**Chapter 6**

**File Permissions**

**1. Understanding file permissions on Unix: a brief tutorial**

Every user on a Unix system has a unique username, and is a member of at least one group (the primary group for that user). This group information is held in the password file (/etc/passwd). A user can also be a member of one or more other groups. The auxiliary group information is held in the file /etc/group. Only the administrator can create new groups or add/delete group members (one of the shortcomings of the system).

Every directory and file on the system has an owner, and also an associated group. It also has a set of permission flags which specify separate read, write and execute permissions for the 'user' (owner), 'group', and 'other' (everyone else with an account on the computer) The 'ls' command shows the permissions and group associated with files when used with the ‘-l’ option. On some systems (e.g. Coos), the '-g' option is also needed to see the group information.

An example of the output produced by 'ls -l' is shown below.

**drwx------ 2 richard staff 2048 Jan 2 1997 private**

**drwxrws--- 2 richard staff 2048 Jan 2 1997 admin**

**-rw-rw---- 2 richard staff 12040 Aug 20 1996 admin/userinfo**

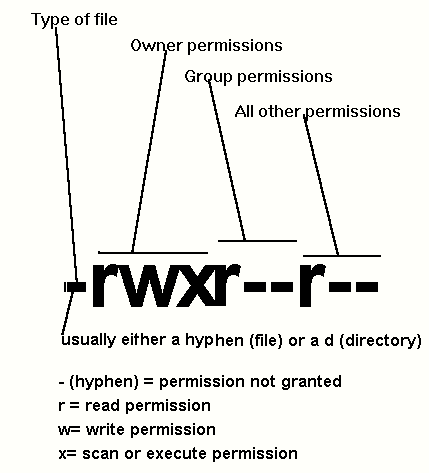
**drwxr-xr-x 3 richard user 2048 May 13 09:27 public**

Understanding how to read this output is useful to all unix users, but especially people using group access permissions.

* Field 1: a set of ten permission flags.
* Field 2: link count (don't worry about this)
* Field 3: owner of the file
* Field 4: associated group for the file
* Field 5: size in bytes
* Field 6-8: date of last modification (format varies, but always 3 fields)
* Field 9: name of file (possibly with path, depending on how ls was called)

The permission flags are read as follows (left to right)

|  |  |
| --- | --- |
| **position** | **Meaning** |
| **1** | Directory flag, 'd' if a directory, '-' if a normal file, something else occasionally may appear here for special devices. |
| **2,3,4** | read, write, execute permission for User (Owner) of file |
| **5,6,7** | read, write, execute permission for Group |
| **8,9,10** | read, write, execute permission for Other |
| **value** | **Meaning** |
| **-** | In any position means that flag is not set |
| **R** | file is readable by owner, group or other |
| **W** | File is writeable. On a directory, write access means we can add or delete files |
| **X** | File is executable (only for programs and shell scripts - not useful for data files). Execute permission on a directory means we can list the files in that directory |
| **S** | In the place where 'x' would normally go is called the set-UID or set-groupID flag. |



Graphical representation of file permission field

**2. Difference in access permissions for files and folders**

Access permissions for files and folders mean different things from the user standpoint. The table below shows the difference.

|  |  |  |
| --- | --- | --- |
| Access type | File | Directory |
| Read (r) | the file can be read by the corresponding userid or group to which this set of symbols applies | If the directory listing can be obtained |
| Write (w) | The file can be changed or deleted by the corresponding user or group to which this set of symbols applies | If user or process can change directory contents somehow: create new or delete existing files in the directory or rename files. |
| Execute (x) | The file is considered executable, and may be executed by the user or group to which this set of symbols applies | If user or process can access the directory, that is, go to it (make it to be the current working directory) |

On an executable program with set-UID or set-groupID, that program runs with the effective permissions of its owner or group.

For a directory, the set-groupID flag means that all files created inside that directory will inherit the group of the directory. Without this flag, a file takes on the primary group of the user creating the file. This property is important to people trying to maintain a directory as group accessible. The subdirectories also inherit the set-groupID property.

