University of Texas at El Paso
Computational Science (CPS)

A short tutorial

Chmod Command Examples for Beginners

1 References

http://www.yourownlinux.com/2013/09/chmod-basics-of-filesdirectories.html
http://www.thegeekstuff.com/2010/06/chmod-command-examples/
http://askubuntu.com/questions/303593/how-can-i-chmod-777-all-subfolders-of-var-www
http://superuser.com/questions/91935/how-to-chmod-all-directories-except-files-recursively

Introduction

Earlier we discussed about how to use octal permission bits with chmod. In this article, let us review how to use symbolic representation with chmod. Following are the symbolic representation of three different roles: CHMOD stands for "Change Mode" and it is the Linux command which is used to change the access permissions of files and directories. In Linux/UNIX, the concept of user (owner) and group is very fundamental, as everybody wants things he uses to be kept secure and properly organized. That is why every file and directory has an owner and a group associated with it and they have different permissions to access that particular file.

- u is for user,
- g is for group,
- and o is for others.

Following are the symbolic representation of three different permissions:

- r is for read permission,
- w is for write permission,
- x is for execute permission.

Following are few examples on how to use the symbolic representation on chmod.

1. Add single permission to a `file/directory` Changing permission to a single set. + symbol means adding permission. For example, do the following to give execute permission for the user irrespective of anything else:

   ```
   $ chmod u+x filename
   ```

2. Add multiple permission to a file/directory Use comma to separate the multiple permission sets as shown below.

   ```
   $ chmod u+r,g+x filename
   ```

3. Remove permission from a file/directory Following example removes read and write permission for the user.

   ```
   $ chmod u-rx filename
   ```

4. Change permission for all roles on a file/directory Following example assigns execute privilege to user, group and others (basically anybody can execute this file).

   ```
   $ chmod a+x filename
   ```

5. Make permission for a file same as another file (using reference). If you want to change a file permission same as another file, use the reference option as shown below. In this example, file2s permission will be set exactly same as file1s permission.

```
$ chmod --reference=file1 file2
```

6. Apply the permission to all the files under a directory recursively. Use option -R to change the permission recursively as shown below.

```
$ chmod -R 755 directory-name/
```

7. Change execute permission only on the directories (files are not affected). On a particular directory if you have multiple sub-directories and files, the following command will assign execute permission only to all the sub-directories in the current directory (not the files in the current directory).

8.

```
$ chmod u+X *
```

Note: If the files has execute permission already for either the group or others, the above command will assign the execute permission to the user

Owner permissions: They determine what operations the owner of the file can execute on the file.

Group permissions: They determine what operations a user belonging to the group associated with that file can execute on the file.

Other permissions: They indicate what operations all other users can execute on the file.

So, there are Three basic file/directory operations that a user/group/other users can perform on the files and directories.

- Read (r): Permission to read the contents of the file/directory. In case of directories, a person can view all the files and sub-directories belonging to the directory.
- Write (w): Permission to modify the contents of the file/directory. In case of directories, a person can create a file or sub-directory in that directory.
- Execute (x): Permission to execute a file as a script/program. Executing a directory! Well, it does not make any sense. In case of directories, a person can enter that directory. In order to use ls and cd commands in /bin directory, a user should have Execute permissions.

CHMOD assigns numeric values to the Read, Write and Execute permissions which are as follows:

- Read : 4
- Write : 2
- Execute : 1

Numeric	Permission Type	Permission To
400	Read	Owner
040	Read	Group
004	Read	Others
200	Write	Owner
020	Write	Group
002	Write	Others
100	Execute	Owner
010	Execute	Group
001	Execute	Others

So, the permissions associated with any file/directory in Linux have a 3x3 format i.e. Three types of permissions (Read, Write and Execute) that are available for three types of users (Owner, Group and Other).

Octal	Decimal	Permission	Representation
000	0(0+0+0)	No permissions	---
001	1(0+0+1)	Execute	--x
010	2(0+1+0)	Write	-w-
011	3(0+1+1)	Write+Execute	-wx
100	4(1+0+0)	Read	r--
101	5(1+0+1)	Read + Execute	r-x
110	6(1+1+0)	Read + Write	rw-
111	7(1+1+1)	Read + Write + Execute	rwx

To observe this, just enter ls l command that displays 9 characters for every file/directory representing the permissions for all the three types of users.

For Example:

2 Changing File/Directory Permissions with CHMOD

To change the permissions associated with files and directories, you may either use Octal Representation (using numeric) or Symbolic Representation (using alphabets). We will restrict this part of our discussion up to the use of octal representation for changing files and directories permissions.

