# FilePermissions

## Understanding and Using File Permissions

In Linux and Unix, everything is a file. Directories are files, files are files and devices are files. Devices are usually referred to as a node; however, they are still files. All of the files on a system have permissions that allow or prevent others from viewing, modifying or executing. If the file is of type Directory then it restricts different actions than files and device nodes. The super user "root" has the ability to access any file on the system. Each file has access restrictions with permissions, user restrictions with owner/group association. Permissions are referred to as bits.

To change or edit files that are owned by root, **sudo** must be used - please see RootSudo for details.

If the owner read & execute bit are on, then the permissions are:

```
-r-x------
```

There are three types of access restrictions:

| Permission | Action | chmod option |
|---|---|---|
| read | (view) | r or 4 |
| write | (edit) | w or 2 |
| execute | (execute) | x or 1 |

There are also three types of user restrictions:

| User | ls output |
|---|---|
| owner | `-rwx------` |
| group | `----rwx---` |
| other | `-------rwx` |

**Note:** The restriction type scope is not inheritable: the file owner will be unaffected by restrictions set for his group or everybody else.

Contents



## Folder/Directory Permissions

Directories have directory permissions. The directory permissions restrict different actions than with files or device nodes.

| Permission | Action | chmod option |
|---|---|---|
| read | (view contents, i.e. ls command) | r or 4 |
| write | (create or remove files from dir) | w or 2 |
| execute | (cd into directory) | x or 1 |

1. read restricts or allows viewing the directories contents, i.e. ls command
2. write restricts or allows creating new files or deleting files in the directory. (Caution: **write access for a directory allows deleting of files in the directory even if the user does not have write permissions for the file!**)
3. execute restricts or allows changing into the directory, i.e. cd command

**Folders (directories) must have 'execute' permissions set (x or 1), or folders (directories) will NOT FUNCTION as folders (directories) and WILL DISAPPEAR from view in the file browser (Nautilus).**

## Permissions in Action

```
user@host:/home/user$ ls -l /etc/hosts
-rw-r--r--  1 root root 288 2005-11-13 19:24 /etc/hosts
user@host:/home/user$
```

Using the example above we have the file "/etc/hosts" which is owned by the user root and belongs to the root group.

What are the permissions from the above /etc/hosts ls output?

```
-rw-r--r--

owner = Read & Write (rw-)
group = Read (r--)
other = Read (r--)
```

# Changing Permissions

The command to use when modifying permissions is chmod. There are two ways to modify permissions, with numbers or with letters. Using letters is easier to understand for most people. When modifying permissions be careful not to create security problems. Some files are configured to have very restrictive permissions to prevent unauthorized access. For example, the /etc/shadow file (file that stores all local user passwords) does not have permissions for regular users to read or otherwise access.

```
user@host:/home/user# ls -l /etc/shadow
-rw-r-----  1 root shadow 869 2005-11-08 13:16 /etc/shadow
user@host:/home/user#

Permissions:
owner = Read & Write (rw-)
group = Read (r--)
other = None (---)

Ownership:
owner = root
group = shadow
```

## chmod with Letters

```
Usage: chmod {options} filename
```

| Options | Definition |
|---|---|
| u | owner |
| g | group |
| o | other |
| a | all (same as ugo) |
| x | execute |
| w | write |
| r | read |
| + | add permission |
| - | remove permission |
| = | set permission |

Here are a few examples of chmod usage with letters (try these out on your system).

First create some empty files:

```
user@host:/home/user$ touch file1 file2 file3 file4
user@host:/home/user$ ls -l
total 0
-rw-r--r--  1 user user 0 Nov 19 20:13 file1
-rw-r--r--  1 user user 0 Nov 19 20:13 file2
-rw-r--r--  1 user user 0 Nov 19 20:13 file3
-rw-r--r--  1 user user 0 Nov 19 20:13 file4
```

Add owner execute bit:

```
user@host:/home/user$ chmod u+x file1
user@host:/home/user$ ls -l file1
-rwxr--r--  1 user user 0 Nov 19 20:13 file1
```

Add other write & execute bit:

```
user@host:/home/user$ chmod o+wx file2
user@host:/home/user$ ls -l file2
-rw-r--rwx  1 user user 0 Nov 19 20:13 file2
```

Remove group read bit:

```
user@host:/home/user$ chmod g-r file3
```

```
user@host:/home/user$ ls -l file3
-rw----r--  1 user user 0 Nov 19 20:13 file3
```

Add read, write and execute to everyone:

```
user@host:/home/user$ chmod ugo+rwx file4
user@host:/home/user$ ls -l file4
-rwxrwxrwx  1 user user 0 Nov 19 20:13 file4
user@host:/home/user$
```

## chmod with Numbers

```
Usage: chmod {options} filename
```

| Options | Definition |
|---|---|
| #-- | owner |
| -#- | group |
| --# | other |
| 1 | execute |
| 2 | write |
| 4 | read |

Owner, Group and Other is represented by three numbers. To get the value for the options determine the type of access needed for the file then add.

