14.2.4 User Variables and Shell Arithmetic

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

User Variables and Shell Arithmetic 0:00-0:20

In this demonstation we look at bash variables that we define at the shell prompt and variables inside of a script. We sometimes call these local variables or user variables. We'll pay particular attention to integer variables and how you can use shell arithmetic to manipulate them.

Variable Basics 0:21-1:05

Let's start with some basics. Remember that variables are like a container with a name. Here at our bash shell prompt, we can use the ‘declare' command like this to define a container and give it the name ‘myvar'. We reference the variable using the dollar sign ($) expansion. Let's do a simple echo command. This container, or variable, is empty.

Notice that we named ‘myvar' using lowercase characters. Bash lets us name them using any combination of upper and lowercase. While serious programmers standardize the way they name variables, the only suggestion we might want to consider is that when we name them in something other than full uppercase, we're less likely to overwrite any of our environment variables.

Assign a Value 1:06-2:10

We don't have to formally declare every variable we want to use. Variables are automatically declared when they are assigned a value. To assign a value we use the equal sign (=). The name of the variable is on the left and the value is on the right.

Notice that we can't put spaces before or after the equal sign (=). If we do, bash thinks the variable is a command. If our value needs to have spaces in it, we enclose it using quotes. We use single quotes when we want the value to be the literal characters between the quotes. We use double quotes if we want to use special character like the dollar sign ($) and backslash (\) to retain their special meaning.

Bash variables are untyped, meaning bash doesn't store characters, numbers, strings or other values differently. In fact, bash always stores data as character strings. Even though we assign this variable ‘x' a number, it stores it as the character ‘1' followed by a period (.) and the characters ‘2', and ‘3'.

Shell Arithmetic 2:10-3:03

But what if we need to do arithmetic? Bash will do that if we declare a variable using the ‘-i' option. Bash denotes variables that are declared this way as integer variables. What that really means is that when we assign it a value using the equal sign (=), the part on the right is considered an arithmetic equation, rather than a character string. We call this shell arithmetic. And there are a lot of rules.

Let's try some shell arithmetic with our ‘x' variable that currently is not an integer variable. When we assign it 4+5, we get the literal characters ‘4', the plus sign (+) and ‘5'.

Let's start over by deleting ‘x' using the unset command. Now, let's declare x as an integer variable. And we can give ‘x' an assignment in the same command. Notice that now the value of ‘x' is nine (9).

Integer vs. Non-Integer Variables 3:04-3:36

Shell arithmetic is only performed when the variable on the left of the equal (=) sign is an integer variable. Here we might expect that ‘y' would be assign the value of 10, because x is 9, and 9 plus 1 is 10. But y is not an integer variable, so we just get the literal characters.

Let's make ‘y' an integer and try our equation again. Now, ‘y' is assigned the value 10 because ‘y' is an integer variable and shell arithmetic was used.

Shell Arithmetic Tips 3:37-4:19

Here are some tips about shell arithmetic. First, Integer values can be negative. Next, we can form equations using star (*) for multiply, and forward slash (/) for divide. There are other arithmetic operators you can use. We can find them in the bash man page or other references.

We can also use parentheses '()' to force the order of the arithmetic. Also, notice that the dollar sign ($) isn't required to expand the variables in an equation. It will still work, but it isn't needed in shell arithmetic.

Integer Arithmetic 4:20-5:30

One important thing we need to cover is that shell arithmetic is integer arithmetic. We can't assign decimal values to integers. This is particularly noticeable when we divide. Any remainder from a division operation is ignored. Sometimes we refer to this as the value was truncated. In this example, 'x+2' would be 11, and 11 divided by 3 is 3 with a remainder of 2. So 'y' is assigned the value of 3 and the remainder of 2 is discarded.

Another important thing to understand is that shell arithmetic will substitute a zero (0) in an equation, when the value is non-numeric characters. You can add "Dave" to 4, but shell arithmetic substitutes a zero (0) for "Dave", giving 4 as the result.

We can use the 'declare -p' command to see both the value of a variable and whether it has been declared as an integer. In the case of our 'y' variable, the 'i' shows that it's an integer. Also notice the double quotes (") around the value of 4. This helps us remember that the value is stored as the character "4", and not the number 4.

Local Variables 5:31-6:29

Before we finish, we need to emphasize that all of these variables are considered local variables. The same thing applies to variables within a script. If we exit this terminal window and re-open it, our variables are gone.

Let's look at this script. It simply assigns a value to a variable named myvar and then echos the value. Let's set a variable named myvar before we run the script and echo it to the screen.

Now let's run the script and re-echo the value of myvar. We notice that even though myvar was assigned a different value inside the script, it didn't affect the variable in the terminal window. We remember that user variables in a parent shell aren't inherited by a child shell. Essentially, there are two myvar variables, one in our shell that is our terminal window, and the other inside the child shell that ran the script.

Summary 6:30-6:46

That's it for this demonstration. We showed how local or user variables are defined and how the 'declare -i' command creates an integer variable. We also showed how shell arithmetic is used to assign a value to integer variables. We finished by showing that user variables are local to the shell where they are created.
