COMP 2718: Permissions: Part 2

By: Dr. Andrew Vardy

Adapted from the notes of Dr. Rod Byrne

Outline

- Permissions: Part 2
- `umask` - Set Default Permissions
- Special Permissions
- Changing Identities
- `su` - Run a New Shell as Another User
- `sudo` - Execute a Command as Another User
- `chown` - Change File Owner and Group

Permissions: Part 2

Here we will continue to look at file permissions and cover the following commands:

- `umask` – Set the default file permissions
- `su` – Run a shell as another user
- `sudo` – Execute a command as another user
- `chown` – Change a file's owner

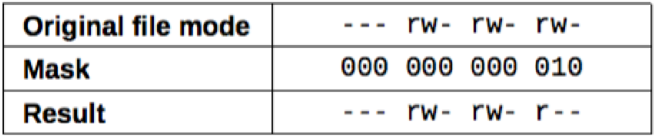
`umask` - Set Default Permissions

The umask is a number that alters the default permissions for a new file. It uses the concept of a “bit mask”. Each bit that is set (value 1) is meant to prevent a new file from having that permission. For example, here are two common values:

umask (octal)	umask (binary)	Meaning
0002	000 000 000 010	Not writable by others
0022	000 000 010 010	Not writable by group or others

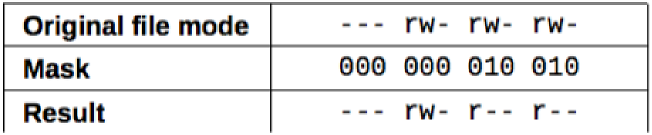
`umask` is a command that shows the current umask value (if no argument is specified). If called with a single argument, that argument is then interpreted as the new umask value for the shell.

Note that `umask` acts like a filter, turning off certain permissions. So it has to be understood in relation to the default permissions—usually 666 meaning `rw-rw-rw-`. Consider the application of the umask 0002:



Original file mode	`--- rw- rw- rw-`
Mask	`000 000 000 010`
Result	`--- rw- rw- r--`

. . . and umask 0022:



Original file mode	`--- rw- rw- rw-`
Mask	`000 000 010 010`
Result	`--- rw- r-- r--`

`umask` can also accept a symbolic notation similar to `chmod` but most of the time you will see the octal form.

Special Permissions

Notice that the octal numbers used by `chmod` and `umask` are usually expressed as 4 digits whereas only 3 seem necessary. This is because the first digit is used to control the following special permissions:

setuid (octal 4000) Sets the *effective user ID* to the program's owner rather than the real user. Used to give a program "superpowers" to act with the priviledges of another user (usually, the superuser).

setgid (octal 2000) Like the setuid bit, but sets the *effective group ID* to that of the file owner.

sticky bit (octal 1000) When set on directories restricts access so that only the file's owner or `root` can rename or delete files. Often used for `/tmp`.

You are unlikely to need this, but here is how you can set these special permissions using `chmod`:

```
$ chmod u+s program     # setuid
$ chmod g+s dir         # setgid
$ chmod +t dir          # sticky bit
```

More important is to recognize special permissions. Here are some example file modes:

-rw**s**r-xr-x	A file with setuid
drwxrw**s**r-x	A directory that has the setgid attribute:
drwxrwxrw**t**	A directory with the sticky bit set

For real examples, take a look at `passwd`, `su`, `sudo`, and `\tmp`

Changing Identities

The super user refers to a special account for system administration. The super user's `uid` is 0 and the username is usually `root`.

To perform system adminsitration tasks a regular user may be able to assume the identity of the super user. It might also be necessary to assume the identity of another user. The two programs used for these tasks are `su` and `sudo`...

`su` - Run a New Shell as Another User

```
su [-[l]] [user]
```

The `-l` means to use the environment (more on this later) of the target user. Strangely, for su, `-l` can be abbreviated as `-`. If no user is specified, it is assumed that `root` is intended.

Using `su` has some disadvantages over `sudo`. In particular, there is the temptation to run all commands as `root`, which degrades security. Because of this the `root` account is locked by default on the following OS's:

- Ubuntu Linux
- Mac OS X
- Probably others. . . ?

`su` remains useful for system administrators but has largely been supplanted by `sudo` (at least for typical users).

sudo - Execute a Command as Another User

sudo provides a more constrained ability to execute commands as the superuser. Use of sudo does not require knowing the root user's password (which su does require). Instead, sudo asks the user to enter his or her own password.

```
sudo command
```

The command is then run with superuser privileges. On Linux the file /etc/sudoers is used to administer sudo privileges. The commands that can be run with sudo can also be restricted.

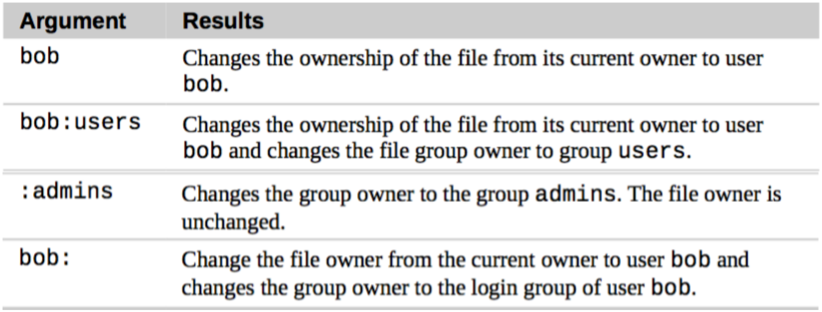
Fortunately, you don't need to type your password constantly. After entering your password you can execute subsequent sudo commands without password for 15 minutes.

chown - Change File Owner and Group

chown can change the owner and group owner of a file or directory. Superuser privileges are required (i.e. use sudo to run it).

```
chown [owner][:[group]] file...
```

Here are some example arguments to illustrate usage:



Argument	Results
bob	Changes the ownership of the file from its current owner to user bob.
bob:users	Changes the ownership of the file from its current owner to user bob and changes the file group owner to group users.
:admins	Changes the group owner to the group admins. The file owner is unchanged.
bob:	Change the file owner from the current owner to user bob and changes the group owner to the login group of user bob.