the new connection `static-addr`.

   ## Warning

   Because access to your machine is provided over the primary network connection, setting incorrect values during network configuration might make your machine unreachable. If you machine is unreachable, then use the **Reset** button above what used to be your machine's graphical display and try again.

```
[root@servera ~]# nmcli con add con-name static-addr \
ifname eth0 type ethernet ipv4.method manual ipv4.dns 172.25.250.220 \
ipv4.addresses 172.25.250.10/24 ipv4.gateway 172.25.250.254
Connection 'static-addr' (dc519805-48c4-4b31-b9e9-d3631cf9082c) successfully a
dded.
```

9. Display and activate the new connection.
   1. View all connections.

```
[root@servera ~]# nmcli con show
NAME                UUID                                  TYPE      DEVI
CE
Wired connection 1  ec3a15fb-2e26-3254-9433-90c66981e924  ethernet  eth0
```

```
static-addr         dc519805-48c4-4b31-b9e9-d3631cf9082c  ethernet  --
```

   5. View the active connections.

```
[root@servera ~]# nmcli con show --active
NAME                UUID                                  TYPE      DEVI
CE
```

```
Wired connection 1  ec3a15fb-2e26-3254-9433-90c66981e924  ethernet  eth0
```

   8. Activate the new `static-addr` connection.

```
[root@servera ~]# nmcli con up static-addr
```

```
Connection successfully activated (D-Bus active path: /org/freedesktop/N
etworkManager/ActiveConnection/2)
```

   10. Verify the new active connection.

```
[root@servera ~]# nmcli con show --active
NAME         UUID                                  TYPE      DEVICE
```

```
static-addr  dc519805-48c4-4b31-b9e9-d3631cf9082c  ethernet  eth0
```

10. Update the previous connection so that it does not start at boot. Verify that the `static-addr` connection is used when the system reboots.
    1. Disable the original connection so that it does not start automatically at boot.

```
2. [root@servera ~]# nmcli con mod "Wired connection 1" \
```

```
connection.autoconnect no
```

    3. Reboot the system.

```
4. [root@servera ~]# systemctl reboot
5. Connection to servera closed by remote host.
6. Connection to servera closed.
```

```
[student@workstation ~]$
```

    7. Log in to the `servera` machine and verify that the `static-addr` connection is the active connection.

```
8. [student@workstation ~]$ ssh student@servera
9. ...output omitted...
10. [student@servera ~]$ nmcli con show --active
11. NAME         UUID                                  TYPE      DEVICE
```

```
static-addr  dc519805-48c4-4b31-b9e9-d3631cf9082c  ethernet  eth0
```

11. Test connectivity by using the new network addresses.
    1. Verify the IP address.

```
2. [student@servera ~]$ ip -br addr show eth0
```

```
eth0             UP             172.25.250.10/24 fe80::eb21:9a:24de:e8fe
/64
```

    3. Verify the default gateway.

```
4. [student@servera ~]$ ip route
5. default via 172.25.250.254 dev eth0 proto static metric 100
```

```
172.25.250.0/24 dev eth0 proto kernel scope link src 172.25.250.10 metri
c 100
```

6. Ping the DNS address.

```
[student@servera ~]$ ping -c3 172.25.250.220
PING 172.25.250.220 (172.25.250.220) 56(84) bytes of data.
64 bytes from 172.25.250.220: icmp_seq=1 ttl=64 time=0.777 ms
64 bytes from 172.25.250.220: icmp_seq=2 ttl=64 time=0.431 ms
64 bytes from 172.25.250.220: icmp_seq=3 ttl=64 time=0.272 ms

--- 172.25.250.220 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2045ms
```

```
rtt min/avg/max/mdev = 0.272/0.493/0.777/0.210 ms
```

15. Return to the `workstation` system as the `student` user.

```
[student@servera ~]$ exit
logout
Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish net-configure
```

This concludes the section.

Previous Next

Edit Network Configuration Files

Objectives

Modify network configuration by editing configuration files.

Connection Configuration Files

Starting with Red Hat Enterprise Linux 8, network configurations are stored in the `/etc/NetworkManager/system-connections/` directory. This configuration location uses the key file format instead of the `ifcfg` format. However, the previously stored configurations at `/etc/sysconfig/network-scripts/` continue to work.
The `/etc/NetworkManager/system-connections/` directory stores any changes with the `nmcli con mod` *`name`* command.

