

Unix & Linux Stack Exchange is a question and answer site for users of Linux, FreeBSD and other Un*x-like operating systems. Join them; it only takes a minute:

[Sign up](#)

Here's how it works:

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top

UNIX & LINUX

Understanding UNIX permissions and file types

[Ask Question](#)



65

I've never really got how `chmod` worked up until today. I followed a tutorial that explained a big deal to me.



★

60

For example, I've read that you've got three different permission groups:

- owner (u)
- group (g)
- everyone (o)

Based on these three groups, I now know that:

- If the file is owned by the user, the user permissions determine the access.
- If the group of the file is the same as the user's group, the group permission determine the access.
- If the user is not the file owner, and is not in the group, then the other permission is used.

I've also learned that you've got the following permissions:

- read (r)

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

I created a directory to test my newly acquired knowledge:

```
mkdir test
```

Then I did some tests:

```
chmod u+rw test/  
# drwx-----  
chmod g+rx test/  
# drwxr-x---  
chmod u-x test/  
# drw-r-x---
```

After fooling around for some time I think I finally got the hang of `chmod` and the way you set permission using this command.

But...

I still have a few questions:

- What does the `d` at the start stand for?
- What's the name and use of the containing slot and what other values can it hold?
- How can I set and unset it?
- What is the value for this `d` ? (As you only have $7=4+2+1$ $7=4+2+1$ $7=4+2+1$)
- Why do people sometimes use `0777` instead of `777` to set their permissions?

But as I shouldn't be asking multiple questions, I'll try to ask it in one question.

In UNIX based system such as all Linux distributions, concerning the permissions, what does the first part (`d`) stand for and what's the use for this part of the permissions?

I would like to add that I am new to the whole Unix/Linux platform and if this is a stupid question, please excuse me for asking.

[linux](#)[permissions](#)[ls](#)[chmod](#)

edited Jan 29 '18 at 21:52




[Stephen Kitt](#)

167k 24 376 454

asked Feb 10 '15 at 11:51

migrated from serverfault.com
Feb 10 '15 at 11:52

This question came from our site for
system and network administrators.

-
- 3 Please next time try to ask only one question, multiple question are usually bad for referencing and almost never addressed all at the same time. – [Kiwiy](#) Feb 10 '15 at 12:12
-
- 1 @Kiwiy I'm sorry, I thought i'd do it like this because it concerns the same subject. If you want, you can suggest a better title for my question to improve it's reference. – [Peter](#) Feb 10 '15 at 13:09 
-
- 1 Incidentally, the value for the "d" is 040000 - it can be found in header files under the name `S_IFDIR`. You don't use it when setting the file mode, but the `stat()` function actually returns the value 040750 for `drwxr-x---`. – [Random832](#) Feb 10 '15 at 18:07
-
- 2 @jamesqf Actually, now that I understand how the octal codes map, it's simpler for me to think about that way. – [HalosGhost](#) Feb 10 '15 at 22:12
-
- 4 @Peter: there are no stupid questions - just stupids that don't learn because they're afraid to ask. – [mgarciaisai](#) Feb 11 '15 at 14:05
-

5 Answers



110



+100

I'll answer your questions in three parts: file types, permissions, and use cases for the various forms of `chmod`.

File types

The first character in `ls -l` output represents the file type; `d` means it's a directory. It can't be set or unset, it depends on how the file was created. You can find the complete list of file types in the [ls documentation](#); those you're likely to come across are

- `-` : "regular" file, created with any program which can write a file
- `b` : block special file, typically disk or partition devices, can be

/dev for examples)

- `d` : directory, can be created with `mkdir`
- `l` : symbolic link, can be created with `ln -s`
- `p` : named pipe, can be created with `mkfifo`
- `s` : socket, can be created with `nc -U`
- `D` : [door](#), created by some server processes on Solaris/openIndiana.

Permissions

`chmod 0777` is used to set all the permissions in one `chmod` execution, rather than combining changes with `u+` etc. Each of the four digits is an octal value representing a set of permissions:

- `suid`, `sgid` and “sticky” (see below)
- user permissions
- group permissions
- “other” permissions

The octal value is calculated as the sum of the permissions:

- “read” is 4
- “write” is 2
- “execute” is 1

For the first digit:

- `suid` is 4; binaries with this bit set run as their owner user (commonly `root`)
- `sgid` is 2; binaries with this bit set run as their owner group (this was used for games so high scores could be shared, but it's often a security risk when combined with vulnerabilities in the games), and files created in directories with this bit set belong to the directory's owner group by default (this is handy for creating shared folders)
- “sticky” (or “restricted deletion”) is 1; files in directories with this bit set can only be deleted by their owner, the directory's owner, or `root` (see `/tmp` for a

See [the `chmod` manpage](#) for details. Note that in all this I'm ignoring other security features which can alter users' permissions on files (SELinux, file ACLs...).

