Introduction to Unix

May 24, 2008

Exercises: Privileges

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
login: inst
```

That is, log out and log back in as the *inst* user.

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. CREATE A NEW GROUP

To create a group you must be logged in as the *root*. From your past exercises you should be able to do this:

```
     # pw groupadd team1
```

Prove that it really exists:

```
     # grep team1 /etc/group
```

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

```
     # pw usermod inst -G wheel,team1
```

This can be useful for working in teams on a project, giving access to web directories, etc.

3. GIVE GROUP ACCESS TO A FILE

Log in as *inst* first (either log out of root, or use another virtual terminal and log in there). We'll create a file and allow members of the team1 group r/w access to the file:

```
     $ 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 inst  team1  16 May 24 14:14 group.txt
```

How would you give members of the group team1 read/write access to this file? Before 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:

```
-rw-rw-r--  1 inst  team1  16 May 24 14:14 group.txt
```

5. PROGRAM EXECUTION, PRIVILEGES & SUDO

For this exercise you will need to use both the *inst* and the *root* account. You should log in to one virtual terminal as *root* and another as *inst*.

As a general user you can see the status of network devices on your machine:

```
$ ifconfig
```

Your machines come with Intel Gigabit network cards. This is represented by the "em0" network interface, or "the first Intel Gigabit network card" in your machine. If you had two such cards you would see em0 and em1.

The output of the above command looks something like this:

```
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=1b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING>
        inet6 fe80::21e:4fff:fe94:4de%em0 prefixlen 64 scopeid 0x1
        inet 0.0.0.0 netmask 0xffffff00 broadcast 0.255.255.255
        ether 00:1e:4f:94:04:de
        media: Ethernet autoselect
        status: no carrier
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
        inet 127.0.0.1 netmask 0xff000000
```

The `ifconfig` command can execute privileged instructions as well. Try to do this as the *inst* user:

```
    $ ifconfig em0 196.200.218.nnn/24
```

Note that “nnn” is the IP address we are assigning to your PC. We start at 100 with the first PC by the classroom entrance, and we go around the room and end with the last PC, which has the address 123.

You should see an error like this:

```
    ifconfig: ioctl (SIOCDIFADDR): permission denied
```

This is your indication that the user *inst* does not have the permissions to change the configuration of the em0 device. By default you must be *root* to make this change.

Now go to your virtual terminal where you are logged in as the *root* user and do the following:

```
    # ifconfig em0 196.200.218.nnn/24
    # route add default 196.200.218.254
```

If you don't understand the details of this don't worry. We'll go over it on later.

Now let's quickly install a package to let us do some system administration as a regular user. Do the following:

```
    # pkg_add ftp://196.200.223.1:/pub/sae/sudo-1.6.9.6.tbz
    # rehash                          [tell your shell about the new package]
```

Now we need to quickly edit a configuration file and make a single, small change:

```
    # visudo
```

Search for the line that reads:

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

And, remove the first character, the “#” from the line, and the space after it. In vi place your cursor over the “#” and press the x key twice to delete the character “#” and the space.

The line should now look like:

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

Press `:wq` to write your changes and quit.

Remember when you created your *inst* account you used the command:

```
    # pw useradd inst -G wheel -m -s /usr/local/bin/bash
```

This means that we placed the *inst* account in the group called “wheel”. Two ways to verify this include:

```
# groups inst
# grep wheel /etc/group
```

What does `sudo` do?

```
# man sudo
```

As you can see `sudo` let's a regular user execute privileged commands. Let's go do this. Switch to your virtual terminal where you are logged in as the *inst* user and do the following:

```
$ sudo ifconfig em0 down
```

The first time you run the `sudo` command during a login session you will be prompted like this:

```
We trust that you have received the usual lecture from the
local System Administrator. It usually boils down to these
three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

Password:
```

At this point you type in the *inst* user password and you can execute the `ifconfig` command with privileges. The next time you use `sudo` you may not need to type in a password depending on how long it's been since you last used `sudo`.

The `sudo` utility is a useful way to execute privileged commands in Unix without having to run as the superuser. This is a good thing as it is easy to make mistakes as the *root* user and possibly cause problems.

6. MAKE A FILE EXECUTABLE

Do this exercise as the *inst* 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:

```
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 *inst* 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 *inst* 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?

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.

Why didn't "hello" execute?

The FreeBSD kernel first determines your uid. If your uid matches the User (owner) field of the file, then the permissions for User are checked and FreeBSD *stops checking* at this point. Thus, the owner cannot execute the file. Another user, however, *can* execute the file as the kernel will check to see if the user is a member of the file's Group, if not, then it will see how the files Other (world) permissions are set. In this case Other was set with the execute bit on, thus another user *can* execute the file. This may seem odd, but it's one way to give more explicit control over who can do what to a file.

What about setuid, setgid and sticky bits?

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