Key File Format

The NetworkManager uses the INI-style key format for storing network connection profiles. The key-value pairs store configurations as sections (groups). Each configuration key/value pair in the section is one of the listed properties in the settings specification. This configuration file stores most of the settings in the same format as the INI-style format. For example, writing IP addresses as `192.168.0.1/24` is easier to read than as integer arrays.

Although the recommended way to manage profiles is with the `nmcli` command, users might still manually create or modify the configuration files. After editing the configuration file, run the `nmcli con reload` command to inform NetworkManager about these changes.

Table 11.7. Comparison of NetworkManager Settings and Key File Format File

`nmcli con mod`	**`*.nmconnection` file**	**Effect**

`ipv4.method manual`	`[ipv4]`
	
	`method=manual`
		Configure IPv4 addresses statically.

`ipv4.method auto`	`[ipv4]`
	
	`method=auto`
		Look for configuration settings from a DHCPv4 server. It shows static addresses only when it has information from DHCPv4.

`ipv4.addresses 192.0.2.1/24`	`[ipv4]`
	
	`address1=192.0.2.1/24`
		Set a static IPv4 address and network prefix. For more than one connection address, the `address2` key defines the second address, and the `address3` key defines the third address.

`ipv4.gateway 192.0.2.254`	`[ipv4]`
	
	`gateway=192.0.2.254`
		Set the default gateway.

`nmcli con mod`	`*.nmconnection` file	Effect

`ipv4.dns 8.8.8.8`	`[ipv4]`
	
	`dns=8.8.8.8`
		Modify `/etc/resolv.conf` to use this name server.

`ipv4.dns-search example.com`	`[ipv4]`
	
	`dns-search=example.com`
		Modify `/etc/resolv.conf` to use this domain in the `search` directive.

`ipv4.ignore-auto-dns true`	`[ipv4]`
	
	`ignore-auto-dns=true`
		Ignore DNS server information from the DHCP server.

`ipv6.method manual`	`[ipv6]`
	
	`method=manual`
		Configure IPv6 addresses statically.

`ipv6.method auto`	`[ipv6]`
	
	`method=auto`
		Configure network settings with SLAAC from router advertisements.

`ipv6.method dhcp`	`[ipv6]`
	
	`method=dhcp`
		Configure network settings by using DHCPv6, but not by using SLAAC.

`ipv6.addresses 2001:db8::a/64`	`[ipv6]`
	
	`address1=2001:db8::a/64`
		Set a static IPv6 address and network prefix. When using more than one address for a connection, the `address2` key defines the second address, and the `address3` key defines the third address.

`ipv6.gateway 2001:db8::1`	`[ipv6]`
	
	`gateway=2001:db8::1`
		Set the default gateway.

`ipv6.dns fde2:6494:1e09:2::d`	`[ipv6]`
	
	`dns=fde2:6494:1e09:2::d`
		Modify `/etc/resolv.conf` to use this name server. The same as IPv4.

`ipv6.dns-search example.com`	`[ipv6]`
	
	`dns-search=example.com`
		Modify `/etc/resolv.conf` to use this domain in the `search` directive.

`ipv6.ignore-auto-dns true`	`[ipv6]`
	
	`ignore-auto-dns=true`
		Ignore DNS server information from the DHCP server.

`connection.autoconnect yes`	`[connection]`
	
	`autoconnect=true`
		Automatically activate this connection at boot.

`connection.id ens3`	`[connection]`
	
	`id=Main eth0`
		The name of this connection.

`connection.interface-name ens3`	`[connection]`
	
	`interface-name=ens3`
		The connection is bound to the network interface with this name.

`nmcli con mod`	`*.nmconnection` **file**	**Effect**
`802-3-ethernet.mac-address …`	`[802-3-ethernet]` `mac-address=`	The connection is bound to the network interface with this MAC address.

Modify Network Configuration

You can also configure the network by directly editing the connection configuration files. Connection configuration files control the software interfaces for individual network devices. These files are usually called `/etc/NetworkManager/system-connections/`*`name`*`.nmconnection`, where *name* refers to the device's name or connection that the configuration file controls.

