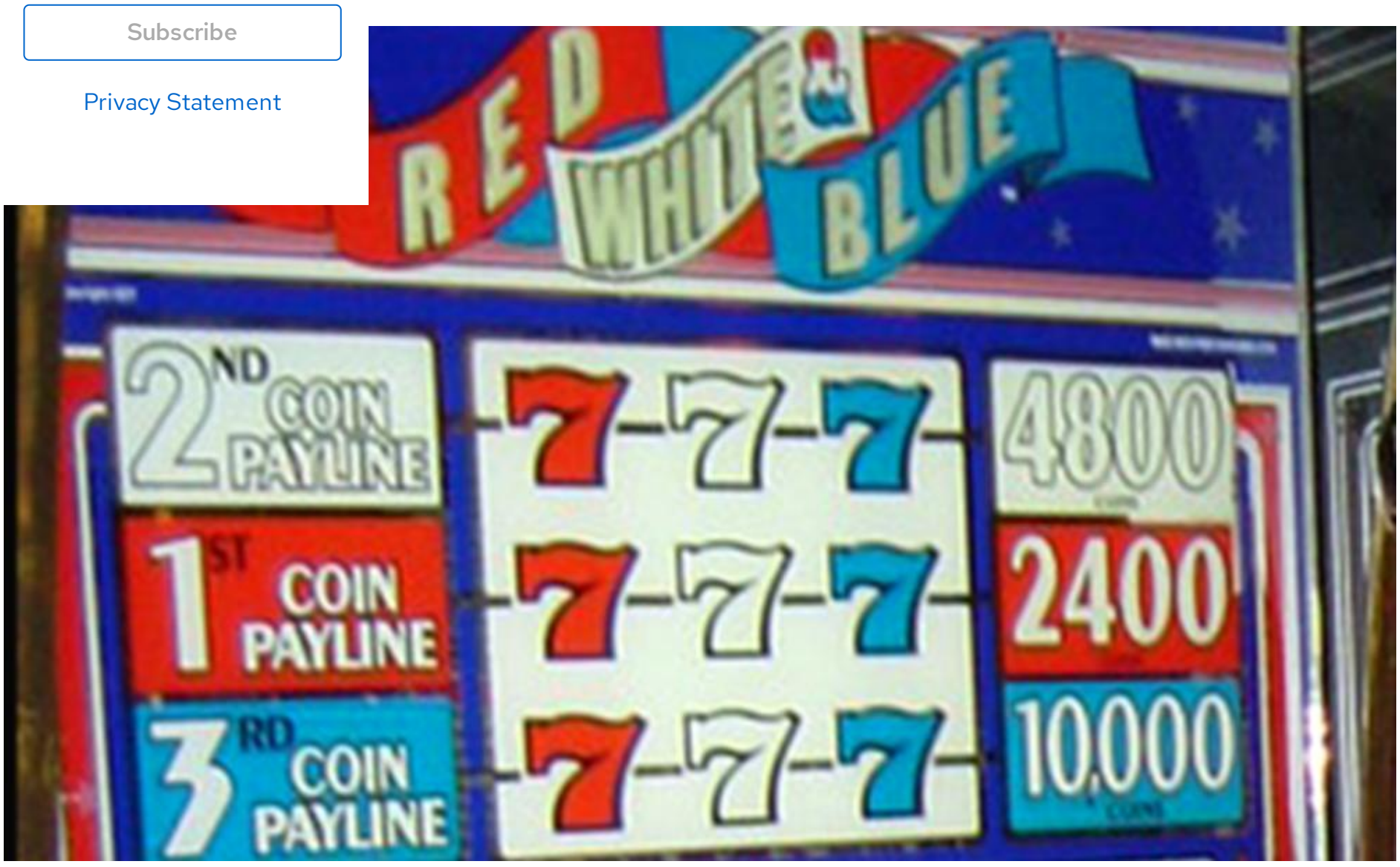
sions: An introduction to chmod

x file permissions and how to use chmod to modify them.

shank Nandishwar Hegde (Red Hat, Sudoer)



[Want to try out Red Hat Enterprise Linux? Download it now for free.]

If you have ever worked Linux system, you for sure have dealt with files, and that means that you might have encountered messages like this one below:

```
localhost@user1$ cat /etc/sudoers
cat: /etc/sudoers: Permission denied
```

Or, similar to this, error messages like "You do not have the permissions to upload files to this folder," which might have prevented you from reading, writing, or accessing a particular file. And, naturally, this error might have made you wonder—the first time you encountered this problem, at least—as to why you were denied access.

Let's take a look into Linux file permissions and the ways to restrict them, plus play with files a little bit. When you list files in a particular directory in Linux, you might have seen r, w, and x, and wondered what these letters mean. They have tremendous significance in determining what exactly a particular user can do with a file.