So, in octal representation of the permissions:

- First digit is for Owner
- Second digit is for Group
- Third digit is for Others

As an example, we have seen in our one of the previous articles- Getting Started with Linux Shell Scripting Language, we have used a command as chmod 744 helloworld.sh. Indirectly, we have given Read + Write + Execute (4+2+1) permissions to the Owner, Read (4) permission to the group and Read (4) permission to the others.

Now, if we wish to give Read + Write (4+2) permissions to the owner, Read (4) permissions to the group and others, then we need to enter following command:

```
chmod 644 <file_name>
```

Another example, to give Read + Execute permission (4 + 1 = 5) to user and no permission (0) to group, and Write (2) permission to others, enter following command:

```
chmod 502 <file_name>
```

3 What is UMASK..?

UMASK, along with default permission of file/directory, is responsible for determining the final value of the default permission of a file/directory. The default permission for a file is 777 and for a directory, it is 666. From these default permissions, the umask value is subtracted to get the final default permission for newly created files or directory. The default value of umask is 022.

Final default permissions for file and directories are determined as follows:

- Default file permission: 666
- Default directory permission: 777
- Default umask : 022

- Final default file permission: 644
- Final default directory permission: 755

You may change the umask to an appropriate value based on your purpose. For example, if you wish no one but the owner can do anything with the file/directory then you can set umask as 0077.

```
umask 0077
```

After this action, when you make a new file/directory, the permissions associated with them will be as shown below:

4 Symbolic Representation

The symbolic representation used for three different types of users is as follows:

- **u** is used for user/owner
- **g** us used for group
- **o** is used for others

5 Usage of Symbolic Representation

5.1 Adding Single Permission

To change single permission of a specific set of users (owner, group or others), we can use '+' symbol to add a permission.

```
Syntax: chmod <user>+<permission> <file_name>
```

```
Example: chmod u+x my_file
```

Using above command, we can add Execute permission to the owner of the file.

5.2 Adding Multiple Permissions

This is similar to command explained above, you just need to separate those multiple permissions with a comma (,).

```
Syntax: chmod <user>+<permission>,<user>+<permission> <file_name>
```

```
Example: chmod g+x,o+x my_file
```

Using above command, we can add Execute permissions to the group and other users of the file.

5.3 Removing a Permission

Removing a permission is as easy as adding a permission, just remember to use '-' symbol instead of '+'.

```
Syntax: chmod <user>-<permission> <file_name>
```

```
Example: chmod o-x my_file
```

Above command removes Execute permission from the other users of the file.

5.4 Making the Changes for All

In case we add or remove some permissions for all the users (owner, group and others), we can use a notation 'a' which denotes "All users".

```
Syntax: chmod a<+ or -><permission> <file_name>

Example: chmod a+x my_file
```

Above command will add Execute permission to all the users.

5.5 Copying the Permissions

If we wish to make permissions of two files/directories same, we can do it using reference option. Consider that, we want to apply permissions of myfile1 to some other file called myfile2, then use following command:

```
Example: chmod --reference=myfile1 myfile2
```

5.6 Applying Changes to All the Content of a Directory

If we want to apply some specific changes to all the files inside a directory, we can make use of option -R denoting that the operation is recursive.

```
Syntax: chmod -R <directory_name>/
```

That's enough for this article. In this article, I tried to cover the basics of files and directory permissions and the fundamental use of CHMOD command that helps us change those permissions associated with files and directories. Please feel free to comment for the feedback. Stay tuned for some more interesting articles.

6 OTHERS 1

This is bad practice, but hopefully you are just using this for development, or you have another good reason. You can specify the permissions when you create a directory using the -m option:

```
mkdir -m 777 dirname
```

Or you can set the permissions recursively.

```
sudo chmod -R 777 /var/www
```

Before using either of these, really consider if you want your filesystem to be so accessible.

7 OTHER 2

A common reason for this sort of thing is to set directories to 755 but files to 644. In this case there's a slightly quicker way than nik's find example:

```
chmod -R u+rwX,go+rX,go-w /path
```

Meaning:

- `-R` = recursively;
- `u+rwX` = Users can read, write and execute;
- `go+rX` = group and others can read and execute;
- `go-w` = group and others can't write

The important thing to note here is that X acts differently to x. In manual we can read:

The execute/search bits if the file is a directory or any of the execute/search bits are set in the original (unmodified) mode.

In other words, chmod u+X on a file won't set the execute bit; and g+X will only set it if it's already set for the user.