Depending on the purpose of the connection profile, NetworkManager uses the following directories to store the configuration files:

- The `/etc/NetworkManager/system-connections/` directory stores persistent profiles that the user created and edited. NetworkManager copies them automatically to the `/etc/NetworkManager/system-connections/` directory.
- The `/run/NetworkManager/system-connections/` directory stores temporary profiles, which are automatically removed when you reboot the system.
- The `/usr/lib/NetworkManager/system-connections/` directory stores predeployed immutable profiles. When you edit such a profile with the NetworkManager API, NetworkManager copies this profile to either the persistent or the temporary storage.

Sample configuration file content for static IPv4 configuration:

```
[connection]
id=Main eth0
uuid=27afa607-ee36-43f0-b8c3-9d245cdc4bb3
type=802-3-ethernet
autoconnect=true

[ipv4]
method=manual

[802-3-ethernet]
```

```
mac-address=00:23:5a:47:1f:71
```

Table 11.8. IPv4 Configuration Options for Key File Format

Static	*Dynamic*	*Either*

`[ipv4]`
 	
`address1=172.25.0.10/24`
 	
`gateway=172.25.0.254`
 	
`dns=172.25.254.254`
	`method=auto`	`[connection]`
		
		`interface-name=ens3`
		
		`id=Main eth0`
		
		`autoconnect=true`
		
		`uuid=f3e8(…)ad3e`
		
		`type=ethernet`

After you modify the configuration files, run the `nmcli con reload` command so that NetworkManager loads the configuration changes. If the `autoconnect` variable is set to `false` in the connection profile, then activate the connection manually.

```
[root@host ~]# nmcli con reload
[root@host ~]# nmcli con up "static-ens3"
```

References

`nmcli`(1), `nm-settings`(5), and `nm-settings-keyfile`(5) man page

For more information, refer to the *Manually Creating NetworkManager Profiles in Key File Format* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/configuring_and_managing_networking/assembly_manually-creating-networkmanager-profiles-in-key-file-format_configuring-and-managing-networking

Previous Next

Guided Exercise: Edit Network Configuration Files

In this exercise, you manually modify network configuration files and ensure that the new settings take effect.

Outcomes

- Configure additional network addresses on each system.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares your environment and ensures that all required resources are available.

```
[student@workstation ~]$ lab start net-edit
```

Instructions

1. On the `workstation` machine, use the `ssh` command to log in to the `servera` machine as the `student` user.

```
2. [student@workstation ~]$ ssh student@servera
3. ...output omitted...
```

```
[student@servera ~]$
```

4. Locate network interface names with the `ip link` command.

   ### Important

   Network interface names are determined by their bus type and the detection order of devices during boot. Your network interface names might vary according to the course platform and hardware in use.

   Locate the network interface name that is associated with the Ethernet address on your system. Record or remember this name, and use it to replace the *`enx`* placeholder in subsequent commands. The active connection is called `Wired connection 1`, and the configuration is in the `/etc/NetworkManager/system-connections/"Wired connection 1.nmconnection"` file.

```
[student@servera ~]$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAU
LT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mo
de DEFAULT group default qlen 1000
    link/ether 52:54:00:00:fa:0a brd ff:ff:ff:ff:ff:ff
    altname enp0s3
    altname ens3
[student@servera ~]$ nmcli con show --active
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  a98933fa-25c0-36a2-b3cd-c056f41758fe  ethernet  eth0
[student@servera ~]$ ls /etc/NetworkManager/system-connections/
'Wired connection 1.nmconnection'
```

5. On the `servera` machine, switch to the `root` user, and then edit the `/etc/NetworkManager/system-connections/"Wired connection 1.nmconnection"` file to add the `10.0.1.1/24` address.
   1. Use the `sudo -i` command to switch to the `root` user.

```
[student@servera ~]$ sudo -i
[sudo] password for student: student
```

```
[root@servera ~]#
```

