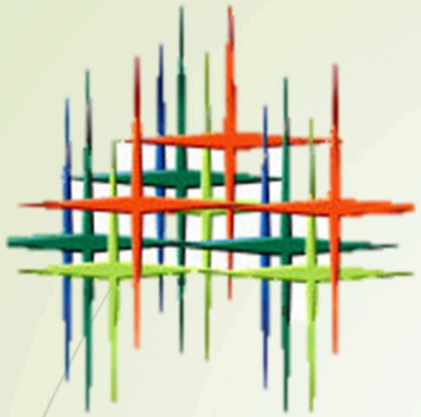

ABC IT Education

WE'LL TAKE YOU FROM ZERO TO HERO IN A SNAP

Linux Systems Administration

Linux File and Directory Permissions

- We'll be looking at symbolic permissions, numeric permissions, file versus directory permissions, how to change permissions, how to work with groups, and the file creation mask.
- Here below is output from an 'ls -l' command.

```
drwxrwxr-x@    3 bryce  staff    96B Apr  9 18:35 Creative Cloud Files
drwx------+    7 bryce  staff   224B Aug 21  2017 Pictures
-rw-r--r--     1 bryce  staff     0B Jul 25  2017 first_day.html
```

- If you look at the permissions string, the first character will indicate whether it's a regular file by beginning with a '-', or it will begin with the 'd' if it's a directory, or an 'l' if it's a symbolic link.
- Some other characters that you'll encounter in the permissions string include are r, w, and x.

- r, w and x represent the 3 main types of permissions.
- They are read (r), write (w), and execute (x).
- If you have read permission to a file, that means you can see its contents. For example, you could run cat against the file and you would see the file's contents.
- If you have write permissions to a file, you can modify it. You can change its contents.
- If you have execute permissions, you can run that file as a program.
- Read, write and execute are fairly self explanatory when applied to files.

- When read, write, execute permissions are applied to directories, they have a slightly different meaning.
- For example, when read permissions are applied to a directory, that means you can see the file names in the directory.
- If you don't have read permissions to a directory, we'll not be able to see the directory's contents.
- The write permission when applied to directories, allows entries to be modified within the directory, so you can edit files that are in the directory.
- The execute permission gives you the ability to see metadata about the files that are in the directory.

- Read permissions give you the ability to see filenames, whereas the execute permissions give you the ability to see modification dates and owner and group information just like you would see in an 'ls -l' long listing output.
- There are categories of users that these read, write and execute permissions can be applied to. And these categories are user, group, other and all.
- Like the permission types, each one is represented by a single letter, 'u' represents the user that owns the file.
- The users that are in the files group are represented by 'g'.
- Users that are not the owner nor are not in the files group are considered other represented by 'o‘ and 'a' is used to represent all or everybody.

- The ‘**groups**’ command will show what groups you are a member of, and if you supply another user's ID as an argument to the ‘groups’ command, you'll see a list of groups to which that user belongs.
- You can also use **'id -Gn**' as a synonym for the ‘groups’ and it will give you the same output.
- In this example, running the **groups** command shows that I am in the adminuser group and the sales group.
- You can also see that **'id -Gn**' returns the same value as the groups command does.
- If you run the **groups** command, followed by another username, you'll see the groups that user is in.
- For instance, msaks' is in the finance and msaks groups.

- You now have enough background information to start decoding the permission string.
- We said that the first character is the type via a directory (d), a file (-) or a link(l).
- The next 3 characters represent the user (owner of the file) permissions.
- The next 3 characters represent the permissions for the members of the files group.
- The final 3 characters represent the permissions available to all users.
- The order of permissions is significant, so permissions will always be displayed in the order: user, then group, and then others.

- The read, write and execute permissions are also always displayed in that order, so if a particular permission is not granted, a hyphen will take its place.
- The command used to change permissions is called '**chmod**', which is a short for change mode, so permissions are also known as modes.
- The format of the chmod command is 'chmod' mode file and the mode is either symbolic notation or numeric notation.
- To specify modes or permissions with symbolic notation, run the chmod command, followed by user, group, other, or all, an operator to add (+), subtract (-) or set (=) permissions, then the permissions - read and/or write and/or execute.

- You can add, subtract, or set permissions using user category, user (u), group (g), other (o), or all (a) and permission pairs, read (r), write (w) or execute (x).
- **Examples:**
- (1) Add execute permissions to other for file named kkk
- $ chmod o+x kkk
- (2) Add write, execute permissions user, group for file named myDocs
- $ chmod ug+wx myDocs
- (3) Add read permissions to group and remove write from others for file named userGroups
- $ chmod g+r,o-w userGroups

- (4) Remove all permissions from group and other for file named userOnly
- $ chmod go-rwx userOnly
- (5) Add write and execute permissions to all users for file named fullAccess
- $ chmod ugo+wx fullAccess or chmod a+rw fullAccess

- From the examples above you see that you can also change more than just one permission at a time, so you can specify the group have write and execute permissions.
- If you want to set different permissions for different user categories, you separate the specifications with a comma.

