8.10.2 Use umask

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

Use umask 0:00-0:24

In this demonstration, we're going to work with umask. You can use the value assigned to umask to modify the default permissions that are assigned to files and to directories when they're created in the Linux file system. With that in mind, we need to look at the default permissions that are assigned to files and directories when they're created in the file system.

Create a File 0:25-1:19

Let's first create a file. Let's just do a 'touch' command and make a new file called 'mytestfile', and let's also make a new directory. Let's do 'mkdir temp'. Let's run the 'ls -l' command so we can see the mode, the permissions, that were automatically assigned to this file and this directory.

Okay, first of all, let's look at the file. We can see that the owner of the file, the rtracy user, is assigned read-write access to the file but no execute permission, which makes sense because it's not an executable file. The owning group also received read-write access to the file, and all other authenticated users on the system received read access to the file.

If I were to put something in this file, I could open it and edit it, and anyone who is a member of this group--it's just me--would be able to open it and edit it. Any other user on the system would still be able to see the contents of the file but not be able to modify it.

Use Default Modes 1:20-3:10

Let's look at the /temp directory down here that we've just created. By default, the user that owns the directory, me, is granted read, write, and execute permissions to the folder; meaning that they can list the file contents, they can add and delete files from the folder, and they can also enter into the folder.

The same permissions were assigned to the owning group--which again, is my group, and I'm the only member of it--but they receive the same read, write, and execute permissions to the folder.

All other authenticated users to the system have read access to the folder and they have execute access to the folder, so they can list folder contents and they can go into the folder, but they're not allowed to add or remove files from the folder because they did not receive the write permission.

Understand that the Linux kernel, whenever you create a file in the file system, actually assigns a mode of 666 to files and 777 to directories when they're created. However, this is not the mode that we're seeing. If we had 666 permissions assigned to mytestfile, we would have read-write, read-write, and read-write here.

Remember, read is assigned value of 4, write is assigned value of 2; that's where we get 6 from. There's 6, there's 6, but for others, we have a value of 4. Somewhere along the line, the write permission was blocked. That's where we lost the 2. That's why it's 664 instead of 666.

Likewise, with the /temp directory, there's our first 7 assigned to the owner (4, 2, and 1)--seven, and likewise to group (4, 2, and 1)--seven. But notice for other authenticated users, we have 4 and 1, so that's 5. We have 775 instead of 777. Once again, the write permission, which has a value of 2, was removed. This was done by umask.

View the Value of umask 3:11-7:08

You can view the value of umask by just entering 'umask' at the shell prompt, and here you can see that we have a value of 0002. This 0 specifies which permissions are subtracted from the owner. This digit right here specifies the permissions that are subtracted from owner. This digit specifies the permissions that are subtracted from group, and this digit specifies the permissions that are subtracted from other.

It doesn't matter if we're dealing with files or directories; the same umask values are applied to customize the mode. As you can see, because we have 0 here, no permissions are removed from the owner and no permissions are removed from group, but we have a value of 2 here.

This value of 2 specifies that the write permission be removed. And because the 2 is in this place right here, it specifies that the write permission be removed from others right here.

Be aware that not all distributions will use the same umask. Most of them that you work with will probably use this one: 002. However, I know that there are some that use a value of 022 instead of 002, which would remove the write permission from group as well by default.

If, for some reason, you're not happy with the default modes that are being assigned to your folders and files, then you can actually modify the value of umask--customize what mode is automatically assigned to the files and folders in your file system when they're created.

For example, over here under directory, notice that by default, we are giving other authenticated users permission to enter into, basically, all the directories when they're created in the file system, unless we manually go in and customize their permissions after the fact. From a security standpoint, that may not be the greatest idea.

Maybe we don't want to give them the ability to go into all these different directories and browse through the file system. Because remember up here, because of the mode that we assigned, they do have read-accessed files, meaning that they could poke around through all the different files in the file system and actually view their contents.

They won't be able to change them, but they'll be able to see them. From a security standpoint, that could be concerning.

Perhaps a better security posture would be to actually remove the execute permission from others when a mode is assigned to a directory. That way, they cannot go down a level into those directories and open up files within them.

What is the value assigned to x? It has a value of 1; therefore, if we want to remove the x permission from others with umask, what would we set the value of umask to? We would have to add another 1 to it, because we still want it to remove group, so we have the 2, but we also want it to remove the execute permission as well. We add the value of execute to the number that is being used as the mask for others.

What we would do is go 'umask 003', we're adding a 1 to this last digit, which will remove both the write and the execute permission from others. Hit Enter. If we run the 'umask' command, we should see that it has a value of 003 now, whereas before it was 002.

Let's go ahead and test to make sure that this worked. Let's create another new directory. 'mkdir', and let's call this directory 'newdir', and let's create a new file, 'touch newfile'. Let's run the 'ls -l' command, and we see that the mode is now different.

For the new directory we just created, /newdir, owner still gets read, write, execute, just like they should; and group gets read-write execute, just like we said that they should. Notice what happened over here to others. They have read, but they do not have execute, which will prevent them from actually entering into that directory.

However, if we look at the mode for newfile, it's really not any different than it was before when we created mytestfile. That's because the execute permission was not assigned in the first place by the Linux kernel.

Subtract Permissions 7:09-7:48

Remember, the Linux kernel, by default, will assign a mode of 666 to files when they're created in the file system. That means they're given read-write, read-write, read-write. Execute was never even in the picture, so by subtracting it with umask, there was nothing to subtract. It didn't affect our file modes at all, but it did affect our directory modes.

Before we end, I need to point out that the value you assign to umask from the command line is not persistent, meaning that as long as this system stays running, this value we assigned will remain in effect, but as soon as we reboot the system, it'll jump back to its default. If you want to make that value persistent across reboots, you'll have to go into a shell configuration file and add the necessary command.

Summary 7:49-8:07

That's it for this demonstration. In this demo we talked about using umask. We first looked at the default modes that are assigned to files and folders when they're created in the file system. Then we discussed the role of umask in subtracting permissions from the default mode assigned to files and folders, and then we customized the value of umask to modify the default mode that's assigned to files and folders.
