14.3.2 Looping

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

Looping 0:00-0:32

In this demonstration, we will show looping. Looping is mostly done inside of scripts. However, for this demonstration, I'm going to show it to you interactively, so that I can make changes quickly, and you can see the results.

Looping is used quite a bit in programming so that you can extrapolate information from a file and, you can add a sequence of numbers. There's many, many uses for looping.

The first loop will uses the 'for' loop.

The for Loop 0:32-1:43

The 'for' loop is probably one of the more common loops that's used. The first thing were going to do is to do just a plain easy loop. We will specify the 'for' command. We'll specify a variable name. Then we'll use the keyword 'in'. 'in' is describing what is going to be coming next. 'i' will take the place of everything that is in the list were about to give it. We can just use the numbers 1, 2, 3, 4, 5, making it fairly simple. 'i' will take on the values of 1, then 2, then 3, then 4, then 5.

As in most scripting and most commands we're going to use a semicolon to stop that part of the command, and then we use the keyword 'do'. What this is telling us is we are executing a command called 'for'. We are specifying the variable. Telling it what to do and then we're stopping the command. Now, we need the rest of the construct for the 'for' loop, then that next command is 'do'.

Looped Commands 1:44-2:20

Notice we have a continuation prompt. What this tells us is that the command is not complete. We need to finish the command. I'll just do a simple echo of the variable as it goes through the changes. We're still not complete. To finish our loop, we match the 'do' with the word 'done'. That tells us that were finished with our loop.

Pressing enter, we should see of numbers 1, then 2, then 3, then 4, then 5, which is exactly what we see. It worked just as we thought it should.

Interation Options 2:21-3:39

There's different iterations of this looping structure, different ways to describe and provide the variable 'i' with different information. So here's another loop. We can do 'for i in', and and then can specify a set. We'll choose the number 1, dot, dot, 5 (1..5). So what that means is the number 1 through the number 5 inclusive. We use the same construct. We add the 'do', 'echo $i', 'done'. Then we see the exact same output.

This is a little bit different because one of the parameters wasn't added here. We can actually add an increment. I'm going to change that a little bit. 'for i in' the set 1, dot, dot, 5 (1..5), which means inclusive. But this time, rather than an increment of 1 were can use an increment of 2. We'll go ahead and loop through and see it only shows 1, 3, and 5 because we're starting at 1, we're going to 5, but we're skipping every other number. In this case, we get all odd numbers.

The C Contruct 3:40-5:07

The last construct that we can do is a very 'C' like construct was originally in the C programming language and Linux is very much written in C, so it just makes sense. In this case, we use an expansion construct. Again, we use 'for', and this time we use a double open parethesis '(('. That tells us were going to be using expansion. At that point, we again use our variable. We'll use 'i' again, But, this time we have to define what the starting number will be. We'll use 1 again. We'll use a semicolon (;) as a separator.

Now, we run a test, and the test is, "At what point do we stop the loop." We're going to say, "When 'i' is either less than or equal to the number 10." We'll change that up a little bit, then a semicolon (;). Now, what we do is we specify the increment. In this case, what were going to do is were going to use the incrementer, 'i++'. Again, this is a C construct meaning increment 'i' buy 1. We'll close the parentheses '))', specify the semicolon (;), and the word 'do'. Then, the same thing, 'echo $i', and then 'done'.

This time, we should see the numbers 1 through 10, which is exactly what we see.

The seq Command 5:08-6:06

There is an older way of doing things. It's really not used anymore. But just for the sake of things that you might see in some older scripts, we're going to use the sequence command. It's the same idea as we have before, but this time were using an external program to specify our numbers and our increment. So, again, 'for i in', and this time we're going to use another expansion. We're going to use the keyword, 'seq'. That's an external program with the sequencer. We're going to go ahead and specify a start number, an increment, and an end number. Then a semicolon (;), 'do'. We're going to 'echo $i', and 'done'. You see that again, we're starting at 1, we're incrementing by 2 until we get to the number 20.

The while Loop 6:05-8:06

The 'while' loop is a little different than the 'for' loop. It's used for different purposes. It's used for traversing a directory and looking for files and while a file exists, it continues on to the next file. It might be used for user validation, or user input validation. While the input is wrong, continue to loop through and continue to ask the user to enter information.

The construct is similar. But in the 'for' loop, we set the beginning value of the variable and we test against the ending value of the variable. Then we specify an increment. In the 'while' loop, we really have no way to set the variable so we have to do that ahead of time. In this case we're going to go ahead and set the variable 'i' to the value of 1. Now we can do our loop.

Start with the keyword 'while', and now we need to do our test. What we want to do is say, "Is the value of the variable 'i' less than or equal to a number." And in this case, we'll use 10. Then, we will bracket our test. Again, we add a semicolon (;) to tell the system were done. The keyword 'do' is the same in 'while' as it is in 'for'. We'll go ahead and display the value of the variable 'i' as we did before. But, this time we have no way to increment so we have to actually do that inside of our while loop. We'll go ahead and enter the expansion double open parenthesis '((', variable name, and plus, plus. What were doing is we're incrementing the value of the variable 'i'. At that point, were done. We go ahead and issue the keyword 'done'.

Again, you'll see that indeed, the 'while' loop did cycle through.

The until Loop 8:07-9:55

While has a brother called 'until'. 'until' is just the opposite of 'while'. We want to use 'until' when we know that we're going to go ahead and increment at least once. Were going to go ahead and execute at least once. But, we need to do our test after we go through the process at least once. It's very, very similar in the way we do things. But, where while, we're waiting for something to happen and were saying, "'i' is less than 10." What were looking for now is the result.

Let's do the same thing. We'll do and 'i' equals 1. The same as we did before. But now, we're going to say 'until' the value of 'i' is equal to 10. We'll go ahead and put our semicolon (;) in again. We'll do a 'do'. Then, everything else is the same. We'll still echo the value of 'i', and will increment 'i', and then were 'done'.

You'll see that the results are the same. But actually, they aren't, are they? I use the keyword equal 10. Well, as soon as this became true, it didn't loop anymore. So it looped through 1, 2, 3, all the way to 9. It echoed 9, then it incremented i to 10, and then looped again. It went until the value of 'i' equals 10, 'do'--well it equals 10 now, so it goes ahead and exits. I would have needed to put in is greater than or equal to 10.

Summary 9:56-9:58

In this demonstration, we demonstrated the 'for' loop, the 'while' loop, and the 'until' loop.