Chapter 6

Controlling Access to Files with Linux File System Permissions

Goal

To set Linux file system permissions on files and interpret the security effects of different permission settings.

Objectives

- Explain how the Linux file permissions model works.
- Change the permissions and ownership of files using command-line tools.
- Configure a directory in which newly created files are automatically writable by members of the group which owns the directory, using special permissions and default umask settings.

Sections

- Linux File System Permissions (and Quiz)
- Managing File System Permissions from the Command Line (and Guided Exercise)
- Managing Default Permissions and File Access (and Guided Exercise)

Lab

Controlling Access to Files with Linux File System Permissions

Linux File System Permissions

Objectives

After completing this section, you should be able to 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 simple but flexible, easy to understand and apply, yet still able to handle most normal permission cases easily.

Files have three user categories to which permissions apply. The file is owned by a user, normally the one who created the file. The file is also owned by a single group, usually the primary group of the user who created the file, but this can be changed. Different permissions can be set for the owning user, the owning group, and for all other users on the system that are not the user or a member of the owning group.

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

In *Figure 6.1*, **`joshua`** is a member of the groups **`joshua`** and **`web`**, and **`allison`** is a member of **`allison`**, **`wheel`**, and **`web`**. When **`joshua`** and **`allison`** need to collaborate, the files should be associated with the group **`web`** and group permissions should allow the desired access.

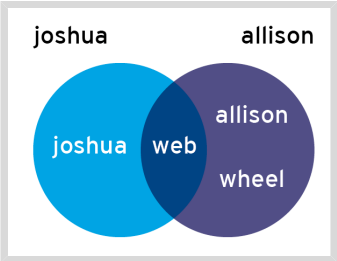

Figure 6.1: Example group membership to facilitate collaboration

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

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.

Permission	Effect on files	Effect on directories
x (execute)	Files can be executed as commands.	The directory can become the current working directory. (You can **cd** into it, but also require read permission to list files found there.)

Users normally have both read and execute permissions on read-only directories so that they can list the directory and have full read-only access to its contents. If a user only has read access on a directory, the names of the files in it can be listed, but no other information, including permissions or time stamps, are available, nor can they be accessed. If a user only has execute access on a directory, they cannot list file names in the directory. If they know the name of a file that they have permission to read, they can access the contents of that file from outside the directory by explicitly specifying the relative file name.

A file may be removed by anyone who has ownership of, or write permission to, the directory in which the file resides, regardless of the ownership or permissions on the file itself. This can be overridden with a special permission, the *sticky bit*, discussed later in this chapter.

Note

Linux file permissions work differently than the permissions system used by the NTFS file system for Microsoft Windows.

On Linux, permissions apply only to the file or directory on which they are set. That is, permissions on a directory are not inherited automatically by the subdirectories and files within it. However, permissions on a directory can block access to the contents of the directory depending on how restrictive they are.

The **read** permission on a directory in Linux is roughly equivalent to **List folder contents** in Windows.

The **write** permission on a directory in Linux is equivalent to **Modify** in Windows; it implies the ability to delete files and subdirectories. In Linux, if **write** and the **sticky bit** are both set on a directory, then only the file or subdirectory owner may delete it, which is similar to the Windows **Write** permission behavior.

The Linux root user has the equivalent of the Windows **Full Control** permission on all files. However, root may have access restricted by the system's SELinux policy using process and file security contexts. SELinux will be discussed in a later course.

Viewing File and Directory Permissions and Ownership

The **-l** option of the **ls** command shows detailed information about permissions and ownership:

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

Use the **-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 Jan 31 22:00 /home
```

The first character of the long listing is the file type, interpreted like this:

- **-** is a regular file.
- **d** is a directory.
- **l** is a soft link.
- Other characters represent hardware devices (**b** and **c**) or other special-purpose files (**p** and **s**).

The next nine characters are the file permissions. These are in three sets of three characters: permissions that apply to the user that owns the file, the group that owns the file, and all other users. If the set shows **rwx**, that category has all three permissions, read, write, and execute. If a letter has been replaced by **-**, then that category does not have that permission.

After the link count, the first name specifies the user that owns the file, and the second name the group that owns the file.

So in the example above, the permissions for user **student** are specified by the first set of three characters. User **student** has read and write on **test**, but not execute.

Group **student** is specified by the second set of three characters: it also has read and write on **test**, but not execute.

Any other user's permissions are specified by the third set of three characters: they only have read permission on **test**.

The most specific set of permissions apply. So if user **student** has different permissions than group **student**, and user **student** is also a member of that group, then the user permissions will be the ones that apply.

Examples of Permission Effects

The following examples will help illustrate how file permissions interact. For these examples, we have four users with the following group memberships:

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

Those users will be working with files in the **dir** directory. This is a long listing of the files in that directory:

```
[database1@host dir]$ ls -la
total 24
drwxrwxr-x.  2 database1 consultant1   4096 Apr  4 10:23 .
drwxr-xr-x. 10 root      root          4096 Apr  1 17:34 ..
-rw-rw-r--.  1 operator1 operator1     1024 Apr  4 11:02 lfile1
-rw-r--rw-.  1 operator1 consultant1   3144 Apr  4 11:02 lfile2
-rw-rw-r--.  1 database1 consultant1  10234 Apr  4 10:14 rfile1
-rw-r-----.  1 database1 consultant1   2048 Apr  4 10:18 rfile2
```

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

What are the permissions of **rfile1**? The user that owns the file (**database1**) has read and write but not execute. The group that owns the file (**consultant1**) has read and write but not execute. All other users have read but not write or execute.

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

Effect	Why is this true?
The user **operator1** can change the contents of **rfile1**.	User **operator1** is a member of the **consultant1** group, and that group has both read and write permissions on **rfile1**.
The user **database1** can view and modify the contents of **rfile2**.	User **database1** owns the file and has both read and write access to **rfile2**.
The user **operator1** can view but not modify the contents of **rfile2** (without deleting it and recreating it).	User **operator1** is a member of the **consultant1** group, and that group only has read access to **rfile2**.
The users **database2** and **contractor1** do not have any access to the contents of **rfile2**.	**other** permissions apply to users **database2** and **contractor1**, and those permissions do not include read or write permission.
operator1 is the only user who can change the contents of **lfile1** (without deleting it and recreating it).	User and group **operator1** have write permission on the file, other users do not. But the only member of group **operator1** is user **operator1**.
The user **database2** can change the contents of **lfile2**.	User **database2** is not the user that owns the file and is not in group **consultant1**, so **other** permissions apply. Those grant write permission.
The user **database1** can view the contents of **lfile2**, but cannot modify the contents of **lfile2** (without deleting it and recreating it).	User **database1** is a member of the group **consultant1**, and that group only has read permissions on **lfile2**. Even though **other** has write permission, the group permissions take precedence.
The user **database1** can delete **lfile1** and **lfile2**.	User **database1** has write permissions on the directory containing both files (shown by **.**), and therefore can delete any file in that directory. This is true even if **database1** does not have write permission on the file itself.

References

ls(1) man page

info coreutils (*GNU Coreutils*)

- Section 13: Changing file attributes

Quiz

Interpreting File and Directory Permissions

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

The system has four users assigned to the following groups:

- User **`consultant1`** is in groups **`consultant1`** and **`database1`**
- User **`operator1`** is in groups **`operator1`** and **`database1`**
- User **`contractor1`** is in groups **`contractor1`** and **`contractor3`**
- User **`operator2`** is in groups **`operator2`** and **`contractor3`**

The current directory (`.`) contains four files with the following permissions information:

