Unix security

Creating and deleting directories,and copying, moving and deleting files raises a question: how do you keep other users from messing around with your files? Or the system files?

Recall that each user has an identity, given by its username. Moreover, each user can belong to one or more groups. Membership in a group is set by the system administrator. For example, everyone in this class is a member of group `cs114`. I am also a member of group `cs114`, and I may also be a member of group `instructors`. Thus, you can be a member of more than one group.

The following commands give you identity information on yourself or someone else:

- `id` gives information on yourself
- `id` *`username`* gives information on user *`username`*
- `groups` returns the groups your are a member of
- `groups` *`username`* returns the groups user *`username`* is a member of

Permissions

Going back to security, each file and each directory has an *owner* (usually, the creator) and a group associated with it.

There are three ways in which a file or directory can be accessed: read, write or execute. They mean different things for files and for directories:

	For a file	For a directory
read (**r**)	view content	list content
write (**w**)	modify content	create, remove, delete files in directory
execute (**x**)	run program	enter directory (via `cd`)

From a file or directory point of view, there are three kind of people: the owner, members of the group, and others. Each file has read, write, and execute permissions (which I'll abbreviate as r/w/x

permissions) for each of these kind of people: r/w/x permissions for the owner, r/w/x permissions for members of the group, and r/w/x permissions for everyone else. This kind of information is summarized by a string of 9 characters of the form `xxxyyyzzz` where `xxx` represent the r/w/x permissions for the owner, `yyy` the r/w/x permissions for the members of the group, and `zzz` the r/w/x permissions for everyone else. Each set of r/w/x permissions is of the form `abc`, where `a` is either `r` or `-`, `b` is either `w` or `-`, and `c` is either `x` or `-` (you will sometimes see `s` instead of `x`; for the time being, you can assume it means the same as `x`). A `-` indicates simply that the corresponding permission is denied.

Consider the following examples:

- `rw-------` only the owner can read or write
- `rw-rw-rw-` everyone can read or write
- `---rw----` all members of the group (excluding the owner) can read or write
- `rwxrwx---` the owner and all members of the group can read, write, or execute

How do you check the permissions of a file or a directory? The command `ls` has an option that shows you the permissions of the files and directories it lists. If you type `ls -l` (the option `-l` stands for "long display"), you get output that looks like this:

```
babbage% ls -l
total 12
drwx------   6 cs114    cs114        512 Feb 18  2001 2000FA
drwx------   9 cs114    cs114        512 Oct  3 16:17 2001SP
drwxrwx---   5 cs114    cs114        512 Oct  5 11:34 HW1
drwxr-xr-x   2 cs114    cs114        512 Oct  9 13:48 bin
drwx--x---   3 cs114    cs114        512 Oct  1  2000 man
drwx--x---   3 cs114    cs114        512 Oct  1  2000 share
```

The leftmost string of characters on each line gives you type and permission information for the corresponding file. The first character is either `d` for a directory, or `-` for a file. (You will sometimes see `l` as well; this says that file is a link to another file. We'll cover links later in the course.) The following 9 characters are the permissions, as described above. Later on the line, you get the owner of the file or directory (`cs114` in all the examples above), as well as the group associated with the file or directory (`cs114` as well in all the examples above). For example, you see that the owner has read, write and execute permissions on directory `bin/`, while members of the `cs114` group have read and execute access, as do everyone else for that matter.

Changing owner and group

How do you change things such as the owner or group of a file or directory? Unix provides the following commands:

- `chown` *`username arg1 arg2 ...`* changes the owner of the files/directories *`arg1`*, ... to *`username`*
- `chgrp` *`groupname arg1 arg2 ...`* changes the group associated with the files/directories *`arg1`*, ... to *`groupname`*

To recursively change the owner (or the group) of all the files in all the subdirectories of a given directory, you can write `chown` *`username`* `-R` *`directory`* (similarly with `chgrp`).

Changing permissions

How do you change permissions on a file or a directory? The command `chmod` does this for you. The command is invoked as follows: `chmod` *`spec arg1 arg2 ...`*, changing the permissions of *`arg1`*, ... according to the specification *`spec`*.

A specification has the form ⟨*user*⟩⟨*mode*⟩⟨*permissions*⟩, meaning that you are changing according to ⟨*mode*⟩ the permissions ⟨*permissions*⟩ of the users ⟨*user*⟩, where:

- ⟨*user*⟩ is any combination of the letters `u` (the owner), `g` (the group), and `o` (all others). The letter `a` (for all) can also be used. Using `a` is like using `ugo`
- ⟨*mode*⟩ is either `+` (adding permissions), `-` (removing permissions), or `=` (setting permissions)
- ⟨*permissions*⟩ is any combination of the letters `r`, `w`, and `x`

For example,

- `chmod a+r` *`file`* adds read permissions to all (owner, group, others) to file *`file`*
- `chmod og-rwx` *`file`* removes read, write and execute permissions for the group members and everyone else to file *`file`*

You can combine multiple specifications by separating them by a comma (without any space). Hence,

- `chmod ug+w,o-w` *`file`* adds write permissions for the owner and the group, and removes write permissions for everyone else, to file *`file`*

As with `chown` and `chgrp`, you can recursively change permissions for all the files in all the subdirectories of a directory by using the `-R` option. For example, `chmod -R o-rwx foo`.