   4. Edit the configuration file. Add the `10.0.1.1/24` address as the second address below the first address in the file.

```
[root@servera ~]# vim /etc/NetworkManager/system-connections/"Wired conn
ection 1.nmconnection"
...output omitted...
[ipv4]
address1=172.25.250.10/24,172.25.250.254
address2=10.0.1.1/24
```

```
...output omitted...
```

6. Activate the new network address with the `nmcli` command.
    1. Reload the configuration changes for NetworkManager to read the changes.

```
[root@servera ~]# nmcli con reload
```

    2. Activate the connection with the changes.

```
3. [root@servera ~]# nmcli con up "Wired connection 1"
```

```
Connection successfully activated (D-Bus active path: /org/freedesktop/N
etworkManager/ActiveConnection/2)
```

7. Verify that the new IP address is assigned successfully.

```
8. [root@servera ~]# ip -br addr show eth0
```

```
eth0:       UP       172.25.250.10/24 10.0.1.1/24 fe80::6fed:5a11:4ad4:1bcf/64
```

9. Return to the `workstation` machine as the `student` user.

```
10. [root@servera ~]# exit
11. logout
12. [student@servera ~]$ exit
13. logout
14. Connection to servera closed.
```

```
[student@workstation ~]$
```

15. On the `serverb` machine, edit the `/etc/NetworkManager/system-connections/"Wired connection 1.nmconnection"` file to add an address of `10.0.1.2/24` and load the new configuration.
    1. Log in to the `serverb` machine as the `student` user and switch to the `root` user.

```
2. [student@workstation ~]$ ssh student@serverb
3. ...output omitted...
4. [student@serverb ~]$ sudo -i
5. [sudo] password for student: student
```

```
[root@serverb ~]#
```

6. Edit the configuration file. Add the `10.0.1.2/24` address as the second address below the first address in the file.

```
7. [root@serverb ~]# vim /etc/NetworkManager/system-connections/"Wired conn
   ection 1.nmconnection"

8. address1=172.25.250.11/24,172.25.250.254
```

```
address2=10.0.1.2/24
```

9. Reload the configuration changes for NetworkManager to read the changes.

```
[root@serverb ~]# nmcli con reload
```

10. Activate the connection with the changes.

```
11. [root@serverb ~]# nmcli con up "Wired connection 1"
```

```
Connection successfully activated (D-Bus active path: /org/freedesktop/N
etworkManager/ActiveConnection/2)
```

12. Verify that the new IP address is assigned successfully.

```
13. [root@serverb ~]# ip -br addr show eth0
```

```
eth0        UP        172.25.250.11/24 10.0.1.2/24 fe80::6be8:6651:4280:89
2c/64
```

16. Test connectivity between the `servera` and `serverb` machines by using the new network addresses.
    1. From the `serverb` machine, ping the new address of the `servera` machine.

```
2. [root@serverb ~]# ping -c3 10.0.1.1

3. PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.

4. 64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=1.30 ms

5. 64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.983 ms

6. 64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.312 ms

7.

8. --- 10.0.1.1 ping statistics ---
```

```
9.  3 packets transmitted, 3 received, 0% packet loss, time 2003ms
```

```
rtt min/avg/max/mdev = 0.312/0.864/1.297/0.410 ms
```

10. Return to the `workstation` machine as the `student` user.

```
11. [root@serverb ~]# exit
12. logout
13. [student@serverb ~]$ exit
14. logout
15. Connection to serverb closed.
```

```
[student@workstation ~]$
```

16. Access the `servera` machine as the `student` user to ping the new address of the `serverb` machine.

```
17. [student@workstation ~]$ ssh student@servera ping -c3 10.0.1.2
18. PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
19. 64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.876 ms
20. 64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.310 ms
21. 64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=0.289 ms
22.
23. --- 10.0.1.2 ping statistics ---
24. 3 packets transmitted, 3 received, 0% packet loss, time 2047ms
```

```
rtt min/avg/max/mdev = 0.289/0.491/0.876/0.271 ms
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish net-edit
```

This concludes the section.

Previous Next

Configure Hostnames and Name Resolution

Objectives

Configure a server's static hostname and its name resolution and test the results.