```
drwxrwxr-x.   operator1     database1     .
-rw-rw-r--.   consultant1   consultant1   lfile1
-rw-r--rw-.   consultant1   database1     lfile2
-rw-rw-r--.   operator1     database1     rfile1
-rw-r-----.   operator1     database1     rfile2
```

1. Which regular file is owned by `operator1` and readable by all users?

a. **`lfile1`**
b. **`lfile2`**
c. **`rfile1`**
d. **`rfile2`**

2. Which file can be modified by the `contractor1` user?

a. **`lfile1`**
b. **`lfile2`**
c. **`rfile1`**
d. **`rfile2`**

3. Which file cannot be read by the `operator2` user?

a. **`lfile1`**
b. **`lfile2`**
c. **`rfile1`**
d. **`rfile2`**

4. **Which file has a group ownership of `consultant1`?**
   a. **`lfile1`**
   b. **`lfile2`**
   c. **`rfile1`**
   d. **`rfile2`**

5. **Which files can be deleted by the `operator1` user?**
   a. **`rfile1`**
   b. **`rfile2`**
   c. All of the above.
   d. None of the above.

6. **Which files can be deleted by the `operator2` user?**
   a. **`lfile1`**
   b. **`lfile2`**
   c. All of the above.
   d. None of the above.

Solution

Interpreting File and Directory Permissions

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

The system has four users assigned to the following groups:

- User **`consultant1`** is in groups **`consultant1`** and **`database1`**
- User **`operator1`** is in groups **`operator1`** and **`database1`**
- User **`contractor1`** is in groups **`contractor1`** and **`contractor3`**
- User **`operator2`** is in groups **`operator2`** and **`contractor3`**

The current directory (`.`) contains four files with the following permissions information:

```
drwxrwxr-x.   operator1     database1     .
-rw-rw-r--.   consultant1    consultant1   lfile1
-rw-r--rw-.   consultant1    database1     lfile2
-rw-rw-r--.   operator1     database1     rfile1
-rw-r-----.   operator1     database1     rfile2
```

1. Which regular file is owned by `operator1` and readable by all users?

a. `lfile1`
b. `lfile2`
c. **`rfile1`**
d. `rfile2`

2. Which file can be modified by the `contractor1` user?

a. `lfile1`
b. **`lfile2`**
c. `rfile1`
d. `rfile2`

3. Which file cannot be read by the `operator2` user?

a. `lfile1`
b. `lfile2`
c. `rfile1`
d. **`rfile2`**

4. Which file has a group ownership of `consultant1`?

a. **`lfile1`**
b. `lfile2`
c. `rfile1`
d. `rfile2`

5. Which files can be deleted by the `operator1` user?

a. `rfile1`
b. `rfile2`
c. All of the above.
d. None of the above.

6. Which files can be deleted by the `operator2` user?

a. `lfile1`
b. `lfile2`
c. All of the above.
d. None of the above.

Managing File System Permissions from the Command Line

Objectives

After completing this section, you should be able to change the permissions and ownership of files using command-line tools.

Changing File and Directory Permissions

The command used to change permissions from the command line is **chmod**, which means "change mode" (permissions are also called the *mode* of a file). The **chmod** command takes a permission instruction followed by a list of files or directories to change. The permission instruction can be issued either symbolically (the symbolic method) or numerically (the numeric method).

Changing Permissions with the Symbolic Method

```
chmod WhoWhatWhich file|directory
```

- *Who* is u, g, o, a *(for user, group, other, all)*
- *What* is +, -, = *(for add, remove, set exactly)*
- *Which* is r, w, x *(for read, write, execute)*

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

With the symbolic method, it is not necessary to set a complete new group of permissions. Instead, you can change one or more of the existing permissions. Use **+** or **-** to add or remove permissions, respectively, or use **=** to replace the entire set for a group of permissions.

The permissions themselves are represented by a single letter: **r** for read, **w** for write, and **x** for execute. When using **chmod** to change permissions with the symbolic method, using a capital **X** as the permission flag will add execute permission only if the file is a directory or already has execute set for user, group, or other.

Note

The **chmod** command supports the **-R** option to recursively set permissions on the files in an entire directory tree. When using the **-R** option, it can be useful to set permissions symbolically using the **X** option. This allows the execute (search) permission to be set on directories so that their contents can be accessed, without changing permissions on most files. Be cautious with the **X** option, however, because if a file has any execute permission set, **X** will set the specified execute permission on that file as well. For example, the following command recursively sets read and write access on **demodir** and all its children for their group owner, but only applies group execute permissions to directories and files that already have execute set for user, group, or other.

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

Examples

- Remove read and write permission for group and other on **file1**:

```
[user@host ~]$ chmod go-rw file1
```

- Add execute permission for everyone on **file2**:

```
[user@host ~]$ chmod a+x file2
```

Changing Permissions with the Numeric Method

In the example below the # character represents a digit.

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

- Each digit represents permissions for an access level: user, group, other.
- The digit is calculated by adding together numbers for each permission you want to add, 4 for read, 2 for write, and 1 for execute.

Using the *numeric* method, permissions are represented by 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 (numeric) representation of permissions, each digit stands for one access level, from left to right: user, group, and other. To determine each digit:

1. Start with 0.
2. If the read permission should be present for this access level, add 4.
3. If the write permission should be present, add 2.
4. If the execute permission should be present, add 1.

Examine the permissions **-rwxr-x---**. For the user, **rwx** is calculated as 4+2+1=7. For the group, **r-x** is calculated as 4+0+1=5, and for other users, **---** is represented with 0. Putting these three together, the numeric representation of those permissions is 750.

This calculation can also be performed in the opposite direction. Look at the permissions 640. For the user permissions, 6 represents read (4) and write (2), which displays as **rw-**. For the group

part, 4 only includes read (4) and displays as **r--**. The 0 for other provides no permissions (**---**) and the final set of symbolic permissions for this file is **-rw-r-----**.

Experienced administrators often use numeric permissions because they are shorter to type and pronounce, while still giving full control over all permissions.

Examples

- Set read and write permissions for user, read permission for group and other, on **samplefile**:

```
[user@host ~]$ chmod 644 samplefile
```

- Set read, write, and execute permissions for user, read and execute permissions for group, and no permission for other on **sampledir**:

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

Changing File and Directory User or Group Ownership

A newly created file is owned by the user who creates that file. By default, new files have a group ownership that is the primary group of the user creating 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, the group that owns the file may need to be changed.

Only **root** can change the user that owns a file. Group ownership, however, can be set by **root** or by the file's owner. **root** can grant file ownership to any group, but regular users can make a group the owner of a file only if they are a member of that group.

File ownership can be changed with the **chown** (change owner) command. For example, to grant ownership of the **test_file** file to the **student** user, use the following command:

```
[root@host ~]# chown student test_file
```

chown can be used with the **-R** option to recursively change the ownership of an entire directory tree. The following command grants ownership of **test_dir** and all files and subdirectories within it to **student**:

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

The **chown** command can also be used to change group ownership of a file by preceding the group name with a colon (:). For example, the following command changes the group **test_dir** to **admins**:

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

The **chown** command can also be used to change both owner and group at the same time by using the *owner:group* syntax. For example, to change the ownership of **test_dir** to **visitor** and the group to **guests**, use the following command:

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

Instead of using **chown**, some users change the group ownership by using the **chgrp** command. This command works just like **chown**, except that it is only used to change group ownership and the colon (:) before the group name is not required.

Important

You may encounter examples of **chown** commands using an alternative syntax that separates owner and group with a period instead of a colon:

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

You should not use this syntax. Always use a colon.

A period is a valid character in a user name, but a colon is not. If the user **enoch.root**, the user **enoch**, and the group **root** exist on the system, the result of **chown enoch.root filename** will be to have **filename** owned by the user **enoch.root**. You may have been trying to set the file ownership to the user **enoch** and group **root**. This can be confusing.

If you always use the **chown** colon syntax when setting the user and group at the same time, the results are always easy to predict.

References

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

Guided Exercise

Managing File Security from the Command Line

In this exercise, you will create a collaborative directory for pre-existing users.

Outcomes

- Create a directory with permissions that make it accessible by all members of the **ateam** group.
- Create a file owned by user **andy** that can be modified by **alice**.

Before You Begin

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

Steps

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

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

2. Create a group, **ateam**. Create two new users, **andy** and **alice**, who are members of that group.

```
[root@ip-192-0-2-1 ~]# groupadd ateam
[root@ip-192-0-2-1 ~]# useradd -G ateam andy
[root@ip-192-0-2-1 ~]# useradd -G ateam alice
[root@ip-192-0-2-1 ~]# id andy; id alice
uid=1010(andy) gid=1010(andy) groups=1010(andy),40001(ateam)
uid=1011(alice) gid=1011(alice) groups=1011(alice),40001(ateam)
```

3. Create a directory in **/home** called **ateam-text**.

```
[root@ip-192-0-2-1 ~]# mkdir /home/ateam-text
```

4. Change the group ownership of the **ateam-text** directory to **ateam**.

