2.1.3 Shell Commands

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

Shell Commands 0:00-0:58

In this lesson, we're going to talk about using shell commands. Understand that even though most modern Linux distributions include some type of graphical user interface, most, if not all, of the work you're going to do to manage a Linux System will be done from the shell prompt. For example, you may need to enter commands to run a program, or to find certain pieces of information, or to manage files, or to create file systems, or even create new partitions on a storage device. Therefore, you have to be familiar with how to enter commands at the shell prompt.

Running a program or command from the shell prompt is fairly easy. It's done in much the same way as you would on other operating systems, such as using a command window or PowerShell prompt in Windows. All you have to do is type the command or the script file name that you want to run or the program file name that you want to run in the shell prompt, and then you press Enter.

Enter Commands at the Shell Prompt 0:59-1:45

For example, let's suppose you want to display a listing of all the files and directories within the current directory, and you want that output displayed on the screen. To do this, you would type the 'ls' command right here and hit Enter at the shell prompt. When you do, a listing of all the files and folders is displayed.

As you can see, entering commands at the shell prompt is pretty straightforward. Understand that there are a couple of issues that you have to be aware of when you're entering commands at the shell prompt. The first one, and the one that drives a lot of new system administrators crazy, is the fact that the Linux shell handles the path to the executable that you want to run in a manner that is different from other operating systems. If you've come from a different operating system environment, like a Windows environment, this can be a problem.

PATH Environment Variable 1:46-2:56

Linux systems work in a different manner. Linux does also employ a PATH environment variable, just like a Windows system does. But here's the key thing that you have to remember: if you enter a command at the shell prompt, the Linux shell does not look in the current directory first. This is why this is so confusing to new Linux system administrators. You expect to be able to switch to the directory where the executable file resides and then run it from the command line. On Linux, this doesn't work. With Linux, the shell only searches for the file you're trying to run in the directories in your PATH environment variable.

As you can see here, the PATH environment variable contains a list of directories. Each directory is separated by a full colon (:) in the PATH. As you can see here, if you want to view the PATH environment variable, you can type 'echo $PATH' at the shell prompt. The thing that trips people up is the fact that even if the executable you're trying to run resides in the current directory, this shell won't be able to find it if that directory is not in your PATH environment variable. Instead, the shell will return an error.

Execute from the Current Directory 2:57-3:42

For example, let's suppose you have an executable file named zombie right here, in your user accounts home directory. We use the cd command to switch to our user's home directory. Then we enter —˜zombie' right here, at the shell prompt. Well, notice here, when we do that, it generates an error. This happens because the shell can't find this file, here, that we're trying to run.

Why is this? It's because this user's home directory is not included by default,within the PATH environment variable. This is the default configuration on most Linux distributions. It drives people crazy because they can see the file. It's right there. They try to run it, and the shell says, "Hey, I don't know where this is at." And you say, "I'm looking right at it. it's right there!" How do you deal with this?

Options for Running Executables 3:43-4:57

Well, there are three different options for running executable files. The first one is to enter the full path to the executable file. In the previous example, because it's in my home directory and my username was rtracy, I could have entered '/home/rtracy/zombie' at the shell prompt. By doing that, the shell would then know exactly where that executable file resided in the file system, and then it could run it, whereas before, it could not.

A second option is to switch to the directory where the executable resides and then add a dot slash (./) to the beginning of the command. The dot slash characters specify the current directory, whatever it happens to be. By adding these characters to the beginning of the command, you essentially tell the shell to look for the command that you're entering in the current directory. Basically, you're telling shell, "Hey, look in here. It's right here."

The final option is that you could add the directory where the executable resides to the list of directories in your PATH environment variable. There's really a fourth option that we don't have listed here, and that is to copy the executable file to a directory that already is listed in the PATH environment variable. That would work as well.

Case Sensitivity 4:58-5:53

In addition to path issues, you also need to be aware of the fact that Linux file names and directory names are case sensitive. That means Linux commands are also case sensitive. Therefore, if the executable file that you're going to run is named something like this zombie with all lowercase characters, like we just saw, then you must enter zombie with all lowercase characters at the shell prompt.

If you entered zombie with a capital 'Z', or wrote out zombie with all capital letters, or maybe used a capital 'B' and a capital 'E' in the command, it won't work. That's because the shell--because file names are case sensitive--will interpret each of those names as completely different files. Remember, this rule applies not just to file names, but to directory names as well. If you use the wrong case in a directory name in a path, it will point that shell to a completely different place in the file system.

Use exec 5:54-6:29

Now, you have another option for running commands from the shell prompt. In addition to running commands directly from the prompt itself, you can also use the exec command to run a program. The syntax for using exec is shown here. We enter 'exec' first, and then we enter the name of the executable that we want to run. For example, if I wanted to run the zombie file that we just looked at before, I would enter 'exec zombie'. Of course, I would have to provide the full path to zombie because it's in my home directory. We already talked about that. When you do, the zombie file would be executed.

Exec and Processes 6:30-7:43

This command isn't actually used very often because it's usually easier just to run the command right from the shell prompt, as we already talked about. However, there are instances when exec is more appropriate. That's because the exec command does have one useful feature. That is the fact that when you execute a command directly from the shell prompt without using exec, the new process created by the command runs alongside the shell process. If you were to use the —˜ps' command to view running processes, you'd actually see two different ones, one for the original shell where the command was run from, and a separate process for the command that's running within the shell.

That's not the case when you run a command with the exec command. If you execute a command using exec, the new process created by the command actually replaces the shell process from which it was launched. This has some significant impacts, key of which is when you exit out of the application, you basically terminate the shell itself because it assumed the shell's process.

