14.1.1 Bash Scripting Overview

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

Scripting Overview 0:00-0:35

Let's spend some time talking about scripting. One of the cool things about Linux is the fact that it allows you to create your own powerful shell scripts that you can run right from the shell prompt. A shell script is nothing more than a text file that contains a series of commands that are executed by the shell in the order that they appear in the script.

You can use shell scripts to run one single command, or you can use them to run multiple commands all at once. They can also be used to read input from the end user or from other shell commands and then make decisions based on the input that it receives.

Sample Shell Script 0:36-0:48

A shell script, as we said, is a text file that contains a series of commands, and they're executed in the order that they appear in the file. A sample shell script is shown here. Notice that the script contains several key parts.

Specify Shell 0:49-1:20

First, we have a pound sign (#), an exclamation point (!), and then the path to the bash shell. It's very important that this be included within your shell script. The very first line of any shell script must specify which shell the script is written to run under. In this case, we're saying, "We're going to run this using the '/bin/bash' shell." That way, when the script is run, a new subshell will be created using the shell that we specify here, and then the script's contents will be processed within that sub-shell.

Script Comments 1:21-2:06

Next, we have a comment. Notice there's a pound sign here followed by some text. This part of the script is just a comment that describes what the script does. Any time you see a line of text within a script file that begins with a pound sign, it indicates that the text that comes after that is a comment. Because it's a comment, this part of the script is not displayed or processed in any way when the script is run.

You don't have to include comments in your scripts; they are optional. The script will run just fine without them. But it is considered good form to include comments where necessary. For example, it's a good idea to include a comment right at the beginning of your scripts--right after the shell declaration--that describes what the script does and, possibly, even who wrote the script.

Shell Commands 2:07-2:54

Next, we have two commands. We have the echo command. We have the date command. These elements in the script are just simple shell commands that we normally use at the shell prompt, and we're going to run them from within the script.

First, we're going to run the echo command. We're going to use echo to display this text on the screen. And then, after that command is done, we're going to run the date command. That will display the current date and time on the screen. Your shell scripts may or may not include the echo and date commands. You can put whatever commands you want inside of your shell script.

The last part of the shell script is ‘exit 0'. This part of the script indicates that we have reached the end. It tells the shell what to do after it's done running the commands in the script. In this case, it tells the shell to exit.

Run the Script 2:55-3:44

When we run the script, the output is sent to the standard out, which, by default, is the screen, as you can see here. Notice that the script first displayed the text that was indicated by the echo command. Then, when it was done, the next command was processed, which directed the shell to run the date command. The output of the date command is shown here.

When the script was done, the shell exited the script and returned us to the command prompt. Shell scripting is extremely flexible. We could add any other shell command we wanted to the shell script. You just have to remember to put each command on a separate line.

This script was a really simple script. It executed from beginning to end. It didn't include any branching. No decision-making within the script. And it also did not include any kind of end user interaction.

Capture User Input 3:45-4:28

This was a great start for a script. But understand that the bash shell actually allows you to create much more complex scripts than this. For example, we could make our scripts much more interactive by having the script ask the user a question and then capture their input for processing. This is typically done using the echo command we already looked at along with the read command.

The echo command that you see right here is used to present the user with a question. In this case, What directory do you want to add to the path?" The read command is used to pause the script, present a prompt on the screen, and then read in the information that the user enters into a variable that we define.

The read Command 4:29-5:36

In this script, the user is prompted for the name of a directory that he or she wants to add to the path environment variable. The read command, right here, provides the user with a prompt to enter in a directory name. Then, when the user presses the Enter key, whatever value he or she typed is assigned to the variable, here, named NEWPATH.

Once that variable is stored in memory, we use the echo command a second time to display the value that they entered. We say, ‘"You want to add"', and then we reference the NEWPATH variable.

Notice, in this example, that we're asking the end user to specify a directory path to be added to the path environment variable. We didn't actually do it in this script. All we did was ask them what path they want to add. We said, "Great. This is the path you want to add," and then we ended. We didn't actually modify the path environment variable.

If we want to do this, then we need to add some more commands to the shell script. The best approach to doing this when writing shell scripts is to first ask yourself, "If I were doing this from the shell prompt, what commands would I use?" Enter those commands in the script and try it out.

Apply User Input 5:37-6:23

In this example, we're going to ask the user the same questions and read the input into the same variable. This time, we're going to add the directory specified by the user to the path environment variable, and then we're going to export path.

Notice, here, we've added the variable assignment command to the script where we say "PATH=", and then we include whatever the current paths are in the path environment variable, and then append to the end of those paths a colon (:) followed by the path that the end user specified.

Then we export it. And then, so the end user can see what the path environment variable looks like, we use the echo command to display, "Your path environment variable is now:" followed by the current value of the path environment variable.

Dynamic Variable Declaration 6:24-7:07

If you've ever done any programming, you probably noticed that we didn't have to declare the NEWPATH variable anywhere in the script. With many scripting and programming languages, you do have to declare a variable before you can use it. You have to set its size and specify what type of information that that variable can contain. It could be a text string, a real number, an integer, a Boolean value, and so on.

The bash shell's a lot more forgiving. In this instance, the bash shell will create the new path variable for me in memory dynamically. Then, using the read command, it will assign whatever the end user inputs into new path.

Declare Variable Type 7:08-8:26

Of course, the bash shell does let you declare and type variables within a script if you want to. This is done using the declare command in the script. This could be useful in scripts where you want to have the end user enter a number with the read command instead of text.

The key issue here that you've got to remember when creating shell scripts is the fact that the bash shell interprets anything entered in at the read command by the end user as text, even if the user enters a number. However, if we declare the variables first and then type them as integers, something very different will happen. The variables will be interpreted as numbers on which you can perform arithmetic operations. You can't do that with text.

In the example that you see here, we've added declare statements to the beginning of a script to declare three different variables named NUM1, NUM2, and TOT. Because we used the -i option, we've specified that each of these variables will be integers, meaning that the bash shell will interpret the values assigned to these variables as whole numbers and not as text.

That means we can then manipulate the values assigned to these variables. We can add them together, we can perform subtraction on them, we can multiply them, we can divide them--whatever arithmetic operations we want.

Summary 8:27-8:41

That's it for this lesson. In this lesson, we introduced you to basic scripting concepts on a Linux system. We talked about the components that comprise the shell script. We then discussed how you can use the read command to read input from the end user in a script. And then we ended this lesson by reviewing how to declare and type variables within a script.
