2.7.1 Redirection

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

Redirection 0:00-0:21

Let's talk about using redirection at the shell prompt. Now, the Linux shell is extremely powerful and very, very flexible. One of the features that makes it this way is its ability to manipulate command input and command output. In this lesson, we're going to discuss how you redirect the output from a shell command.

Bash Shell File Descriptors 0:22-0:34

However, before you can learn how to do this, you first have to understand the concept of file descriptors. Understand that there are three file descriptors that are used for every single command that you enter at the shell prompt.

Standard Input (stdin) 0:35-1:04

The first one is called the standard in, or stdin. This file descriptor stands for standard input, which is the input that's sent to a particular command for it to process. The standard in for any command is represented by the number 0. Now, the standard in could come from a file. It could come from a text stream sent from another command. Or it could come from information that the user types on the keyboard.

Standard Output (stdout) 1:05-1:32

In addition to standard in, we also have standard out. This file descriptor stands for the standard output, which is, simply, the output from a particular command. Notice that the standard out is identified with the number 1. By default, the standard output is always written to the screen. For example, if I were to issue the pwd command at the shell prompt, then it would write the output from that command, which is just the current directory, to the screen.

Standard Error (stderr) 1:33-2:02

Finally, we have the standard error, which is assigned a number of 2. This is the output generated by a command if an error situation is encountered. For example, if I were to use the cd command to try to change to a directory in the file system that doesn't exist, instead of actually performing that function, the command would return an error, basically, saying, "I don't know what directory you're talking about. I can't find it." That would come out down here as the standard error.

Manipulate Input and Outputs 2:03-2:19

Now, the Bash shell on Linux allows you to manipulate both the input and the two outputs for any given command. Using these three descriptors, you can manipulate where a command gets its input from and where it sends its output to.

Redirect Output 2:20-4:14

Let's first look at redirecting the output from a command. The Bash shell allows you to manipulate where the output from a command goes after it's generated. By default, it gets just written on the screen. However, you can specify that it be sent elsewhere. For example, it's very common to redirect the output of a command from the screen to a text file in the file system, especially if the output from the command is really, really long or is something that we want to analyze. Maybe we're troubleshooting a problem in the system.

Now, redirection is accomplished using the > character on the command line. The syntax for redirecting output is to specify the command itself and then the output number. Remember, we said that the standard out is represented by the number 1, and then a > sign, and then we specify where we want to write that information to, such as a file in the file system. In this example, we want to use the tail command to view the last few lines of our system boot log file, which is /var/log/boot.log. We want to save the output in a file named lastboot in the current

directory. We enter tail /var/boot.log 1> lastboot. This tells the shell to redirect the standard out, number 1, to a file named lastboot. The output from the command is not sent to the screen at all. Instead, it is written to the file we specified here.

Now, before we go any further, I do need to point out that if you fail to enter a file descriptor right here, this number, then the shell is going to automatically assume that you want to redirect the standard out. In this example, I could actually remove the 1 altogether, and I would get the same result. I could just use the > sign pointing to the lastboot file.

Redirect Errors 4:15-5:23

Now, you can use the same technique to redirect the standard error from the screen to a file. Notice here, all we did was replace the 1 with a 2. That way, any error messages generated by the command will be redirected to this file, right here. In this case, we changed the name of the file to errmsg. Now, it's important to note that this command will only write data to this file if an error is generated. For example, if this file didn't exist. If this file does exist and tail is able to execute properly, then nothing will be written to the error message file because there was no error message generated. Because we're not redirecting the standard out, it will just be written on the screen. And as long as there's no error message, nothing will go in the errmsg file.

If, on the other hand, there was an error message generated--let's say boot.log doesn't exist for some reason--then, the resulting error message generated by the tail command gets written to a file in the file system named errmsg, and, of course, nothing happens for the standard out because tail couldn't find the file to generate the output from.

Redirect to a Nonexistent File 5:24-5:55

Be aware that if you redirect the output from a command to a file that doesn't exist, right here, that's not a problem, because the shell will automatically create it for you. Here's the thing you have to remember: if this file does already exist and we use one > sign, then the shell will actually erase the existing file and replace it with the new output by default. Now, there may be situations where this is fine; this is what you want it to do. But there may be other situations where this would not be a good thing.

Append Output to an Existing FIle 5:56-6:18

If you want to not erase the existing file, but to, instead, append the new output to the end of the existing file without erasing anything already there, you should use two > instead of one (>>). In this case, whatever was already in lastboot will be preserved, and then the output of the tail command will be added on to the very end of the lastboot file.

Redirect Output and Errors 6:19-7:33

Now, you can also redirect both the standard error and the standard out to text files at the same time. There are two different ways to do this. One way is to add two redirection instructions to the same command. That's what's shown in this first line. The first one is for the standard out, and the second one is for the standard error. We enter the name of the command, in this case, the tail /var/log/boot.log, and we specify that the standard out go to the lastboot file. And if there's a problem somewhere and tail isn't able to run properly, then we specify that any error messages generated go to the errmsg file.

Remember that, because of the way redirection works, only one of these files will be affected by this command, depending upon whether the command was successful or not. If the command was successful, then this file gets written to. If the command was not successful and an error was generated, then the first file will not be written to, but the second file will be written to. If the first file is written to, that means the command was successful. Therefore, no error messages were written, so nothing will be written to the error message file. It's one or the other, not both.

Redirect Output and Errors to the Same File 7:34-8:36

Now, there's a second way to do this, and it's shown in the second example on this screen. We're basically doing the same thing as we did in the first example. We're redirecting both the standard out and the standard error in the same command, but what we're doing is redirecting both outputs to the same file. Notice that we're writing the standard out to a file named lastboot. But then, instead of specifying a different file name over here for the standard error, we put an &, and then the number 1, which says, basically, "Take whatever error messages generated and write it to the same file that we're writing the standard error to."

If you're going to use this option, it's very important to remember to use that & before the number 1. This tells the shell that the character that follows is a file descriptor and not a file name. If I were to omit the & and just put a 1 there, the shell would think that I wanted to write the standard error output to a separate file named 1.

Redirect Input 8:37-9:40

Now that you understand how to redirect outputs from a command, we next need to discuss how to redirect command inputs. Now, just as you can specify where the output from a command is sent, you can also specify where a command's input comes from. To do this, all you have to do is reverse the character we used previously for redirecting the output. When we're working with input, we use a < sig, instead of a > sign. The syntax is shown here. We enter the command, then we enter a < sign, and then we specify where we're getting the input from. In this case, it is coming from a file named employees. This option is useful in situations where you need to send a lot of text to a command. In this example, I could already have a very large text file that has the list of all the names of the employees in my organization in it. I can send it to the input of the sort command. Then the sort command does its thing, and the sorted output is displayed on the screen.

Summary 9:41-9:51

That's it for this lesson. In this lesson, we discussed redirection at the shell prompt. We first reviewed the three Bash shell file descriptors, and then we discussed how you can use the > and the < characters to redirect both the input and the output for a command.
