Part Workbook 4. File Ownerships and Permissions

Table of Contents

Chapter 1. Regular File Ownerships and Permissions

Key Concepts

- Files have a user owner, a group owner, and a set of permissions.
- Three permission types: (r)ead, (w)rite, and e(x)ecute
- Three access classes: (u)ser owner, (g)roup owner, and (o)ther
- Viewing ownerships and permissions: **ls -l**
- Modifying ownerships and permissions: **chmod**, **chgrp**, and **chown**

Discussion

User Owners, Group Owners, and Permissions

Linux is a multiuser environment, so many different people are often working with a single set of files and directories at the same time. Some of these files are intended to be shared, so that many people can view and edit the file. For example, someone might be listing suggested gifts they would like to receive for an upcoming birthday. The more people who are able to read the file, the better. Other files, the user might want to keep private. Few people would want anyone else to be able to read their diary. Someone might want a file to be readable to everyone on the system, and allow a few select users on the system to modify the file as well.

Every file in Linux has three properties that allows users to control who can access the file and how: a user owner, a group owner, and a set of allowed permissions. When a user creates a file, that user becomes the file's user owner, and (usually) the primary group of the user becomes the file's group owner. A user cannot change the user owner of a file, but a user has some abilities to change the group owner. The permission bits define how three different classes of users can use the file: the file's owner, members of the group that owns the file, and everyone else.

Consider the following long listing of the `/var/gigs` directory.

Figure 1.1. long listing of the `/var/gigs` directory.

```
[student@station student]$ ls -l /var/gigs/
total 28
-rw-rw-r--    1 elvis    elvis         129 Jan 13 14:48 atlanta
-rw-r--r--    1 blondie  music         142 Jan 13 14:46 boston
-rw-rw-r--    1 elvis    music         143 Jan 13 14:48 chicago
-rwxr-x---    1 root     music          55 Jan 13 15:25 generate_report.sh
-rw-rw-r--    1 root     music        2057 Jan 13 14:47 los_angeles
-rw-rw-r--    1 elvis    music         142 Jan 13 14:47 san_francisco
-rw-rw-r--    1 blondie  blondie       135 Jan 13 14:47 springfield
```

The last column is the name of the file, while the third and fourth columns are the user and group owners of the file, respectively. The following sections discuss how to use the the first column to determine the permissions of the file.

The Three Permission Types: (r)ead, (w)rite, and e(x)ecute

When deciding who can access a file, a user can distinguish between three types of permissions. Someone can view a file if they have read permissions, but must have write permissions in order to modify the file. Execute permissions allow someone to use the file as a command. In order to start up an application or run a script, the file containing the application or script must be executable. Normal (data) files do not use the executable permission type.

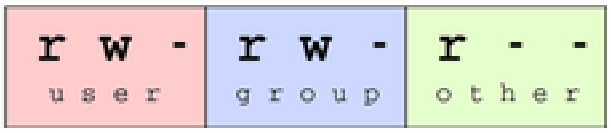
Table 1.1. Regular File Permissions

(r)ead	(w)rite	e(x)ecute
view the file	modify the file	use the file as a command

Three Access Classes: (u)ser owner, (g)roup owner, and (o)ther.

Each file has a set of read, write, and execute permissions for three different classes of file access. The first set is used for the owner of a file. The second set is used for members of the group that owns the file, and the last set is used for everyone who doesn't fit into the first two groups. A file's permissions are usually shown using a series of nine characters, such as `rwxr-xr-x`. The first three letters represent the "user" permissions, the next three the "group" permissions, and the last three the "other" permission. A letter indicates that the corresponding permission is enabled, while a disabled permission is represented by a dash.

Figure 1.2. Permissions for three classes of file access



When someone tries to access a file, Linux asks the following questions in order:

1. Does the user own the file? If so, the user permissions are used.
2. Is the user a member of the group that owns the file? If so, the group permissions are used.
3. Otherwise, the other permissions are used.

Examples

Newly created files

Consider the following verse that elvis is composing:

```
[elvis@station elvis]$ echo "There once was a man from Peru," > /tmp/limerick
[elvis@station elvis]$ ls -l /tmp/limerick
-rw-rw-r--    1 elvis    elvis          32 Jan 14 13:42 /tmp/limerick
```

Note that, by default, elvis owns the file, and as the file owner, he has read and write permissions. Users other than elvis can only read the file. (Because Red Hat Enterprise Linux uses the User Private Group scheme, the user elvis is also the only member of the group elvis.) Because elvis has writable permissions, he can modify the file by appending the next line of the verse:

```
[elvis@station elvis]$ echo "who told limericks of lines only two." >> /tmp/lime
rick
```

Other users, such as blondie, however, are only able to read the file. When blondie tries to overwrite the file with her own verse, she finds that she does not have appropriate permissions.

```
[blondie@station blondie]$ cat /tmp/limerick
There once was a man from Peru,
who told limericks of lines only two.
[blondie@station blondie]$ echo "There once was a man from Verdun." > /tmp/limer
ick
-bash: /tmp/limerick: Permission denied
```

Group shared files

Consider the following users, and their respective group memberships:

user	groups
blondie	blondie,music
elvis	elvis,music,wrestle,physics,emperors
hogan	hogan,wrestle
bob	bob

The following is a listing of files in the /tmp directory:

```
[elvis@station elvis]$ ls -l /tmp
total 28
-rw-------    1 bob      bob           136 Jan 14 15:58 diary
-rw-rw-r--    1 elvis    music         142 Jan 14 15:58 gigs
-rw-rw-r--    1 elvis    elvis          70 Jan 14 13:42 limerick
-rw-rw----    1 blondie  blondie       134 Jan 14 15:59 lyrics
-rw-------    1 blondie  blondie        29 Jan 14 16:00 lyrics.old
-rw-r-----    1 hogan    wrestle       146 Jan 14 15:58 routines
```

❶ Bob's `diary` is only readable and writable by Bob.

❷ Elvis owns the file `gigs`, but it is readable and writable by anyone in the group music. Any musician can add (or remove) an event to the schedule, and anyone can read the file to find out when the musicians are playing.

❸ In contrast, blondie has been working on `lyrics` which she is not yet ready to share. Although the file is group writable, it is owned by the group blondie. Because Red Hat implements the User Private Group scheme, the user blondie should be the only member of the group blondie. The permissions on `lyrics` have the same effect as those on `lyrics.old`

❹ Hulk Hogan has been working on `routines` for an upcoming wrestling match. So as not to catch any other wrestlers by surprise, he has left the file readable by all members of the group wrestle. Note that other wrestlers can read the file, but not change any of the routines. People who are not members of the group wrestlers have no access to the file.

Executable files

Files which are to be interpreted as commands need to have executable permissions.

```
[elvis@station elvis]$ ls -l /bin/ls /usr/bin/firefox
-rwxr-xr-x    2 root     root        67884 Sep  2 07:21 /bin/ls
-rwxr-xr-x    1 root     root         5607 Oct  3 13:58 /usr/bin/firefox
```

The `ls` command is executable by anyone. Note that read permission is not necessary to execute the command, but allows users to view the (often binary) content of the file. Readable permissions for `firefox`, for example, allow observant users to realize that the command is really a browseable text script.

Symbolic Links

While symbolic links have permissions, the permissions are always `rwxrwxrwx`. Access decisions "fall through" to the file to which the symbolic link refers.

```
[elvis@station elvis]$ ls -l /bin/view /bin/vi
-rwxr-xr-x    1 root     root       451628 Aug 27 19:09 /bin/vi
lrwxrwxrwx    1 root     root            2 Sep 11 11:32 /bin/view -> vi
```

The command `view`, which is really a symbolic link, would effectively have the same permissions as vi: `rwxr-xr-x`.

Online Exercises

Explore permissions on newly created files

Lab Exercise

Objective: Examine default behavior of newly created files

Estimated Time:10 minutes

1. Create a subdirectory in the `/tmp` with the same name as your username. For example, if your username is *elvis*, create the directory `/tmp/elvis`. This directory will hereafter be referred to as `/tmp/username`.