```
[root@ip-192-0-2-1 ~]# chown :ateam /home/ateam-text
```

5. Ensure the permissions of **ateam-text** allows group members to create and delete files.

```
[root@ip-192-0-2-1 ~]# chmod g+w /home/ateam-text
```

6. Ensure the permissions of **ateam-text** forbids others from accessing its files.

```
[root@ip-192-0-2-1 ~]# chmod 770 /home/ateam-text
[root@ip-192-0-2-1 ~]$ ls -ld /home/ateam-text
drwxrwx---.  2 root ateam 6 Jul 24 12:50 /home/ateam-text
```

7. Exit the root shell and switch to the user **andy**.

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

8. Navigate to the **/home/ateam-text** folder (remember to open a terminal window first).

```
[andy@ip-192-0-2-1 ~]$ cd /home/ateam-text
```

9. Create an empty file called **andyfile3**.

```
[andy@ip-192-0-2-1 ateam-text]$ touch andyfile3
```

10. Record the default user and group ownership of the new file and its permissions.

```
[andy@ip-192-0-2-1 ateam-text]$ ls -l andyfile3
-rw-rw-r--.  1 andy andy 0 Jul 24 12:59 andyfile3
```

11. Change the group ownership of the new file to **ateam** and record the new ownership and permissions.

```
[andy@ip-192-0-2-1 ateam-text]$ chown :ateam andyfile3
[andy@ip-192-0-2-1 ateam-text]$ ls -l andyfile3
-rw-rw-r--.  1 andy ateam 0 Jul 24 12:59 andyfile3
```

12. Exit the shell and switch to the user **alice**.

```
[andy@ip-192-0-2-1 ateam-text]$ exit
[ec2-user@ip-192-0-2-1 ~]$ sudo su - alice
[alice@ip-192-0-2-1 ~]$
```

13. Navigate to the **/home/ateam-text** folder.

```
[alice@ip-192-0-2-1 ~]$ cd /home/ateam-text
```

14. Determine **alice**'s privileges to access and/or modify **andyfile3**.

```
[alice@ip-192-0-2-1 ateam-text]$ echo "text" >> andyfile3
[alice@ip-192-0-2-1 ateam-text]$ cat andyfile3
text
```

If you didn't set the permissions correctly, the above commands will instead result in something like the following:

```
[alice@ip-192-0-2-1 ateam-text]$ echo "text" >> andyfile3
-bash: /home/ateam-text/andyfile3: Permission denied
```

Important

In the preceding example, the command **echo "text" >> andyfile3** is using a technique called *shell I/O redirection* to append the line **text** to the end of the file **andyfile3**. The output of the **echo** command is appended to the end of the file that the **>>** points at. Be careful to use **>>** and not just one **>**, since the **>** operator will overwrite the entire file and replace its contents with only the line **text**.

If you are interested in more information on shell I/O redirection, an overview is available at http://wiki.bash-hackers.org/syntax/redirection.

Be careful if you choose to test this by having **alice** edit **andyfile3** with **vim**. If the permissions are wrong on the file, **vim** will warn you that you're editing a read-only file. But if the **/home/ateam-test** *directory* is writable by **alice**, then **vim** will still let you use **:wq!** to write the file, even if **alice** doen't have write permission on the file!

How is this possible? It turns out **vim** is "smart" enough to recognize that it can overwrite the file by deleting the file from the directory and creating a new copy. This is allowed because having write on a directory means you can delete any file in that directory, even if you can't write the file directly. The **!** on the **vim** command **:wq!** indicates that you want it to do everything it can to write the file.

Note that one side effect of this is that the file's owner will change to **alice**, and other permissions may change as well.

15. This completes this exercise. Log out and stop your Amazon EC2 instance.

Managing Default Permissions and File Access

Objectives

After completing this section, students should be able to:

- Control the default permissions of new files created by users.
- Explain the effect of special permissions.
- Use special permissions and default permissions to set the group owner of files created in a particular directory.

Special Permissions

Special permissions constitute a fourth permission type in addition to the basic user, group, and other types. As the name implies, these permissions provide additional access-related features over and above what the basic permission types allow. This section details the impact of special permissions, summarized in the table below.

Effects of Special Permissions on Files and Directories

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

The *setuid* permission on an executable file means that commands run as the user owning the file, not 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** where you would normally expect the **x** (owner execute permissions) to be. If the owner does not have execute permissions, this is replaced by an uppercase **S**.

The special permission *setgid* on a directory means that files created in the directory inherit their group ownership from the directory, rather than inheriting it from the creating user. This is commonly used on group collaborative directories to automatically change a file from the default

private group to the shared group, or if files in a directory should be always owned by a specific group. An example of this 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, commands run as the group that owns that file, not as the user that ran the command, in a similar way to 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** where you would normally expect the **x** (group execute permissions) to be. If the group does not have execute permissions, this is replaced by an uppercase **S**.

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

```
[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** where you would normally expect the **x** (other execute permissions) to be. If other does not have execute permissions, this is replaced by an uppercase **T**.

Setting Special Permissions

- Symbolically: setuid = **`u+s`**; setgid = **`g+s`**; sticky = **`o+t`**
- Numerically (fourth preceding digit): setuid = 4; setgid = 2; sticky = 1

Examples

- Add the setgid bit on **`directory`**:

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

- Set the setgid bit and add read/write/execute permissions for user and group, with no access for others, on **`directory`**:

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

Default File Permissions

When you create a new file or directory, it is assigned initial permissions. There are two things that affect these initial permissions. The first is whether you are creating a regular file or a directory. The second is the current *umask*.

If you create a new directory, the operating system starts by assigning it octal permissions 0777 (**`drwxrwxrwx`**). If you create a new regular file, the operating system assignes it octal permissions 0666 (**`-rw-rw-rw-`**). You always have to explicitly add execute permission to a regular file. This

makes it harder for an attacker to compromise a network service so that it creates a new file and immediately executes it as a program.

However, the shell session will also set a umask to further restrict the permissions that are initially set. This is an octal bitmask used to clear the permissions of new files and directories created by a process. 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 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 will display the current value of the shell's umask:

```
[user@host ~]$ umask
0002
```

Use the **umask** command with a single numeric argument to change the umask of the current shell. The numeric argument should be an octal value corresponding to the new umask value. You can omit any leading zeros in the umask.

The system's default umask values for Bash shell users are defined in the **/etc/profile** and **/etc/bashrc** files. Users can override the system defaults in the **.bash_profile** and **.bashrc** files in their home directories.

umask Example

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. The owner and group both have read and write permission on files, and other is set to read. The owner and group both have read, write, and execute permissions on directories. The only permission for other is read.

```
[user@host ~]$ umask
0002
[user@host ~]$ touch default
[user@host ~]$ ls -l default.txt
-rw-rw-r--. 1 user user 0 May  9 01:54 default.txt
[user@host ~]$ mkdir default
[user@host ~]$ ls -ld default
drwxrwxr-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 changes 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 write access 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
```

The default umask for users is set by the shell startup scripts. By default, if your account's UID is 200 or more and your username and primary group name are the same, you will be assigned a umask of 002. Otherwise, your umask will be 022.

As **root**, you can change this by adding a shell startup script named **/etc/profile.d/local-umask.sh** that looks something like the output in this example:

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

The preceding example will set the umask to 007 for users with a UID greater than 199 and with a username and primary group name that match, and to 022 for everyone else. If you just wanted to set the umask for everyone to 022, you could create that file with just the following content:

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

To ensure that global umask changes take effect you must log out of the shell and log back in. Until that time the umask configured in the current shell is still in effect.

References

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

Guided Exercise

Controlling New File Permissions and Ownership

In this exercise, you will control default permissions on new files using the **umask** command and **setgid** permission.

Outcomes

- Create a shared directory where new files are automatically owned by the group **ateam**.
- Experiment with various umask settings.
- Adjust default permissions for specific users.
- Confirm your adjustment is correct.

Before You Begin

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

It is also assumed that you have completed the steps from the preceding exercise in this chapter. The users **alice** and **andy** should exist on your system. The primary group for both users should be a user private group with the same name as the user's username. Both users should also be members of the group **ateam**.

Steps

1. Use **sudo** to switch user to **alice**.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo su - alice
Last login: Fri Jul 24 17:01:55 EDT 2020 on pts/0
[alice@ip-192-0-2-1 ~]$
```

2. Use the **umask** command without arguments to display Alice's default umask value.