For example, if I were to enter —˜exec zombie' from the shell prompt, and the zombie process would be loaded, and then I exited out of the program, the entire shell session would terminate because the shell's process was replaced by the zombie process.

Command Completion 7:44-8:30

When you're entering commands at the bash shell, there's a feature offered by the shell that you are going to grow to love, and it's called command completion. Some people call it tab complete--it's the same thing.

This feature is very helpful when you need to enter a very long file name, or very long directory name, or even a very long command name at the command prompt. Basically, what happens is you start typing a few characters of the command, or the file name, or the directory name in question, and then you press the Tab key. When you do this, the bash shell will guess what it is that you actually wanted to type and then automatically complete the command for you. This saves a lot of trouble, especially if you make a lot of typographical errors, like I do.

Tab Completion Example 8:31-9:47

For example, let's suppose that you have a file in your user's home directory named 'averylongfilenamefordemonstrationpurposes.txt'. That's a long file name. You really don't want to have to type that all out if you want to edit it. Let's suppose you do need to edit it for some reason.

You want to pull it up in your vi editor. Well, one option would be to type out the full command: —˜vi averylongfilenamefordemonstrationpurposes.txt' at the shell prompt. However, if you're like me, the chances of typing the file name wrong are very high. To prevent this, you can use command completion to take care of the typing for you.

In this example, we start off by typing 'vi', and then we enter the path of the file, which is in our home directory, so we enter '~/'. And then we type the first three, or four, or five letters of the file name right here; we type 'avery'. Then, instead of finishing out the entire file name, I just press the Tab key at that point. When I do this, the bash shell will look at all the files within the directory specified, and it will look for any of them that begin with A-V-E-R-Y. When it does this, it will determine that I probably wanted to open the file with the name of 'averylongfilenamefordemonstrationpurposes.txt'. So it will add the full file name right here, on to the end of the command. Then, all I have to do is press Enter.

Tab Completion Multiple Files 9:48-10:51

Command completion is great. It saves time, and it prevents mistakes. Be aware that there may be situations when there are multiple files within the path specified that match the pattern. What happens with Tab completion in the scenario? Let's take a look at the example here.

I'm using the same command that I did before. I'm running the vi command to load my text editor. I specify that I'm looking in my user's home directory. I type the first few characters of the file name, 'avery', and then I hit Tab. Well, in this scenario, I actually have two different files within that same directory that match this pattern. I have my text file that I actually want to edit, and I also have a backup file, that I made previously, that has a different extension, '.bak'. When I type this command, right here, the Tab complete feature actually doesn't know which of these two files I'm referring to. By default, after pressing Tab once, nothing is actually displayed on the screen.

Press Tab Twice 10:52-11:38

If you're trying to use Tab complete and this happens, you press Tab and nothing is displayed, it indicates to you that one of two different conditions exist: either no files in the directory specified match the pattern, or more than one file in that directory matches the pattern.

How do you know which one is true? Well, you hit Tab a second time. If there are no matching files in the path specified, nothing will still happen. But if there are more than one matches in the path specified, then the shell will bring up a list of files within that directory that match the file name that you started typing. That allows you to say, "Well, I want this one, and not that one." Before you can use Tab complete, though, you do have to type a few more characters to specify which in the two files you want, and then you can press Tab again.

Command History 11:39-13:06

The next feature of the shell that we want to talk about is command history. Understand that every time you enter a command at the shell prompt, that command is saved in a file in your home directory. The name of the file is '.bash_history'. This file is just a simple text file. It's a hidden file, and it contains all of your previously entered shell commands, one on each line. This file is continuously updated every single time you enter a shell command.

You can display the contents of the '.bash_history' file by entering the 'history' command at the shell prompt. When you do, a list of all your recent commands is displayed. This is really useful because if I need to run one of these commands again, all I have to do is press the Up arrow key at the shell prompt. When I do, the shell's going to read this file, and it will display the last command that I entered. If I press the Up arrow key again and again, it will scroll through this list of previously used commands. When I hit the one that I want, all I have to do is press Enter to execute it.

This is so useful, especially if you need to retype a very long, very complex command. Be aware that if you don't want to arrow through all of your past commands to find the one you want, you can also just enter part of the command that you need and press 'Ctrl+R'. When you do that, the bash shell is going to look through your command history and display the most recent command that matches the partial command that you started entering.

HISTSIZE 13:07-13:36

Your shell will use two different environment variables to manage the entries going into your history file. The first one is HISTSIZE. On some distributions, it's HISTFILESIZE. This environment variable configures the size of your history file. By default, on most distributions, this is set to 1,000 entries. You're not stuck with that. If you wanted to, you could customize the size of your history file by changing the value of this variable. You could set it to 2,000, or you could crank it down to 100, or whatever it is you need to do.

HISTCONTROL 13:37-14:29

The other environment variable you need to be aware of is HISTCONTROL. This environment variable controls how your command history is stored. There are a couple of different values that you can set this variable to. If you set this variable to a value of ignoredups, it tells the shell to ignore any duplicate commands within your command history. As you saw in the previous example, it had multiple instances of the su command.

I probably don't need five or six different instances of that command. I only need one. We can set it to ignoredups to eliminate any of the duplications. If you set HISTCONTROL to a value of ignorespace, it tells the shell to ignore commands in the history that start with a space. If you set it to a value of ignoreboth, it basically includes both ignorespace and ignoredups at the same time. The last option is to set it to erasedups, which will remove duplicate entries in the history file instead of just ignoring them.

Summary 14:30-14:41

That's it for this lesson. In this lesson, we discussed how to enter commands at the shell prompt. We talked about how commands are entered. We talked about the command completion feature of the shell. We talked about using command history. Then we looked at several commonly used shell commands.