2. Compose a short list of new year's resolutions in your preferred text editor, or simply from the command line. Store the file in your newly created directory, as `/tmp/username/resolutions.txt`.

```
[student@station student]$ nano /tmp/student/resolutions.txt
... (compose and save your text) ...
[student@station student]$ cat /tmp/student/resolutions.txt
keep room clean
don't eat all of the pop tarts
use less proprietary software
[student@station student]$ ls -l /tmp/student/resolutions.txt
-rw-rw-r--    1 student  student        77 Jan 15 07:12 /tmp/student/resolutions.txt
```

3. Become one of your alternate user accounts. You may either login again from another virtual console, from a network connection, or simply **su -** to the alternate user.

```
[student@station student]$ su - student_a
Password:
[student_a@station student_a]$
```

4. As the alternate user, confirm that you can view the file. Try adding an new item to the list. Why can you not modify the file as the alternate user?

```
[student_a@station student_a]$ cat /tmp/student/resolutions.txt
keep room clean
don't eat all of the pop tarts
use less proprietary software
[student_a@station student_a]$ echo "lose weight" >> /tmp/student/resolutions.txt
```

```
-bash: /tmp/student/resolutions.txt: Permission denied
```

Deliverables

1.
   1. A newly created file titled `/tmp/username/resolutions.txt`, with default permissions.

Questions

Analyzing File Permissions

Objective: Determine read and write accessibility for shared files.

Use the following users and group memberships for this exercise.

user	**subscribed groups**
elvis	elvis,music,wrestle,physics,emperors
ventura	ventura,wrestle,governor
prince	prince,music
pataki	pataki,governor
einstein	einstein,physics
alice	alice, games, mail

The following questions refer to the following files, found in a `/tmp` directory:

```
[elvis@station elvis]$ ls -l /tmp
total 32
-rw-rw-r--    1 alice    alice         44 Jan 15 10:31 adventures
-rw-rw----    1 pataki   governor      29 Jan 15 10:30 budget
-rw-rw-r--    1 nero     emperors     142 Jan 15 10:30 census
-rw-rw-rw-    1 elvis    elvis         29 Jan 15 10:33 pacman.highscores
-rw-r-----    1 einstein physics      352 Jan 15 10:30 theory
```

1. Which users can read einstein's file `theory`?
   a. elvis
   b. ventura
   c. prince
   d. pataki
   e. einstein
   f. alice
2. Which users can modify the `census` taken for nero?
   a. elvis
   b. ventura
   c. prince

   d. pataki
   e. einstein
   f. alice

3. Which users can add their own entry to elvis's `pacman.highscores`?
   a. elvis
   b. ventura
   c. prince
   d. pataki
   e. einstein
   f. alice

4. Which users can read governor pataki's `budget`?
   a. elvis
   b. ventura
   c. prince
   d. pataki
   e. einstein
   f. alice

5. Which users can read alice's `adventures`?
   a. elvis
   b. ventura
   c. prince
   d. pataki
   e. einstein
   f. alice

6. Which users can modify alice's `adventures`?
   a. elvis
   b. ventura
   c. prince
   d. pataki
   e. einstein

f. alice

Chapter 2. Changing File Permissions: chmod

Key Concepts

- The **chmod** command is used to modify file permissions
- The first argument to **chmod** uses a [ugoa]+/-[rwx] syntax to describe how the permissions should be changed

Discussion

In the previous lesson, we learned that files have three types of permissions ((r)ead, (w)rite, and e(x)ecute) and three classes of access ((u)ser, (g)roup, and (o)ther) that define how the file can be used. These permissions are managed by the **chmod** command. In Linux, the permissions of a file are often referred to as the "mode" of the file. The name **chmod** is a shortcut for **ch**ange **mod**e.

For example, when someone creates a file, by default the file can only be modified by that person, but is readable by everyone on the system. Suppose elvis were working on the lyrics for a new song, that he wanted no one else to read until he was finished. After creating the file, he would need to use the **chmod** command to disallow others from reading the file.

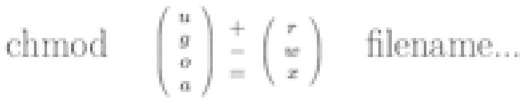
```
[elvis@station elvis]$ echo "Well, it's one for the honey," > blue_suede
[elvis@station elvis]$ echo "Two for the snow," >> blue_suede
[elvis@station elvis]$ ls -l blue_suede
-rw-rw-r--    1 elvis    elvis          48 Jan 16 08:09 blue_suede
[elvis@station elvis]$ chmod o-r blue_suede
[elvis@station elvis]$ ls -l blue_suede

-rw-rw----    1 elvis    elvis          48 Jan 16 08:09 blue_suede
```

Note that in the first use of the **ls -l** command, the file was readable by others. After the **chmod** command, however, the **ls -l** shows that others no longer can read the file. Once elvis has settled on the wording, he can now restore readability for others to the file, again using the **chmod** command.

```
[elvis@station elvis]$ echo "Well, it's one for the money," > blue_suede
[elvis@station elvis]$ echo "Two for the show," >> blue_suede
[elvis@station elvis]$ echo "Three to get ready," >> blue_suede
[elvis@station elvis]$ echo "Now go, cat, go." >> blue_suede
[elvis@station elvis]$ ls -l blue_suede
-rw-rw----    1 elvis    elvis          48 Jan 16 08:10 blue_suede
[elvis@station elvis]$ chmod o+r blue_suede
[elvis@station elvis]$ ls -l blue_suede
-rw-rw-r--    1 elvis    elvis          85 Jan 16 08:11 blue_suede
```

The **chmod** command allows a user to change any permissions associated with a file. The first argument is composed of a sequence of letters specifying the access classes, followed by a plus, minus, or equals sign, followed by another sequence of letters specifying which permission types to change. Any subsequent arguments specify a list of files to apply the changes to. The syntax is summarized in the following figure.

Figure 2.1. Using the chmod command



abbreviation	interpretation
u	user
g	group
o	other
a	all
+	add
-	remove
=	set
r	read
w	write
x	execute

Examples

Using chmod on files

The following table gives several examples of how the **chmod** command can be used to modify permissions of a file named `foo`, with default permissions of `rw-rw-r--`. The first column is an example of the **chmod** command in use, and the last column is the permissions that the file would have after running the command.

```
[elvis@station elvis]$ ls -l foo
-rw-rw-r--    1 elvis    elvis          42 Jan 16 08:09 foo
```

command	effect	resulting permissions
chmod o-r foo	remove readability for others	`rw-rw----`
chmod g-w foo	remove writability for group	`rw-r--r--`
chmod ug+x foo	add executability for user and group	`rwxrwxr--`
chmod o+w foo	add writability for other	`rw-rw-rw-`
chmod go-rwx foo	remove readability, writability, and executability for group and other	`rw-------`
chmod a-w foo	remove writability for all	`r--r--r--`
chmod uo-r foo	remove readability for user and other	`-w-rw----`
chmod go=rx foo	set readability and executability but no writability for group and other	`rw-r-xr-x`

Even though the last two entries would work, why is it difficult to imagine a use for a file with either of those resulting permissions?

In the first case, it is hard to imagine a file not readable by the user owner. In the second case, it is hard to imagine a file executable by others but not by owner.

Online Exercises

Making a File Private

Lab Exercise

Objective: Change permissions on a file, such that others are not able to read the file.

Expected time: 10 minutes

1. Create the directory `/tmp/`*`username`*, if it does not already exist. For example, if your username is elvis, create the directory `/tmp/elvis`.

2. Create a simple list of resolutions in the file `/tmp/`*`username`*`/resolutions.txt`. You may use a text editor, your file from the previous lesson's exercise if it is still available, or simply create a new one as in the following example.

```
[student@station student]$ cat > /tmp/student/resolutions.txt
keep room clean
don't eat all of the pop tarts
use less proprietary software
Ctrl+D
[student@station student]$ ls -l /tmp/student/resolutions.txt
-rw-rw-r--    1 student  student        77 Jan 16 17:52 /tmp/student/resolutions.txt
```

3. Note that the permissions on a newly created file allow all users on the system to read the file. Assume that you want to keep your resolutions private. Modify the file's permissions, such that read access for others is removed.

```
[student@station student]$ chmod o-r /tmp/student/resolutions.txt
[student@station student]$ ls -l /tmp/student/resolutions.txt
-rw-rw----    1 student  student        77 Jan 16 17:52 /tmp/student/resolutions.txt
```

4. Using one of your alternate accounts, confirm that other users on the system are not able to read your resolutions.

```
[student@station student]$ su - student_a
Password:
[student_a@station student_a]$ cat /tmp/student/resolutions.txt
cat: /tmp/student/resolutions.txt: Permission denied
```

   How does this differ from the previous Lesson's Exercise?

Deliverables

1.
   1. A newly created file titled `/tmp/resolutions.txt`, readable only by the file owner.

Questions

1. Which of the following commands is used to modify a file's permissions?

   a. **chperm**

   b. **perms**

   c. **chmod**

d. **chown**

e. None of the above.