```
[alice@ip-192-0-2-1 ~]$ umask
0002
```

3. Create a new directory **/tmp/shared** and a new file **/tmp/shared/defaults** to see how the default umask affects permissions.

```
[alice@ip-192-0-2-1 ~]$ mkdir /tmp/shared
[alice@ip-192-0-2-1 ~]$ ls -ld /tmp/shared
drwxrwxr-x. 2 alice alice 6 Jul 24 18:43 /tmp/shared
[alice@ip-192-0-2-1 ~]$ touch /tmp/shared/defaults
[alice@ip-192-0-2-1 ~]$ ls -l /tmp/shared/defaults
-rw-rw-r--. 1 alice alice 0 Jul 24 18:43 /tmp/shared/defaults
```

4. Change the group ownership of **/tmp/shared** to **ateam** and record the new ownership and permissions.

```
[alice@ip-192-0-2-1 ~]$ chown :ateam /tmp/shared
[alice@ip-192-0-2-1 ~]$ ls -ld /tmp/shared
drwxrwxr-x. 2 alice ateam 21 Jul 24 18:43 /tmp/shared
```

5. Create a new file in **/tmp/shared** and record the ownership and permissions.

```
[alice@ip-192-0-2-1 ~]$ touch /tmp/shared/alice3
[alice@ip-192-0-2-1 ~]$ ls -l /tmp/shared/alice3
-rw-rw-r--. 1 alice alice 0 Jul 24 18:46 /tmp/shared/alice3
```

6. Ensure the permissions of **/tmp/shared** cause files created in that directory to inherit the group ownership of **ateam**.

```
[alice@ip-192-0-2-1 ~]$ chmod g+s /tmp/shared
[alice@ip-192-0-2-1 ~]$ ls -ld /tmp/shared
drwxrwsr-x. 2 alice ateam 34 Jul 24 18:46 /tmp/shared
[alice@ip-192-0-2-1 ~]$ touch /tmp/shared/alice4
[alice@ip-192-0-2-1 ~]$ ls -l /tmp/shared
total 0
-rw-rw-r--. 1 alice alice 0 Jul 24 18:46 alice3
-rw-rw-r--. 1 alice ateam 0 Jul 24 18:48 alice4
-rw-rw-r--. 1 alice alice 0 Jul 24 18:43 defaults
```

7. Change the umask for **alice** such that new files are created with read-only access for the group and no access for other users. Create a new file and record the ownership and permissions.

```
[alice@ip-192-0-2-1 ~]$ umask 027
[alice@ip-192-0-2-1 ~]$ touch /tmp/shared/alice5
[alice@ip-192-0-2-1 ~]$ ls -l /tmp/shared
total 0
-rw-rw-r--. 1 alice alice 0 Jul 24 18:46 alice3
-rw-rw-r--. 1 alice ateam 0 Jul 24 18:48 alice4
-rw-r-----. 1 alice ateam 0 Jul 24 18:50 alice5
-rw-rw-r--. 1 alice alice 0 Jul 24 18:43 defaults
```

8. Log out and log back in again as **alice**, starting a new shell, and view her umask.

```
[alice@ip-192-0-2-1 ~]$ umask
0027
[alice@ip-192-0-2-1 ~]$ exit
logout
[ec2-user@ip-192-0-2-1 ~]$ sudo su - alice
Last login: Fri Jul 24 18:31:25 EDT 2020 on pts/0
[alice@ip-192-0-2-1 ~]$ umask
0002
```

Note that **alice**'s umask reverted to her default settings.

9. Change the default umask for **alice** to prohibit all access to "other" on files the user creates, by appending **umask 007** to the end of the **~/.bashrc** file.

```
[alice@ip-192-0-2-1 ~]$ echo "umask 007" >> ~/.bashrc
[alice@ip-192-0-2-1 ~]$ cat ~/.bashrc
# .bashrc

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

# User specific environment
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
umask 007
```

Important

Rather than using redirection, you could instead simply edit the file with the command **vim ~/.bashrc**, and add **umask 007** as the last line of the file, as shown in the output of **cat ~/.bashrc** from the example.

If you are interested in more in-depth information on shell I/O redirection, an overview is available at http://wiki.bash-hackers.org/syntax/redirection.

10. Log out of **alice**'s **su** session, and then log back into **alice**'s account and confirm that the umask changes you made are persistent.

```
[alice@ip-192-0-2-1 ~]$ exit
logout
[ec2-user@ip-192-0-2-1 ~]$ sudo su - alice
Last login: Fri Jul 24 18:54:02 EDT 2020 on pts/0
[alice@ip-192-0-2-1 ~]$ umask
0007
```

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

Lab

Controlling Access to Files with Linux File System Permissions

In this lab, you will configure a directory that users in the same group can use to easily share files that are automatically writable by the entire group.

Outcomes

- A directory called **/home/operators** where the members of group **operators** can work collaboratively on files.
- Only the **root** user and **operators** group can access, create, and delete files in this directory.
- Files created in this directory should automatically be assigned a group ownership of **operators**.
- New files created in this directory will not be accessible to normal users who are not in the group or own the files.

Before You Begin

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

Steps

1. Use **sudo** to get an interactive shell as **root**.
2. Create a new group named **operators**.
3. Create three users, **yasmine**, **louis**, and **liza**. The primary group for each of these users should be a user private group that has the same name as their username. They should also be members of group **operators**.

   You may set passwords for these users so that you can use commands like **su - yasmine** to switch between the users to test your work, or you may leave the accounts locked with invalid passwords and switch between them from the **ec2-user** account with **sudo**.
4. Create the **/home/operators** directory.
5. Change group permissions on the **/home/operators** directory so that it belongs to the **operators** group.
6. Set permissions on the **/home/operators** directory so it is set GID, the owner and group have full read/write/execute permissions, and other users have no permissions on the directory.
7. Check that the permissions were set properly.
8. When you finish, check your work to ensure you have done everything correctly.
9. This concludes this lab. Log out and stop your Amazon EC2 instance.

Solution

Controlling Access to Files with Linux File System Permissions

In this lab, you will configure a directory that users in the same group can use to easily share files that are automatically writable by the entire group.

Outcomes

- A directory called **/home/operators** where the members of group **operators** can work collaboratively on files.
- Only the **root** user and **operators** group can access, create, and delete files in this directory.
- Files created in this directory should automatically be assigned a group ownership of **operators**.
- New files created in this directory will not be accessible to normal users who are not in the group or own the files.

Before You Begin

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

Steps

1. Use **sudo** to get an interactive shell as **root**.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo su -
Last login: Fri Jul 24 20:57:08 EDT 2020 on pts/0
[root@ip-192-0-2-1 ~]#
```

2. Create a new group named **operators**.

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

3. Create three users, **yasmine**, **louis**, and **liza**. The primary group for each of these users should be a user private group that has the same name as their username. They should also be members of group **operators**.

   You may set passwords for these users so that you can use commands like **su - yasmine** to switch between the users to test your work, or you may leave the accounts locked with invalid passwords and switch between them from the **ec2-user** account with **sudo**.

```
[root@ip-192-0-2-1 ~]# useradd -G operators yasmine
[root@ip-192-0-2-1 ~]# useradd -G operators louis
[root@ip-192-0-2-1 ~]# useradd -G operators liza
```

4. Create the **/home/operators** directory.

```
[root@ip-192-0-2-1 ~]# mkdir /home/operators
```

5. Change group permissions on the **`/home/operators`** directory so that it belongs to the **`operators`** group.

```
[root@ip-192-0-2-1 ~]# chown :operators /home/operators
```

6. Set permissions on the **`/home/operators`** directory so it is set GID, the owner and group have full read/write/execute permissions, and other users have no permissions on the directory.

```
[root@ip-192-0-2-1 ~]# chmod 2770 /home/operators
```

7. Check that the permissions were set properly.

```
[root@ip-192-0-2-1 ~]# ls -ld /home/operators
drwxrws---. 2 root operators 6 Jul 24 11:38 /home/operators
```

8. When you finish, check your work to ensure you have done everything correctly.

```
[root@ip-192-0-2-1 ~]# exit
logout
[ec2-user@ip-192-0-2-1 ~]$ touch /home/operators/ec2-user-test
touch: cannot touch '/home/operators/ec2-user-test': Permission denied
[ec2-user@ip-192-0-2-1 ~]$ sudo su - yasmine
[yasmine@ip-192-0-2-1 ~]$ touch /home/operators/yasmine-test
[yasmine@ip-192-0-2-1 ~]$ ls -l /home/operators
total 0
-rw-rw-r--. 1 yasmine operators 0 Jul 24 11:47 yasmine-test
[yasmine@ip-192-0-2-1 ~]$ exit
logout
[ec2-user@ip-192-0-2-1 ~]$ sudo su - louis
[louis@ip-192-0-2-1 ~]$ touch /home/operators/louis-test
[louis@ip-192-0-2-1 ~]$ ls -l /home/operators
total 0
-rw-rw-r--. 1 louis   operators 0 Jul 24 11:48 louis-test
-rw-rw-r--. 1 yasmine operators 0 Jul 24 11:47 yasmine-test
[louis@ip-192-0-2-1 ~]$ exit
logout
[ec2-user@ip-192-0-2-1 ~]$
```

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

Chapter 8

Installing and Updating Software Packages

Goal

To download, install, update, and manage software packages from Red Hat and YUM package repositories.