- As an example we can specify something for the user add read, write, execute, then for group, take away execute, and the command will be chmod u+rwx,g-x <files[s]>
- Specifying an equal sign, sets the permission to exactly what you specify.
- Assume all (a) is set to read as such chmod a=r ccc
- Executing such a command, the owner or the user, the group and other, are all set to the read permission only.
- **Note**: if you don't specify permissions after the equal sign, then all the permissions are removed.
- **Exercise**: In your adminuser home directory create an empty file **$ touch myFile** What are the permissions of myFile? Set the permissions of myFile to read for all
  $ chmod a=r myFile

- Now let's set the user to read, write, execute and group to read, execute, and no permissions to others all in one command.

- In addition to symbolic mode, there is an octal mode or numeric mode to set file and directory permissions using chmod.
- In octal mode permissions are based in binary, each permission type is treated as a bit that is either off - 0, or on - 1.
- As before, with permissions, order has meaning, so the permissions are always in read, write and execute order.
- In octal mode, if r, w and x are all set to off, the binary representation is a 000 and if all set on, the binary is 111.

- If you want to allow write permissions the binary representation would be 010 and to get the number that you can use to chmod we convert the binary representation into base ten or decimal.
- With all this in mind, what we need to do here is to remember that read=4 (100 in binary), write=2 (010), and execute=1 (001).

Numeric Based Permissions

r	w	x	
0	0	0	Value for off
1	1	1	Binary value for on
4	2	1	Base 10 value for on

- The permissions number is then determined by adding up all the values for each permission type, and there are 8 possible values from 0 to 7, hence the name, octal mode.
- Below is a lists all 8 possible options.

Octal	Binary	String	Description
0	0	---	No permissions
1	1	--x	Execute only
2	10	-w-	Write only
3	11	-wx	Write and execute (2+1)
4	100	r--	Read only
5	101	r-x	Read and execute (4+1)
6	110	rw-	Read and write (4+2)
7	111	rwx	Read, write, and execute (4+2+1)

- The user categories are always in the user, group and other order, so once you determine the octal value for each category, you specify them in that order.
- If for example you want read, write, execute for user, read and execute for group, and just read for other, permission would be set by 'chmod 754 filename'.
- **Here are the most commonly used permissions.**
- 700 ensures that a file can be read, edited and executed by the owner and no one else on the system will have access to that file.
- The 755 permission allows everyone on the system to execute the file but only the user or the owner of the file can edit that file.

- 664 allows a group of people to modify the file and others read it.
- 660 allows a group of people to modify the file and not let others read it.
- Mode 644 allows everyone on the system to read the file but only the user or the owner of that file can edit that file.
- If Giving 777 permissions gives everyone on the system full access to that file or directory, so if you're ever tempted to use 777 or 666 for permissions, think twice and look for a better way to set the permissions.
- If a program or a script is set to 777 permission, then anyone on the system can make changes to that script or program and since the execute bit is set for everyone that program can be executed by anyone on the system.

- If, for whatever reason, malicious code was inserted in the script or program it can cause unnecessary trouble.
- If multiple people need write access to a file, consider using groups and limiting that access to the members of that group.
- **NOTE**: it's good to avoid 777 and 666 permission modes.
- **Groups**
- When you create a file, it's group is set to your primary group.
- Remember a user may belong to more than one group, so if you're a member of multiple groups, say sales and training groups and sales is your primary or first group, then all files you create will be in the sales group.

- If you want to change the group of a file, use the '**chgrp**' command.
- You can see that when I created the sales.dat file,
- that it was put in my primary group, which is adminuser, but I am also a member of the sales group so let me change the group to sales for this file.
- $ chgrp sales sales.dat
- Now do an ls –l on the file and see that the group is sales.
- Exercise:
- Give permission to sales.dat so that other people in the sales group can edit the file.

- The beauty of this is that, instead of keeping files in different people's home directories, you could have a common place to store these files
- So if there's a /usr/local/sales/directory on your system, you can put all the sales files there and give the sales group read, write and execute permissions for those files and anyone that is a member of the sales group can edit that file.
- **Directory Permissions**
- Permissions on a directory can affect the files in that directory, and a common problem is having proper permission set on a file within the directory, only to have the incorrect permission set on the directory itself.

- Not having the correct permissions on a directory can prevent the execution of the file for example.
- If you are sure that a file's permission is correct, start looking at the directory it's in and then look at that directory's parent directory and work your way all the way towards the root of the file system.
- Create a directory called **dirPerms** in your home and make sure the permissions are set to 755 on this directory, and if they are not set them to 755.
- Now create a file in the new directory and make it executable.
- If you do an ‘ls’ in that directory, you will see one file and that it’s an executable file, so it works.

