Permissions

The Unix operating system (and likewise, Linux) differs from other computing environments in that it is not only a *multitasking* system but it is also a *multi-user* system as well.

What exactly does this mean? It means that more than one user can be operating the computer at the same time. While your computer will only have one keyboard and monitor, it can still be used by more than one user. For example, if your computer is attached to a network, or the Internet, remote users can log in via **`telnet`** or **`ssh`** (secure shell) and operate the computer. In fact, remote users can execute X applications and have the graphical output displayed on a remote computer. The X Windows system supports this.

The multi-user capability of Unix is not a recent "innovation," but rather a feature that is deeply ingrained into the design of the operating system. If you remember the environment in which Unix was created, this makes perfect sense. Years ago before computers were "personal," they were large, expensive, and centralized. A typical university computer system consisted of a large mainframe computer located in some building on campus and *terminals* were located throughout the campus, each connected to the large central computer. The computer would support many users at the same time.

In order to make this practical, a method had to be devised to protect the users from each other. After all, you could not allow the actions of one user to crash the computer, nor could you allow one user to interfere with the files belonging to another user.

This lesson will cover the following commands:

- **`chmod`** - modify file access rights
- **`su`** - temporarily become the superuser
- **`chown`** - change file ownership
- **`chgrp`** - change a file's group ownership

File permissions

Linux uses the same permissions scheme as Unix. Each file and directory on your system is assigned access rights for the owner of the file, the members of a group of related users, and everybody else. Rights can be assigned to read a file, to write a file, and to execute a file (i.e., run the file as a program).

To see the permission settings for a file, we can use the **`ls`** command as follows:

```
[me@linuxbox me]$ ls -l some_file
```

```
-rw-rw-r-- 1 me   me   1097374 Sep 26 18:48 some_file
```

We can determine a lot from examining the results of this command:

- The file "some_file" is owned by user "me"
- User "me" has the right to read and write this file
- The file is owned by the group "me"
- Members of the group "me" can also read and write this file
- Everybody else can read this file

Let's try another example. We will look at the **bash** program which is located in the **/bin** directory:

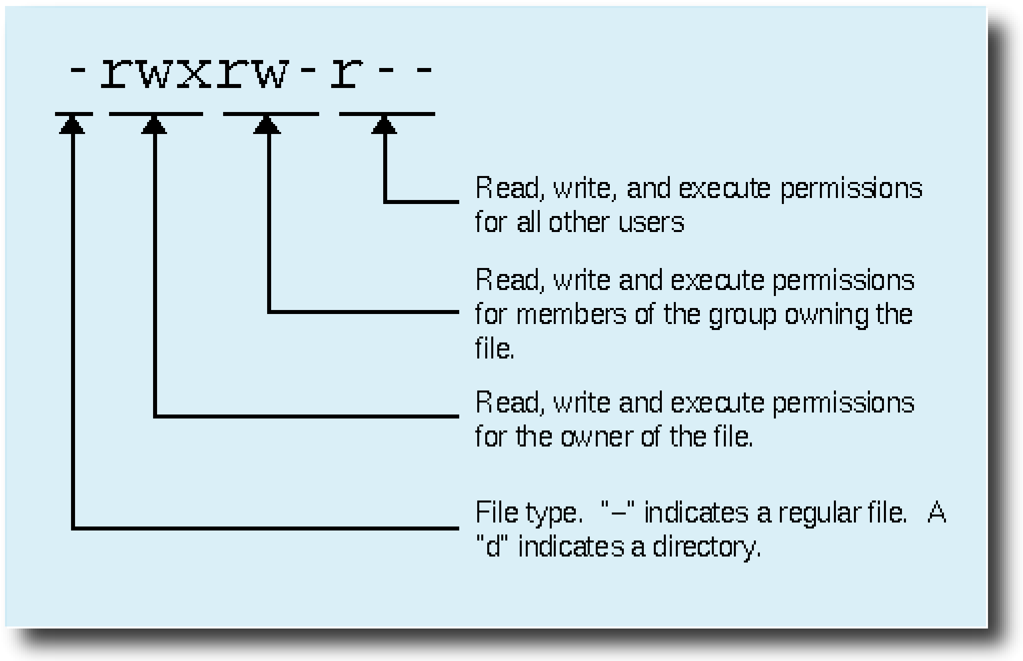
```
[me@linuxbox me]$ ls -l /bin/bash

-rwxr-xr-x 1 root root  316848 Feb 27  2000 /bin/bash
```

Here we can see:

- The file "/bin/bash" is owned by user "root"
- The superuser has the right to read, write, and execute this file
- The file is owned by the group "root"
- Members of the group "root" can also read and execute this file
- Everybody else can read and execute this file

In the diagram below, we see how the first portion of the listing is interpreted. It consists of a character indicating the file type, followed by three sets of three characters that convey the reading, writing and execution permission for the owner, group, and everybody else.


