Unix security model

In Unix, each file and each directory has an owner (usually — the user who created it) and a group associated with it.

There are three different ways a file or a directory can be accessed:

	File	Directory
r (**r**ead)	view contents	list contents
w (**w**rite)	modify contents	create, rename and delete files
x (e**x**ecute)	run a program	enter (access) a directory

This creates 9 access flags:

	Owner	Group	Others
Read	r or -	r or -	r or -
Write	w or -	w or -	w or -
Execute	x or -	x or -	x or -

`ls` revisited

`ls` *[options]* arg_1 arg_2 ... will for each argument:

- if it's a file, `ls` will list it
- if it's a directory, `ls` will list its **contents** (files and subdirectories). But `ls -d` would list the directory itself.

`ls -l` will give additional information about each file and directory listed: type, permissions, owner, group and modification time.

The first field will look like `drwxrwxrwx` or `-rw-r--r--`. The first letter is "`d`" if it's a directory and "`-`" if it's a file. The next 9 letters are permissions with `r/w/x` meaning that a specific permission is granted and "`-`" meaning that it's denied.

Getting identity information — `id`, `groups`

`id` *user* gives information (numerical and symbolic) about the user, his/her primary group and the other groups he/she belongs to. Without the *user* argument, you'll get the information about user who is running `id` (yourself).

`groups` is similar to `id`, but just gives the list of groups.

Changing owner and group — `chown`, `chgrp`

`chown` *user* arg_1 arg_2 ...
`chgrp` *group* arg_1 arg_2 ...
`chown` *user.group* arg_1 arg_2 ...

`-R` option can be used to **r**ecursively traverse a directory.

Changing permissions — `chmod`

`chmod` <*permission specification*> arg_1 arg_2 ...

permission specification should have the form *u*sers mode permissions (no spaces). You can repeat several permission specifications in a single `chmod` command by putting them together separated by comas (again, no spaces).

users is a combination of letters "`u`" (owner/**u**ser) "`g`" (**g**roup) and "`o`" (**o**thers). You can use "`a`" (**a**ll) in place of "`ugo`"

mode is either "`=`" (set), "`+`" (add to existing permissions) or "`-`" (subtract from existing permissions).

permissions is a combination of letters "`r`", "`w`" and "`x`"

Examples: `chmod a+r` gives everybody read access. `chmod ug+w,o-w` grants owner and group read/write access while removing it for everybody else.

Special characters in file and directory names

What if name starts with a "-"? For example, `rmdir -l` will say "rmdir: invalid option – l" instead of removing directory "`-l`". It's even worse if you "accidentally" pick up a real option! **Solution:** use `rmdir -- -l`

What if name contains a space? For example, `rmdir a b` would try to remove two directories named "a" and "b" instead of removing directory "a b". Solution: use `rmdir "a b"` or `rmdir 'a b'`

There is an important distinction between the two cases — in the former example, the `--` option is recognized by `rmdir` itself, but in the latter one, it's the ***shell*** who recognizes quotation.