PET: Understanding File Permissions Part 2 - Beyond “rwx”

Paul A. Marshall

1 Introduction

Ok, so you have got the hang of those read/write/execute and user/group/other concepts (if not, it’s best to get those nailed before you read this). Also you know how to view these attributes on your existing files. However, you have seen some strange letters that do not fit the model thus far, and you are curious as to their meaning. This is my attempt to fill in the gaps.

2 “s” is for “Set User ID”

2.1 How to set it

The “Set User ID” or SUID bit can be set as follows :

```
chmod 4nnn <filename>
```

or

```
chmod u+s <filename>
```

2.2 How to recognise it

If we change the permissions of a file to 4777 and list it back (in long format) the permissions will be shown as “-rwsrwxrwx”. We can now see that the SUID bit for this file has been set by the presence of the “s”. That’s fine and dandy, but now we can’t tell if the user execute bit is set, can we? Well actually, the case gives it away. A lower case “s” means that the execute bit is set, an upper case “S” means that it is clear. If we change the permissions of our file to 4677 the permissions will be shown as “-rwSrwxrwx”.

2.3 What is it for?

The SUID bit only comes in to play if the file has execute permission. When such a file is executed, the resulting process takes on the effective user ID of the owner of that file (assuming the file is not a script - Linux will not allow this for security reasons).

For exmaple, say we have a program file owned by user “paul” with permissions “rwsrwxrwx”. This file can be run by any user, however, the resulting process will have all the same access capabilities as paul. If it so chooses, it can read all the files that paul can read, it can write to all the files that paul can write to, and it can execute all the files that paul can execute. In other words, it is as if the process had been run by paul in the first place!

It is worth mentioning root SUID programs at this point. Such programs are infamous for security exploits whereby end users can derail the program and gain root access. The moral here is:

1. Only make a file a root owned SUID if it absolutely has to be.

2. Keep up-to-date with those security fixes!

3 "s" is also for "Set Group ID"

3.1 How to set it

The "Set Group ID" or SGID bit can be set as follows:

```
chmod 2nnn <filename>
```

or

```
chmod g+s <filename>
```

3.2 How to recognise it

A file with permissions set to 2777 will be displayed as "-rwxrwsrwx". As before, a lower case "s" signifies that the group execute bit is set.

3.3 What is it for?

On executable files, SGID has similar function as SUID, but as you might expect, the resulting process takes on the effective group ID of that of the file.

When applied to directories, SGID takes on a special meaning. Any files created in such a directory will take on the same group ID as that of the directory, regardless of the group ID of the user creating the file.

For example, let's say we have a directory with permissions "drwxrwsrwx" owned by the group "rockers" and a user belonging to the group "mods" (we are talking about the user's main group ID here) comes along and creates a file in this directory. The resulting file will have a group ID of "rockers", not "mods" as would be the case in a normal directory. Naturally, if our user doesn't like this, he can always issue a "chgrp" command.

On non-executable files and non-directories, the SGID bit has no effect.

4 "t" is for sTicky

4.1 How to set it

The sticky bit can be set as follows:

```
chmod 1nnn <filename>
```

or

```
chmod +t <filename>
```

4.2 How to recognise it

A file with permissions set to 1777 will be displayed as "-rwxrwsrwt". A lower case "t" signifies that the other execute bit is set.

4.3 What is it for?

On Linux systems, the sticky bit only has an effect when applied to directories. A directory with this bit set will allow users to be able to rename or remove only those files which they own within that directory (other directory permissions permitting). It is usually found on tmp directories and prevents users from tampering with one another's files.

The sticky bit can have other functions on other UNIX systems. However, apart from it's use with directories described above, there does not appear to be a universally accepted implementation.

5 Those Other Mysterious Letters - "d", "l", "b", "c", "p"

You may have come across these little fellows in your travels through your file system. I will give only a brief explanation here as they are not permission letters per se and some would justify PETs of their own.

d - Example "drwxrwxrwx". You probably haven't managed to get this far without knowing that this is a directory. I mention it here for completeness.

l - Example "lrwxrwxrwx". This is a symbolic link. A symbolic link is a file that links to another file and can be used as an alternative way of accessing that file. It is analogous to a Windows shortcut. The permissions on a symbolic link are irrelevant as it is the permissions on the target file that count.

b and c - Examples "brwxrwxrwx" and "crwxrwxrwx". These are found on special files called device files, located in the /dev directory (although there is nothing to stop them from being created elsewhere). "b" refers to block devices (such as hard drives), "c" refers to character devices (such as printers).

p - Example "pwrxrwxrwx". This is a special type of file called a "pipe". It allows two processes to pass data - one places data into the pipe, the other takes it out. This type of named pipe file is not often used.

6 Also Worth Knowing

6.1 Ext2 Specific

Ext2 file systems (the most common type used with Linux) have some extra attributes which can be set. The most notable of these from a security point of view is the "immutable" attribute which can be set as follows:

```
chattr +i <filename>
```

A file with this attribute set cannot be modified, deleted, linked to or renamed. Only root can clear this attribute. These special file attributes can be viewed via the “lsattr” command.

6.2 A Note on Other File Systems

In order for permissions to be used, the underlying file system must support it. Native Linux/UNIX file systems such as Ext2 do store file permissions, whilst others, such as Windows FAT, do not. Linux uses an abstraction layer called the “Virtual File System” or VFS which makes all file systems appear the same (ie the end user or developer does not need to get too embroiled in how each file system works). For this reason, a mounted Windows partition may appear to have permissions on it’s files, but using chmod on them is pointless.

6.3 Finally

As with read, write, execute permissions, it is possible to mix and match SUID, SGID and sticky bit settings when using the octal style parameter to chmod. An extreme, if somewhat impractical example would be:

```
chmod 7777 myfile
```

but there you have it, that’s a file with all bits set and all guns blazing ;-)