2. Select all of the following which are legitimate invocations of the **chmod** command.

a. **`chmod u+r /tmp/foo`**

b. **`chmod ug+rw /tmp/foo`**

c. **`chmod rw-g /tmp/foo`**

d. **`chmod g+wu-r /tmp/foo`**

e. **`chmod a-a /tmp/foo`**

f. **`chmod go+rwx /tmp/foo`**

Use the output of the following command to answer the next 2 questions.

```
[elvis@station elvis]$ ls -l
total 4
-rw-rw-r--    1 elvis    elvis         138 May  8 16:15 sample.txt
```

3. Which command would result in `sample.txt` having permissions of `rw-------`?

a. **`chmod a-rw sample.txt`**

b. **`chmod og-rw sample.txt`**

c. **`chmod u+rw sample.txt`**

d. **`chmod u-rw sample.txt`**

e. **`chmod o-rw sample.txt`**

4. Which command would result in `sample.txt` having permissions of `rw-r--r--`?

a. **`chmod a-w sample.txt`**

b. **`chmod go-w sample.txt`**

c. **`chmod go-rw sample.txt`**

d. **`chmod g-w sample.txt`**

e. Both B and D

Use the output of the following command to answer the next 2 questions.

```
[elvis@station elvis]$ ls -l
total 4
-rw-------    1 elvis    elvis         138 May  8 16:15 diary.txt
```

5. Which of the following commands would result in `diary.txt` having permissions of `rw-r--r--`?

a. **`chmod a+r diary.txt`**

b. **`chmod go+r diary.txt`**

  c. **`chmod u-w diary.txt`**

  d. **`chmod o+rw diary.txt`**

  e. **`chmod ugo+r diary.txt`**

  f. **`chmod ugo+rw diary.txt`**

6. Which command would result in `diary.txt` having permissions of `rw-rw-r--`?

   a. **`chmod a+w diary.txt`**

   b. **`chmod go+w diary.txt`**

   c. **`chmod ug+rw diary.txt`**

   d. **`chmod g+rw o+r diary.txt`**

   e. The desired permissions cannot be obtained using the **chmod** command only once.

Use the output of the following command to answer the next 2 questions.

```
[elvis@station elvis]$ ls -l
total 4
-rw-rw-r--    1 elvis    elvis         138 Jul  8 16:15 sample.sh
```

7. Which of the following commands would result in `sample.sh` having permissions of `rwxrwxr-x`?

   a. **`chmod a+rw sample.sh`**

   b. **`chmod go+x sample.sh`**

   c. **`chmod a+x sample.sh`**

   d. **`chmod x+a sample.sh`**

   e. Both C and D

8. Which of the following best explains why someone would want the file `sample.sh` to have permissions of `rwxrwxr-x`?

   a. The owner would like anyone to be able to remove the file.

   b. The owner would like anyone to be able to modify the file.

   c. The owner would like anyone to be able execute the file as a script.

   d. The owner would like anyone to be able to rename the file.

   e. None of the above reasons apply.

Chapter 3. Changing File Ownerships with chgrp and chown

Key Concepts

- The **chgrp** command changes group owners
- The **chown** command changes user owners

Discussion

Changing File Group Owners with chgrp

In the previous Lesson, we learned how to modify a file's permissions, and looked at an example of how to make a file private: the file's owner could read the file, but no one else could. What if you would like to share a file with a group of people, but not the everyone? Or what if you would like everyone to be able to read the file, but only a certain group of people to modify it? In these situations, you make use of the file's group owner and group permissions.

The previous Workbook discussed how Linux users belong to a collection of groups. To review, every user belongs to one primary group, and may belong to a number of secondary groups as well. When a new file is created, the file's group owner is set to the primary group of the creator. In Red Hat Enterprise Linux, this would usually be the creator's *private group*. Because you are the only member, your private group is not useful for collaborations. In order to share the file, you must change the file's group owner. This is done with a command called **chgrp**.

`chgrp` GROUP FILE...

Change group owner of FILE(s) to GROUP

The first argument specifies the new group owner of the file, while subsequent arguments list the files whose group owner is to be changed. The group owner can only be changed by the user who owns the file, and the user must be a member of the new group owner of the file.

In the following example, the user ventura is a member of the group wrestler and the group governor (in addition to his private group, ventura). He first wants to create a tax plan that other governors should be able to read but not modify, and to which no one else should have any access. He must perform the following steps:

1. Create the file.
2. Change the group owner of the file to the group governor using **chgrp**.
3. Change permissions on the file from the default `rw-rw-r--` to `rw-r-----` using **chmod**.

```
[ventura@station ventura]$ id
uid=2306(ventura) gid=2308(ventura) groups=2308(ventura),302(wrestle),305(govern
or)

[ventura@station ventura]$ echo "Raise tax on oatmeal" > taxplan.txt
[ventura@station ventura]$ ls -l taxplan.txt
-rw-rw-r--    1 ventura  ventura        21 Jan 16 09:55 taxplan.txt
```

```
[ventura@station ventura]$ chgrp governor taxplan.txt
[ventura@station ventura]$ ls -l taxplan.txt
-rw-rw-r--    1 ventura  governor       21 Jan 16 09:55 taxplan.txt

[ventura@station ventura]$ chmod g-w taxplan.txt
[ventura@station ventura]$ chmod o-r taxplan.txt
[ventura@station ventura]$ ls -l taxplan.txt
-rw-r-----    1 ventura  governor       21 Jan 16 09:55 taxplan.txt
```

Can steps two and three above be swapped? *yes.*

Changing File User Owners with chown

In rare instances, it is necessary for an administrator to change the user owner of a file. This can be done using the **chown** command, whose syntax is almost identical to the chgrp command:

`chown` USER FILE...

Change the owner of FILE(s) to USER.

The first argument is the name of the new user owner of the file, and subsequent arguments are the names of the files to change. Only the administrative user, root, is able to use the **chown** command. If any user could change the file owner, access permissions would be meaningless.

Because only root can execute the **chown** command, it will not be discussed in depth.

Who can Modify a File's Owners and Permissions?

In general, only a file's owner can change the permissions or group ownerships of a file. Additionally, when using **chgrp**, the file's new group owner must be one of the user's subscribed groups. The administrative user, root, can perform any of these operations. These abilities are summarized in the table below.

Table 3.1. Who Can Modify File Ownerships and Permissions?

Operation	**Allowed Users**
chmod	root and the file owner
chgrp	root and the file owner (only to subscribed groups)
chown	only root

Examples

Managing Group Files

Consider the user ventura, who is a member of the governor and wrestler secondary groups (in addition to his private group, ventura). He would like to share his wrestling plans with the user hogan, as well as all other members of the group wrestle. Non-members of the group wrestlers, however, should have no access. He accomplishes this with the following sequence of commands.

```
[ventura@station ventura]$ echo "beat chest and scream." > /tmp/plans.txt
[ventura@station ventura]$ ls -l /tmp/plans.txt
-rw-rw-r--    1 ventura  ventura        23 Jan 20 07:16 /tmp/plans.txt

[ventura@station ventura]$ chmod o-rw /tmp/plans.txt
[ventura@station ventura]$ chgrp wrestle /tmp/plans.txt
```

```
[ventura@station ventura]$ ls -l /tmp/plans.txt
-rw-rw----    1 ventura  wrestle        23 Jan 20 07:16 /tmp/plans.txt
```

Could ventura have instead used **chmod o-r /tmp/plans.txt** to the same effect in the above sequence? *yes*.

The user hogan would now like to add to ventura's plans. As a member of the group wrestle, he has permissions to do so.

```
[hogan@station hogan]$ echo "throw large objects." >> /tmp/plans.txt
[hogan@station hogan]$ cat /tmp/plans.txt
beat chest and scream.
throw large objects.
```

Proud of his contribution, hogan would now like to make the plans available to the rest of the world by making the file world readable.

```
[hogan@station hogan]$ chmod o+r /tmp/plans.txt
chmod: changing permissions of `/tmp/plans.txt': Operation not permitted
```

Even though hogan is a member of the group wrestle, he does not own the file. Who are the only two users on the system that can change the group owner of the file? *ventura and root*

Online Exercises

Sharing a file with a group

Lab Exercise

Objective: Create a file that is shared between members of a given group.

Estimated Time:15 minutes

Specifications

1. Replacing *`username`* with your username, create the directory `/tmp/`*`username`*, if it does not already exist. For example, if your username is student, create the directory `/tmp/student`.

