14.2.3 Bash Shell Variables and Parameters

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

Bash Shell Variables and Parameters 0:00-0:40

In this demonstration, we will show the Bash shell variables and parameters.

The first thing we need to do is open the terminal and by doing so will have access to produce information that we can see. The first thing we do, is we run the 'ps' command which shows us all the processes that are running. In this case, you can see that I'm running a Bash shell. That's what this terminal is. The process ID is 21836. To show other information, like when we run a script, we're running an additional Bash shell with an additional process ID.

Process IDs and Scripts 0:41-1:32

So I wrote a script. Let me show to you here. It's called 'show-IDs', and you can see it's very simple. I want to see what my script process ID is, and that's done with the special variable of dollar sign, dollar sign ($$). I'm also showing the parent process ID, which is the built-in variable dollar sign PPID ($PPID).

Let's go ahead and run that, show-IDs, and you'll see that indeed the two processes are different. The process ID for the script is a little higher than that of the parent process. I'll go ahead and run it again and again, just to show you that indeed, that number does change. The script process ID is different than the parent, and that goes to show that when a script is run a different Bash shell is executed.

Scripts and Environment Variables 1:33-2:19

Another thing that we need to take into account are the environmental variables. If I look at the environmental variables here, these are variables that are set by our system. You can see here that my login name is 'TestUser', my home directory is '/home/TestUser', and there's lots of other information here, as well. Now, this too, can change between what we see interactively. When we run the script, it's noninteractive. The parameters may change based on Linux processes that are running between a parent and the child. The system itself may not run certain jobs or certain parameters within the child process.

Special Parameters 2:20-3:40

I also want to show how different special parameters work within scripts. Scripts themselves are, as i said, very powerful. There a very powerful part of Linux. That's where we really gain the power within Linux. I've written a couple of scripts. Let me to show you what I have here. You see here, I have 'showargs' and I have star-at_diff'. Here's that show IDs that I did, but let's go ahead and do the 'showargs' scripts. Let's take a look at that, so I'll go ahead and show you the 'showargs' script.

It's very, very straightforward. I'm going to be using some special variables. You can see here, I have a dollar sign star ($*). That shows me all the arguments that were placed on the command line when I started the script. Dollar sign pound ($#) shows me the number of arguments that were used when I started the script. The show program name--so whatever I use to execute the program, will be showed with a dollar sign zero ($0). Each parameter after that, dollar sign one ($1), dollar sign two ($2), and so on, will show the arguments that I'm specifying. '$1' will show me the first argument. '$2' will show me the second. '$3' will show me the third, and so forth.

First Look at $* and $@ 3:41-4:48

Then we have this other one. It's a dollar sign at ($@), and that's going to show me all the arguments starting with the first. When you look here at the dollar sign star ($*) and the dollar sign at ($@), initially they look the same. But I'm can show you another script where it actually shows you what the differences between them. Let's go ahead and run this, 'showargs.sh', and I'm just going to add a few parameters: 'one', 'two', 'three', and 'four'. Nothing special.

Press enter, and it's going to basically go through and show me all of those parameters and what they do.

We see all of my arguments, 'one', 'two', 'three' and 'four'. The number of arguments is 4. The program, the executable name is './showargs.sh'. The first argument is 'one'. The second argument is 'two', and display all argument, starting with the first is 'one two three four'. Again, initially the '$*' produces, for the most part, the same output as '$@'. There is a difference. I'll demonstrate what those differences are as well.

Second Look at $* and $@ 4:49-6:12

Remember that because I have this number of arguments, I can create a loop like a for loop that will show me all of the arguments that I have, in a format that I can use. That's what my next script shows, as well. So remember--the things to remember are this. One is a '$*' shows me all the arguments. $# or number sign, shows me the number of arguments, and then '$1', '$2', those show me the positional argument. The first argument will be $1, second argument will be $2, etc.

Now let me show you the script that really delineates and shows a difference between the '$*' and the '$@'. I'll go ahead and show you that one now. Again I'm showing all parameters, '$*', and the number of parameters. Then I'm doing two loops here. The first thing that I'm going to do is I'm going to use the '$*', and I'm going to go ahead and create this script that displays all of the parameters. Then, I'm going to do the same thing but in this case, I'm using the '$@'. But, you can see that the loops are pretty identical.

How Backslash is Interpreted 6:13-7:36

Now, if you're confused by the backslashes (\) here. It's actually pretty straightforward. This '\n' here means new line. In other words, just give me a blank space. This '\' here tells the script interpreter that the next character should not be interpreted, but displayed. That's all that that means. Same thing here. This '\' here tells this quote sign (') just be printed not interpreted. The same thing here for the '\$'. Okay. Again pretty straightforward. A little bit different.

But, let's go ahead and run the script. So 'start-at_diff. Then I'll add the exact same parameters, 'one', 'two', 'three', and 'four'. So you see, basically, there's no difference. Right? Because here's the '$*'. There are my parameters. And here's the '$@' and my parameters. Now, here is where the difference comes in. If I change that command just a little bit. I'll go ahead and backspace here, and now I'll combine 'three' and 'four' together. So I now have 'one', 'two', and then another parameter, 'three four'. Watch what happens.

$* and $@ Results 7:37-8:21

As expected, it shows all the parameters the same way. So there's really no way to see the difference there. But look here. Now instead of four parameters, I only have three. When I look at the '$*', I have 'one', 'two', 'three', 'four'--nothing changes. Here's where the change takes place. When I use the '$@', it sees the three parameters, 'one', 'two', and 'three four'. So the way the '$*' interprets in the way the '$@ interprets is indeed, different. The '$@' uses an array format to print everything out and that's not true of the '$*'

Summary 8:22-8:31

In this demonstration, I've shown the differences between the script process IDs. We've shown variables, and we've shown the difference in how you can use special variables in scripting.