Update the System Hostname

The `hostname` command displays or temporarily modifies the system's fully qualified hostname.

```
[root@host ~]# hostname
host.example.com
```

Specify a static hostname in the `/etc/hostname` file. Use the `hostnamectl` command to modify this file, and view the system's fully qualified hostname. If this file does not exist, then the hostname is set by a reverse DNS query when an IP address is assigned to the interface.

```
[root@host ~]# hostnamectl hostname host.example.com
[root@host ~]# hostnamectl status
   Static hostname: host.example.com
         Icon name: computer-vm
           Chassis: vm 🖴
        Machine ID: ace63d6701c2489ab9c0960c0f1afe1d
           Boot ID: 0edf5ba1830c48adbd6babfa08f0b867
    Virtualization: kvm
  Operating System: Red Hat Enterprise Linux 9.0 (Plow)
       CPE OS Name: cpe:/o:redhat:enterprise_linux:9::baseos
            Kernel: Linux 5.14.0-70.13.1.el9_0.x86_64
      Architecture: x86-64
   Hardware Vendor: Red Hat
    Hardware Model: OpenStack Compute
[root@host ~]# cat /etc/hostname
host.example.com
```

Important

In Red Hat Enterprise Linux 7 and later versions, the static hostname is stored in the `/etc/hostname` file. Red Hat Enterprise Linux 6 and earlier versions store the hostname as a variable in the `/etc/sysconfig/network` file.

Configure Name Resolution

The stub resolver converts hostnames to IP addresses or the reverse. It determines where to look based on the configuration of the `/etc/nsswitch.conf` file. By default, it attempts to resolve the query by first using the `/etc/hosts` file.

```
[root@host ~]# cat /etc/hosts
127.0.0.1       localhost localhost.localdomain localhost4 localhost4.localdomain4
::1             localhost localhost.localdomain localhost6 localhost6.localdomain6
172.25.254.254 classroom.example.com
172.25.254.254 content.example.com
```

The `getent hosts` *`hostname`* command tests hostname resolution with the `/etc/hosts` file. If an entry is not found in the `/etc/hosts` file, then the stub resolver uses a DNS name server to look up the hostname. The `/etc/resolv.conf` file controls how this query is performed:

- **`search`** : A list of domain names to try with a short hostname.
  Either `search` or `domain` should be set in the same file; if they are both set, then only the last entry takes effect. See `resolv.conf`(5) for details.
- **`nameserver`** : The IP address of a name server to query. Up to three name server directives can be given to provide backups if one name server is down.

```
[root@host ~]# cat /etc/resolv.conf
# Generated by NetworkManager
domain example.com
search example.com
nameserver 172.25.254.254
```

NetworkManager uses DNS settings in the connection configuration files to update the `/etc/resolv.conf` file. Use the `nmcli` command to modify the connections.

```
[root@host ~]# nmcli con mod ID ipv4.dns IP
[root@host ~]# nmcli con down ID
[root@host ~]# nmcli con up ID
[root@host ~]# cat /etc/NetworkManager/system-connections/ID
...output omitted...
[ipv4]
...output omitted...
dns=8.8.8.8;
...output omitted...
```

The default behavior of the `nmcli con mod ID ipv4.dns IP` command is to replace any previous DNS settings with the new IP list that is provided. A plus (+) or minus (-) character in front of the `nmcli` command `ipv4.dns` option adds or removes an individual entry, respectively.

```
[root@host ~]# nmcli con mod ID +ipv4.dns IP
```

In the following example, add the DNS server with an IPv6 IP address of `2001:4860:4860::8888` to the list of name servers on the `static-ens3` connection.

```
[root@host ~]# nmcli con mod static-ens3 +ipv6.dns 2001:4860:4860::8888
```

Note

Static IPv4 and IPv6 DNS settings become `nameserver` directives in `/etc/resolv.conf`. On a dual-stack system, keep listed at least one IPv4-reachable and an IPv6 name server (assuming a dual-stack system), in the event of networking issues with either stack.

Test DNS Name Resolution

The `host` *`HOSTNAME`* command can test DNS server connectivity.

```
[root@host ~]# host classroom.example.com
classroom.example.com has address 172.25.254.254
[root@host ~]# host 172.25.254.254
254.254.25.172.in-addr.arpa domain name pointer classroom.example.com.
```

Important

DHCP automatically rewrites the `/etc/resolv.conf` file when interfaces are started, unless you specify `ignore-auto-dns = yes` in the relevant interface configuration file in the `/etc/NetworkManager/system-connections/` directory.