2. Compose a short shopping list in your preferred text editor, or simply from the command line. Store the file as `/tmp/`*`username`*`/shopping.txt`. Change the group owner of the file to wrestle.

```
[student@station student]$ nano /tmp/student/shopping.txt
... (compose and save your text) ...
[student@station student]$ chgrp wrestle /tmp/student/shopping.txt
[student@station student]$ cat /tmp/student/shopping.txt
eggs
bacon
milk
M and M's
[student@station student]$ ls -l /tmp/student/shopping.txt
-rw-rw-r--    1 student  wrestle        26 Jan 16 10:48 /tmp/student/shopping.txt
```

3. Recall that your first alternate account is also a member of the group wrestle. Become your first alternate user (i.e., student_a). You may either login again from another virtual console, from a network connection, or simply **su -** to the alternate user.

```
[student@station student]$ su - student_a
Password:
[student_a@station student_a]$ groups
student_a wrestle physics
```

4. As the alternate user, confirm that you can view the file. Also, note that you can modify the file, by adding an additional item to the shopping list.

```
[student_a@station student_a]$ cat /tmp/student/shopping.txt
eggs
bacon
milk
M and M's
[student_a@station student_a]$ echo "wheaties" >> /tmp/student/shopping.txt
[student_a@station student_a]$ cat /tmp/student/shopping.txt
eggs
bacon
milk
M and M's
wheaties
```

5. Recall that your second alternate account (i.e, student_b) is not a member of the group wrestle. Try becoming your second alternate user, and repeat the previous steps. You should be able to view, but not modify, the file.

```
[student@station student]$ su - student_b
Password:
[student_b@station student_b]$ groups
student_b emperors
[student_b@station student_b]$ cat /tmp/student/shopping.txt
eggs
bacon
milk
M and M's
wheaties
[student_b@station student_b]$ echo "chips" >> /tmp/student/shopping.txt
-bash: /tmp/student/shopping.txt: Permission denied
```

Deliverables

1.
    1. The file `/tmp/`*`username`*`/shopping.txt`, owned by your primary user, group owned by the group wrestle, writable by members of the group wrestle and readable by all.

Questions

1. Which of the following commands is used to change the group owner of a file?

    a. **group**

    b. **chgroup**

    c. **chgrp**

    d. **changegroup**

    e. None of the above

2. Which of the following commands is used to change the user owner of a file?

    a. **owner**

    b. **chowner**

    c. **chown**

d. **changeowner**

e. None of the above

Use the output from the following command to answer the next 6 questions.

```
[student@station grps]$ ls -l
total 16
-rw-rw----    1 ventura  wrestle      2027 May  9 07:38 antics
-rw-------    1 ventura  ventura       126 May  9 07:40 donors
-rw-rw-r--    1 madonna  madonna       138 May  9 07:38 playlist
-rw-rw-r--    1 ventura  wrestle        29 May  9 07:38 slogans
```

3. the user madonna wants members of the group music to be able to read and modify her `playlist`, and everyone else to only be able to read it. What steps must she perform?

   a. **`chgrp music playlist`**

   b. **`chown music playlist`**

   c. **`chmod g+w playlist`**

   d. A, then C

   e. She does not need to do anything, this is already the case.

4. The user ventura wants members of the group wrestle to be able to read his `slogans`, and everyone else to have no access to the file. What steps must he perform?

   a. **`chgrp wrestle slogans`**

   b. **`chmod g+w slogans`**

   c. **`chmod o-r slogans`**

   d. A and B

   e. All of the above

5. The user ventura wants members of the group governor to be able to read his `donors`, and everyone else to have no access to the file. What steps must he perform?

   a. **`chgrp governor donors`**

   b. **`chmod g+r donors`**

   c. **`chmod o-rw donors`**

   d. A and B

   e. B and C

The user ventura would like to change the group owner of the file `antics` to music. He attempts the following:

```
[ventura@station grps]$ chgrp music antics
chgrp: changing group of `antics': Operation not permitted
```

6. What is the most likely explanation of this error?

   a. ventura used the wrong syntax on the command line.

b. ventura does not own the file antics.

c. ventura is not a member of the group music.

d. Only root can use the **chgrp** command.

e. None of the above.

7. What must ventura do to successfully perform the operation?

a. ventura must add himself to the group music.

b. ventura must get the system administrator, as root, to add him to the group music.

c. ventura must first run the **newgrp** command.

d. ventura must run **`chmod g-rw antics`**

e. None of the above.

8. Which users on the system can modify the permissions on the file `slogans`?

a. ventura

b. madonna

c. all members of the group wrestle

d. root

e. all members of the group music

9. You have just created the file `foo.txt` (with default permissions). You would now like members of the group music to be able to read and modify the file, and everyone else to be able to read but not modify it. What steps must you perform?

a. **`chgrp music foo.txt`**

b. **`chmod g+w foo.txt`**

c. **`chmod o-w foo.txt`**

d. **`A and B`**

e. **`A and C`**

10. You have just created the file `foo.txt` (with default permissions). You would now like members of the group music to be able to read (but not modify) the file, and everyone else to have no access. What steps must you perform?

a. **`chgrp music foo.txt`**

b. **`chmod g-w foo.txt`**

c. **`chmod o-r foo.txt`**

d. **`A and B`**

e. All of the above

Chapter 4. Directory Ownerships and Permissions

Key Concepts

- Because directories are also files, they have a user owner, a group owner, and a set of permissions.
- Read permissions allow a user to list the contents of a directory.
- Write permissions allow a user to add or remove files.
- Execute permissions allow a user to access a file within the directory.
- Directory permissions are modified with the **chmod** command.

Discussion

Directory Permissions

When someone is using a file within Linux, they are generally either reading its information, modifying its information, or trying to execute the file as a script or application. Therefore the permission types already discussed, namely (r)ead, (w)rite, and e(x)ecute have very natural interpretations.

To Linux, a directory is just a special type of file, therefore it also has the same types of permissions ((r)ead, (w)rite, and e(x)ecute), a user owner, a group owner, and the same classes of access ((u)ser, (g)roup, and (o)ther.) However, directories are obviously used differently. Would it be meaningful to open a directory in an editor, such as **nano /home/elvis**? Because people use directories differently, directory permissions have different interpretations.

What do people do with directories? They list their contents with the **ls** command. They remove files from them, create new files within them, and move files from one directory to another. Directory permissions should allow a directory owner to control who can perform which of these operations.

Linux considers listing a directory's contents (as with the **ls** command) analogous to "read"ing a directory, and therefore someone must have (r)ead permissions to list its contents. Adding or removing a file from a directory is considered "write"ing to the directory, and therefore someone must have (w)rite permissions in order to shuffle files within the directory.

There is no reasonable analogy to "execute"ing a directory, so Linux doesn't try to define a similar behavior. Instead, the e(x)ecute permission controls a behavior for directories which has nothing to do with command execution. In order to access any file within a directory, a user must have e(x)ecute permission. This permission is known as the "search" permission, but because the third permission was already called "execute" for regular files, the same word (and letter) is used for directories as well. In order to refer to any file within a directory (including subdirectories!), a user must have e(x)ecute permissions.

The first row of the following table should look familiar. It restates how to interpret permissions for regular files, as presented earlier. A row for directories has been added in order to compare and contrast permission interpretation for both file types.

Table 4.1. Permissions for Regular Files and Directories

	(r)ead	**(w)rite**	**e(x)ecute**
regular file	view the file	modify the file	use the file as a command

	(r)ead	(w)rite	e(x)ecute
directory	list directory contents	add or remove files	"search" for a known file within the directory

Examples

New Directory Defaults

Newly created regular files are readable by everybody, but can only be modified by the user and group owner of the file. How are newly created directories handled? Consider nero, who is collecting census data from his various provinces. He decides to create a directory called `/tmp/census` to hold all of his data.

```
[nero@station nero]$ mkdir /tmp/census
[nero@station nero]$ ls -ld /tmp/census/
drwxrwxr-x    2 nero     nero         4096 Jan 16 15:33 /tmp/census/
```

(Why did nero need to add the **-d** command line switch to the **ls** command?)

Without -d, ls would have shown the directory contents, not the directory itself.

Note that the default permissions for newly created directories are `rwxrwxr-x`. These permissions have the following implications:

1. Anyone can search files within the directory.
2. Anyone can list the files within the directory.
3. Only the directory owner (or members of the group owner) can add or remove files from within the directory.