For example if you want a file that has -rw-rw-rwx permissions you will use the following:

| Owner | Group | Other |
|---|---|---|
| read & write | read & write | read, write & execute |
| 4+2=6 | 4+2=6 | 4+2+1=7 |

```
user@host:/home/user$ chmod 667 filename
```

Another example if you want a file that has --w-r-x--x permissions you will use the following:

| Owner | Group | Other |
|---|---|---|
| write | read & execute | execute |
| 2 | 4+1=5 | 1 |

```
user@host:/home/user$ chmod 251 filename
```

Here are a few examples of chmod usage with numbers (try these out on your system).

First create some empty files:

```
user@host:/home/user$ touch file1 file2 file3 file4
user@host:/home/user$ ls -l
total 0
-rw-r--r--  1 user user 0 Nov 19 20:13 file1
-rw-r--r--  1 user user 0 Nov 19 20:13 file2
-rw-r--r--  1 user user 0 Nov 19 20:13 file3
-rw-r--r--  1 user user 0 Nov 19 20:13 file4
```

Add owner execute bit:

```
user@host:/home/user$ chmod 744 file1
user@host:/home/user$ ls -l file1
-rwxr--r--  1 user user 0 Nov 19 20:13 file1
```

Add other write & execute bit:

```
user@host:/home/user$ chmod 647 file2
user@host:/home/user$ ls -l file2
-rw-r--rwx  1 user user 0 Nov 19 20:13 file2
```

Remove group read bit:

```
user@host:/home/user$ chmod 604 file3
user@host:/home/user$ ls -l file3
-rw----r--  1 user user 0 Nov 19 20:13 file3
```

Add read, write and execute to everyone:

```
user@host:/home/user$ chmod 777 file4
user@host:/home/user$ ls -l file4
-rwxrwxrwx  1 user user 0 Nov 19 20:13 file4
user@host:/home/user$
```

### chmod with sudo

Changing permissions on files that you do not have ownership of: (**Note** that changing permissions the wrong way on the wrong files can quickly mess up your system a great deal! Please be careful when using **sudo**!)

```
user@host:/home/user$ ls -l /usr/local/bin/somefile
-rw-r--r--  1 root root 550 2005-11-13 19:45 /usr/local/bin/somefile
user@host:/home/user$

user@host:/home/user$ sudo chmod o+x /usr/local/bin/somefile

user@host:/home/user$ ls -l /usr/local/bin/somefile
-rw-r--r-x  1 root root 550 2005-11-13 19:45 /usr/local/bin/somefile
user@host:/home/user$
```

## Recursive Permission Changes

To change the permissions of multiple files and directories with one command. Please note the warning in the chmod with sudo section and the Warning with Recursive chmod section.

### Recursive chmod with -R and sudo

To change all the permissions of each file and folder under a specified directory at once, use sudo chmod with -R

```
user@host:/home/user$ sudo chmod 777 -R /path/to/someDirectory
user@host:/home/user$ ls -l
total 3
-rwxrwxrwx  1 user user 0 Nov 19 20:13 file1
drwxrwxrwx  2 user user 4096 Nov 19 20:13 folder
-rwxrwxrwx  1 user user 0 Nov 19 20:13 file2
```

### Recursive chmod using find, pipemill, and sudo

To assign reasonably secure permissions to files and folders/directories, it's common to give files a permission of 644, and directories a 755 permission, since chmod -R assigns to both. Use sudo, the find command, and a pipemill to chmod as in the following examples.

To change permission of only files under a specified directory.

```
user@host:/home/user$ sudo find /path/to/someDirectory -type f -print0 | xargs -0 sudo chmod 644
user@host:/home/user$ ls -l
total 3
-rw-r--r--  1 user user 0 Nov 19 20:13 file1
drwxrwxrwx  2 user user 4096 Nov 19 20:13 folder
-rw-r--r--  1 user user 0 Nov 19 20:13 file2
```

To change permission of only directories under a specified directory (including that directory):

```
user@host:/home/user$ sudo find /path/to/someDirectory -type d -print0 | xargs -0 sudo chmod 755
user@host:/home/user$ ls -l
total 3
-rw-r--r--  1 user user 0 Nov 19 20:13 file1
drwxr-xr-x  2 user user 4096 Nov 19 20:13 folder
-rw-r--r--  1 user user 0 Nov 19 20:13 file2
```

## Warning with Recursive chmod

WARNING: Although it's been said, it's worth mentioning in context of a gotcha typo. Please note, Recursively deleting or chown-ing files are extremely dangerous. You will not be the first, nor the last, person to add one too many spaces into the command. This example will hose your system:

```
user@host:/home/user$ sudo chmod -R / home/john/Desktop/tempfiles
```

Note the space between the first / and home.