Objectives

- Explain what an RPM package is and how RPM packages are used to manage software on a Red Hat Enterprise Linux system.
- Find, install, and update software packages using the **`yum`** command.

Sections

- RPM Software Packages and Yum (and Quiz)
- Managing Software Updates with Yum (and Guided Exercise)

RPM Software Packages and Yum

Objectives

After completing this section, you should be able to explain how software is provided as RPM packages, and investigate the packages installed on the system with Yum and RPM.

Software packages and RPM

The RPM Package Manager, originally developed by Red Hat, provides a standard way to package software for distribution. Managing software in the form of *RPM packages* is much simpler than working with software that has simply been extracted into a file system from an archive. It lets administrators track which files were installed by the software package and which ones need to be removed if it is uninstalled, and check to ensure that supporting packages are present when it is installed. Information about installed packages is stored in a local RPM database on each system. All software provided by Red Hat for Red Hat Enterprise Linux is provided as an RPM package.

RPM package files names consist of four elements (plus the **`.rpm`** suffix): **`name-version-release.architecture`**:

Figure 8.1: RPM file name elements

- NAME is one or more words describing the contents (coreutils).
- VERSION is the version number of the original software (8.30).
- RELEASE is the release number of the package based on that version, and is set by the packager, who might not be the original software developer (4.el8).
- ARCH is the processor architecture the package was compiled to run on. **`noarch`** indicates that this package's contents are not architecture-specific (as opposed to **`x86_64`** for 64-bit, **`aarch64`** for 64-bit ARM, and so on).

Only the package name is required for installing packages from repositories. If multiple versions exist, the package with the higher version number is installed. If multiple releases of a single version exist, the package with the higher release number is installed.

Each RPM package is a special archive made up of three components:

- The files installed by the package.
- Information about the package (metadata), such as the name, version, release, and arch; a summary and description of the package; whether it requires other packages to be installed; licensing; a package change log; and other details.

- Scripts that may run when this package is installed, updated, or removed, or are triggered when other packages are installed, updated, or removed.

Typically, software providers digitally sign RPM packages using GPG keys (Red Hat digitally signs all packages it releases). The RPM system verifies package integrity by confirming that the package was signed by the appropriate GPG key. The RPM system refuses to install a package if the GPG signature does not match.

Updating Software with RPM Packages

Red Hat generates a complete RPM package to update software. An administrator installing that package gets only the most recent version of the package. Red Hat does not require that older packages be installed and then patched. To update software, RPM removes the older version of the package and installs the new version. Updates usually retain configuration files, but the packager of the new version defines the exact behavior.

In most cases, only one version or release of a package may be installed at a time. However, if a package is built so that there are no conflicting file names, then multiple versions may be installed. The most important example of this is the **`kernel`** package. Since a new kernel can only be tested by booting to that kernel, the package is specifically designed so that multiple versions may be installed at once. If the new kernel fails to boot, the old kernel is still available and bootable.

References

`rpm`(8) man page

Quiz

RPM Software Packages

Choose the correct answer to the following questions:

1. Which portion of an RPM references the version of the upstream course code?

a. Repository
b. Version
c. Release
d. Changelog

2. Which portion of an RPM lists the reasons for each package build?

a. Errata
b. Version
c. Release
d. Changelog

3. Which portion of an RPM references the version of the package build?

a. Errata
b. Version
c. Release
d. Changelog

4. Which portion of an RPM references the processor type required for a specific package?

a. Architecture
b. Release
c. Version
d. Errata

5. Which term is used to descibe a collection of RPM packages and package groups?

a. Software Collection
b. Distribution
c. Repository
d. Changelog

6. Which term is used to verify the source and integrity of a package?

a. Release
b. GPG signature
c. Repository Certificate
d. Changelog

Solution

RPM Software Packages

Choose the correct answer to the following questions:

1. **Which portion of an RPM references the version of the upstream course code?**
   a. Repository
   b. Version
   c. Release
   d. Changelog

2. **Which portion of an RPM lists the reasons for each package build?**
   a. Errata
   b. Version
   c. Release
   d. Changelog

3. **Which portion of an RPM references the version of the package build?**
   a. Errata
   b. Version
   c. Release
   d. Changelog

4. **Which portion of an RPM references the processor type required for a specific package?**
   a. Architecture
   b. Release
   c. Version
   d. Errata

5. **Which term is used to descibe a collection of RPM packages and package groups?**
   a. Software Collection
   b. Distribution
   c. Repository
   d. Changelog

6. **Which term is used to verify the source and integrity of a package?**
   a. Release
   b. GPG signature
   c. Repository Certificate
   d. Changelog

Managing Software Updates with Yum

Objectives

After completing this section, you should be able to find, install, and update software packages, using the **yum** command.

Managing Software Packages with Yum

The low-level **rpm** command can be used to install packages, but it is not designed to work with package repositories or resolve dependencies from multiple sources automatically.

Yum is designed to be a better system for managing RPM-based software installation and updates. The **yum** command allows you to install, update, remove, and get information about software packages and their dependencies. You can get a history of transactions performed and work with multiple Red Hat and third-party software repositories.

Finding Software with Yum

- **yum help** displays usage information.
- **yum list** displays installed and available packages.

```
[user@host ~]$ yum list 'http*'
Available Packages
http-parser.i686              2.8.0-2.el8                      rhel8-appstream
http-parser.x86_64            2.8.0-2.el8                      rhel8-appstream
httpcomponents-client.noarch  4.5.5-4.module+el8+2452+b359bfcd  rhel8-appstream
httpcomponents-core.noarch    4.4.10-3.module+el8+2452+b359bfcd  rhel8-appstream
httpd.x86_64                  2.4.37-7.module+el8+2443+605475b7  rhel8-appstream
httpd-devel.x86_64            2.4.37-7.module+el8+2443+605475b7  rhel8-appstream
httpd-filesystem.noarch       2.4.37-7.module+el8+2443+605475b7  rhel8-appstream
httpd-manual.noarch           2.4.37-7.module+el8+2443+605475b7  rhel8-appstream
httpd-tools.x86_64            2.4.37-7.module+el8+2443+605475b7  rhel8-appstream
```

- **yum search *KEYWORD*** lists packages by keywords found in the name and summary fields only.

  To search for packages that have "web server" in their name, summary, and description fields, use **search all**:

```
[user@host ~]$ yum search all 'web server'
================= Summary & Description Matched: web server ====================
pcp-pmda-weblog.x86_64 : Performance Co-Pilot (PCP) metrics from web server logs
nginx.x86_64 : A high performance web server and reverse proxy server
======================== Summary Matched: web server ===========================
libcurl.x86_64 : A library for getting files from web servers
libcurl.i686 : A library for getting files from web servers
libcurl.x86_64 : A library for getting files from web servers
====================== Description Matched: web server =========================
httpd.x86_64 : Apache HTTP Server
```

```
git-instaweb.x86_64 : Repository browser in gitweb
...output omitted...
```

- **`yum info`** ***`PACKAGENAME`*** returns detailed information about a package, including the disk space needed for installation.