Let's take a look at an example:

```
localhost@user1$ ls -ltr chmod_directory/
```

```
-rw-r--r--. 1 creator creator 0 Jul 29 21:55 I_Can_Access.txt
```

Default file permissions are rw-r--r-- (from the umask value (covered later in the article)), as shown in the example

value assigned to it:

1

ach "triplet", meaning that a file permission of rw- has the value of 6 and rwx has the any file that's newly created, the default value is 644 (rw-r--r--), meaning that the file's ll others can only read this file. The first triplet is the permission for the file group permissions, and the third is for others (users outside of the owner/creator or a etting makes sense for obvious reasons: The owner should have higher control over the ead and write to it. Others might want to read the contents but not modify them. Of course, you can change this setting with the `chmod` command, which is the focus of this article.

So to understand this concept in a simpler way, think of file permissions as a 3x3 matrix, where owners, groups, and others each have r, w, and x settings. In the above example:

- The file's *creator* (owner/user) has read and write permissions: -**rw**-r--r--.
- The file's group creator (group) has read permissions: -rw-**r--**r--.
- Others have read permissions represented by the last bits: -rw-r--**r--**.

Now, let's see the default permission values for a directory. Let's say the directory `chmod_directory` was created with the default permissions of 755. Unlike files, a directory has files in it. In order for anyone other than the owner to '`cd`' into the directory, it needs an execute permission, which in turn makes the directory:

- Readable, writable and executable by the owner (rwx is 7).
- Readable and executable by the group (r-x is 5).
- Readable and executable for others (r-x is 5).

Note: The r-x designation does NOT mean r minus x, it means read and execute but missing write. The - is a placeholder for a permission.

(Please take a minute to think about why this is the default behavior.)

Ok, now that you have learned the basics of file and directory permissions, let's take a look into the `chmod` command, which helps with making permission changes for files and directories.

As mentioned in the man page:

```
This manual page documents the GNU version of chmod. chmod changes the file mode bits of each given
ich can be either a symbolic representation of changes to make, or an octal
t pattern for the new mode bits.

ode is [ugoa...][[+-=][perms...]...], where perms is either zero or more
t, or a single letter from the set ugo. Multiple symbolic modes can be
.

rs ugoa controls which users' access to the file will be changed: the user
rs in the file's group (g), other users not in the file's group (o), or all
e are given, the effect is as if a were given, but bits that are set in the
```

sentation

ou can either use octal representation (numeric), or symbolic representation (the letters). In octal representation, the first digit is for the user, the second digit is for the group, and the third digit is for others. Let's look at two examples of setting permissions with octal representation to understand this concept.

Example 1: If you want to give read (4), write (2), and execute (1) permissions to both the user and group, and only read (4) permission to others, you can use:

```
localhost@user1$ chmod 664 <file-name>
```

Example 2: If you want to restrict write permissions to all others except the file's owner, you can use:

```
localhost@user1$ chmod 744 <file-name>
```

Using symbolic representation

You can also change permissions using symbolic representation rather than numeric. Symbolic representation is assigning permissions to user (u), group (g), and others (o) using letters (symbols) and the letter designations: r, w, and x.

Let's look at these examples again, but using symbolic representation.

Example 1: Read, write, and execute for the user and group, plus only read for others, maps as:

```
localhost@user1$ chmod ug+rw,o+r <filename>
```

Example 2: Read, write, and execute for the user and only read permissions for group and others maps as:

```
localhost@user1$ chmod u+rwx,go+r <file-name>
```

Awesome, I'm proud of you all: You have now mastered file permission concepts. But I'll caution you that there are two dangerous scenarios that you might want to avoid, so keep this as a best practice while using chmod. Avoid using boundary cases, such as `chmod 777 <file-name>` and `chmod 000 <filename>`. Using `chmod 777 <file-name>` gives everyone rwx permissions, and it is generally not a good practice to give full powers to all the users in a system. The second case, I will leave you guys to figure out.

Using umasks

I will leave you guys with one more concept that you need to be aware of (umask) that decides the default permissions for a file. Overall, the default values are:

- Directory: 0777

As you might remember, the default file permission value is 0644, and the default directory's is 0755. The default the overall file/directory default value. You can set the umask values in `/etc/profile`

nd for manipulating file and directory permissions. With the concepts mentioned in this ufficient knowledge to handle permissions in Linux-based distros.

Shashank Nandishwar Hegde

I work as a Solutions Engineer at Red Hat and my day-to-day work involves OpenShift and Ansible. I'm highly passionate about open source software, cloud, security, and networking technologies. More about me

Related Content

An introduction to web application firewalls for Linux sysadmins

You've used host-based firewalls and network firewalls, but have you ever considered implementing a web application firewall? You should.

Posted: September 30, 2020
Author: Jason Frisvold (Sudoer)

How to manage cgroups with CPUShares

Managing access to CPU time using CPUShares with cgroups in part two in this four-part series covering cgroups and resource management.

Posted: October 2, 2020
Author: Steve Ovens (Red Hat)

A Linux sysadmin's introduction to cgroups

Defining cgroups and how they help with resource management and performance tuning in this first article kicking off a four-part series covering cgroups and resource management.

Posted: September 29, 2020
Author: Steve Ovens (Red Hat)

OUR BEST CONTENT, DELIVERED TO YOUR INBOX

Subscribe now

Get the highlights in your inbox every week.

x

Enter your email address...

Select your country or region

Subscribe

Privacy Statement