

File and Directory Permissions in Linux - Basics.

In Linux, everything is a file.

One of the properties of a file is a set of permission bits, which determine who has access and what type of access they have. Note that this scheme was originally developed to protect users from each other, and the system from all users.

There are three categories of permissions defined for a linux file system.

User **Group** **Other**

The User is the person who created the file, and is often referred to as the owner.

The Group is one or more users who have been granted separate access by the user (owner).

Every one else who is able to log in to the computer, and is neither the user nor a member of the associated group, is classified as "other".

File and Directory Permissions in Linux – Nomenclature.

There are three distinct types of permission bits for each category

read

write

execute

and these permission types define how a user can access a file.

There are two types of permissions nomenclature, namely symbolic and absolute.

Lets look at the easy one first – **symbolic** nomenclature.

Symbolic nomenclature uses the symbols **r,w** and **x** to show what type of access is permitted, and a hyphen (-) to show access is denied.

The actions allowed by these permissions depend on whether we are considering a file or a directory (folder).

An additional character is used to show the type of file. This character is usually displayed as the leftmost character of the permissions bits.

File and Directory Permissions in Linux – Symbolic Nomenclature

for **files**

Read permission (**r**) means you can look at the files' content.

Write permission (**w**) means that you change or delete the file.

Execute permission (**x**) means that you can run the file as a program.

for **directories**

Read permission (**r**) means you can list the contents of the directory

Write permission (**w**) means you can add or remove files in the directory

Execute permission (**x**) means you can list information about the files in the directory

A few examples using the `ls -l` command in an X-Terminal ..

```
-rw-r--r--  1 nslinux users      21393 Apr 26 13:56 rpms.list
drwxrwxr-x  3 nslinux nslinux    224 May 14 15:15 Documents
brw-rw----  1 nslinux cdrom      22, 0 May 14 17:33 /dev/hdc
crw-rw----  1 root lp           6, 0 May 15 2006 /dev/lp0
lrwxrwxrwx  1 root root          3 May 14 17:33 /dev/cdrom -> hdc
```

File and Directory Permissions in Linux – Absolute Nomenclature

The Absolute nomenclature scheme uses octal numbers to show the permission bits which are set. The overall permissions of a file are then expressed as a number which is the sum of the numeric values assigned to each category.

The numeric values for each permission bit can be expressed as follows ..

| user | | | group | | | other | | |
|------------|------------|------------|-----------|-----------|-----------|----------|----------|----------|
| read | write | execute | read | write | execute | read | write | execute |
| 400 | 200 | 100 | 40 | 20 | 10 | 4 | 2 | 1 |

The overall permissions are expressed as a number, which is determined by the sum of the numbers representing those permissions which are turned on

File and Directory Permissions in Linux – Absolute Nomenclature

Suppose you want to set the permissions of a non executable file that you have created so that - you (the user) have read and write permissions, the group “users” has read permissions, and everyone else also has read permissions.

This would translate to

user = 400 + 200 = 600
group = 40
other = 4

which gives a total = 600 + 40 + 4 = 644

or, in symbolic nomenclature, **rw-r--r--**

For a directory, lets assume you want to give yourself full permissions, the group users just read and execute permission, and no permission at all for everyone else.

Here, permissions would be

user = 400 + 200 + 100 = 700
group = 40 + 10 = 50
other = 0

for a total of 750

or, in symbolic nomenclature, **rwxr-x---**

File and Directory Permissions in Linux - Defaults.

Default Permissions

In console or X-Terminal, the default permissions set on file or directory creation are determined by the shell in use. The usual shell is the Bash shell, which has the function `umask` to view and/or set the default permissions.

Consider the following command issued in a bash shell ...

```
$ umask  
0022
```

Here, we have the `umask` set to 0022.

We ignore the leading zero, and take the complement of 022, that is we subtract each digit from 7.

```
  777  
-  022  
-----  
  755
```

File and Directory Permissions in Linux – Defaults (cont).

Now 755 corresponds to ...

| | | | | | | | | | | | | | |
|-------|---|-----|---|-----|---|-----|---|-----|---|---|---|---|---|
| user | = | 600 | + | 200 | + | 100 | = | 700 | (| r | w | x |) |
| group | = | 40 | + | 10 | | | = | 50 | (| r | | x |) |
| other | = | 4 | + | 1 | | | = | 5 | (| r | | x |) |
| | | | | | | | | --- | | | | | |
| | | | | | | | | 755 | | | | | |

or in symbolic nomenclature - **rxr-rx-rx**

Note that **for files** the default permissions set at file creation time usually do not set the execute bit, even if the umask implies that this is so.

for example

```
$ umask
0022
$ touch test3.txt
$ ls -l test3.txt
-rw-r--r-- 1 nslinux nslinux 0 Apr 26 13:15 test3.txt
```

Note also that application programs may set permissions when they save files which may be different from the defaults.

File and Directory Permissions – Command Line Manipulation.

There are three bash shell commands to manipulate permissions

chown - change owner **chgrp** - change group **chmod** - change mode

The chown command often requires a shell with root privileges. For example ..

```
# ls -ld misc-security
drwxr-xr-x  2 root root      72 Apr 26 15:15 misc-security
```

We want to change this to – owner = nslinux, group = users.

```
# chown nslinux misc-security
# chgrp users misc-security

# ls -ld misc-security
drwxr-xr-x  2 nslinux users 72 Apr 26 15:15 misc-security
```

```
# chown nslinux:users misc-security

# ls -ld misc-security
drwxr-xr-x  2 nslinux users 72 Apr 26 15:15 misc-security
```

Note that here we have changed both owner and group at the one time.

File and Directory Permissions – Command Line Manipulation (cont).

We can change the group on files that we own ..

```
$ ls -l test.txt
-rw-rw---- 1 nslinux nslinux 186 Mar 12 16:41 test.txt
$ chgrp users test.txt
$ ls -l test.txt
-rw-rw---- 1 nslinux users 186 Mar 12 16:41 test.txt
```

And we can also change the permission bits using chmod ..

```
$ ls -l test.txt
-rw-rw---- 1 nslinux users 186 Mar 12 16:41 test.txt
$ chmod g-w,o+r test.txt
$ ls -l test.txt
-rw-r--r-- 1 nslinux users 186 Mar 12 16:41 test.txt
```