Set this entry by using the `nmcli` command.

```
[root@host ~]# nmcli con mod "static-ens3" ipv4.ignore-auto-dns yes
```

Use the `dig` *`HOSTNAME`* command to test DNS server connectivity.

```
[root@host ~]# dig classroom.example.com

; <<>> DiG 9.16.23-RH <<>> classroom.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3451
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 13, ADDITIONAL: 27

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 947ea2a936353423c3bc0d5f627cc1ae7147460e10d2777c (good)
;; QUESTION SECTION:
;classroom.example.com.          IN      A

;; ANSWER SECTION:
classroom.example.com.   85326   IN      A       172.25.254.254
...output omitted...
```

Neither the `host` nor the `dig` commands view the configuration in the `/etc/hosts` file. To test the `/etc/hosts` file, use the `getent hosts` *`HOSTNAME`* command.

```
[root@host ~]# getent hosts classroom.example.com
172.25.254.254  classroom.example.com
```

References

`nmcli`(1), `hostnamectl`(1), `hosts`(5), `getent`(1), `host`(1), `dig`(1), `getent`(1), and `resolv.conf`(5) man pages

For more information, refer to the *Configuring and Managing Networking Guide* at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/configuring_and_managing_networking/index

Previous Next

Guided Exercise: Configure Hostnames and Name Resolution

In this exercise, you manually configure the system's static hostname, the `/etc/hosts` file, and the DNS name resolver.

Outcomes

- Set a customized hostname.
- Configure name resolution settings.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that all required resources are available.

```
[student@workstation ~]$ lab start net-hostnames
```

Instructions

1. Log in to `servera` as the `student` user and switch to `root` user.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@testa ~]$ sudo -i
[sudo] password for student: student
```

```
[root@testa ~]#
```

6. View the current hostname settings.
   1. Display the current hostname.

```
2. [root@testa ~]# hostname
```

```
testa
```

   3. Display the hostname status. The persistent, locally configured hostname is displayed in the `Static hostname` field. The current runtime hostname, which is obtained from DHCP or DNS network services, is displayed in the `Transient hostname` field.

```
4. [root@testa ~]# hostnamectl status
5.    Static hostname: servera.lab.example.com
6. Transient hostname: testa
7.          Icon name: computer-vm
8.            Chassis: vm 🖴
9.          Machine ID: ace63d6701c2489ab9c0960c0f1afe1d
10.            Boot ID: 03bf1d5518bd43b4a25cfe9a18d5a46a
11.     Virtualization: kvm
12.   Operating System: Red Hat Enterprise Linux 9.0 (Plow)
13.        CPE OS Name: cpe:/o:redhat:enterprise_linux:9::baseos
14.             Kernel: Linux 5.14.0-70.13.1.el9_0.x86_64
15.       Architecture: x86-64
16.    Hardware Vendor: Red Hat
```

```
    Hardware Model: OpenStack Compute
```

7. Set a static hostname to match the current static hostname.
   1. Change the hostname and the hostname configuration file.

```
2. [root@testa ~]# hostnamectl hostname \
```

```
servera.lab.example.com
```

   3. View the content of the `/etc/hostname` file, which provides the hostname at network start.

```
[root@testa ~]# **cat /etc/hostname**
```

```
servera.lab.example.com
```

5. Log out and log in to `servera` as the `student` user. Switch to the `root` user to change the command prompt to show the updated hostname.

```
[root@testa ~]# exit
logout
[student@testa ~]$ exit
logout
Connection to servera closed.
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera ~]$ sudo -i
[sudo] password for student: student
```

```
[root@servera ~]#
```

15. Display the hostname status. The transient hostname is not shown, now that a static hostname is configured.