Special bits are handled differently depending on the type of file (regular file or directory) and the underlying system. (This is mentioned in the `chmod` manpage.) On the system I used to test this (with `coreutils` 8.23 on an `ext4` filesystem, running Linux kernel 3.16.7-ckt2), the behaviour is as follows. For a file, the special bits are always cleared unless explicitly set, so `chmod 0777` is equivalent to `chmod 777`, and both commands clear the special bits and give everyone full permissions on the file. For a directory, the special bits are never fully cleared using the four-digit numeric form, so in effect `chmod 0777` is also equivalent to `chmod 777` but it's misleading since some of the special bits will remain as-is. (A previous version of this answer got this wrong.) To clear special bits on directories you need to use `u-s`, `g-s` and/or `o-t` explicitly or specify a negative numeric value, so `chmod -7000` will clear all the special bits on a directory.

In `ls -l` output, `suid`, `sgid` and "sticky" appear in place of the `x` entry: `suid` is `s` or `S` instead of the user's `x`, `sgid` is `s` or `S` instead of the group's `x`, and "sticky" is `t` or `T` instead of others' `x`. A lower-case letter indicates that both the special bit and the executable bit are set; an upper-case letter indicates that only the special bit is set.

The various forms of `chmod`

Because of the behaviour described above, using the full four digits in `chmod` can be confusing (at least it turns out I was confused). It's useful when you want to set special bits as well as permission bits; otherwise the bits are cleared if you're manipulating a file, preserved if you're manipulating a directory. So `chmod 7750` ensures you'll get at

won't necessarily clear the special bits.

Using numeric modes instead of text commands (`[ugo][=+-][rwxXst]`) is probably more a case of habit and the aim of the command. Once you're used to using numeric modes, it's often easier to just specify the full mode that way; and it's useful to be able to think of permissions using numeric modes, since many other commands can use them (`install` , `mknod ...`).

Some text variants can come in handy: if you simply want to ensure a file can be executed by anyone, `chmod a+x` will do that, regardless of what the other permissions are. Likewise, `+X` adds the execute permission only if one of the execute permissions is already set or the file is a directory; this can be handy for restoring permissions globally without having to special-case files v. directories. Thus, `chmod -R ug=rX,u+w,o=` is equivalent to applying `chmod -R 750` to all directories and executable files and `chmod -R 640` to all other files.

edited Jul 14 '18 at 7:54

answered Feb 10 '15 at 12:02



[Stephen Kitt](#)

167k 24 376 454

Though, the other answers were pretty good, you really spent some time answering this question. Thanks. – [Peter](#) Feb 12 '15 at 8:56

On *BSD directories behave as if their `sgid` bit was always set, regardless of its actual value. On FreeBSD the `suid` bit can be configured to act analogously to `sgid` (i.e. files and subdirectories created inside will have the same owner as the directory), provided that the underlying filesystem supports that, and is mounted with the `suidir` option. – [lcd047](#) Jun 2 '15 at 4:53

" `+X` adds the execute permission only if one of the execute permissions **is already set** or the file

is a directory" thank you @stephen-kitt missing the "**is already set**"



33

So, permissions in Linux are very important. I will try to make a short explanation.



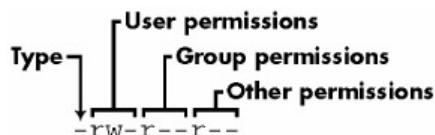
For pieces of a file mode

+50

Every Unix file has a set of permissions that determine whether you can read, write, or run the file. Running `ls -l` displays the permissions. Here's an example of such a display:

```
-rw-r--r-- 1 user somegroup 7041 M
```

I attach a image of pieces of a file mode:



Type can be different thing. For example:

- d (directory)
- c (character device)
- l (symlink)
- p (named pipe)
- s (socket)
- b (block device)
- D (door, not common on Linux systems, but has been ported)

If you want to set some permissions for all directory you can use R attribute, for example:

```
chmod -R 777 /some/directory/
```

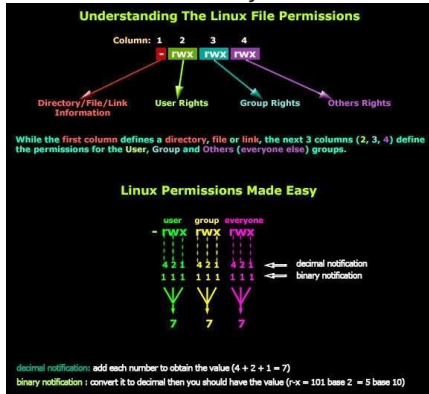
For chmod 777 vs 0777

The `chmod` command usually expects the input to be an octal number, the leading zero refers to the value of the sticky/sgid/suid bit triplet. In C however, it would make a difference, since `777` would be translated to `01411` (octal), thus setting the sticky bit (see the `chmod(2)` man page), read

(which is a rather strange combination).