For example, julius decides he would like to browse nero's census information. Notice that julius can browse the directories, and the files within the directories, but because of the default directory permissions, he cannot add or remove new files. Because of the default file permissions, he can view, but not modify, the contents of the files.

```
[julius@station julius]$ ls -al /tmp/census/
total 20
drwxrwxr-x    2 nero     nero         4096 Jan 16 15:48 .
drwxrwxrwt   23 root     root         4096 Jan 16 15:45 ..
-rw-rw-r--    1 nero     nero           42 Jan 16 15:48 egypt.dat
-rw-rw-r--    1 nero     nero           42 Jan 16 15:48 gaul.dat
-rw-rw-r--    1 nero     nero           42 Jan 16 15:47 iberia.dat
[julius@station julius]$ rm /tmp/census/iberia.dat
rm: remove write-protected regular file `/tmp/census/iberia.dat'? y
rm: cannot remove `/tmp/census/iberia.dat': Permission denied
[julius@station julius]$ echo "110 CE     42" > /tmp/census/thrace.dat
-bash: /tmp/census/thrace.dat: No such file or directory

[julius@station julius]$ cat /tmp/census/gaul.dat
110 CE          45430
120 CE          53200
130 CE          55820
[julius@station julius]$ echo "140 CE     583420" >> /tmp/census/gaul.dat
-bash: /tmp/census/gaul.dat: Permission denied
```

Home Directories

In Red Hat Enterprise Linux, a user's home directory does not follow the default permissions.

```
[nero@station nero]$ ls -ld ~
drwx------    3 nero     nero         4096 Jan 16 16:04 /home/nero
[nero@station nero]$ ls -l /home/
total 120
drwx------    3 alice    alice        4096 Jan 15 08:04 alice
drwx------    3 augustus augustus     4096 Jan 14 15:22 augustus
drwx------    3 austin   austin       4096 Jan 14 15:22 austin
drwx------    3 blondie  blondie      4096 Jan 14 13:46 blondie
...
```

In Red Hat Enterprise Linux, home directories are "protected". By default, only the user that owns a home directory has search permissions.

```
[nero@station nero]$ ls -l ~augustus
ls: /home/augustus: Permission denied
```

Have you noticed that most of our exercises involving multiple users accessing a file have used the `/tmp/username` directory rather than a user's home directory? Why have we not used a user's home directory instead?

Because we had not talked about directory permissions yet, a user had no way to allow other users to access files within their home directory.

Creating a ~/pub Directory

Nero would now like to make his census data available to the world at large. In Red Hat Enterprise Linux, there are generally only two places where users can create files, the `/tmp` directory and `~` (the user's home directory.) In the first example, nero chose to create a `census` directory within `/tmp`. In Red Hat Enterprise Linux, however, the `/tmp` directory is "swept." If a file within the `/tmp` is not accessed for 10 days, it is removed from the system. Nero needs to come up with a better place.

In order to create a permanent, publicly accessible location for his census data, nero chooses to create a public subdirectory within his home directory. Conventionally, such a subdirectory in Linux is often called `pub`. As the following sequences will reveal, sharing files from a user's home directory is not as easy as just creating a world (r)eadable and e(x)ecutable directory.

First nero creates the `~/pub` directory, and copies the census information from `/tmp/census` over to it.

```
[nero@station nero]$ mkdir pub
[nero@station nero]$ cp /tmp/census/* pub
[nero@station nero]$ ls -al /home/nero/pub/
total 20
drwxrwxr-x    2 nero     nero         4096 Jan 16 16:13 .
drwx------    4 nero     nero         4096 Jan 16 16:12 ..
-rw-rw-r--    1 nero     nero           42 Jan 16 16:13 egypt.dat
-rw-rw-r--    1 nero     nero           42 Jan 16 16:13 gaul.dat
-rw-rw-r--    1 nero     nero           42 Jan 16 16:13 iberia.dat
```

❶ Recall that "." always refers to the current directory, in this case `/home/nero/pub`.
❷ Recall that ".." always refers to the current directory's parent, in this case `/home/nero`.

Being the conscientious sort, nero double checks the permissions on the newly created directory and files. On `/home/nero/pub`, he finds permissions of `rwxrwxr-x`, implying that others can search and list files from within the directory. The data files themselves have permissions of `rw-rw-r--`, implying that others have read access to the files, so all seems in order. He tells julius where to find the information.

Interested in the data, julius tries to access one of the files. Unfortunately, all does not go well:

```
[julius@station julius]$ cat /home/nero/pub/egypt.dat
cat: /home/nero/pub/egypt.dat: Permission denied
```

What has nero neglected? Recall that in order to access a file within a directory, including subdirectories, a user must have search permissions for the directory. The permissions on `/home/nero/pub` are fine, but consider the permissions on `/home/nero` (".." in the above listing, or listed again below):

```
[nero@station nero]$ ls -ld /home/nero/
drwx------    4 nero     nero         4096 Jan 20 14:05 /home/nero/
```

If julius were able to access `/home/nero/pub`, all would be fine. But because julius does not have search permissions for `/home/nero`, he cannot access `/home/nero/pub`! In order to create a publicly accessible directory within a home directory, a user must allow people to search their home directory. Nero fixes the problem in the following sequence of commands.

```
[nero@station nero]$ chmod o+x /home/nero/
[nero@station nero]$ ls -al /home/nero/pub/
total 20
drwxrwxr-x    2 nero     nero         4096 Jan 16 16:13 .
drwx-----x   4 nero     nero         4096 Jan 16 16:14 ..
-rw-rw-r--    1 nero     nero           42 Jan 16 16:13 egypt.dat
-rw-rw-r--    1 nero     nero           42 Jan 16 16:13 gaul.dat
-rw-rw-r--    1 nero     nero           42 Jan 16 16:13 iberia.dat
```

❶ Now others have e(x)ecute permissions to `/home/nero`.

julius tries again to examine the file. Because the permissions on `/home/nero/pub` allow others both (r)ead and e(x)ecute access, julius can both search for files and get a directory listing.

```
[julius@station julius]$ cat /home/nero/pub/egypt.dat
110 CE          45430
120 CE          53200
130 CE          55820
[julius@station julius]$ ls /home/nero/pub
egypt.dat  gaul.dat  iberia.dat
```

In contrast, nero's home directory, `/home/nero`, now has permissions of `rwx-----x`, allowing julius only e(x)ecute permissions. Therefore, if julius already knows a file is in the directory (because, for example, nero told him it was there), julius can get to it; however, julius cannot browse the contents of the directory with the **ls** command.

```
[julius@station julius]$ ls /home/nero
ls: /home/nero: Permission denied
```

Protecting Home's Subdirectories

Often, users elect to allow other users to have access to their home directories (consider, for example, the previous example). By allowing others e(x)ecute, but not (r)ead, permissions to their home directory, other users must know that a directory exists within the home directory in order to access it. Because other users may not use the **ls** command to discover the contents of their home directory, users home directories remain private, and only the parts they choose to expose are available to other users.

While not adding (r)ead permissions to a home directory provides some protection against the browsing of other users, it is not foolproof. Other users can still "guess" at the contents of a directory to which they have e(x)ecute but not (r)ead permissions. For example, users commonly create a directory called `~/mail`, for storing mail messages. Suppose nero has given others e(x)ecute access to his home directory (as in the previous example), and later creates such a `~/mail` directory. If elvis were to guess that such a directory existed, the default permissions on `~nero/mail` would allow him to browse its contents. This is played out in the following transcript:

```
[nero@station nero]$ ls -ld ~
drwx-----x    3 nero     nero         4096 Jan 20 16:41 /home/nero
[nero@station nero]$ mkdir mail
```

```
[nero@station nero]$ cal 2002 > mail/sent
[nero@station nero]$ ls -al mail
total 12
drwxrwxr-x    2 nero     nero         4096 Jan 20 16:41 .
drwx-----x    4 nero     nero         4096 Jan 20 16:41 ..
-rw-rw-r--    1 nero     nero         2027 Jan 20 16:41 sent
```

```
[elvis@station elvis]$ cat ~nero/mail/sent
                               2002

       January               February                 March
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
       1  2  3  4  5                   1  2                   1  2
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16
20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23
27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30
                                              31

...
```

What is the lesson for nero? If you allow others access to your home directory, as is often the case, you must consider the permissions on newly created subdirectories. By default, if someone can guess the directory name, they will be able to browse the subdirectory.

Nero's solution would be to close off his `mail` subdirectory.

```
[nero@station nero]$ chmod o-rx mail
```

Online Exercises

Creating Public Directories for Distinct Groups

Lab Exercise

Objective: Create group accessible directories within a user's home directory.

Estimated Time: 30 mins.

Specifications

Your account should be a member of two secondary groups, music and wrestle, in addition to your private group. Additionally, your first alternate account should be a member of the groups wrestle and physics, while your third alternate account should be a member of the group music.

```
[student@station student]$ groups student
student : student wrestle music
[student@station student]$ groups student_a
student_a : student_a wrestle physics
[student@station student]$ groups student_c
student_c : student_c music
```

From within your home directory, you would like to share information with other musicians and wrestlers, but you would not like musicians to see wrestlers' information, and *vice versa*. You would prefer if one group didn't even realize that the other group's directory even existed. Adopt the following plan.