  To get information on the Apache HTTP Server:

```
[user@host ~]$ yum info httpd
Available Packages
Name         : httpd
Version      : 2.4.37
Release      : 7.module+el8+2443+605475b7
Arch         : x86_64
Size         : 1.4 M
Source       : httpd-2.4.37-7.module+el8+2443+605475b7.src.rpm
Repo         : rhel8-appstream
Summary      : Apache HTTP Server
URL          : https://httpd.apache.org/
License      : ASL 2.0
Description  : The Apache HTTP Server is a powerful, efficient, and extensible
             : web server.
```

- **`yum provides`** ***`PATHNAME`*** displays packages that match the path name specified (which often include wildcard characters).

  To find packages that provide the **`/var/www/html`** directory, use:

```
[user@host ~]$ yum provides /var/www/html
httpd-filesystem-2.4.37-7.module+el8+2443+605475b7.noarch : The basic directory
 layout for the Apache HTTP server
Repo        : rhel8-appstream
Matched from:
Filename    : /var/www/html
```

Installing and removing software with `yum`

- **`yum install`** ***`PACKAGENAME`*** obtains and installs a software package, including any dependencies.

```
[user@host ~]$ yum install httpd
Dependencies resolved.
================================================================================
 Package                  Arch       Version             Repository        Size
================================================================================
Installing:
 httpd                    x86_64     2.4.37-7.module...  rhel8-appstream   1.4 M
Installing dependencies:
 apr                      x86_64     1.6.3-8.el8         rhel8-appstream   125 k
 apr-util                 x86_64     1.6.1-6.el8         rhel8-appstream   105 k
...output omitted...
Transaction Summary
================================================================================
Install  9 Packages
```

```
Total download size: 2.0 M
Installed size: 5.4 M
Is this ok [y/N]: y
Downloading Packages:
(1/9): apr-util-bdb-1.6.1-6.el8.x86_64.rpm          464 kB/s |  25 kB     00:00
(2/9): apr-1.6.3-8.el8.x86_64.rpm                   1.9 MB/s | 125 kB     00:00
(3/9): apr-util-1.6.1-6.el8.x86_64.rpm              1.3 MB/s | 105 kB     00:00
...output omitted...
Total                                               8.6 MB/s | 2.0 MB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : apr-1.6.3-8.el8.x86_64                                 1/9
  Running scriptlet: apr-1.6.3-8.el8.x86_64                                 1/9
  Installing       : apr-util-bdb-1.6.1-6.el8.x86_64                        2/9
...output omitted...
Installed:
  httpd-2.4.37-7.module+el8+2443+605475b7.x86_64 apr-util-bdb-1.6.1-6.el8.x86_64
  apr-util-openssl-1.6.1-6.el8.x86_64            apr-1.6.3-8.el8.x86_64
...output omitted...
Complete!
```

- **`yum update`** ***`PACKAGENAME`*** obtains and installs a newer version of the specified package, including any dependencies. Generally the process tries to preserve configuration files in place, but in some cases, they may be renamed if the packager thinks the old one will not work after the update. With no PACKAGENAME specified, it installs all relevant updates.

```
[user@host ~]$ sudo yum update
```

Since a new kernel can only be tested by booting to that kernel, the package is specifically designed so that multiple versions may be installed at once. If the new kernel fails to boot, the old kernel is still available. Using **`yum update kernel`** will actually *install* the new kernel. The configuration files hold a list of packages to *always install* even if the administrator requests an update.

Note

Use **yum list kernel** to list all installed and available kernels. To view the currently running kernel, use the **uname** command. The **-r** option only shows the kernel version and release, and the **-a** option shows the kernel release and additional information.

```
[user@host ~]$ yum list kernel
Installed Packages
kernel.x86_64          4.18.0-60.el8          @anaconda
kernel.x86_64          4.18.0-67.el8          @rhel-8-for-x86_64-baseos-htb-rpms
[user@host ~]$ uname -r
4.18.0-60.el8.x86_64
[user@host ~]$ uname -a
Linux host.lab.example.com 4.18.0-60.el8.x86_64 #1 SMP Fri Jan 11 19:08:11 UTC
 2019 x86_64 x86_64 x86_64 GNU/Linux
```

- **yum remove *PACKAGENAME*** removes an installed software package, including any supported packages.

```
[user@host ~]$ sudo yum remove httpd
```

Warning

The **yum remove** command removes the packages listed *and any package that requires the packages being removed* (and packages which require those packages, and so on). This can lead to unexpected removal of packages, so carefully review the list of packages to be removed.

Installing and removing groups of software with yum

- **yum** also has the concept of *groups*, which are collections of related software installed together for a particular purpose. In Red Hat Enterprise Linux 8, there are two kinds of groups. Regular groups are collections of packages. *Environment groups* are collections of regular groups. The packages or groups provided by a group may be **mandatory** (they must be installed if the group is installed), **default** (normally installed if the group is installed), or **optional** (not installed when the group is installed, unless specifically requested).

  Like **yum list**, the **yum group list** command shows the names of installed and available groups.

```
[user@host ~]$ yum group list
Available Environment Groups:
   Server with GUI
   Minimal Install
   Server
...output omitted...
Available Groups:
   Container Management
   .NET Core Development
```

```
   RPM Development Tools
...output omitted...
```

Some groups are normally installed through environment groups and are hidden by default. List these hidden groups with the **`yum group list hidden`** command.

- **`yum group info`** displays information about a group. It includes a list of mandatory, default, and optional package names.

```
[user@host ~]$ yum group info "RPM Development Tools"
Group: RPM Development Tools
 Description: These tools include core development tools such rpmbuild.
  Mandatory Packages:
    redhat-rpm-config
    rpm-build
  Default Packages:
    rpmdevtools
  Optional Packages:
    rpmlint
```

- **`yum group install`** installs a group that installs its mandatory and default packages and the packages they depend on.

```
[user@host ~]$ sudo yum group install "RPM Development Tools"
...output omitted...
Installing Groups:
 RPM Development Tools

 Transaction Summary
 ================================================================================
 Install  64 Packages

 Total download size: 21 M
 Installed size: 62 M
 Is this ok [y/N]: y
...output omitted...
```

Important

The behavior of Yum groups changed starting in Red Hat Enterprise Linux 7. In RHEL 7 and later, groups are treated as *objects*, and are tracked by the system. If an installed group is updated, and new mandatory or default packages have been added to the group by the Yum repository, those new packages are installed upon update.

RHEL 6 and earlier consider a group to be installed if all its mandatory packages have been installed, or if it had no mandatory packages, or if any default or optional packages in the group are installed. Starting in RHEL 7, a group is considered to be installed *only* if **`yum group install`** was used to install it. The command **`yum group mark install GROUPNAME`** can be used to mark a group as installed, and any missing packages and their dependencies are installed upon the next update.

Finally, RHEL 6 and earlier did not have the two-word form of the **`yum group`** commands. In other words, in RHEL 6 the command **`yum grouplist`** existed, but the equivalent RHEL 7 and RHEL 8 command **`yum group list`** did not.

Viewing transaction history

- All install and remove transactions are logged in **`/var/log/dnf.rpm.log`**.

```
[user@host ~]$ tail -n 5 /var/log/dnf.rpm.log
2019-02-26T18:27:00Z SUBDEBUG Installed: rpm-build-4.14.2-9.el8.x86_64
2019-02-26T18:27:01Z SUBDEBUG Installed: rpm-build-4.14.2-9.el8.x86_64
2019-02-26T18:27:01Z SUBDEBUG Installed: rpmdevtools-8.10-7.el8.noarch
2019-02-26T18:27:01Z SUBDEBUG Installed: rpmdevtools-8.10-7.el8.noarch
2019-02-26T18:38:40Z INFO --- logging initialized ---
```

- **`yum history`** displays a summary of install and remove transactions.

```
[user@host ~]$ sudo yum history
ID     | Command line             | Date and time    | Action(s)      | Altered
-------------------------------------------------------------------------------
     7 | group install RPM Develo | 2019-02-26 13:26 | Install        |   65
     6 | update kernel            | 2019-02-26 11:41 | Install        |    4
     5 | install httpd            | 2019-02-25 14:31 | Install        |    9
     4 | -y install @base firewal | 2019-02-04 11:27 | Install        |  127 EE
     3 | -C -y remove firewalld - | 2019-01-16 13:12 | Removed        |   11 EE
     2 | -C -y remove linux-firmw | 2019-01-16 13:12 | Removed        |    1
     1 |                          | 2019-01-16 13:05 | Install        |  447 EE
```

- The **`history undo`** option reverses a transaction.

```
[user@host ~]$ sudo yum history undo 5
Undoing transaction 7, from Tue 26 Feb 2019 10:40:32 AM EST
    Install apr-1.6.3-8.el8.x86_64                              @rhel8-appstream
    Install apr-util-1.6.1-6.el8.x86_64                         @rhel8-appstream
    Install apr-util-bdb-1.6.1-6.el8.x86_64                     @rhel8-appstream
    Install apr-util-openssl-1.6.1-6.el8.x86_64                 @rhel8-appstream
    Install httpd-2.4.37-7.module+el8+2443+605475b7.x86_64      @rhel8-appstream
...output omitted...
```

Summary of Yum Commands

Packages can be located, installed, updated, and removed by name or by package groups.

