Understanding File Permissions

UNIX is a multi-user system, and supports a simple scheme that allows users to share certain files, and keep other files private. File permissions determine which users can access each file or directory, and how they can access them. There are three types of permissions associated with each file or directory:

- *read*: Allows the contents of a file or directory to be viewed or copied. We can use cat, more or cp on a file if we have read permission. A directory for which we have read permissions can be listed using ls.

- *write*: Allows changes to be made to the contents of a file (however we may not be able to move it, rename it, or delete it). Having write permissions for a directory means that we can create, move or remove files in that directory.

- *execute*: If a file has execute permissions, then we can run the program represented by the file. If a directory has execute permissions, then we can cd into that directory.

These permissions are indicated by r, w, and x respectively. Every file and directory has an owner associated with it. The owner is usually the person who created the file or directory. Only the owner is allowed to change permissions. To see the file or directory permissions, we can use the ls -alF command. In the following sample output, the first column shows the permissions.

```
drwx--x--x   5 ema      guest        512 May 03 15:52 ./
drwxr-xr-x   4 root     root         512 May 03 15:20 ../
-rw-------   1 ema      guest       5194 May 13 09:59 .cshrc
-rw-------   1 ema      guest        526 May 27 15:53 .login
-rw-------   5 ema      guest        124 May 28 15:28 .logout
-rw-r-----   4 ema      guest       9512 May 10 15:21 banner.gif
drwx------   1 ema      guest        512 May 13 09:59 document
-rwx------   1 ema      guest    5275326 May 27 15:53 emacs*
-rw-------   5 ema      guest      11512 May 28 15:28 hello.c
-rwxr--r--   4 ema      guest        512 May 19 15:27 home.html
drwx------   1 ema      guest      25194 Jun 16 09:59 temp/
```

The first character in this column is d if the name represents a directory, or a - if it represents a file. The next three characters (e.g., rw- in the case of banner.gif) tell us how the owner of the file can access it: r means the owner has read permissions, w means the owner has write permission, and x means that he/she has execute permissions. Hence the owner of the file banner.gif can read it or write to it, but can not execute it. In this case, since the file is not an executable program, the owner does not need execute permissions.

The next three characters (r-- for the file banner.gif) show the group permissions for the file. A group is created by the system administrator and represents a collection of one or more existing users (in this case the group is called guest). Users who are part of the guest group can read (but cannot write or execute) banner.gif. Finally, the last three characters --- specify the permissions for all other users. Other users can not read, write or execute file banner.gif.

The chmod Command

To change permissions, we use the chmod command. A user can only use chmod on the files or directories that he/she owns. The general format of the command is:

```
chmod instruction filename
```

where filename is the name of the file whose permissions are to be changed and instruction indicates the change(s) desired. The instruction is specified as follows:

```
[ugoa][+|-|=][rwxa]
```

The character(s) before the plus + or minus - sign specify whose permissions are being modified: u for the owner (user), g for the group, o for other users, or a for all (that is, user, group and other).

The next character is + to add permissions, or - to remove permissions.

The character(s) after the plus + or minus - sign specify which permissions to add or remove: r for read permissions, w for write permissions, x for execute permissions, or a for all permissions (that is, read, write and execute). The table below shows a summary of the possible values for the permissions instruction as a sum of its parts:

Scope	Action	Permission
u owner (user)	+ add permission - remove permission = set permission	r read
g group	+ add permission - remove permission = set permission	w write
o other	+ add permission - remove permission = set permission	x execute
a all users	+ add permission - remove permission = set permission	a all permissions

For example, using the example directory illustrated above,

```
% chmod a+r hello.c
```

will give everybody (including the owner and other members of the group *guest*), read permission for the file hello.c. If we did a ls -l command, we would see:

```
-rw-r--r--   5 ema      guest      11512 May 28 15:28 hello.c
```

It is possible to set more than one class and more than one permission at the same time. For example,

```
% chmod u+x,a+rw hello.c
```

would add execute permission for the owner, and give everybody read and write permissions for the file hello.c.

Alternatively, the first argument to chmod can be digits that represent the octal (base 8) equivalent of the eight possible permission combinations, with 0 being no permissions set and 7 being all permissions set for a given set of users. For example, the command chmod 640 name sets the user permissions to read and write, and sets the group permissions to read only, since the bit representation of this permissions configuration is 100 or r--. In the same way, 1 represents executable only (--x) and 5 represents read and executable permissions only (r-x). The following table demonstrates permission settings represented as binary and octal numbers:

Method	User	Group	Others
Setting	rwx	r-x	---
Binary	111	101	000
Octal	7	5	0

Giving other users *write* permission to a file can be dangerous, since they can accidentally change, corrupt or delete the contents of the file.

The groups Command

Let us now discuss the purpose of the group category of Unix file permissions. Suppose that a user (say, Paul) is working on a software project as part of an eight person team. Every person in Paul's team needs access to the source code for the project. One way of doing this would be to use the command chmod a+rwx file for every file that the team is sharing. The problem with this is that all users on the system would be able to access the team's source code. The solution to this problem is the concept of a group.

If Paul and his team members were all part of the same Unix group, then granting team access to files would become somewhat simpler. Paul could simply issue the command chmod g+rwx file.

The groups command is used to list all of the groups that a user is a part of. For example,

```
% whoami
t5h2

% groups
undergrad cc015C csuse
```

In this example the whoami command prints out the login ID of the user who started the current shell. The groups command with no arguments lists the groups for the current userID. We can also specify a userID to get a listing of groups for the specific user to which that userID belongs.

```
% groups cs300
ta
```

System administrators for Unix systems, known as super-user or root, are able to create and delete groups on the system, as well as add and delete users from pre-existing groups.

The chgrp Command

This utility allows a user to change the group of the files that he/she owns. For instance, consider the following example:

```
% whoami
joeluser
% groups joeluser
undergrad novicegroup
% ls -laF
drwx------ 5 joeluser undergrad 512  Jun  2 11:48 cs250/
drwx------ 2 joeluser undergrad 512  May 17 13:32 cs300/
drwxrwx--- 2 joeluser undergrad 512  May  2 10:53 smallproject/
-rw------- 1 joeluser undergrad 1158 Jun  2 16:01 email.txt
```

At this point in time, all of the files and directories in the account called joeluser have a group permission attribute that refers to group undergrad. This is because all of these files were created by someone, (presumably Joel) using a shell with userID joeluser, and group undergrad. Suppose that joeluser wants to change the groups of all of the files in the smallproject directory to have their group permissions refer to group novicegroup. He could accomplish this with the following command:

```
% chgrp -R novicegroup smallproject
```

The -R option will recursively change the group of all the files in the smallproject directory. Alternatively, if joeluser only wanted to change the group of the file email.txt, he could omit the -R option:

```
% chgrp novicegroup email.txt
```

Unix permissions are important when developing software as part of a team. Many conflicts can arise when permissions are poorly understood or forgotten. Consider the following scenario: a team member finishes implementing a system critical piece of software, copies all files into his group's directory, then leaves on vacation. His fellow team members would like to integrate his files into the system, but they are not group readable (the team member forgot to chmod g+rw ... before he left). Now development will be delayed until the guilty team member returns from his vacation. For this and many other reasons, it is important to understand and remember Unix file permissions. Here is a table that summarizes the commands we discussed:

Command	**Description**
chmod permission filename	Change the permissions for a file
chgrp -R group filename	Change the group ownership of a file
groups userID	List the groups for the specified user name
newgrp [-] groupname	Start a new shell with a specified group name. If a dash is used instead of a group name, a shell with the same properties will be used as login shell.