

Most people who deal with \*nix systems occasionally get confused about file and directory permissions. Sometimes it’s the order of the read, write, and execute bits. Other times it’s the octal notation, or maybe how to decipher the setuid and sticky bit trickery.

My goal is for this page to serve as an instant UNIX/Linux permissions refresher and/or primer for those who either 1) never quite got it in the first place, or 2) sometimes get tripped up on the details.

**Basics**

Let’s start some output from ls:

**# ls -lah // my favorite ls options**

-rwxr-xr-- 1 daniel consultants 5K Mar 10 06:55 scanner.rb

-rwxr-xr-x 1 sarah teachers 18M Jul 30 10:07 papers.tar.bz2

* **Three Sets Of Three**(blue)  
  Since we’re looking at a file and not a directory, the first character we see starts the nine (9) character string defining three (3) types of permissions for three (3) different access levels (3 x 3 = 9). Remember to count the dashes.

* **UGO (Like the car)** (blue)  
  The way the three sets break down is according to the following order: User, Group, Other. *UGO*. So the first set of rwx‘s is for the user, the second is for the group, and the third is for everyone else.

* **See Who Owns The File** (red)  
  You can always see the user and group owners of a file but never the “other”. That’s because “other” permissions simply apply to those who are not one of the other two.

**The Three Numbers**

Quite a few people get tripped up when they see the permissions listed as a set of three numbers. There’s no cause for alarm. Just remember that this number is simply three binary placeholders, each one holding the number that corresponds to the access level in r, w, x order. Observe the chart:

**Permissions on Directories**

Also remember that permissions are different on directories than they are on files. On files they’re pretty much like you’d expect them: read the file, write/delete the file, and execute the file. But for directories it’s a bit less intuitive.

* **Read** means you can get a directory listing.
* **Write** says you can create or delete contents within the directory.
* **Execute** means you are able to enter the directory, i.e. cd into it.

**Setuid, Setguid and Sticky Bits**

This is one of the bits (it’s British for parts, lol) that gives people the most trouble. No worries.

**Setuid**

Setuid was designed to solve a very basic problem: user’s not having enough permission to run certain programs. The answer was to add an option within files to say, “Regardless of who runs this program, run it as the user who **owns** it, not the user that executes it.”

In today’s world this is a lot like getting poked in the eye with a stick. It’s just not cool, so don’t partake. Here’s the way it looks, though, in case you come across it in the wild:

-rw**s**r-xr-- 1 daniel consultants 5K Mar 10 06:55 scanner.rb

You see that little “s” in there? It stands for Satan. Notice that it resides where the “x” (execute) would have been for the user: that’s because it’s set**UID**, and it relates to execution. Also, it’s a lowercase “s” in this example because both the setuid bit and the execute bit are set. If only the setuid bit is set (and the user doesn’t have execute permissions himself) it shows up as a *capital* “S”.

[ Note: This capitalization issue applies to all of the “special” permission bits. The general rule is this: If it’s lowercase, that user **HAS** execute. If it’s uppercase, the user **DOESN’T**have execute. ]

**Setguid**

Setguid isn’t used nearly as much, but it’s the same exact concept applied to the *group* that the file belongs to instead of the user. Here’s how it looks:

-rwxr-**S**r-x 1 bjones principals 101K Aug 16 04:01 grades.xml

Notice that this time the “s” is in the group’s set instead of the user’s. Also notice that it’s a capital “S” instead of a lowercase one. Again, that means that the group (principals in this case) doesn’t have execute permissions themselves, but when the program is executed by either bjones or any random user with access to the file, it’ll run with the rights of the principles group.

**Sticky Bits**

The sticky bit had [a funky purpose](https://en.wikipedia.org/wiki/Sticky_bit) originally, but it’s now used to keep people from modifying or deleting the contents of directories that the user or group owns. You can apply it to files as well, but it’s usually employed on directories.

An example application of the sticky bit is a school assignment in which you have shared space on a server, with each student having a directory. Using the sticky bit, the administrator could ensure that once a given student created content in their respective folder, no other student could come in and delete it.

Here’s how it looks:

drwxr-xr-**t** 1 alice alice 4.4K 2007-01-01 09:21 Alice

Remember that if “everyone” (other) doesn’t have execute permissions on the directory it’ll be a **capital** T instead of lowercase.

**Changing Permissions**

To change permissions you use the chmod command and simply lay out what you want the permissions to look like on the file directory.

Change the permissions on your web directory to 755.

chmod 755 web\_directory

Change the permissions on your grocery list to 644.

chmod 644 grocery\_list.txt

**Assigning Uid, Guid, and Sticky Bits**

You can also assign the special properties to a file or directory using chmod.

Set setuid on a script

chmod 4644 script.rb

Set guid on a script

chmod 2644 script.rb

Set the sticky bit on a directory

chmod 1644 myfiles

**Conclusion**

Well, hopefully this was able to refresh you on the basic permissions and/or the special bits. If you have any comments, corrections, or any suggestions for how I could improve this document, please [let me know](mailto:daniel@danielmiessler.com).