Task:	Command:
List installed and available packages by name	`yum list [NAME-PATTERN]`
List installed and available groups	`yum group list`
Search for a package by keyword	`yum search KEYWORD`
Show details of a package	`yum info PACKAGENAME`
Install a package	`yum install PACKAGENAME`
Install a package group	`yum group install GROUPNAME`
Update all packages	`yum update`
Remove a package	`yum remove PACKAGENAME`
Display transaction history	`yum history`

References

yum(1) and **yum.conf**(5) man pages

For more information, refer to the *Installing software packages* chapter in the *Red Hat Enterprise Linux 8 Configuring basic system settings* guide at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/configuring_basic_system_settings/index#installing-software-packages_managing-software-packages

Guided Exercise

Installing and Updating Software with Yum

In this exercise, you will install and remove software packages and package groups.

Outcomes

- Install and remove packages which have dependencies.

Before You Begin

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

It is also assumed that the AMI that you are using for your Amazon EC2 instance has been pre-configured with a Red Hat Enterprise Linux subscription and can get software packages and updates from the Red Hat Update Infrastructure (RHUI) in AWS. This should be the case for all official AMIs as discussed at https://aws.amazon.com/partners/redhat/faqs/.

Steps

1. Attempt to run the **gnuplot** command.

   1.1. Attempt to run the **gnuplot** command. You should find that it is not installed.

```
[ec2-user@ip-192-0-2-1 ~]$ gnuplot
-bash: gnuplot: command not found
```

2. Search for the package that provides **gnuplot**.

   2.1. Get an interactive shell as the **root** user.

```
[ec2-user@ip-192-0-2-1 ~]$ sudo su -
Last login: Fri Jul 24 15:42:55 EDT 2020 on pts/0
[root@ip-192-0-2-1 ~]#
```

   2.2. Search for plotting packages.

```
[root@ip-192-0-2-1 ~]# yum search plot
Last metadata expiration check: 0:51:39 ago on Tue 28 Jul 2020 07:00:56 AM UTC.
========================= Name & Summary Matched: plot =========================
gnuplot-common.x86_64 : The common gnuplot parts
texlive-pst-plot.noarch : Plot data using PSTricks
gnuplot.x86_64 : A program for plotting mathematical expressions and data
[root@ip-192-0-2-1 ~]#
```

   2.3. Find out more information about the *gnuplot* package.

```
[root@ip-192-0-2-1 ~]# yum info gnuplot
Last metadata expiration check: 0:51:53 ago on Tue 28 Jul 2020 07:00:56 AM UTC.
Available Packages
Name         : gnuplot
Version      : 5.2.4
Release      : 1.el8
Architecture : x86_64
Size         : 892 k
Source       : gnuplot-5.2.4-1.el8.src.rpm
Repository   : rhel-8-for-x86_64-appstream-rpms
Summary      : A program for plotting mathematical expressions and data
URL          : http://www.gnuplot.info/
License      : gnuplot and MIT
Description  : Gnuplot is a command-line driven, interactive function plotting
             : program especially suited for scientific data representation.
             : Gnuplot can be used to plot functions and data points in both
             : two and three dimensions and in many different formats.
             :
             : Install gnuplot if you need a graphics package for scientific
             : data representation.
             :
             : This package provides a Qt based terminal version of gnuplot.

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

3. Install the *gnuplot* package.

```
[root@ip-192-0-2-1 ~]# yum install gnuplot
Last metadata expiration check: 0:53:39 ago on Tue 28 Jul 2020 07:00:56 AM UTC.
Dependencies resolved.
================================================================================
 Package          Arch   Version        Repository                       Size
================================================================================
Installing:
 gnuplot          x86_64 5.2.4-1.el8    rhel-8-for-x86_64-appstream-rpms 892 k
Installing dependencies:
 avahi-libs       x86_64 0.7-19.el8     rhel-8-for-x86_64-baseos-rpms     63 k
 cairo            x86_64 1.15.12-3.el8  rhel-8-for-x86_64-appstream-rpms 721 k
...output omitted...
Transaction Summary
================================================================================
Install  59 Packages

Total download size: 28 M
Installed size: 88 M
Is this ok [y/N]: y
Downloading Packages:
(1/59): fontpackages-filesystem-1.44-22.el8.noa  39 kB/s |  16 kB     00:00
(2/59): dejavu-fonts-common-2.35-6.el8.noarch.r 157 kB/s |  74 kB     00:00
...output omitted...
  Installing       : libjpeg-turbo-1.5.3-10.el8.x86_64                    1/59
  Installing       : libX11-xcb-1.6.8-3.el8.x86_64                        2/59
  Installing       : mesa-libglapi-19.3.4-2.el8.x86_64                    3/59
```

```
...output omitted...
  Verifying        : fontpackages-filesystem-1.44-22.el8.noarch            1/59
  Verifying        : dejavu-fonts-common-2.35-6.el8.noarch                 2/59
  Verifying        : dejavu-sans-fonts-2.35-6.el8.noarch                   3/59
...output omitted...
Installed:
  avahi-libs-0.7-19.el8.x86_64
  cairo-1.15.12-3.el8.x86_64
  cups-libs-1:2.2.6-33.el8.x86_64
...output omitted...
Complete!
[root@ip-192-0-2-1 ~]#
```

4. Test the **gnuplot** command.

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

 G N U P L O T
 Version 5.2 patchlevel 4    last modified 2018-06-01

 Copyright (C) 1986-1993, 1998, 2004, 2007-2018
 Thomas Williams, Colin Kelley and many others

 gnuplot home:     http://www.gnuplot.info
 faq, bugs, etc:   type "help FAQ"
 immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> quit
[root@ip-192-0-2-1 ~]#
```

5. Test the **yum remove** command.

5.1. Use the **yum remove** command to remove the *gnuplot* package, but respond with **no** when prompted. How many packages would be removed?

```
[root@ip-192-0-2-1 ~]# yum remove gnuplot
Dependencies resolved.
================================================================================
 Package          Arch   Version       Repository                         Size
================================================================================
Removing:
 gnuplot          x86_64 5.2.4-1.el8   @rhel-8-for-x86_64-appstream-rpms 2.1 M
Removing unused dependencies:
 avahi-libs       x86_64 0.7-19.el8    @rhel-8-for-x86_64-baseos-rpms    159 k
 cairo            x86_64 1.15.12-3.el8 @rhel-8-for-x86_64-appstream-rpms 1.8 M
...output omitted...
Transaction Summary
================================================================================
Remove  59 Packages

Freed space: 88 M
```

```
Is this ok [y/N]: n
Operation aborted.
[root@ip-192-0-2-1 ~]#
```

5.2. Use the **`yum remove`** command to remove the *gnuplot-common* package, but respond with **no** when prompted. How many packages would be removed?

```
[root@ip-192-0-2-1 ~]# yum remove gnuplot-common
Dependencies resolved.
================================================================================
 Package          Arch   Version       Repository                        Size
================================================================================
Removing:
 gnuplot-common   x86_64 5.2.4-1.el8   @rhel-8-for-x86_64-appstream-rpms 1.7 M
Removing dependent packages:
 gnuplot          x86_64 5.2.4-1.el8   @rhel-8-for-x86_64-appstream-rpms 2.1 M
Removing unused dependencies:
 avahi-libs       x86_64 0.7-19.el8    @rhel-8-for-x86_64-baseos-rpms    159 k
 cairo            x86_64 1.15.12-3.el8 @rhel-8-for-x86_64-appstream-rpms 1.8 M
...output omitted...
Transaction Summary
================================================================================
Remove  59 Packages

Freed space: 88 M
Is this ok [y/N]: n
Operation aborted.
[root@ip-192-0-2-1 ~]#
```