EDIT 1

I found other picture about Linux permissions and I will attach to understand more easy:



edited Feb 10 '15 at 17:44



peterph

23.4k 2 44 57

answered Feb 10 '15 at 12:18



ValeriRangelov

633 5 12

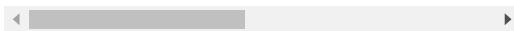
5 You are wrong about the 777 vs 0777. Both are octal (decimal makes no sense anyway in this case), but in four-character form, the first digit sets the special bits (sticky & setuid). – [orion](#) Feb 10 '15 at 14:01

3 @orion Occasionally it actually is true, e.g. in a C-like code `chmod(777)` would actually be the equivalent of running `chmod 1411` (i.e. the `chmod` command with argument `1411`). – [peterph](#) Feb 10 '15 at 17:39

2 ... which in the case of the `syscall` (or its wrapper) and the binary bearing the same name can be a bit confusing. – [peterph](#) Feb 10 '15 at 17:46

3 Python Master Race solved this backwards problem by banning all "numbers" that start with 0, forcing the user to be explicit if they want anything other than decimal, e.g. hex: `0x1FF`, binary: `0b11111111` or octal: `0o777`. PYTHON POWER – [Nick T](#) Feb 11 '15 at 3:42

Picture source? – [rugk](#) Jun 12 '17 at 23:04



d means it is a directory, if you have



If you use 0777 as permissions you are giving full control (read+write+execute) to every user/group of the system. It is a lazy way to solve problems when you have users/groups that can't access directories/files.

For example, if you list the content of a directory and get this:

```
-rw-r--r--  1 root    root
42596 jun  7  2012
preloadable_libintl.so
```

preloadable_libintl.so is a file owned by user root and group root. The *owner* has read and write access, the *group* has only read access and any *other user* has read access. This can be translated as 644.

If I change it to 777 it will look like this:

```
-rwxrwxrwx  1 root    root
42596 jun  7  2012
preloadable_libintl.so
```

[edited Feb 10 '15 at 13:12](#)

[answered Feb 10 '15 at 11:55](#)



[jcbemu](#)

3,322 8 19



6



After getting my question answered here and doing some research about the outcome I found an article which explains it all very well. I would like to share some parts of this article here for future references.

Viewing permissions

In order to use `chmod` to change permissions of a file or directory, you will first need to know what the current mode of access is. You can view the contents of a directory in the terminal by `cd` to that directory and then use:

```
$ ls -l
```

The `-l` switch is important because using `ls` without it will only display the names of files or folders in the directory.

```
total 128
drwxr-xr-x 2 peter users 4096 Jul
drwxr-xr-x 6 peter users 4096 Jul
drwxr-xr-x 2 peter users 4096 Jul
drwxr-xr-x 2 peter users 4096 Jun
drwxr-xr-x 2 peter users 4096 Jun
drwxr-xr-x 2 peter users 4096 Jun
-rw-r--r-- 1 peter users 354 Jul
```

What the columns mean

The first column is the type of each file:

- `-` denotes a normal file.
- `d` denotes a directory, i.e. a folder containing other files or folders.
- `p` denotes a named pipe (aka FIFO).
- `l` denotes a symbolic link.

The letters after that are the permissions, this first column is what we will be most interested in. The second one is how many links there are in a file, we can safely ignore it. The third column has two values/names: The first one (in my example 'peter') is the name of the user that owns the file. The second value ('users' in the example) is the group that the owner belongs to (Read more about groups).

The next column is the size of the file or directory in bytes and information after that are the dates and times the file or directory was last modified, and of course the name of the file or directory.

What the permissions mean

The first three letters, after the first `-` or `d`, are the permissions the owner has. The next three letters are permissions that apply to the group. The final three letters are the permissions that apply to everyone else.

Each set of three letters is made up of `r`, `w` and `x`. `r` is always in the first position, `w` is always in the second position, and `x` is always in the third position. `r` is the read permission, `w` is the write permission, and `x` is the execute permission. If there is a hyphen (`-`)

and if the letter is present then it is granted.

Folders

In case of folders the mode bits can be interpreted as follows:

- `r` (read) stands for the ability to read the table of contents of the given directory,
- `w` (write) stands for the ability to write the table of contents of the given directory (create new files, folders; rename, delete existing files, folders) if and only if execute bit is set. Otherwise, this permission is meaningless.
- `x` (execute) stands for the ability to enter the given directory with command `cd` and access files, folders in that directory.