**3. The default file permissions (umask):**

Each user has a default set of permissions which apply to all files created by that user, unless the software explicitly sets something else. This is often called the 'umask', after the command used to change it. It is either inherited from the login process, or set in the .cshrc or .login file which configures an individual account, or it can be run manually.

Typically the default configuration is equivalent to typing 'umask 22' which produces permissions of:

**-rw-r--r-- for regular files, or**

**drwxr-xr-x for directories.**

In other words, user has full access, everyone else (group and other) has read access to files, lookup access to directories.

When working with group-access files and directories, it is common to use 'umask 2' which produces permissions of:

**-rw-rw-r-- for regular files, or**

**drwxrwxr-x for directories.**

For private work, use 'umask 77' which produces permissions:

**-rw------- for regular files, or**

**drwx------ for directories.**

The logic behind the number given to umask is not intuitive.

The command to change the permission flags is "chmod". Only the owner of a file can change its permissions.

The command to change the group of a file is "chgrp". Only the owner of a file can change its group, and can only change it to a group of which he is a member.

See the online manual pages for details of these commands on any particular system (e.g. "man chmod").

The basic form of the chmod command is:

**chmod who add-or-remove what\_permissions filename**

**Note:**

there should not be any spaces between the "who", "add-or-remove", and "what\_permissions" portions of the command, in a real chmod command. The spaces were included in the above diagram to make it more readable. See the following examples for samples of proper syntax.)

We'll break that diagram down a little further, and then give some examples.

**Command "Breakdown": chmod**

**chmod**

This is the name of the command.

**who**

Any combination of u (for "user"), g (for "group"), or o (for "others"), or a (for "all"--that is, user, group, and others).

**add-or-remove**

Use + to add the attribute (set the flag), or - to remove the attribute (clear the flag).

**what\_permissions**

Any combination of r (for Read), w (for Write), or x (for Execute).

**filename**

A file or directory name (or wildcard pattern) to which we wish to apply the listed permission changes.

Examples of typical useage are given below:

1. Files in the current directory readable by anyone.

**$chmod a+r \***

NOTE: -rw-r--r-- - - - myfile (myfile is having these permissions)

2. Give group write permission to "myfile", leaving all other permission flags alone

**$ chmod g+w myfile (symbolic mode)**

**$ chmod 664 myfie (numeric mode)**

3. Remove read and write access to "myfile", leaving all other permission flags alone

**$ chmod g-rw myfile (symbolic mode)**

**$ chmod 604 myfile (numeric mode)**

4. Give group read write access to this directory, and everything inside of it (-R = recursive)

**$ chmod -R g+rw (symbolic mode)**

**$ chmod –R 664 (numeric mode)**

5. To give read, write to user(owner) , to give read permission and remove write permission to Group to remove all permission for others to the file named ‘bigfile’ $ chmod u+rw, g+r, g-w, o-a bigfile (symbolic mode)

**$ chmod 6 4 0 bigfile (numeric mode)**

6. Change the ownership of this directory to group 'medi' and everything inside of it (-R = recursive). The person issuing this command must own all the files or it will fail.

**$ chgrp -R medi . (symbolic mode)**

**$ There is no numeric mode equality command for this**

**Warnings:**

Putting 'umask 2' into a startup file (.login or .cshrc) will make these settings apply to everything we do unless manually changed. This can lead to giving group access to files such as saved email in your home directory, which is generally not desirable.

Making a file group read/write without checking what its group is can lead to accidentally giving access to almost everyone on the system. Normally all users are members of some default group such as "users", as well as being members of specific project-oriented groups. Don't give group access to "users" when we intended some other group.

Remember that to read a file, we need execute access to the directory it is in AND read access to the file itself. To write a file, your need execute access to the directory AND write access to the file. To create new files or delete files, we need write access to the directory. We also need execute access to all parent directories back to the root. Group access will break if a parent directory is made completely private.