1. Allow others access files in your home directory by granting them "search" permissions.

2. Create a publicly searchable, but not listable, subdirectory to your home directory, `~/pub`.

3. Create two subdirectories of `~/pub`, both searchable and listable to only the members of the groups *music* and *wrestle*, respectively: `~/pub/music` and `~/pub/wrestle`. Note that you will need to consider the group owners of the directories, as well as permissions.

4. Create the files `~/pub/music/lyrics` and `~/pub/wrestle/plan`. The files should be readable and writable by members of the music and wrestle groups, respectively.

When you are finished, you should be able to confirm the appropriate behavior using your first and third alternate accounts (student_a and student_c).

```
[student_a@station student_a]$ ls /home/student/pub/wrestle
plan
[student_a@station student_a]$ cat /home/student/pub/wrestle/plan
pin the other guy
[student_a@station student_a]$ ls /home/student/pub
ls: /home/student/pub: Permission denied

[student_c@station student_c]$ ls ~student/pub/music
lyrics
[student_c@station student_c]$ cat ~student/pub/music/lyrics
row, row, row your goat
[student_c@station student_c]$ ls ~student/pub
ls: /home/student/pub: Permission denied
```

Deliverables

1.
    1. A Home directory searchable to all, listable only by you.
    2. A `~/pub` directory searchable to all, listable only by you.
    3. A `~/pub/music` directory, searchable and listable only by members of the group music.
    4. A file `~/pub/music/lyrics`, readable and writable by members of the group music.
    5. A `~/pub/wrestle` directory, searchable and listable only by members of the group wrestle.
    6. A file `~/pub/wrestle/plan`, readable and writable by members of the group wrestle.

Hints

The following series of commands demonstrates one possible solution for making the `~/pub/music` directory.

```
[student@station student]$ ls -ld ~
drwx------    3 student  student      4096 Jan 20 15:17 /home/student

[student@station student]$ chmod o+x ~
[student@station student]$ mkdir ~/pub
[student@station student]$ chmod o-r ~/pub
[student@station student]$ ls -al ~/pub
total 8
drwxrwx--x    2 student  student      4096 Jan 20 15:17 .
drwx-----x    4 student  student      4096 Jan 20 15:17 ..

[student@station student]$ mkdir ~/pub/music
[student@station student]$ chmod o-rx ~/pub/music/
[student@station student]$ chgrp music ~/pub/music
[student@station student]$ ls -al ~/pub
total 16
drwxrwx--x    4 student  student      4096 Jan 20 15:18 .
drwx-----x    4 student  student      4096 Jan 20 15:17 ..
```

```
drwxrwx---    2 student  music        4096 Jan 20 15:18 music
```

Protecting Subdirectories within Home

Lab Exercise

Objective: Protect a newly created subdirectory within your home directory from unintended browsing.

Estimated Time: 10 mins.

This lab assumes that others already have e(x)ecute permissions on your home directory. Create a `memos` subdirectory in your home directory, and modify its permissions so that other users on the system have no access to the directory. Create a file within the directory, and use one of your alternate accounts to confirm that other users cannot access the file.

Solution: The following series of commands proved one possible solution to the above specifications. (The output assumes you have just completed the previous exercise.)

```
[student@station student]$ ls -ld ~
drwx-----x    4 student  student      4096 Jan 20 16:50 /home/student
[student@station student]$ mkdir memos
[student@station student]$ chmod o-rx memos/
[student@station student]$ ls -l
total 8
drwxrwx---    2 student  student      4096 Jan 20 16:50 memos
drwxrwx--x    4 student  student      4096 Jan 20 15:18 pub
[student@station student]$ echo "feed cat" > memos/todo

[student_a@station student_a]$ ls ~student/memos
ls: /home/student/memos: Permission denied
[student_a@station student_a]$ cat ~student/memos/todo
cat: /home/student/memos/todo: Permission denied
```

Deliverables

1.
    1. A directory called `~/memos`, which is browseable by only the directory owner.

Questions

Analyzing Directory Permissions

Use the following table of users (with group memberships) and files (with user owner, group owner, and permissions) to answer the following questions. Note that the permissions includes an initial letter that indicates if the file is a regular file ("-") or a directory ("d").

user	**group memberships**
elvis	elvis, wrestle, physics, emperors, music
ventura	ventura,wrestle,governor
pataki	pataki,governor
blondie	blondie, music
prince	prince, music
einstein	einstein, physics

user	**group memberships**
maxwell	maxwell, physics

filename	**user**	**group**	**permissions**
`governor/`	ventura	governor	`drwxrwx---`
`governor/budget2001`	ventura	governor	`-rw-r--r--`
`governor/budget2002`	ventura	governor	`-rw-r-----`
`music/`	root	music	`drwxr-xr-x`
`music/contrib/`	root	music	`drwxrwxrwx`
`music/contrib/stand_by_your_spam.txt`	blondie	blondie	`-rw-rw-r--`
`music/drafts/`	prince	music	`drwxrwx---`
`music/drafts/1998.txt`	prince	prince	`-rw-rw-r--`
`music/drafts/little_orange_corvette.txt`	prince	music	`-rw-rw-r--`
`physics/`	einstein	physics	`drwxrwx--x`
`physics/published/`	maxwell	physics	`drwxrwxr-x`
`physics/published/equations`	maxwell	maxwell	`-rw-r--r--`
`physics/published/relativity`	einstein	einstein	`-rw-rw-r--`
`physics/working/`	einstein	physics	`drwxr-x---`
`physics/working/time_travel`	einstein	physics	`-rwxr--r--`

1. Which user can read the file `governor/budget2001`?

   a. elvis

   b. ventura

   c. prince

   d. maxwell

2. Which user can modify the file `physics/published/relativity`?

   a. einstein

   b. maxwell

   c. elvis

   d. blondie

3. Which users can modify the file `music/drafts/little_orange_corvette`?

   a. elvis and einstein

   b. elvis, prince, and blondie

   c. prince and einstein

    d. only prince

4. Which users can create a new file in the directory `physics/published/`?
    a. only maxwell
    b. maxwell and ventura
    c. all users can
    d. elvis, maxwell, and einstein

5. Which users can list files in the directory `physics/`?
    a. elvis, einstein, and maxwell
    b. only einstein
    c. all users can
    d. elvis, prince, and blondie

6. Which users can list files in the directory `music/`?
    a. elvis, einstein, and maxwell
    b. only root
    c. all users can
    d. elvis, prince, and blondie

7. Who can read the file `governor/budget2001`?
    a. All users
    b. Only ventura
    c. ventura and pataki
    d. ventura and maxwell

8. Who can read the file `governor/budget2002`?
    a. All users
    b. Only ventura
    c. ventura and pataki
    d. ventura and maxwell

9. Which users can read the file `physics/working/time_travel`?
    a. elvis, maxwell, and einstein
    b. only einstein
    c. all users

d. einstein and maxwell

10. Which users can remove the file `music/contrib/stand_by_your_spam.txt`?

a. only blondie

b. all users

c. elvis, maxwell, and einstein

d. prince, blondie, and elvis

Chapter 5. chmod Revisited: Octal Notation

Key Concepts

- The **chmod** command can use an alternate octal syntax
- The octal syntax is often quicker and more exacting.

Discussion

Why a different syntax?

In the previous lessons, the **chmod** command was used to modify a file's permissions using a symbolic syntax, as in the following.

```
[student@station student]$ chmod og-r diary
```

This syntax has some advantages, but also some disadvantages. Because the syntax is very readable, the intent of the command is fairly evident: the student does not want other users reading the `diary.` What are the resulting permissions on the file `diary?` A drawback of the syntax is that, knowing only that this command was run, you cannot say! Is the file writable to group members? others? Without knowing the original permissions, you do not know.

This lesson will illustrate an alternate "octal" syntax for the **chmod** command, which overcomes this shortcoming:

```
[student@station student]$ chmod 600 diary
```

After learning the syntax, you would know that, after running the above command, the file `diary` would have permissions of `rw-------` As an added benefit, once learned, the octal syntax tends to be quicker to use.

Where did 600 come from?

