thegeekdiary.com

Linux File/Directory Permissions cheat sheet – The Geek Diary

3-4 minutes

Here is a short note/cheat sheet for Linux directory and file permissions. The table below gives numbers for all permission types of a File/Directory.

Number	Permission Type	Symbol
0	No Permission	—
1	Execute	–x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r–
5	Read + Execute	r-x
6	Read + Write	rw-
7	Read + Write + Execute	rwx

For example **777** permission to **/etc** folder means the folder has all the **read, write and executable** permissions for owner, group and all users.

Owner - The Owner permissions apply only the owner of the file or

directory, they will not impact the actions of other users.
Group - The Group permissions apply only to the group that has been assigned to the file or directory, they will not effect the actions of other users.
All users - The All Users permissions apply to all other users on the system, this is the permission group that you want to watch the most.

Below is sample output from ls -l; you can see from the first character of each line that foo and bar are directories (indicated by the d) and that meta is a regular file (indicated by the -).

```
$ ls -l
drwxr-xr-x 2 user user    6 Jan  7  2015 Desktop
-rw-rw-r-- 1 user user    0 Feb 16 14:17 file1
-rw-r--r-- 1 user wheel   0 Feb 16 14:22 file2
```

The next nine characters show the file's permissions for user, group, and others (or everyone else) as shown below, with parentheses added for clarity:

```
-(rw-) (r--) (r--) 1 user wheel   0 Feb 16 14:22 file2
```

Now the owner has read and write permissions (rw-), the group and everyone else has only read permissions (r–). This is called symbolic representation because letters such as r, w, and x, are used to indicate permissions. Permissions can also be represented numerically: r = 4; w = 2; x = 1

Add each section so that the permissions of the file meta (from the example above) would be 664. Here is another way to look at how we come to that number:

```
-(rw-)  (rw-)  (r--)
```

-(42-) (42-) (4--)
 6   6   4

Adding the numbers in each section results in permissions of 664.

Changing File Permissions

The **chmod** command is used to alter the permissions of a file. It may be used to add or remove permissions symbolically. For example, to add execute permissions for the owner of a file you would run:

Or, to add read and write permissions for the group that owns the file, you would run:

Instead of adding permissions, the symbolic syntax of chmod can also be used to subtract or set to some absolute value as shown in these examples:

$ chmod u=rwx,g=rx,o= file_name

The chmod command can also explicitly set permissions using a numerical representation. For example, to set permissions on a file to rwxrwxr–, you would run: