

Unix/IP Preparation Course

May 29, 2011

Exercises: Permissions

REFERENCE

Reference: Shah, Steve, "*Linux Administration: A Beginner's Guide*", 2nd. ed., Osborne press, New York, NY.

If you look at files in a directory using "ls -al" you will see the permissions for each file and directories. Here is an example:

```
drwxrwxr-x   3 hervey  hervey      4096 Feb 25 09:49 directory
-rwxr--r--  12 hervey  hervey      4096 Feb 16 05:02 file
```

The left column is important. You can view it like this:

| Type | User | Group | Other | Links | owner | group | size | date | hour | name |
|------|------|-------|-------|-------|--------|--------|------|--------|-------|-----------|
| d | rwX | rwX | r-x | 3 | hervey | hervey | 4096 | Feb 25 | 09:49 | directory |
| - | rwX | r | r | 12 | hervey | hervey | 4096 | Feb 16 | 05:02 | file |

So, the directory has r (read), w (write), x (execute) access for the User and Group. For Other it has r (read) and x (execute) access. The file has read/write/execute access for User and read only access for everyone else (Group and Other).

To change permissions you use the "chmod" command. chmod uses a base eight (octal) system to configure permissions. Or, you can use an alternate form to specify permissions by column (User/Group/Other) at a time.

Permissions have values like this:

| Letter | Permission | Value |
|--------|------------|-------|
| R | read | 4 |
| W | write | 2 |
| X | execute | 1 |
| - | none | 0 |

Thus you can give permissions to a file using the sum of the values for each permission you wish to give for each column. Here is an example:

| Letter | Permission | Value |
|--------|--------------------------|-------|
| --- | none | 0 |
| r-- | read only | 4 |
| rw- | read and write | 6 |
| rwX | read, write, and execute | 7 |
| r-x | read and execute | 5 |
| --X | execute | 1 |

This is just one column. Thus, to give all the combinations you have a table like this:

| Permissions | Numeric equivalent | Description |
|-------------|--------------------|--|
| -rw----- | 600 | User has read & execute permission. |
| -rw-r--r-- | 644 | User has read & execute. Group and Other have read permission. |
| -rw-rw-rw- | 666 | Everyone (User, Group, Other) have read & write permission (dangerous?) |
| -rwx----- | 700 | User has read, write, & execute permission. |
| -rwxr-xr-x | 755 | User has read, write, & execute permission. Rest of the world (Other) has read & execute permission (typical for web pages or 644). |
| -rwxrwxrwx | 777 | Everyone has full access (read, write, execute). |
| -rwx--x--x | 711 | User has read, write, execute permission. Group and world have execute permission. |
| drwx----- | 700 | User only has access to this directory. Directories require execute permission to access. |
| drwxr-xr-x | 755 | User has full access to directory. Everyone else can see the directory. |
| drwx--x--x | 711 | Everyone can list files in the directory, but Group and Other need to know a filename to do this. |

Note: “Other” is often referred to as “World”.

1.) CHANGING FILE PERMISSIONS

If you are logged in as the *root* user please do the following:

```
# exit
```

To become a normal user, like *afnog*. Your prompt should change to include a “\$” sign.

```
$
```

Once logged in we'll create a file and set permissions on it in various ways.

```
$ cd
$ echo "test file" > working.txt
$ chmod 444 working.txt
```

In spite of the fact that the file does not have write permission for the owner, the owner can still change the file's permissions so that they can make it possible to write to it. Do you find this to be strange?

```
$ chmod 744 working.txt
```

Or, you can do this by using this form of `chmod`:

```
$ chmod u+w working.txt
```

To remove the read permission for the User on a file you would do

```
$ chmod u-r working.txt
```

Or, you can do something like:

```
$ chmod 344 working.txt
```

You probably noticed that you can use the "-" (minus) sign to remove permissions from a file. Try reading your file:

```
$ cat working.txt
```

What happened? Uh oh! You can't read your file. Please make the file readable by you:

```
$ chmod ??? working.txt
```

Ask your instructor for help if you don't know what to put in for "???". Or, look at your reference at the start of these exercises to figure this out.

2. PROGRAM EXECUTION, PRIVILEGES & SUDO

As a general user you can see that there is a file called "*master.passwd*":

```
$ ls /etc/master.passwd
```

But, you *cannot* see its contents:

```
$ less /etc/master.passwd
```

As a general user, however, you can see the `/etc/master.passwd` file if you do the following:

```
$ sudo less /etc/master.passwd
```

What is `sudo`? Read about it:

```
$ man sudo
```

3. CREATE A NEW GROUP

```
$ sudo pw groupadd team1
```

Prove that it really exists:

```
$ sudo grep team1 /etc/group
```

Now let's place our *afnog* user in this new group:

```
$ groups
```

You can see that *afnog* is a member of the *wheel* group.

```
$ sudo pw usermod afnog -G team1
```

You won't be able to use your new group until you have logged in and out from your account, or have simulated this process by doing this:

```
$ su - afnog
```

Now try typing:

```
$ groups
```

Using groups like this can be useful for working in teams on a project, giving access to web directories, etc.

4. GIVE GROUP ACCESS TO A FILE

Do the following:

```
$ cd
$ echo "This is our group test file" > group.txt
$ chgrp team1 group.txt
```

What permissions does the file have now?

```
$ ls -l group.txt
```

You should see something like:

```
-rw-r--r--  1 afnog  team1  10B May 29 10:14 group.txt
```

How would you give members of the group *team1* read/write access to this file? Before you look below try solving this on your own.

We'll use the numeric *chmod* functionality.

```
$ chmod 664 group.txt
```

Alternatively you could have typed:

```
$ chmod g+w group.txt
```

Look at the file's permissions:

```
$ ls -l group.txt
```

You should see something like:

```
-rw-rw-r-- 1 sysadmin team1 19 Jul 24 10:14 group.txt
```

5. MAKE A FILE EXECUTABLE

Do this exercise as the *afnog* user.

```
$ cd
$ touch hello
$ vi hello
```

Now add a single line to the file that reads:

```
echo 'Hello, world!'
```

Do this by pressing the “i” key to go in to insert mode in vi. Type in the text, then press the ESCape key, and then press “:wq” to write and quit from the file.

At this point let's try to run this file:

```
$ ./hello
```

You'll probably see something like:

```
bash: ./hello: Permission denied
```

This implies that the file is not executable. We need to set the file's permission to be executable by our *afnog* user. How would you do this?

```
$ chmod 755 hello
```

would work. Now try running the file:

```
$ ./hello
```

You should see

```
Hello, world!
```

on your screen. You've just written your first script!

Now set your `hello` file to be readable, but not executable by the *afnog* user and executable by the Group and by Other. Can you figure out how to do this on your own?

What happens if you now type?

```
$ ./hello
```

Why does this happen? If you execute the file as a different user it will still work! Does this seem odd? (**Hint:** think “left to right”)

CONCLUSION

What's the “.” about?

In our example above when you typed “hello” the file “hello” is in your home directory. Your home directory *is not* in your default path as configured for the bash shell. Thus, bash will not find the hello file, even though it's in the same directory where you are typing the command. By using “.” before the filename we tell bash to explicitly look in the same directory for the file to execute.

What about `setuid`, `setgid` and sticky bits?

Have a look at:

http://www.onlamp.com/pub/a/bsd/2000/09/13/FreeBSD_Basics.html