8.9.1 Permissions

Click one of the buttons to take you to that part of the video.

Permissions 0:00-1:14

In this lesson, we're going to talk about permissions. Understand that managing ownership of files and directories represents only a part of what you have to do in order to control access to the Linux file system.

Basically, ownership only specifies who owns what. Ownership does not say what that person can or can't do with those files or directories that they own.

In order to do this, you have to set up and manage permissions. In this lesson, we're going to talk about how permissions work, and we're going to talk about how to manage permissions from the command line.

Let's begin by discussing how permissions function. Permissions basically specify what a particular user may do with the files and directories in the file system. For example, these permissions may allow a user to view a file but not modify it.

They could allow a user to open and also modify that file and save the changes. Permissions may even allow a user to run an executable file. In fact, permissions can even be configured to prevent a user from even seeing a file within a directory in the file system.

Understand that each file and directory in your Linux file system stores the specific permissions that have been assigned to it in the mode of the file.

Linux Permissions Mode 1:15-3:01

If you hear someone refer to a file or directory's mode, we're talking about the permissions that have been assigned to that file or directory. A list of the possible permissions that can be assigned to either a file or a directory are listed here.

The first one is the read permission. You'll see the read permission represented by the r symbol. If you assign the read permission to a user to a file, it will allow that user to open and view the file, but it does not allow that user to actually modify and save changes to that file.

On the other hand, if we were to assign the read permission for a user to the directory instead of a file, then it allows the user to actually list the contents of that directory. Essentially, it allows them to see what files and subdirectories exist within a given directory.

You can also assign the write permission, which uses a symbol of w. If you grant a user the write permission to a file, it allows that user to open, modify, and save changes to that file. Whereas the read permission allowed them to look at the contents of the file, the write permissions allows them to modify the contents of a file.

On the other hand, if you grant the write permission to a user to a particular directory in the Linux file system, then that user is allowed to either add or remove files from that directory.

The last permission you need to be familiar with is the execute permission, and it's represented by the letter x. If you grant a user the execute permission to a file, then that user is allowed to execute or run that file.

On the other hand, if you grant a user execute permission to a directory, something totally different happens; it just allows the user to actually enter into that directory with the cd command.

We're pretty good thus far. We have the read permission, the write permission, the execute permission, and each of these three different permissions can be assigned to either a file or to a directory in the file system. Here's where things get a little bit confusing.

File System Entities 3:02-5:36

Understand that these three permissions--read, write, and execute--can be assigned to three different entities for each and every file, and each and every directory in the file system.

The first entity is the owner. The owner is the user account that has been assigned to be either a file or a directory's owner. We can assign read, write, or execute permissions to the owner. Those permissions will be applied only to that user account--to no one else.

In addition to owner, you can also assign permissions to group. Now, the group is the group that's been assigned ownership of a particular file or directory. We can assign group read, write, or execute permissions as well.

Any permissions that are assigned to group will be automatically applied to all user accounts that are members of that group. Then the last one, and the one that folks frequently forget from a security perspective--but really shouldn't because it's very important--is others.

Others refers to all other authenticated users on the Linux system. It is anyone who has logged in who is not the files owner, or directory's owner, and is not a member of the group that owns a file or directory. Someone who falls into this category is considered an other.

You can actually assign read, write, and execute permissions to others as well. Remember, I said a minute ago a lot of times we forget about others from a security perspective and we should not. That's because you need to be very careful about what permissions you assign to others.

Basically, every user on the system in one fashion or another belongs to others; therefore, any permission you assign to others basically gets assigned to anybody that authenticates to the system. In some situations this can be really useful. But in other cases, it creates a security hole and can get you in a lot of trouble.

Basically, others should have minimal permissions assigned--only enough to do what they need to do. If every authenticated user on the system does not need access to a particular file or directory, then don't grant unnecessary permission to others.

If you want to view the permissions that have been assigned to a particular file or directory in the file system, you use the ls -l command. When you do, the permissions for each directory or file are displayed. They're displayed over here on the left.

Now, this first column is the mode that we talked about earlier for the particular file or directory. The very first character in the mode identifies whether it is a file, whether it's a directory, or whether it's a symbolic link. If you see a d, as in the case for this entity in the file system, then we know that it's a directory.

Permission List 5:35-9:52

On the other hand, if we see a dash (-), then we know that this entity is a file. If you see an l--the letter l, a lowercase l--then it's a symbolic link.

After that first character, the next three characters identify the permissions that have been assigned to the owner of that directory or file. For the newfile file, the owner of that file has been granted r and w, but not x.

That means that the owner of the file named newfile has read and write permissions to that file. But because there's a dash (-) in the third spot, that means that execute hasn't been assigned. That's really not a problem because this is a text file. It contains data. It's not an executable, so it's not needed anyway.

If this file were an executable, then we would want the owner to be able to execute it. An example of that is shown down here for the file named zombie. Zombie is an executable. Therefore the file owner needs read, write, and execute permissions to this file so that the file owner can not only read and write to it, but can also execute it when necessary.

If you aren't sure who the owner of the file is, you can jump over one column right here. That tells us the name of the owning user. And the next column tells us the name of the owning group. In this case, newfile is owned by the rtracy user and is also owned by a group named rtracy.

By way of comparison, the Project_design.odt file is owned by the rtracy user, but the name of the owning group is users. This is important because the next three characters in a file's mode are the permissions that are assigned to that owning group. For the newfile file, the owning group, which is the rtracy group, has read and write permissions to that file as well.

With this particular file, the user who owns the file can open the file, access it, modify it, and save changes. In addition, any user who's a member of the owning group can also open the file, modify it, and save the changes.

Notice down here for the Project_design.odt file, the owning group has only the r permission assigned--it does not have the w permission assigned--whereas the owning user has both read and write permissions assigned.

