File Maintenance Commands

cp

Copies files. Will overwrite unless otherwise specified. Must also have write permission in the destination directory.

Example:

```
cp  sample.f  sample2.f   - copies sample.f to sample2.f
cp -R dir1 dir2           - copies contents of directory dir1 to dir2
cp -i file.1  file.new    - prompts if file.new will be overwritten
cp *.txt chapt1           - copies all files with .txt suffix to directory
chapt1
cp /usr/doc/README  ~     - copies file to your home directory
cp ~betty/index    .      - copies the file "index" from user betty's
                         home directory to current directory
```

rm

Deletes/removes files or directories if file permissions permit

Example:

```
rm  sample.f    - deletes sample.f
rm  chap?.txt   - deletes all files with chap as the first four
             characters of their name and with .txt as the last
       four characters of their name
rm -i *         - deletes all files in current directory but asks
             first for each file
rm -r /olddir   - recursively removes all files in the directory
             olddir, including the directory itself
```

mv

Moves files. It will overwrite unless otherwise specified. Must also have write permission in the destination directory.

Example:

```
mv  sample.f  sample2.f   - moves sample.f to sample2.f
mv dir1 newdir/dir2       - moves contents of directory dir1 to
                         newdir/dir2
mv -i file.1  file.new    - prompts if file.new will be overwritten
mv *.txt chapt1           - moves all files with .txt suffix to
                         directory chapt1
```

mkdir

Make directory. Will create the new directory in your working directory by default.

Example:

```
mkdir  /u/training/data
mkdir  data2
```

rmdir

Remove directory. Directories must be empty before you remove them.

```
rmdir  project1
```

To recursively remove nested directories, use the rm command with the -r option:

```
rm -r  dirctory_name
```

chgrp

Changes the group ownership of a file or directory.

Syntax

chgrp [**-f**] [**-h**] [**-R**] *Group* { *File* ... | *Directory* ... }
chgrp **-R** [**-f**] [**-H** | **-L** | **-P**] *Group* { *File*... | *Directory*... }

Description

The **chgrp** command changes the group of the file or directory specified by the *File* or *Directory* parameter to the group specified by the *Group* parameter. The value of the *Group* parameter can be a group name from the group database or a numeric group ID. When a symbolic link is encountered and you have not specified the **-h** or **-P** flags, the **chgrp** command changes the group ownership of the file or directory pointed to by the link and not the group ownership of the link itself.

chown

The chown command is used to change the owner and group of files, directories and links. By default, the owner of a filesystem object is the user that created it. The group is a set of users that share the same access permissions (i.e., read, write and execute) for that object. The basic syntax for using chown to change owners is

```
    chown [options] new_owner object(s)
```

new_owner is the user name or the numeric user ID (UID) of the new owner, and object is the name of the target file, directory or link. The ownership of any number of objects can be changed simultaneously.

For example, the following would transfer the ownership of a file named file1 and a directory named dir1 to a new owner named alice:

```
    chown alice file1 dir1
```

In order to perform the above command, most systems are configured by default to require access to the root (i.e., system administrator) account, which can be obtained on a personal computer by using the su (i.e., substitute user) command. An error message will be returned in the event that the user does not have the proper permissions or that the specified new owner or target(s) does not exist (or is spelled incorrectly).

The ownership and group of a filesystem object can be confirmed by using the ls command with its -l (i.e., long) option. The owner is shown in the third column and the group in the fourth. Thus, for example, the owner and group of file1 can be seen by using the following:

```
    ls -l file1
```

The basic syntax for using chown to change groups is

```
    chown [options] :new_group object(s)
```

or

```
  chown [options] .new_group object(s)
```

The only difference between the two versions is that the name or numeric ID of the new group is preceded directly by a colon in the former and by a dot in the latter; there is no functional difference. In this case, chown performs the same function as the chgrp (i.e., change group) command.

The owner and group can be changed simultaneously by combining the syntaxes for changing owner and group. That is, the name or UID of the new owner is followed directly (i.e., with no intervening spaces) by a period or colon, which is followed directly by the name or numeric ID of the new group, which, in turn, is followed by a space and then by the names of the target files, directories and/or links.