You have been warned.

## Changing the File Owner and Group

A file's owner can be changed using the `chown` command. For example, to change the `foobar` file's owner to `tux`:

```
user@host:/home/user$ sudo chown tux foobar
```

To change the `foobar` file's group to `penguins`, you could use **either** `chgrp` or `chown` with special syntax:

```
user@host:/home/user$ sudo chgrp penguins foobar
```

```
user@host:/home/user$ sudo chown :penguins foobar
```

Finally, to change the `foobar` file's owner to `tux` and the group to `penguins` with a single command, the syntax would be:

```
user@host:/home/user$ sudo chown tux:penguins foobar
```

Note that, by default, you must use `sudo` to change a file's owner or group.

# Volume Permissions with umask

This section has been moved to: Fstab#Options

# ACL (Access Control List)

Posix ACLs are a way of achieving a finer granularity of permissions than is possible with the standard Unix file permissions. See the full page on ACLs FilePermissionsACLs

## Setting up ACL

1. Install the acl package:

```
sudo apt-get install acl
```

2. Edit `/etc/fstab` and add option `acl` to partition(s) on which you want to enable ACL. For example:

```
...
UUID=d027a8eb-e234-1c9f-aef1-43a7dd9a2345 /home     ext4    defaults,acl    0    2
...
```

3. Remount partition(s) on which you want to enable ACL. For example:

```
sudo mount -o remount /home
```

4. Verify `acl` is enabled on the partition(s):

```
mount | grep acl
```

The commands, `setfacl` and `getfacl`, set and read ACLs on files and directories.

## Example Usage

This is a simple example for use with a Samba share to ensure that any files or sub-directories created could also be modified by any Samba user.

1. Create a directory with full permission:

```
mkdir shared_dir
chmod 777 shared_dir
```

2. Set the `default` ACL with '-d' and modify with '-m' the permissions for samba `nobody` user `nogroup` group which will apply to all newly created file/directories.

```
setfacl -d -m u:nobody:rwx,g:nogroup:rwx,o::r-x shared_dir
```

## GUI ACL Editor

The Eiciel package allows GUI access to ACLs through the Nautilus file manager.

## Useful ACL Resources

1. http://brunogirin.blogspot.com/2010/03/shared-folders-in-ubuntu-with-setgid.html
2. http://wiki.kaspersandberg.com/doku.php?id=howtos:acl
3. man acl
4. man setfacl
5. man getfacl

# File removal

To remove a file you cannot delete use

```
sudo rm -rf filename
```

where filename is the name and path of the file to delete.

**Nota bene:** Be very careful when using the command `rm` with the `-rf` option since `-r` makes the file removal recursive (meaning it will remove files inside of folders) and `-f` will force the removal even for files which aren't writable. To play it safe, please consider typing in the absolute path to the file

```
sudo rm -rf /path/to/file/filename
```

to prevent any mishaps that can/will occur. It takes longer to type but you can't put a price on peace of mind. See the rm man page for details.

## Sticky Bit

The sticky bit applies only to directories, and is typically used on publicly-writeable directories. Within a directory upon which the sticky bit is applied, users are prevented from deleting or renaming any files that they do not personally own.

To add or remove the sticky bit, use chmod with the "t" flag:

```
chmod +t <directory>
chmod -t <directory>
```

The status of the sticky bit is shown in the other execute field, when viewing the long output of ls. "t" or "T" in the other execute field indicates the sticky bit is set, anything else indicates it is not.

Making a public directory:

```
user@host:/home/user$ mkdir folder
user@host:/home/user$ chmod 777 folder
user@host:/home/user$ ls -l
total 3
drwxrwxrwx  2 user user 4096 Nov 19 20:13 folder
```

Adding the sticky bit (note the "t" in the other execute field):

```
user@host:/home/user$ chmod +t folder
user@host:/home/user$ ls -l
total 3
drwxrwxrwt  2 user user 4096 Nov 19 20:13 folder
```

## See also

1. man chmod
2. man chown
3. man chgrp
4. FindingFiles
5. User Private Groups

---

## ToDo

1. umask (add file and directory umask section, with specific focus on security)
2. The User Private Group scheme. In other words, this page does the nuts and bolts ok, but we need to describe what the permissions should be. The default Ubuntu set up is not agnostic: Every user has their default private group. Directories for collaboration need to have special group and permission set for correct functioning.
3. * Suggestion: I often use `find` instead of `chmod -R`, because it's easier to differentiate between files and directories that way. Yes, I know about the 'X' permission, but I don't trust it.
4. The sticky bit. It's needed for "other" in shared directories like /tmp. It's needed for "group" in shared directories where write permission is given to a group, like /var/www

---

FilePermissions (last edited 2013-11-10 12:51:38 by ti-225-214-154)