- Now let's change the directory permissions on **dirPerms** to 400 so I only have read permission and no write or execute permission on that directory.
- At this point the only information I get from ls, is the filename that is in that directory and that is because of the read permission.
- Since the permissions are not set correctly on the directory, I am unable to execute the file that's in that directory, even though the file itself is executable
- Give execute permissions on **dirPerms** directory, we see that it works. Also 'ls –l' works.
- The file creation mask is what determines the permissions a file or directory will get when created.

- If no mask is used, then the default permissions would be 777 for directories and 666 for files.
- The creation mask is typically set by system administrators to some default value, but it can be overridden on a user by user basis using the '**umask**' command.
- The '**umask**' command sets the file creation masks to the mode that you pass to it, and if you use a '-S', that means **umask** will display and accept symbolic notation.
- The mode supplied **umask** works in the opposite way as the mode given to **chmod**.
- When you give **chmod** 7, that's interpreted to mean read, write and execute permission or all permissions, but when you supply 7 to **umask**, that is interpreted to mean no permission or all permissions off.

- You can actually think chmod as turning on or adding and giving permissions, and umask turning off, subtracting or taking away permissions.
- A quick way to estimate what a umask mode will do to the default permissions is to subtract the octal umask mode from 777 in the case of directories, and from 666 in the case of files.
- If the umask is 022 and you subtract that it from 777, we are left with the default file creation mode of 755 for directories and 666 minus 022 is 644.
- For files, the default file creation mode is 644, with a umask of 002, then we'll have the default permissions for directories of 775 and 664 for files.

- Using a umask of 002 is ideal for working with members of your group since the permissions allow members of the group to manipulate those files and directories that you create.
- Note that this method of subtracting these permissions is an estimation.
- Here's an example of using a umask where this breaks down a little bit so umask 007, if you subtract that from 777, you're left with 770 for directory permissions, which is fine, but 666 minus 007 will leave you with 66 negative 1, but there is no negative one permission, there's just no permission.
- This means it breaks down a little bit, but it gives you a good idea of what to expect.

- Here are some fairly common umask modes - 022, 002, 077, and 007.
- The table below contains all the resulting permissions created by each and everyone of the 8 possible umasks permutations.

Octal	Binary	Dir Perms	File Perms
0	0	rwx	rw-
1	1	rw-	rw-
2	10	r-x	r--
3	11	r--	r--
4	100	-wx	-w-
5	101	-w-	-w-
6	110	--x	---
7	111	---	---

- If you were to run umask without any arguments,
- it will display the umask and the 4 characters
- instead of the 3 that we've been working with.
- The 3 characters we've been working with represent user, group and other, but there is one other class and this class is considered special modes.
- These special modes are setuid, setgid, and sticky.
- These special modes are declared by prepending a character to the auto mode that you normally use with umask or chmod.
- The important point here is to know that umask 0022 is exactly the same as umask 022 or chmod 0644 is the same as chmod 644.

- We'll cover the special modes later in this course but for now I wanted you to be aware (1) that they exist, and (2) - that they are the reason why umask is displayed in 4 characters instead of 3.
- You can see that our umask is set to 0022 and using a capital 'S', we can get symbolic mode.
- Now let's see what the default permissions are.
- The **touch** command either creates a file if it doesn't exist
- or it updates the timestamp of a file.
- So we see the directory was created with 755 permissions
- and the file with 644.

- Let’s set the umask to 007. $ umask 007.
- Now let’s create a directory and a file as follows
- $ mkdir umaskDir and $ touch umaskFile, then check their permissions by doing a long listing.
- You would notice that directories are created with 770 permissions and files are created with 660 permissions.
- Summary: Permissions can be represented by symbols or numbers. The effect permissions have on directories is slightly different than they have on files.
- We talked about how to change permissions with a 'chgrp' command. We talked about some strategies of working with members of your group. Then we covered the file creation mask and the umask command.

- Permissions from a long listing

Numeric Based Permissions

r	w	x	
0	0	0	Value for off
1	1	1	Binary value for on
4	2	1	Base 10 value for on

Octal	Binary	String	Description
0	0	---	No permissions
1	1	--x	Execute only
2	10	-w-	Write only
3	11	-wx	Write and execute (2+1)
4	100	r--	Read only
5	101	r-x	Read and execute (4+1)
6	110	rw-	Read and write (4+2)
7	111	rwx	Read, write, and execute (4+2+1)

	Directory	File
Base Permission	777	666
Subtract Umask	-022	-022
Creations Permission	755	644

	Directory	File
Base Permission	777	666
Subtract Umask	-002	-002
Creations Permission	775	664

Octal Subtraction Is an Estimation

	Directory	File
Base Permission	777	666
Subtract Umask	-007	-007
Creations Permission	770	660 *