Thus, for example, the following would change the owner of a file named file2 to the user with the user name bob and change its group to group2:

```
    chown bob:group2 file2
```

If a user name or UID is followed directly by a colon or dot but no group name is provided, then the group is changed to that user's login group. Thus, for example, the following would change the ownership of file3 to cathy and would also change that file's group to the login group of the new owner (which by default is usually the same as the new owner):

```
    chown cathy: file3
```

Among chown's few options is -R, which operates on filesystem objects recursively. That is, when used on a directory, it can change the ownership and/or group of all objects within the directory tree beginning with that directory rather than just the ownership of the directory itself.

The -v (verbose) option provides information about every object processed. The -c is similar, but reports only when a change is made. The --help option displays the documentation found in the man online manual, and the --version option outputs version information

chmod

Change access permissions, change mode.

```
Syntax
       chmod [Options]... Mode [,Mode]... file...
       chmod [Options]... Numeric_Mode file...
       chmod [Options]... --reference=RFile file...

Options
  -f, --silent, --quiet   suppress most error messages
  -v, --verbose           output a diagnostic for every file processed
  -c, --changes           like verbose but report only when a change is made
      --reference=RFile   use RFile's mode instead of MODE values
  -R, --recursive         change files and directories recursively
      --help              display help and exit
      --version           output version information and exit
```

chmod changes the permissions of each given file according to mode, where mode describes the permissions to modify. Mode can be specified with octal numbers or with letters. Using letters is easier to understand for most people.

Permissions:

	Owner	Group	Other
Read			
Write			
Execute			

Numeric mode:

From one to four octal digits

Any omitted digits are assumed to be leading zeros.

The first digit = selects attributes for the set user ID (4) and set group ID (2) and save text image (1)S

The second digit = permissions for the user who owns the file: read (4), write (2), and execute (1)

The third digit = permissions for other users in the file's group: read (4), write (2), and execute (1)
The fourth digit = permissions for other users NOT in the file's group: read (4), write (2), and execute (1)

The octal (0-7) value is calculated by adding up the values for each digit
User (rwx) = 4+2+1 = 7
Group(rx) = 4+1 = 5
World (rx) = 4+1 = 5
chmode mode = 0755

Examples
chmod 400 file - Read by owner
chmod 040 file - Read by group
chmod 004 file - Read by world

chmod 200 file - Write by owner
chmod 020 file - Write by group
chmod 002 file - Write by world

chmod 100 file - execute by owner
chmod 010 file - execute by group
chmod 001 file - execute by world

To combine these, just add the numbers together:
chmod 444 file - Allow read permission to owner and group and world
chmod 777 file - Allow everyone to read, write, and execute file

Symbolic Mode
The format of a symbolic mode is a combination of the letters +-= rwxXstugoa
Multiple symbolic operations can be given, separated by commas.
The full syntax is [ugoa...][[+-=][rwxXstugo...]...][,...] but this is explained below.

A combination of the letters ugoa controls which users' access to the file will be changed:

User	**letter**
The user who owns it	u
Other users in the file's Group	g
Other users not in the file's group	o
All users	a

If none of these are given, the effect is as if was given, but bits that are set in the umask are not affected.

All users a is effectively user + group + others

The operator '+' causes the permissions selected to be added to the existing permissions of each file; '-' causes them to be removed; and '=' causes them to be the only permissions that the file has.

The letters 'rwxXstugo' select the new permissions for the affected users:

Permission	**letter**
Read	r
Write	w
Execute (or access for directories)	x
Execute only if the file is a directory (or already has execute permission for some user)	X
Set user or group ID on execution	s
Save program text on swap device	t
The permissions that the User who owns the file currently has for it	u
The permissions that other users in the file's Group have for it	g
Permissions that Other users not in the file's group have for it	o

Examples

Deny execute permission to everyone:

```
chmod a-x file
```

Allow read permission to everyone:

```
chmod a+r file
```

Make a file readable and writable by the group and others:

```
chmod go+rw file
```

Make a shell script executable by the user/owner

```
$ chmod u+x myscript.sh
```

Allow everyone to read, write, and execute the file and turn on the set group-ID:

```
chmod =rwx,g+s file
```

Notes:

When chmod is applied to a directory:

read = list files in the directory
write = add new files to the directory
execute = access files in the directory

chmod never changes the permissions of symbolic links. This is not a problem since the permissions of symbolic links are never used. However, for each symbolic link listed on the command line, chmod changes the permissions of the pointed-to file. In contrast, chmod ignores symbolic links encountered during recursive directory traversals

Visit chmod calculator
http://www.onlineconversion.com/html_chmod_calculator.htm