```
[root@servera ~]# hostnamectl status
 Static hostname: servera.lab.example.com
       Icon name: computer-vm
         Chassis: vm
      Machine ID: 63b272eae8d5443ca7aaa5593479b25f
         Boot ID: ef299e0e957041ee81d0617fc98ce5ef
  Virtualization: kvm
Operating System: Red Hat Enterprise Linux 9.0 (Plow)
     CPE OS Name: cpe:/o:redhat:enterprise_linux:9::baseos
          Kernel: Linux 5.14.0-70.el9.x86_64
    Architecture: x86-64
 Hardware Vendor: Red Hat
```

```
  Hardware Model: OpenStack Compute
```

8. Temporarily change the hostname to `testname`.
   1. Change the hostname.

      ```
      [root@servera ~]# hostname testname
      ```

   2. Display the current hostname.

   ```
   3. [root@servera ~]# hostname
   ```

      ```
      testname
      ```

   4. View the content of the `/etc/hostname` file, which provides the hostname at network start.

   ```
   5. [root@servera ~]# cat /etc/hostname
   ```

      ```
      servera.lab.example.com
      ```

   6. Reboot the system.

   ```
   7. [root@servera ~]# systemctl reboot
   8. Connection to servera closed by remote host.
   9. Connection to servera closed.
   ```

      ```
      [student@workstation ~]$
      ```

   10. Log in to `servera` as the `student` user and switch to the `root` user.

   ```
   11. [student@workstation ~]$ ssh student@servera
   12. ...output omitted...
   13. [student@servera ~]$ sudo -i
   14. [sudo] password for student: student
   ```

      ```
      [root@servera ~]#
      ```

   15. Display the current hostname.

   ```
   16. [root@servera ~]# hostname
   ```

      ```
      servera.lab.example.com
      ```

9. Add `class` as a local nickname for the classroom server, and ensure that you can ping the server with that nickname.

    1. Look up the IP address of the `classroom.example.com` server.

```
2. [root@servera ~]# host classroom.example.com
```

```
classroom.example.com has address 172.25.254.254
```

3. Update the `/etc/hosts` file to add the `class` server to access the `172.25.254.254` IP address. The following example shows the expected content of the `/etc/hosts` file.

```
4. [root@servera ~]# vim /etc/hosts
5. 127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
6. ::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

```
172.25.254.254 classroom.example.com classroom class
```

7. Look up the IP address of the `class` server.

```
8. [root@servera ~]# host class
9. Host class not found: 3(NXDOMAIN)
10. [root@servera ~]# getent hosts class
```

```
172.25.254.254  classroom.example.com classroom class
```

11. Use the `ping` command to send packets to the `class` server.

```
12. [root@servera ~]# ping -c3 class
13. PING classroom.example.com (172.25.254.254) 56(84) bytes of data.
14. 64 bytes from classroom.example.com (172.25.254.254): icmp_seq=1 ttl=63 time=1.21 ms
15. 64 bytes from classroom.example.com (172.25.254.254): icmp_seq=2 ttl=63 time=0.688 ms
16. 64 bytes from classroom.example.com (172.25.254.254): icmp_seq=3 ttl=63 time=0.559 ms
17.
18. --- classroom.example.com ping statistics ---
19. 3 packets transmitted, 3 received, 0% packet loss, time 2046ms
```

```
rtt min/avg/max/mdev = 0.559/0.820/1.214/0.283 ms
```

20. Return to the `workstation` system as the `student` user.

```
[root@servera ~]# exit
logout
[student@servera ~]$ exit
logout
Connection to servera closed.
```

```
[student@workstation ~]$
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish net-hostnames
```

This concludes the section.

Previous Next

Summary

- The TCP/IP network model is a simplified, four-layered set of abstractions that describes how different protocols interoperate for computers to send traffic from one machine to another over the internet.
- IPv4 is the primary network protocol on the internet today.
- IPv6 is intended as an eventual replacement for the IPv4 network protocol.
- By default, Red Hat Enterprise Linux operates in dual-stack mode, and uses both network protocols in parallel.
- Network routes determine the correct network interface to send packets to a particular network.
- The `NetworkManager` daemon monitors and manages network configuration.
- The `nmcli` command-line tool configures network settings with the `NetworkManager` daemon.

- Starting in Red Hat Enterprise Linux 9, the default location for network configurations is the `/etc/NetworkManager/system-connections` directory.
- The system's static hostname is stored in the `/etc/hostname` file.
- The `hostnamectl` command modifies or views the status of the system's hostname and related settings.

Previous Next