Recall that a file has three different types of permissions ((r)ead, (w)rite, and e(x)ecute) for three classes of user access ((u)ser owner, (g)roup owner, and (o)ther). With the octal notation, each access class gets a digit: the "hundreds" place for (u)ser, the "tens" place for (g)roup, and the "ones" place for (o)ther. Each permission type gets a value: (r)ead gets 4, (w)rite gets 2, and e(x)ecute gets 1. The digits in the octal notation are just the sum of the allowed permissions for that class of access. The following examples demonstrate how to translate octal numbers into the conventional permissions syntax.

```
755 = rwxr-xr-x

    7 = 4 + 2 + 1 = rwx for (u)ser
    5 = 4 + 0 + 1 = r-x for (g)roup
    5 = 4 + 0 + 1 = r-x for (o)ther

640 = rw-r-----
```

6 = 4 + 2 + 0 = rw- for (u)ser
4 = 4 + 0 + 0 = r-- for (g)roup
0 = 0 + 0 + 0 = --- for (o)ther

701 = rwx-----x

7 = 4 + 2 + 1 = rwx for (u)ser
0 = 0 + 0 + 0 = --- for (g)roup
1 = 0 + 0 + 1 = --x for (o)ther

This notation is called an "octal" notation, because each digit has 8 possible values (0 through 7).

File modes

Once learned, the octal notation provides a quick way to refer to the permissions on a file. For example, experienced Unix administrators will often refer to a file having "644" permissions, or a directory having permissions of "755". In Unix, the state of a file's permissions is often referred to as the mode of the file. In particular, this term is used to refer to a file's permissions in octal notation. For example, a system administrator might say "Directories are created with a default mode of 775".

Examples

Creating a ~/pub Directory (using Octal Notation)

The user elvis would like to set up a shared directory `~/drafts`. He would like members of the group music to have full access to the directory, and other users to have no access. When using **chmod**, elvis will use octal notation.

```
[elvis@station elvis]$ mkdir ~/drafts
[elvis@station elvis]$ chmod 701 ~
[elvis@station elvis]$ chmod 770 ~/drafts/    ❶

[elvis@station elvis]$ echo "One for the honey" > ~/drafts/lyrics
[elvis@station elvis]$ chgrp -R music drafts/ ❷
[elvis@station elvis]$ ls -al drafts/
total 12
drwxrwx---   2 elvis    music        4096 Feb 14 10:16 .  ❸
drwx------   5 elvis    elvis        4096 Feb 14 10:15 .. ❹
-rw-rw-r--   1 elvis    music          18 Feb 14 10:16 lyrics
```

❶ In this use of **chmod**, elvis was able to adjust both the (g)roup and (o)ther permissions with a single command. How would he have had to do this using symbolic notation? *chmod g+w ~/drafts chmod o-rx ~/drafts*

❷ Note that **chgrp -R** tells **chgrp** to recurse through a directory (and all it's subdirectories). **chmod**, **chgrp**, and **chown** all have this command line switch.

❸ "." always refers to the current directory, in this case `/home/elvis/drafts`.

❹ Recall that ".." always refers to the current directory's parent, in this case `/home/elvis`.

Misuse of Recursive Octal Notation

The user einstein has the following collection of papers, some published, some unpublished. All files and directories have default permissions.

```
papers/
|-- published/
|   `-- relativity
`-- unpublished/
    |-- eismcsquared
    |-- photoelectric_effect
    `-- unlikely/
        `-- time_travel
3 directories, 4 files
```

The user einstein now decides to change permissions on all files within his unpublished directory, such that other users will not be able to read them. In order to save time, einstein decides to use the recursive form of **chmod**.

```
[einstein@station einstein]$ chmod -R 660 papers/unpublished/*
chmod: failed to get attributes of `papers/unpublished/unlikely/time_travel': Permission denied
[einstein@station einstein]$ ls papers/unpublished/unlikely/
ls: papers/unpublished/unlikely/time_travel: Permission denied
```

What has einstein done wrong? Why can he not even view his own directory? Pay close attention to the permissions of the files in `papers/unpublished`:

```
[einstein@station einstein]$ ls -l papers/unpublished/
total 4
-rw-rw----    1 einstein einstein        0 Feb 14 10:46 eismcsquared
-rw-rw----    1 einstein einstein        0 Feb 14 10:45 photoelectric_effect
drw-rw----    2 einstein einstein     4096 Feb 14 10:46 unlikely
```

When einstein ran **chmod -R 660 papers/unpublished/***, **chmod** did what it was supposed to. It set the mode of every file (including the directory `unlikely`) to 660. It then went to recurse through the directory `unlikely`, but because the directory now had a mode of 660, einstein no longer has access permission to his own directory, and the command fails!

What is the lesson to learn from this example? Directories and regular files have different "reasonable" permissions. Namely, directories should have the e(x)ecute permission set, while regular files generally should not. Unfortunately, **chmod -R** does not distinguish between the file and directories.

(In order to obtain the desired result, einstein could use the **find** command to recurse more intelligently. **find** will be introduced in a later section.)

Online Exercises

Managing Permissions on Groups of Files

Lab Exercise

Objective: Manage permissions effectively on a large group of files, using octal notation.

Estimated Time: 30 mins.

Setup

Your second alternate account (student_b) should be a member of the group emperors. Login under that account, and make the directory `~/reports`, and **cd** to that directory.

```
[student_b@station student_b]$ groups
student_b emperors
[student_b@station student_b]$ mkdir reports
[student_b@station student_b]$ cd reports
```

Use the following "magic" command to create a large number of files in the current directory. The bracket syntax that you are using will be introduced in a later Workbook.

```
[student_b@station reports]$ touch {2001,2002,2003}.{q1,q2,q3,q4}.{data,draft,final}
[student_b@station reports]$ ls
2001.q1.data   2001.q3.final  2002.q2.draft  2003.q1.data   2003.q3.final
2001.q1.draft  2001.q4.data   2002.q2.final  2003.q1.draft  2003.q4.data
2001.q1.final  2001.q4.draft  2002.q3.data   2003.q1.final  2003.q4.draft
2001.q2.data   2001.q4.final  2002.q3.draft  2003.q2.data   2003.q4.final
2001.q2.draft  2002.q1.data   2002.q3.final  2003.q2.draft
2001.q2.final  2002.q1.draft  2002.q4.data   2003.q2.final
2001.q3.data   2002.q1.final  2002.q4.draft  2003.q3.data
2001.q3.draft  2002.q2.data   2002.q4.final  2003.q3.draft
```

Note that all files should have default permissions.

Specification

You wish to implement the following policies

1. All files (including directories) should be group owned by the group emperors.
2. All the files that end `.data` contain raw data. Anyone can read the files, but only you should be able to modify them.
3. All the files that end `.draft` contain working drafts of your reports, which you want help with. Members of the group emperors should be able to read and modify them, but other users should have no access to them.
4. All the files that end `.final` are final drafts. You want to move them to a dedicated directory `~/reports/final`. Only members of the group emperors should have access to the subdirectory, and they should be able to list the directory contents, and read (but not modify) the reports.

Use the **chmod** command, with octal notation, to obtain these results.

Deliverables

1. The directory `~/reports`, and all of its underlying files and directories, should be group owned by emperor.

   1. A `~/reports` directory, accessible and readable by all.
   2. A collection of files `~/reports/*.data`, readable by all but writable only to you.
   3. A collection of files `~/reports/*.draft`, readable and writable only to members of the group emperor.
   4. A `~/reports/final` directory, accessible and readable only to members of the group emperor.
   5. A collection of files `~/reports/final/*.final`, readable and writable to the file owner, but readable and *not* writable to members of the group emperor. Other users should have no access.

Possible Solution

The following sequence of commands provides one possible solution to this exercise.

```
[student_b@station reports]$ chmod 644 *.data
[student_b@station reports]$ chmod 660 *.draft
[student_b@station reports]$ chmod 640 *.final
[student_b@station reports]$ mkdir final
[student_b@station reports]$ chmod 750 final/
[student_b@station reports]$ mv *.final final/
[student_b@station reports]$ chgrp -R emperors .
```

Questions

Analyzing Directory Permissions

1. Which octal notation below represents permissions of rwxr-x--x?
   a. 751
   b. 762
   c. 007
   d. 267
2. Which octal notation below represents permissions of rw-rw-r--?
   a. 551
   b. 664
   c. 771
   d. 660
3. Which permission represents the octal notation 701?
   a. rwx---r--
   b. r-----rwx
   c. rwx-----x
   d. x-----rwx
4. Which permission represents the octal notation 644?
   a. -w--w-rw-
   b. --x--x-wx
   c. r-xr--r--
   d. rw-r--r--
5. Which of the following would be the most reasonable permissions for a directory that members of only a particularly group should have access to?
   a. 750
   b. 075

   c. 700

   d. 740

6. Which of the following would be the most reasonable permissions for a file that all users can read, but only the owner can write?

   a. 006

   b. 622

   c. 660

   d. 644

Chapter 6. Controlling Default Permissions: umask

Key Concepts

- At the kernel level, Linux creates files with a default mode of 666 (`rw-rw-rw`).
- At the kernel level, Linux creates directories with a default mode of 777 (`rwxrwxrwx`).
- Every process possesses a "umask" parameter which masks out certain of these default permissions.
- In Red Hat Enterprise Linux, the default umask for standard users is 002.
- The bash shell's umask is modified with the **umask** command.