6. Gather information about the "RPM Development Tools" component group and install it.

6.1. Use the **`yum group list`** command to list all available component groups.

```
[root@ip-192-0-2-1 ~]# yum group list
Last metadata expiration check: 0:55:45 ago on Tue 28 Jul 2020 07:00:56 AM UTC.
Available Environment Groups:
   Server with GUI
   Server
   Minimal Install
   Workstation
   Virtualization Host
   Custom Operating System
Available Groups:
   RPM Development Tools
   Container Management
   .NET Core Development
   Graphical Administration Tools
   Network Servers
   System Tools
   Scientific Support
   Smart Card Support
   Headless Management
   Development Tools
```

```
   Legacy UNIX Compatibility
   Security Tools
[root@ip-192-0-2-1 ~]#
```

6.2. Use the **`yum group info`** command to find out more information about the *RPM Development Tools* component group, including a list of included packages.

```
[root@ip-192-0-2-1 ~]# yum group info "RPM Development Tools"
Last metadata expiration check: 0:56:48 ago on Tue 28 Jul 2020 07:00:56 AM UTC.

Group: RPM Development Tools
 Description: Tools used for building RPMs, such as rpmbuild.
 Mandatory Packages:
   redhat-rpm-config
   rpm-build
 Default Packages:
   rpmdevtools
 Optional Packages:
   rpmlint
[root@ip-192-0-2-1 ~]#
```

6.3. Use the **`yum group install`** command to install the *RPM Development Tools* component group.

```
[root@ip-192-0-2-1 ~]# yum group install "RPM Development Tools"
Last metadata expiration check: 0:57:17 ago on Tue 28 Jul 2020 07:00:56 AM UTC.
Dependencies resolved.
================================================================================
 Package           Arch   Version        Repository                       Size
================================================================================
Installing group/module packages:
 redhat-rpm-config noarch 122-1.el8      rhel-8-for-x86_64-appstream-rpms  83 k
 rpm-build         x86_64 4.14.2-37.el8  rhel-8-for-x86_64-appstream-rpms 171 k
 rpmdevtools       noarch 8.10-7.el8     rhel-8-for-x86_64-appstream-rpms  87 k
Installing dependencies:
 binutils          x86_64 2.30-73.el8    rhel-8-for-x86_64-baseos-rpms    5.7 M
 bzip2             x86_64 1.0.6-26.el8   rhel-8-for-x86_64-baseos-rpms     60 k
...output omitted...
Transaction Summary
================================================================================
Install  74 Packages

Total download size: 28 M
Installed size: 90 M
Is this ok [y/N]: y
Downloading Packages:
(1/75): perl-Scalar-List-Utils-1.49-2.el8.x86_6 200 kB/s |  68 kB     00:00
(2/75): perl-PathTools-3.74-1.el8.x86_64.rpm    249 kB/s |  90 kB     00:00
...output omitted...
  Installing       : perl-Carp-1.42-396.el8.noarch                       1/75
  Installing       : perl-Exporter-5.72-396.el8.noarch                   2/75
  Installing       : perl-libs-4:5.26.3-416.el8.x86_64                   3/75
...output omitted...
  Verifying        : perl-Scalar-List-Utils-3:1.49-2.el8.x86_64          1/75
```

```
  Verifying        : perl-PathTools-3.74-1.el8.x86_64                    2/75
  Verifying        : perl-Data-Dumper-2.167-399.el8.x86_64               3/75
...output omitted...
Installed:
  binutils-2.30-73.el8.x86_64
  bzip2-1.0.6-26.el8.x86_64
  dwz-0.12-9.el8.x86_64
...output omitted...
Complete!
[root@ip-192-0-2-1 ~]#
```

7. Explore the **`history`** options of **`yum`**.

7.1. Use the **`yum history`** command to display recent **`yum`** history.

```
[root@ip-192-0-2-1 ~]# yum history
ID     | Command line             | Date and time    | Action(s)      | Altered
-------------------------------------------------------------------------------
     6 | group install RPM Develo | 2020-07-28 07:59 | Install        |   75
     5 | install gnuplot          | 2020-07-28 07:55 | Install        |   59
     4 | -y install vim-enhanced  | 2020-07-28 07:06 | Install        |    4
...output omitted...
[root@ip-192-0-2-1 ~]#
```

7.2. Use the **`yum history info`** command to confirm that the last transaction is the group installation.

```
[root@ip-192-0-2-1 ~]# yum history info 6
Transaction ID : 6
Begin time     : Tue 28 Jul 2020 07:59:12 AM UTC
Begin rpmdb    : 467:81427725b5ac0fd4c0adc6f0b8f799f6fe1315b8
End time       : Tue 28 Jul 2020 07:59:20 AM UTC (8 seconds)
End rpmdb      : 541:da6e11d22bd61700281e7a237e0d18e412b9d9f9
User           : Cloud User <ec2-user>
Return-Code    : Success
Releasever     : 8
Command Line   : group install RPM Development Tools
Packages Altered:
    Install rust-srpm-macros-5-2.el8.noarch         @rhel-8-appstream-rhui-rpms
    Install perl-URI-1.73-3.el8.noarch              @rhel-8-appstream-rhui-rpms
    Install perl-Net-SSLeay-1.88-1.el8.x86_64       @rhel-8-appstream-rhui-rpms
...output omitted...
[root@ip-192-0-2-1 ~]#
```

Important

Note that the transaction ID number (**6** above) may be different for you on your system. Use the most recent ID number that appears in the output of the **`yum history`** command.

7.3. Use the **`yum history undo`** command to remove the set of packages that were installed when the **`gnuplot`** package was installed.

```
[root@ip-192-0-2-1 ~]# yum history undo 5
Last metadata expiration check: 1:01:57 ago on Tue 28 Jul 2020 07:00:56 AM UTC.
Undoing transaction 5, from Tue 28 Jul 2020 07:55:18 AM UTC
    Install libXxf86vm-1.1.4-9.el8.x86_64           @rhel-8-appstream-rhui-rpms
    Install libSM-1.2.3-1.el8.x86_64                @rhel-8-appstream-rhui-rpms
...output omitted...

Transaction Summary
================================================================================
Remove  59 Packages

Freed space: 88 M
Is this ok [y/N]: y
...output omitted...
Complete!
[root@ip-192-0-2-1 ~]#
```

8. Use the **yum update** command to make sure all packages on the system are up to date.

```
[root@ip-192-0-2-1 ~]# yum update
Last metadata expiration check: 1:03:51 ago on Tue 28 Jul 2020 07:00:56 AM UTC.
Dependencies resolved.
================================================================================
 Package              Arch   Version           Repository                  Size
================================================================================
Installing:
 kernel               x86_64 4.18.0-193.13.2.el8_2
                                                rhel-8-baseos-rhui-rpms    2.8 M
 kernel-core          x86_64 4.18.0-193.13.2.el8_2
                                                rhel-8-baseos-rhui-rpms     28 M
 kernel-modules       x86_64 4.18.0-193.13.2.el8_2
                                                rhel-8-baseos-rhui-rpms     24 M
Upgrading:
 NetworkManager       x86_64 1:1.22.8-5.el8_2  rhel-8-baseos-rhui-rpms    2.3 M
 NetworkManager-libnm x86_64 1:1.22.8-5.el8_2  rhel-8-baseos-rhui-rpms    1.7 M
...output omitted...

Transaction Summary
================================================================================
Install   5 Packages
Upgrade  53 Packages

Total download size: 183 M
```

```
Is this ok [y/N]: y
...output omitted...
Complete!
[root@ip-192-0-2-1 ~]#
```

Important

There may be a delay of a few minutes during the cleanup and verification phase of the package update depending on how many packages are installed and what they are. Be patient if this occurs.

Note that the list of packages may be larger or smaller and different than portrayed in the preceding example, depending on the version of your instance's AMI and any additional software updates that become available after this course is published.

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

Important

This also concludes the final exercise of this course. Remember to terminate your Amazon EC2 instance to remove it entirely from the Amazon EC2 cloud and cloud storage once you are done using it.

Thank you for your participation in this course.