CS 114 - Fall 2004

Lecture 2 - Wednesday, September 29, 2004

Files and directories; security

Last time we discussed the `cd`, `pwd`, and `ls` commands. The following commands are also useful for dealing with files and directories. Recall than whenever we talk about a file or a directory, it actually stands for a path to a file or directory.

- `mkdir` *`directory`* - creates a *`directory`*
- `rmdir` *`directory`* - removes a *`directory`* (only if it's empty and you're not inside it)
- `touch` *`file`* - creates an empty *`file`*
- `cat` *`file`* - dumps the contents of *`file`* to the console
- `rm` *`file`* - removes *`file`*
- `mv` *`oldname newname`* - renames (moves) a file
- `mv` *`file1 file2 ... directory`* - moves files into a directory
- `cp` *`oldname newname`* - copies a file
- `cp` *`file1 file2 ... directory`* - copies files into a directory

Some commands have *options*, which affect the way they behave. Options are typically a letter preceded by a `-`. For example, if you give the option `-i` to `rm`, that is, if you write `rm -i` *`file`*, the system will prompt you for confirmation before deleting the file.

```
% rm -i file
rm: remove file (yes/no)?
```

Some commands understand many options. The command `ls` for instance, recognizes, among others, the option `-F`, which makes `ls` give you some indication of the type of each file.

```
% ls -F file3 dir/
```

It appends a `/` at the end of every directory. `ls` also takes the option `-a`, which makes `ls` display *everything* in the directory, including so-called *hidden* files. A file or directory is hidden if its name starts with a dot (`.`), for example `.hidden`. Hidden files aren't really special in any way, except that `ls` by default will not list them. Typically, programs needing configuration files will make those files hidden, otherwise they clutter up your home directory.

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` prints the groups you're a member of
- `groups` *`username`* prints the groups user *`username`* is a member of

Permissions

To ensure only the right people can perform certain operations on a given file or directory, each file or directory has associated with it an *owner* (usually, the creator) and a group, and a set of `permissions`, which describe what these `principals` can do with the file or directory.

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 contents	list contents
write (w)	modify content	create, remove, delete files in directory
execute (x)	run program	enter directory (via cd)

For each file or directory, each of these permissions is associated with three principals: the owner, the group, and everyone else. Thus, each file has read, write, and execute permissions (abbreviated r/w/x permissions) for the owner, r/w/x permissions for members of the group, 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` represents the r/w/x permissions of the owner, `yyy` the r/w/x permissions of the members of the group, `zzz` the r/w/x permissions of 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` (or even `t`) instead of `x`; assume for now that 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 and write
- `rw-rw-rw-` everyone can read and write
- `---rw----` all members of the group (excluding the owner) can read and write
- `rwxrwx---` the owner and all members of the group can read, write, and execute

How do you check the permission of a file or a directory? If you give the `ls` command the `-l` option (`-l` stands for "long display"), it will show you the permissions of the files and directories it lists:

```
% ls -l
total 64
drwxr-sr-x   3 cs114    cs114        4096 Mar  8  2002 Archive
-rw-r-----   1 cs114    cs114        3710 Aug 17  1995 README
-rw-r-----   1 cs114    cs114        5894 Aug  3  1993 X11.ReadMe
-rwxr-x---   1 cs114    cs114        5322 Feb  5  1997 cshrc.proj
-rwxr-x---   1 cs114    cs114        1429 Feb  5  1997 login.proj
drwxr-s---   2 cs114    cs114        4096 Feb 15  2001 mboxes
```

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 `Archive`, while members of the `cs114` group have read and execute access, as do everyone else for that matter.

Changing permissions

The command `chmod` changes permissions on a file or a directory. The command is invoked as follows:

```
% chmod spec arg1 arg2 ...
```

which changes the permissions of the files/directories *arg1*, ... according to the specification `spec`.

A specification has the form `<who><mode><permissions>`, as follows:

- `<who>` is any combination of the letters `u` (the owner, or user), `g` (the group), and `o` (others). The letter `a` (for all) can also be used; it is equivalent to `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 permission to all (user, group, others) to file *file*.

```
chmod go-rwx file
```

removes read write and execute permission for the group and everyone else (execpt the user) for the file *file*.

You can combine multiple specifications by separating them with a comma. Hence,

```
chmod ug+w,o-w file
```

adds write permissions for file *file* for the owner and group, but removes write permissions for everyone else.

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

```
chmod -R o-rwx directory
```

Changing the 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`. Only the super user, *root*, can change the owner of a file.
- `chgrp groupname arg1 arg2 ...` changes the group associated with the files/directories *arg1*, ... to `groupname`. Unless you are the super user, you can only change the group of a file to a group you are in.

To recursively change the owner (or the group) of all the files in all the subdirectories of a given directory, you can give the commands the `-R` option.