COMP 2718: Permissions: Part 1

By: Dr. Andrew Vardy

Adapted from the notes of Dr. Rod Byrne

Outline

- Permissions
- Owners, Group Members, and Everyone Else
- Read, Writing, and Executing
- Digression: Number Systems
- `chmod` - Change File Mode

Permissions

Unix-like OS's have always been **multi-user** systems. At the time of Unix's development computers were large and had to be shared by many users. Therefore, Unix has many features to protect the system from careless or malicious users, and to protect users from each other!

We're going to cover **permissions** which is one of Unix's fundamental security mechanisms, described in chapter 9 of the textbook. Along the way, we'll introduce the following commands:

- `id` – Display user identity
- `chmod` – Change a file's mode

The following commands will be covered in Part 2.

- `umask`, `su`, `sudo`, `chown`, `chgrp`, and `passwd`

Owners, Group Members, and Everyone Else

All files and directories are **owned** by some user. The owner has control over **permissions**, that is the access rights for that file (or directory). Users can belong to **groups** that may also have permission to access the file. The owner can also set the permissions for **everybody else** (a.k.a. **'the world'**).

Who are you?

A user is identified by their **user ID** (**uid**) which is mapped to a username. Each user also has a **primary group ID** (**gid**) and may also belong to other groups. Execute `id` to see this information. e.g.

```
$ id
uid=1001(av) gid=1001(av) groups=1001(av),4(adm),
27(sudo),108(lpadmin),124(sambashare),999(vboxsf)
```

Details

Detailed information on user's and passwords are stored in the following files:

`/etc/passwd` User accounts
`/etc/group` Groups
`/etc/shadow` Passwords

This may differ on some systems. For example, on Mac OS X real human user accounts do not appear in `/etc/passwd`.

Read, Writing, and Executing

We have already seen in "The File System: Part 2" that a file's permissions can be viewed with `ls -l`. e.g.

```
$ ls -l filename.txt
-rw-rw-r-- 1 av av 5 Feb  7 13:34 filename.txt
```

The first 10 characters are the **file attributes**, with the first character having the following possible meanings:

Attribute	File Type
-	A regular file.
d	A directory.
l	A symbolic link. Notice that with symbolic links, the remaining file attributes are always "rwxrwxrwx" and are dummy values. The real file attributes are those of the file the symbolic link points to.
c	A *character special file*. This file type refers to a device that handles data as a stream of bytes, such as a terminal or modem.
b	A *block special file*. This file type refers to a device that handles data in blocks, such as a hard drive or CD-ROM drive.

The remaining 9 characters are called the **file mode** and represent permissions for the owner, the file's group owner, and the world:

Owner	Group	World
rwx	rwx	rwx

Attribute	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated, however this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is determined by directory attributes.	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a program and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered, e.g., cd *directory*.

Some example file mode strings. . .

File Attributes	Meaning
`-rwx------`	A regular file that is readable, writable, and executable by the file's owner. No one else has any access.
`-rw-------`	A regular file that is readable and writable by the file's owner. No one else has any access.
`-rw-r--r--`	A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world-readable.
`-rwxr-xr-x`	A regular file that is readable, writable, and executable by the file's owner. The file may be read and executed by everybody else.
`-rw-rw----`	A regular file that is readable and writable by the file's owner and members of the file's group owner only.
`lrwxrwxrwx`	A symbolic link. All symbolic links have "dummy" permissions. The real permissions are kept with the actual file pointed to by the symbolic link.
`drwxrwx---`	A directory. The owner and the members of the owner group may enter the directory and, create, rename and remove files within the directory.
`drwxr-x---`	A directory. The owner may enter the directory and create, rename and delete files within the directory. Members of the owner group may enter the directory but cannot create, delete or rename files.

Digression: Number Systems

In our introduction to the `chmod` there are references to **octal numbers**. In case you don't know, lets briefly review some important number systems:

System	Base	Symbols	Used by Whom
Binary	2	0, 1	Computers
Decimal	10	0, 1, ... 9	Humans
Octal	8	0, 1, ... 7	Programmers
Hexadecimal	16	0, 1, ... 9, A, B, ... F	Programmers

(Computers use all systems when doing input/output.)
Binary makes sense for computers since the basic means of information representation is electrical (charged or not charged).

Why does anyone use octal and hexadecimal?

Octal and hexadecimal make sense for viewing binary. A binary representation of the colour red using 8-bits for each of the red, green, and blue channels would look like this:

```
11111111 00000000 00000000
```

In octal, each numeral represents 3 bits ($2^3 = 8$). So the above number is 77600000.

In hexadecimal, each numeral represents 4 bits ($2^4 = 16$). So the above number is FF0000.

With either octal or hexadecimal there is an easy mapping to/from binary, which is why programmers use these number systems

chmod - Change File Mode

chmod is used to **ch**ange a file's **mod**e. chmod can take arguments in two forms **octal** and **symbolic notation**.

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

This is an example of the octal form of chmod:

```
$ ls -l fresh.txt
-rw-r--r--  1 av  staff  94  5 Feb 09:46 fresh.txt
$ chmod 620 fresh.txt
$ ls -l fresh.txt
-rw--w----  1 av  staff  94  5 Feb 09:46 fresh.txt
```

Not all 8 possibilities are common. In fact, 2 (`-w-`) is pretty uncommon. Here the ones you will more likely see and use:

`7 (rwx), 6 (rw-), 5 (r-x), 4 (r--), and 0 (---)`

The symbolic notation has three parts: **who** the change will affect, the **operation** to perform, and what **permission** to set. To define who is affected use some combination of 'u', 'g', 'o', and 'a':

Symbol	Meaning
u	Short for "user" but means the file or directory owner.
g	Group owner.
o	Short for "others," but means world.
a	Short for "all." The combination of "u", "g", and "o".

If no character is specified, "all" is assumed.

Next, a “+”, “-”, or “=” specifies the operation to perform:

Character	Operation
+	Permission to be added
-	Permission to be taken away
=	Specified permissions applied; Others taken away

The final character gives the permission to set: “r”, “w”, or “x”.

Some examples of symbolic notation strings:

Notation	Meaning
`u+x`	Add execute permission for the owner.
`u-x`	Remove execute permission from the owner.
`+x`	Add execute permission for the owner, group, and world. Equivalent to `a+x`.
`o-rw`	Remove the read and write permission from anyone besides the owner and group owner.
`go=rw`	Set the group owner and anyone besides the owner to have read and write permission. If either the group owner or world previously had execute permissions, they are removed.
`u+x,go=rx`	Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.