Linux File Permissions

by Scott Morris

http://www.suseblog.com/ (come visit!)

- Let's say I were to ask to borrow car keys and credit card for a trip to Las Vegas for the weekend. You would probably not allow this, correct?

- What about if I asked to borrow your phone for a local call? Or to borrow your computer for a moment to check my email? You would likely permit me to do either of these things, right?

- Inside of companies, employees need to have access to certain things in order to do their jobs properly. Would it be a good idea to allow every employee unrestricted access to everything? Absolutely not.

- The Accounting Department needs access to the payroll, the Development Department needs access to the source code, and Human Resources needs to have access to employee information. It would be borderline illegal to give each of these people access to each other's files and systems, not so?

In Linux, we have full control over file access so that many people can use the system. They can maintain their own files and data without fear of having them be viewed, changed, deleted, or modified by other people.

Let's take a look at how this works.

There are three types of people who may be accessing the files:

- **User** – An account created on the system that a person uses to access that system. Accounts may also be created for machine processes, like the mail, FTP, or web server. Users have direct ownership over files and directories on the system.
- **Group** – Provides the ability to give access to sets of users. Groups have their own permissions over each file or directory.
- **World** – This is anyone else that may attempt to access a file on the machine.

There are also three ways that these types of people may access the file.

- **Read** – Opening a file and looking at its contents.
- **Write** – Overwrite, append, or delete a file. In directories, this may include creation of files.
- **Execute** – The ability to “run” a program (or script).

Let's see an example.

Our company has 3 employees:
Scott – Development
Jenny – Human Resources
Brett – Accounting

We have some files in a directory:

```
laptop:/files # ll
total 48
-rw-rw---- 1 scott development     6527 2007-05-29 10:04 codefile.c
-rw-r--r-- 1 jenny humanresources 21216 2007-05-29 10:04 employees.txt
-rwx------ 1 jenny humanresources   994 2007-05-29 10:05 manage.sh
drwxrwxrwx 2 scott users           4096 2007-05-29 10:31 newsletters
-rw-r----- 1 brett accounting      3224 2007-05-29 10:05 payroll.txt
laptop:/files #
```

We can see that there are some files on this system, each with their different owning users and groups. At the beginning of each line, there are a series of hyphens and letters. Each of these have significance:

- First column – Tells us whether it is a file or a directory. If a “d”, it is a directory. We don't change this. The system manages this for us.
- Columns 2,3,4 – tell us what permissions the user has for the file or directory.
- Columns 5,6,7 – tell us what permissions the group has for the file or directory.
- Columns 8,9,10 – tell us what permissions everyone else, or the “world” has for the file or directory.

Each column shows one of three letters: “r”, “w”, “x”

- “r” - tells us whether the “read” permission is granted, or whether people can open the file and view its contents.
- “w” - tells us whether the “write” permission is granted, or whether people can modify, append to, or save the file
- “x” - tells us whether the “execute” permission is granted, or whether people can “run” the program or script.

Let's look at the files we listed:

- **codefile.c** is a source code file. It only makes sense to have this file accessible by developers. It is also not an executable file, so no one has “execute” permissions on it. Scott and developers can read and write to this file.
- **employees.txt** is an employee address book. Jenny in Human Resources has been given the task of maintaining it. Everyone else in Human Resources can open and view the file. The other employees in the company can also open and view the file to look up phone numbers and email addresses of those listed in the file. But it makes no sense to give access to write to the file to anyone not maintaining it.
- **manage.sh** is an executable script. It adds an entry to the “employees.txt” file. Notice again that Jenny is the only one who has access to add employees.
- **newsletters** is a directory. We can see that everyone has all access to this directory. This makes it possible for any employee to go in and make any announcements as necessary.
- **payroll.txt** is a text file containing payroll records for the other employees in the company. It is maintained by Brett in Accounting, but the rest of the Accounting department can open and view the file's contents.

A good policy to have is to give only permissions that are specifically necessary. No one has access to anything for no reason. This policy can be referred to as “most restrictive access.”

Managing Permissions

There is a command for managing permissions called “chmod”. This means “change mode”.

Basic syntax for this command is:

```
chmod [who][how to change][permissions to modify] [file or directory name]
```

The “who” can be:
- u – user or account that owns the file
- g – group that owns the file
- o – others, or “world”, anyone else who may access the file
- a – all of the above

The “what change” can be:
- = (equals) - set the permissions to exactly what immediately follows
- + (add) - add the permission that immediately follows, leave others as-is
- - (minus) - subtract the permission that immediately follows, leave others as they are

The “permissions to modify” can be, as we've discussed:
- r – read
- w – write
- x – execute

And of course the filename is the name of the file or directory. We can also list multiple permission changes, separated by commas.