Changing permissions using the `chmod` command

`chmod` is a command in Linux and other Unix-like operating systems. It allows you to change the permissions (or access mode) of a file or directory.

You can alter permissions in two different ways: - Text-based `chmod` - Number-based `chmod`

Text method

To change the permissions-or access mode of a file, we use the `chmod` command in a terminal. Below is the command's general structure:

```
chmod who=permissions filename
```

Where Who is any from a range of letters, and each signifies who you are going to give the permission to. They are as follows:

```
u - The user that owns the file.
g - The group the file belongs to.
o - The other users i.e. everyone
a - all of the above - use this in:
```

The permissions are the same as already discussed (`r` , `w` , and `x`).

The `chmod` command lets us add and subtract permissions from an existing set using `+` or `-` instead of `=`. This is different to the above commands,

still need to include `r` as well as `w` after the `=` in the `chmod` command. If you missed out `r`, it would take away the `r` permission as they are being re-written with the `=`. Using `+` and `-` avoid this by adding or taking away from the current set of permissions).

Number method

`chmod` can also set permissions using numbers.

Using numbers is another method which allows you to edit the permissions for all three owner, group, and others at the same time. This basic structure of the code is this:

```
chmod xxx file/directory
```

Where `xxx` is a 3 digit number where each digit can be anything from 1 to 7. The first digit applies to permissions for owner, the second digit applies to permissions for the group, and the third digit applies to permissions for all others.

In this number notation, the values `r`, `w`, and `x` have their own number value:

```
r=4  
w=2  
x=1
```

To come up with a three digit number you need to consider what permissions you want an owner, group, and user to have, and then total their values up. For example, say I wanted to grant the owner of a directory read-write and execution permissions, and I wanted to group and everyone else to have just read and execute permissions. I would come up with the numerical values like so:

```
Owner: rwx = 4+2+1=7  
Group: r-x = 4+0+1=5 (or just 4+1=5)  
Other: r-x = 4+0+1=5 (or just 4+1=5)
```

```
Final number = 755
```

```
$ chmod 755 filename
```

This is the equivalent of using the following:

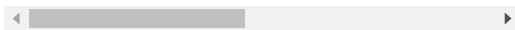
```
chmod u=rwx, filename
```

Most folders/directories are set to 755 to allow reading and writing and execution to the owner, but deny writing to everyone else, and files are normally 644 to allow reading and writing for the owner but just reading for everyone else, refer to the last note on the lack of x permissions with non executable files - its the same deal here.

[edited Jul 14 '18 at 2:56](#)

community wiki
[4 revs](#), [2 users](#) [96%](#)
[Peter](#)

Just copying text from another article is – at least – rude. Maybe also a copyright violation. If it is not only copied or you really used small parts (copyright-friendly thanks to fair use) then please at least add a link to the article you are referencing/quoting. – [rugk Jun 12 '17 at 23:03](#)



Some reading:

1

For the `d` questions:
 This tells you the Unix file type: By default Unix have only 3 types of files. They are..

- Regular files
 - `d` - Directory files
- Special files(This category is having 5 sub types in it.):
- `b` - Block file
 - `c` - Character device file
 - `p` - Named pipe file or just a pipe file
 - `l` - Symbolic link file
 - `s` - Socket file

Read more: [here](#).

For the `0777` vs `777` :
 Sticky bit specified or not. When a directory's sticky bit is set, the filesystem treats the files in such directories in a special way so only the file's owner, the directory's owner, or root user can rename or delete the file. Without the sticky bit set, any user with write and execute permissions for the directory can rename or delete contained files, regardless of the file's owner.

without changing the sticky bit.
Read more: [sticky bit](#) & [chmod](#).

edited Feb 11 '15 at 8:53



PersianGulf

6,935 4 34 61

answered Feb 10 '15 at 15:40



csny

666 2 7 22

-
- 3 While this link may answer the question, it is better to include the essential parts of the answer here and provide the link for reference. Link-only answers can become invalid if the linked page changes. – [jasonwryan](#) Feb 11 '15 at 7:09

@jasonwryan How about now? :) – [csny](#) Feb 11 '15 at 8:38

-
- 2 Better: but it still offers nothing that the other answers haven't already covered... – [jasonwryan](#) Feb 11 '15 at 8:43

(1) There are three basic file types: plain files, directories, and everything else. What? Where are you getting this? In the great scheme of things, plain files and directories are pretty similar. Named pipes and symbolic links are more like plain files and directories than they are like device files or sockets. (2) You think `chmod 777` doesn't clear the `setuid`, `setgid` and sticky bits? Try it. – [G-Man](#) Jun 1 '17 at 5:50