Therefore, the user who owns Project_design.odt can edit the file and save changes, but any user who's a member of the owning group will be able to only look at the contents of the file. They will not be allowed to edit it.

Finally, the last three characters in the mode are the permissions that are assigned to others, which is basically any legitimately authenticated user on the system who is not the owner and is not a member of the owning group. For the newfile file, every user on the system has read access to this file.

The same is true for the Project_design.odt file. All authenticated users on the system who are not the owner and not a member of the owning group will be able to look at the contents of this file. Remember, I said earlier that you need to be very careful about managing the permissions that are assigned to others.

What if this document is very sensitive? What if we don't want every other user on the system to be able to read its contents? We might want to consider removing the r permission to prevent prying eyes from seeing what's in this file.

Before we go any further, you need to be aware that these permissions for each entity can also be represented numerically. This is done to save space. Because, as we saw previously, specifying the three different permissions for each of the three different entities takes up a little bit of space.

To make things a little more concise, we can represent permissions by a number. The read permission is assigned a value of 4, the write permissions is assigned a value of 2, and the execute permission is represented by a value of 1.

Now using these numbers, we can then represent all of the permissions that are assigned to owner, all the permissions that are assigned to group, and all of the permissions that are assigned to others with a single number. All you have to do is add up the value of each permission that's been assigned.

In this example, the owner of a file has been assigned the read and write permissions. Read has a value of 4. Write has a value of 2. We add the two together, and we get a value of 6. If you see 6, you know that the entity represented by that permission has read and write permissions.

Whenever you see numeric permissions identified in this way, remember that the first digit represents owner, the second digit represents group, and the third digit represents others.

Numeric Permissions 9:51-11:45

In this case, the owning group has only the read permission assigned. Therefore, we don't have to add anything up. There's just a value of 4. Same for others. Others has the read permission assigned, so it has a value of 4 as well.

The mode of the file with these permissions assigned to it can be represented more concisely by just the number 6, 4, 4. If we were looking at this mode in the output of the ls -l command, we would see it represented in this way: rw, r, and r for user, group, and others.

So what do you do if the permissions that have been assigned to a file or directory aren't correct? You can modify them using this utility right here: chmod, which stands for change mode.

Be aware that in order to do this, you must either already be the owner of a file or directory, or you must be logged in as the root user. Any other user will not be allowed to change the mode of a file.

That just stands to reason. Imagine the havoc you would cause if any user could go into any file and change its mode. That would be a security nightmare.

Now, there are different syntaxes that can be used with a chmod command. The first one is shown here where we enter chmod, and then we enter the entity that we want to assign permissions to, and then an equal sign (=), and then the permissions that we want to assign to that entity followed by the name of the file or directory whose mode we want to change.

An example is shown down here. We run 'chmod' and we say u=rw, which grants the owner of the directory read, write permissions. Now, because we used an equal sign, whatever we specify here overwrites what already may be in the mode for user.

If the user had rwx permissions to this file for some reason, then by using u=rw, the owner would then have only read-write permissions. The execute permission would be removed. We also specify that the owning group gets read-write permissions. And then o over here refers to others, and others get the read permission.

Now this is a point of confusion right here because o makes us think of owner, right?

Using chmod 11:42-14:35

Well, owner is not o. Owner is u. Others is o. Don't get tripped up by that.

Then we specify the name of the file whose mode we want to modify. Now you can also use the chmod command to simply toggle one particular permission off or on using either the plus or the minus sign for a particular entity.

For example, let's suppose that I want to turn off the write permission that we just gave to the owning group for the Project_design.odt file in the previous example. To do this, I would enter chmod g-w and then the name of the file. That will turn off the write permission.

If I decide later that was the wrong thing to do and I need to turn the write permission back on for group, I would enter the same command again. But this time, I would use a plus sign between g and w to tell the chmod command that we want to turn on the write permission for the owning group.

If I wanted to toggle a permission for the owner of the file, I would have used u instead of g. If, on the other hand, I wanted to toggle on permission for others, I would have used an o instead of a g in the command.

Of course, if I wanted to add the read permission instead of write, I would have added r. If I wanted to manipulate the execute permission I would have used an x instead of w.

There's a third syntax that you can use with the chmod command. This is one where we represent the entire mode that we want to assign to a particular file or directory numerically. To do this, we enter chmod, followed by a numeric representation of the mode that we want to assign to either the file or directory in question.

For example, here we enter chmod and then we enter the numeric mode 660, and then the name of the file, Project_design.odt. So, 660. What permissions are being assigned to the three entities?

The first number is user. If we have 6, and 6 we know that's 4 plus 2, that means we have read, 4, and write, 2.

The second digit applies to group. Again, we have 6. We know that is read, 4, write, 2. And then others gets zero. That means others gets nada. No permissions at all.

Before we end, I want to make you aware that you can use the -R option with the chmod command. That is extremely useful. In the examples we've been working with here, we've been assigning a particular mode to one file at a time.

There may be situations where you would need to change the mode of many files within a directory structure all at once. If you had to go through and manually assign the mode to every single file and directory, it would take a very long time.

If you want to do an entire directory structure all at once, add the -R option to the command, and then it will recursively apply the permissions you specified to every file and every subdirectory within the directory that you specify.

Basically, it takes care of it all at once. But use it at caution. You need to make sure that every file and directory within the directory structure you specify will have exactly the same mode. If there are files that don't have that mode, you're going to have problems because they're going to get assigned to it anyway if you use the -R option.

Summary 14:36-14:47

That's it for this lesson. In this lesson, we reviewed how permissions are used to control access to files and directories in the Linux file system. We also reviewed how you can use the chmod utility in order to manage permission assignments.