Special Note on permissions for directories:

- “Executing” a directory makes no sense. This permission has been modified slightly. The “x” permission on a directory gives permission to access and search its contents. Without this, access to the directory is denied, even if “read” and “write” permissions have been granted. Cannot “cd”, cannot “find”, cannot “touch”, cannot “ls”.
- Sticky bit – This permission is specified with a “t”. It is used on directories to tell the kernel not to allow one user to delete files owned by another user, even if the containing directory is world-writable. Thus, “t” becomes a fourth way to control access permissions.

Let's look at some examples:

Example 1:
We have a file named “booklist.txt” that we want to allow the group to read only:
```
chmod g=r booklist.txt
```

Example 2:
We have a file named “dates.dat” that we want to allow the world to read from and write to:
```
chmod o=rw dates.dat
```

Example 3:
Revoke all permssions on “files.lst” from the group:
```
chmod g= files.lst
```

Example 4:
We have a directory named “photos” that we want to give everyone access to, but not to delete photos owned by other users. In other words, we want to set the sticky bit on the photos directory. Again, this is done with the “t” permission:

```
chmod +t photos
```

Example 4:
We have a file named “bonus.sh” that we want to allow the user to read from, write to, and execute, and everyone else to read only:

```
chmod a=r,u+x bonus.sh
```

Example 5:
We have a file named “inventory.sh” that we want to allow the user to read from, write to, and execute. The group should be able to read from it and execute it, and everyone else should only be able to execute it:

```
chmod u=rwx,g=rx,o=x inventory.sh
```

Example 6:
We can also use the “=” to copy permissions from one type of user to another, as in this example. We have a file called “book_manager.sh”. The owner has access to read, write, and execute this file. We want to grant these exact permissions to the group:

```
chmod g=u book_manager.sh
```

Example 7:
We can use the “+” and “-” to add and subtract permissions while copying them from one type of user to another. We have a file called “manage_payroll.sh”. The owner has access to read, write, and execute this file. We want the group to be able to read and execute this file, but not write:

```
chmod g=u-w manage_payroll.sh
```

Permisison Shortcuts

Sometimes, the chmod commandline can get long and tedious. There is another way to specify permissions for files and directories. The syntax is as follows:

```
chmod xxx [file or directory name]
```

While we are using only three digits in this example, there is a fourth. We can use the fourth digit to control the extra “sticky bit”, and other, less-used permissions. For the sake of simplicity, we will just use three.

The first “x” is for user permissions, the second is for group permissions, and the third is for world permissions.

The “x” characters refer to one of the following:

0 – no permissions whatsoever
1 – execute
2 – write
3 – write, execute

4 – read
5 – read, execute
6 – read, write
7 – read, write, execute

We can see that “execute” has a value of 1, “write” has a value of 2, and “read” has a value of 4. The values 3, 5, 6, and 7 are sums of these “read”, “write”, and “execute” values.

Let's take a look at some examples:

Example 1:
We have a file “list.txt” that we want to give read access to everyone:
```
chmod 444 list.txt
```

Example 2:
We have a file “employees.dat” that we want to give read access to everyone, and also write access to the user:
```
chmod 644 employees.dat
```

Example 3:
We have a file “run.sh” that we want to give read, write, and execute access to the user, read and write access to the group, and read-only access to everyone else:
```
chmod 764 run.sh
```

Changing Owners and Groups

`chown`

There are two commands that we can use to change the owning user and group of a file. The first is `chown` which means “change owner”.

The syntax for this is:

```
chown [user]:[group] [file or directory]
```

Let's look at an example:

I have a file called “systems.lst”. I need to change its owner to a user called `jenny` who is in the `humanresources` group:

```
chown jenny:humanresources systems.lst
```

`chgrp`

If we just want to change the group, we can use another command, called `chgrp` which means “change group”.

The syntax for this command is:

```
chgrp [group name] [file or directory name]
```

Here is an example:
There is a file called “run.sh” that we want to change to be owned by the development group:
```
chgrp development run.sh
```

Other Commands

- `useradd` – add a system user
- `userdel` – remove a system user
- `usermod` – modify an existing system user. This can be used to perform several operations on a user's account.
- `groupadd` – add a group to the system
- `groupdel` – remove a group from the system
- `groupmod` – modify a group

Filesystem Operation Commands

- `rm` – remove a file
- `mv` – move a file
- `cp` – copy a file
- `cat` – view a file's contents
- `less` – view a file's contents in a viewer
- `mkdir` – make a directory
- `rmdir` – remove a directory
- `ls` – view the contents of a directory
- `touch` – Change a file timestamp. (Can also create a file with the specified name)