Discussion

Controlling default permissions with umask

Previous lessons relied on the fact that newly created files have a default mode of 664 (`rw-rw-r--`), implying that anyone can read, but only the file owner can modify, a newly created file. Similarly, newly created directories have a default mode of 775 (`rwxrwxr-x`), implying that any one can access or browse the directory, but only the directory owner can add or remove files.

The default permissions of newly created files can be altered through a standard Unix concept called a umask. Every process, including the **bash** shell, has a 3 digit octal number which is used to "mask out" certain permissions when new files are created. This octal number is referred to as the process's "umask". The umask is composed just like an octal mode, but the meaning is reversed. Like an octal mode, each access class is represented by a single digit: (u)ser owner is the "hundreds", (g)roup owner the "tens", and (o)ther the ones. Like an octal mode, each permission type has a value: 4 for (r)ead, 2 for (w)rite, and 1 for e(x)ectue. Unlike octal modes, however, the umask is composed of the values of *unwanted* permissions.

Default Permissions for files

In order to determine permissions for newly created files, the Linux kernel starts with a global default mode of 666 (`rw-rw-rw`). The kernel then applies the umask for the process that created the file. Any values that are set in the umask are "stripped" from the default permissions of 666. For example, a umask of 002 would result in default permissions of 664:

```
kernel default:         666 --> rw-rw-rw-
umask:                  002 --> -------w-
-----------------------------------------
default permissions:    664 --> rw-rw-r--
```

As another example, a umask of 077 would result in default permissions of 600:

```
kernel default:         666 --> rw-rw-rw-
umask:                  077 --> ---rwxrwx
```

```
        ----------------------------------------
        default permissions:    600 --> rw-------
```

When determining default permissions, any permission that is set in the umask is "masked out" of the kernel default.

Default Permissions for directories

Default permissions for directories are created using the same technique, except that the kernel default mode for directories is 777 (`rwxrwxrwx`). For example, a umask of 002 would result in default directory permissions of 775:

```
        kernel default:         777 --> rwxrwxrwx
        umask:                  002 --> -------w-
        ----------------------------------------
        default permissions:    775 --> rwxrwxr-x
```

As another example, a umask of 077 would result in default permissions of 700:

```
        kernel default:         777 --> rwxrwxrwx
        umask:                  077 --> ---rwxrwx
        ----------------------------------------
        default permissions:    700 --> rwx------
```

Note that, in both of the above examples, a umask of 002 has the same general effect for files and directories: Any one can read, but only the owner may modify. Similarly, a umask of 077 has the same general effect on files as on directories: the user owner may read and modify, but everyone else has no access.

Modifying the shell's umask: the umask command

The umask of the **bash** shell can be examined and modified by a command called **umask**. When called without arguments, the umask command reports the shell's current umask. When called with an octal umask as a single argument, the shell's umask is set to the specified value.

```
[student@station student]$ umask
0002
[student@station student]$ umask 077
[student@station student]$ umask
0077
```

(Note that, on output, the umask is reported as a *four* digit octal number. For now, the leading "0" can be ignored. Its meaning will become clear after the following lesson.)

Examples

Using umask to Create Group Shared Files

In the following sequence, blondie and other musicians are collaborating on song lyrics. blondie would like to create several files that can be viewed and modified by members of the group music, but that no one else can read or write. One approach would be to create the files, and then change the file's permissions with **chmod**. Instead, blondie is going to use **umask** to change her shell's default permissions on newly created files.

```
[blondie@station blondie]$ umask
0002
[blondie@station blondie]$ umask 007
[blondie@station blondie]$ umask
0007

[blondie@station blondie]$ echo "Twinkle, twinkle, little star" > song1.txt
[blondie@station blondie]$ echo "Bah, bah, black sheep" > song2.txt
[blondie@station blondie]$ echo "Mary had a little lamb" > song3.txt
[blondie@station blondie]$ chgrp music song*.txt
[blondie@station blondie]$ ls -l
total 12
-rw-rw----    1 blondie  music          30 Feb 20 14:20 song1.txt
-rw-rw----    1 blondie  music          22 Feb 20 14:20 song2.txt
-rw-rw----    1 blondie  music          23 Feb 20 14:20 song3.txt
```

Note that the shell's umask of 007 "masked out" the write permissions that would have been set for (o)thers.

Using ~/.bashrc to automatically change the bash shell's umask

The emperor nero, in a fit of paranoia, is suspecting that other users are snooping on his files. He would like to configure his **bash** shell, such that every time he logs in, the shell's umask is automatically set to 077. He will do this by editing the file `~/.bashrc` that usually exists by default in a user's home directory.

Whenever a new **bash** shell is started, the commands listed in the specially named file `~/.bashrc` are executed as if they were typed from the command line. By appending "umask 077" to nero's `~/.bashrc` file, he will configure his shell such that the shell's umask will automatically be set to 077 upon startup.

In the following sequence, nero first examines, then appends a new line, and then reexamines his file `~/.bashrc`. For now, do not worry about the original contents of the file, just note that the modified file runs **umask** as its last command.

```
[nero@station nero]$ cat ~/.bashrc
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
[nero@station nero]$ echo "umask 077" >> ~/.bashrc
[nero@station nero]$ cat ~/.bashrc
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
umask 077
```

Now that nero has modified his `~/.bashrc` file, his umask will be automatically set to 077 every time he starts a new shell. When nero logs out, and then back on again, newly created files now have default permissions of 600 (`rw-------`).

```
Red Hat Enterprise Linux release 3.0 (Taroon)
Kernel 2.4.20-4-ES on an i686

station login: nero
Password:
[nero@station nero]$ umask
0077
[nero@station nero]$ echo "All of Gaul is divided into three parts" > notes.txt
[nero@station nero]$ ls -l
total 4
-rw-------    1 nero     nero           40 Feb 20 14:44 notes.txt
```

Online Exercises

Changing the bash shell's default permissions.

Lab Exercise

Objective: Use the `~/.bashrc` file to change the **bash** shell's default permissions.

Estimated Time: 10 mins.

Specification

Every time you start a **bash** shell, the contents of the file `~/.bashrc` are sourced as if you had typed them in at the command line. This file is often used to customize the default behavior of **bash**.

You would like for your shell's default umask to be changed automatically every time you log in. Append a line to the bottom of your `~/.bashrc` file which sets the shell's umask. Newly created files should be readable by all, but writable only to the user owner by default.

Deliverables

1.
    1. A `~/.bashrc` file that sets the **bash** shell's default umask such that files are readable by default to everyone, but only writable by the file's user owner. (In particular, the file should not be writable to the group owner).

Possible Solution

The following sequence of commands provides one possible solution to this exercise.

```
[student@station student]$ echo "umask 022" >> .bashrc
[student@station student]$ cat .bashrc
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
umask 022
```

Upon logging out and logging back in again, you should see the following behavior.

```
[student@station student]$ date > test_file.txt
[student@station student]$ ls -l
```

```
total 4
-rw-r--r--    1 student  student        29 Feb 20 14:52 test_file.txt
```

Note that the newly created files permissions are 644, not the Red Hat default of 664.

Cleaning Up

After you have graded this exercise, remove the line that you added from your `~/.bashrc` file.

Questions

Understanding the implications of a process's umask

1. If a process had a umask of 077, what would be the default permissions for newly created files?

   a. r--r--r--

   b. rw-rw-r--

   c. rwx------

   d. rw-------

2. If you notice that newly created directories have permissions of `rwxr-x---`, what umask is being used?

   a. 077

   b. 027

   c. 227

   d. 002

3. If a process had a umask of 077, what would be the default permissions for newly created directories?

   a. r--r--r--

   b. rw-rw-r--

   c. rwx------

   d. rw-------

4. If you notice that newly created files have permissions of `rw-r-----`, which of the following is a possible umask?

   a. 036

   b. 027

   c. 037

   d. all of the above

5. What umask would result in newly created files with default permissions of `rwxr-xr-x`?

a. 022

b. 033

c. 011

d. It is impossible to have newly created files with the given permissions.

6. Why would you not expect to see someone set a shell's umask to 422?

a. Other users could not read newly created files by default.

b. Users could not read their own newly created files.

c. Newly created files with identical group and other permissions are not allowed.

d. A legitimate umask cannot contain digits higher than 2.