Changing Permissions and Ownership

coursera.org/learn/linux-tools-for-developers/supplement/bq211/changing-permissions-and-ownership

Changing file permissions is done with **chmod**, while changing file ownership is done with **chown**, and changing the group is done with **chgrp**.

There are a number of different ways to use **chmod**. For instance, to give the **owner** and **world** execute permission, and remove the **group** write permission:

```
$ ls -l a_file

-rw-rw-r-- 1 coop coop 1601 Mar  9 15:04 a_file

$ chmod uo+x,g-w a_file

$ ls -l a_file

-rwxr--r-x 1 coop coop 1601 Mar  9 15:04 a_file
```

where **u** stands for user (owner), **o** stands for other (world), and **g** stands for group.

This kind of syntax can be difficult to type and remember, so one often uses a shorthand which lets you set all the permissions in one step. This is done with a simple algorithm, and a single digit suffices to specify all three permission bits for each entity. This digit is the sum of:

- 4 if read permission is desired
- 2 if write permission is desired
- 1 if execute permission is desired.

Thus, 7 means read/write/execute, 6 means read/write, and 5 means read/execute.

When you apply this when using **chmod**, you have to give three digits for each degree of freedom, such as in:

```
$ chmod 755 a_file

$ ls -l a_file

-rwxr-xr-x 1 coop coop 1601 Mar  9 15:04 a_file
```

Changing the group ownership of the file is as simple as doing:

```
$ chgrp aproject a_file
```

and changing the ownership is as simple as:

```
$ chown coop a_file
```

You can change both at the same time with:

```
$ chown coop.aproject a_file
```

where you separate the owner and the group with a period.

All three of these commands can take an **-R** option, which stands for recursive. For example:

```
$ chown -R coop.aproject .
$ chown -R coop.aproject subdir
```

will change the owner and group of all files in the current directory and all its subdirectories in the first command, and in **subdir** and all its subdirectories in the second command.

These commands can only do what you already have the right to do. You cannot change the permissions on a file you do not own, for example, or switch to a group you are not a member of. To do such changes, you must be root, prefixing **sudo** to most of the above commands.