-rwxrw-r--
Read, write, and execute permissions for all other users
Read, write and execute permissions for members of the group owning the file.
Read, write and execute permissions for the owner of the file.
File type. "–" indicates a regular file. A "d" indicates a directory.

chmod

The **chmod** command is used to change the permissions of a file or directory. To use it, you specify the desired permission settings and the file or files that you wish to modify. There are two ways to specify the permissions, but I am only going to teach one way.

It is easy to think of the permission settings as a series of bits (which is how the computer thinks about them). Here's how it works:

```
rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx --- --- = 111 000 000

and so on...

rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4
```

Now, if you represent each of the three sets of permissions (owner, group, and other) as a single digit, you have a pretty convenient way of expressing the possible permissions settings. For example, if we wanted to set `some_file` to have read and write permission for the owner, but wanted to keep the file private from others, we would:

```
[me@linuxbox me]$ chmod 600 some_file
```

Here is a table of numbers that covers all the common settings. The ones beginning with

"7" are used with programs (since they enable execution) and the rest are for other kinds of files.

Value	*Meaning*
777	*(rwxrwxrwx)* No restrictions on permissions. Anybody may do anything. Generally not a desirable setting.
755	*(rwxr-xr-x)* The file's owner may read, write, and execute the file. All others may read and execute the file. This setting is common for programs that are used by all users.
700	*(rwx------)* The file's owner may read, write, and execute the file. Nobody else has any rights. This setting is useful for programs that only the owner may use and must be kept private from others.
666	*(rw-rw-rw-)* All users may read and write the file.
644	*(rw-r--r--)* The owner may read and write a file, while all others may only read the file. A common setting for data files that everybody may read, but only the owner may change.
600	*(rw-------)* The owner may read and write a file. All others have no rights. A common setting for data files that the owner wants to keep private.

Directory permissions

The **`chmod`** command can also be used to control the access permissions for directories. In most ways, the permissions scheme for directories works the same way as they do with files. However, the execution permission is used in a different way. It provides control for access to file listing and other things. Here are some useful settings for directories:

Value	*Meaning*
777	*(rwxrwxrwx)* No restrictions on permissions. Anybody may list files, create new files in the directory and delete files in the directory. Generally not a good setting.
755	*(rwxr-xr-x)* The directory owner has full access. All others may list the directory, but cannot create files nor delete them. This setting is common for directories that you wish to share with other users.
700	*(rwx------)* The directory owner has full access. Nobody else has any rights. This setting is useful for directories that only the owner may use and must be kept private from others.

Becoming the superuser for a short while

It is often useful to become the superuser to perform important system administration tasks, but as you have been warned (and not just by me!), you should not stay logged on as the superuser. Fortunately, there is a program that can give you temporary access to the superuser's privileges. This program is called **su** (short for superuser) and can be used in those cases when you need to be the superuser for a small number of tasks. To become the superuser, simply type the **su** command. You will be prompted for the superuser's password:

```
[me@linuxbox me]$ su
Password:
[root@linuxbox me]#
```

After executing the **su** command, you have a new shell session as the superuser. To exit the superuser session, type **exit** and you will return to your previous session.

Changing file ownership

You can change the owner of a file by using the **chown** command. Here's an example: Suppose I wanted to change the owner of `some_file` from "me" to "you". I could:

```
[me@linuxbox me]$ su
Password:
[root@linuxbox me]# chown you some_file
[root@linuxbox me]# exit
[me@linuxbox me]$
```

Notice that in order to change the owner of a file, you must be the superuser. To do this, our example employed the **su** command, then we executed **chown**, and finally we typed **exit** to return to our previous session.

chown works the same way on directories as it does on files.

Changing group ownership

The group ownership of a file or directory may be changed with **chgrp**. This command is used like this:

```
[me@linuxbox me]$ chgrp new_group some_file
```

In the example above, we changed the group ownership of `some_file` from its previous group to "new_group". You must be the owner of the file or directory to perform a **`chgrp`**.
