OPS235

Bash Shell Scripting - Part 2

Bash Shell Scripting Essentials / Continued
Creating a VM Backup Script

OPS235

Bash Shell Scripting - Part 2

Bash Shell Scripting Essentials / Continued
Creating a VM Backup Script

Additional Bash Shell Essentials

In order to write a useful program (Bash shell script), it should contain several key elements:

- Variables (environment, user-defined, positional parameters)
- Logic (if, if-else, if-elif-else statements)
- Mathematical Operations
- Loops

We have discussed the first two elements in lab1.

In lab2, we will focus on mathematical operations and loops.
At the end of lab2, you will either create or download scripts (using these scripting tools and some using the virsh command) to backup and manipulate your VMs.

Mathematical Operations

Your Bash shell script may be required to perform mathematical operations such as adding, subtracting, multiplying and dividing numbers (most likely stored in variables).

In shell scripting, variables store all data as text (i.e. not numbers). This makes it easier for shell scripting since you don't have to declare the data type of a variable (i.e. integer or a floating point number) as in the C, C++, or Java programming languages. Unfortunately, as a result, you can't simply use math symbols directly in shell scripting.

Example:

```
num1=4; num2=5;
echo $num1 + $num2
```

Output:

```
4 + 5
```

Mathematical Operations

Therefore, you need to have the shell convert the numbers (stored as text) into binary numbers to be used for mathematical calculations.

In Bash shell scripting, you use the syntax $(( )) to perform math operations.

Example:

```
num1=4; num2=5;
echo $(($num1 + $num2))
```

Output:
9

Comments:

When using $(( )) you do NOT have to use the $ inside that math expression to expand the variables. The following work work: echo $((num1 +num2))

Mathematical Operators

There are various mathematical operators that can be used with the $(( )) math expression.

+ addition
- subtraction
* multiplication
** exponentiation (eg: echo $((2**2)) would display 4)
/ division
% modulus (remainder from division)

Note: $(( )) does not handle floating point decimals.
You would need to use other commands for that such as awk or bc.

Mathematical Operators

Here is an "age-old" programming trick to determine if an integer that a user entered is either an odd or even number:

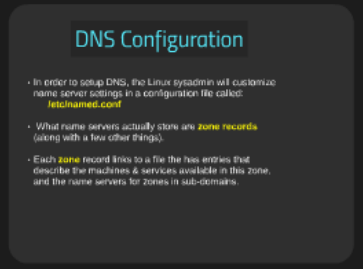
```
read -p "please enter an integer: " myInteger
if [ $((myInteger % 2)) -ne 0 ]
then
  echo "$myInteger is odd"
else
  echo "$myInteger is even"
fi
```

Output (assume user enters the integer 3):

```
please enter an integer: 3
3 is odd
```

Comment:

Module (%) indicates there is a remainder of 1 because 2 does not go into the number 3 evenly when divided. Therefore the result is not equal to zero (which make the condition true) and prints that the number is odd.



Control Flow Statements

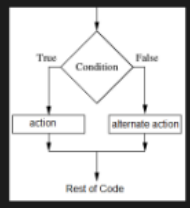
Control Flow Statements are used in shell scripting to make the shell script perform differently based on the value of variables.

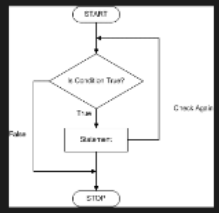
In lab1 notes, we looked at logic control flow statements such as if, if-else, if-elif-else

We will now look at using loops to have sections of the shell script repeat commands based on test conditions.

Using variables, and control flow statements, such as logic and loops are extremely useful for creating powerful shell scripts.

Images:
http://csharp.net-informations.com/statements/img/if_else_csharp.png
http://www.functionx.com/flowcharts/while1.gif





Control Flow Statements (loops)

Wikipedia defines a conditional loop as:

*"... a way for computer programs to **repeat** one or more various steps depending on **conditions** set either by the programmer initially or real-time by the actual program."*

A technical term used to represent looping in programs is called iteration.

https://en.wikipedia.org/wiki/Conditional_loop

Control Flow Statements (loops)

There are different types of loops can be used in programming and shell scripting:

Determinant loops (such as for loops) usually repeat for a preset or "known" number of times

In-determinant loops (such as while or until loops) repeat based on unknown conditions (like waiting for user to enter correct data).

Determinant Loops (for loop)

Here are some examples using for loops:

FOR LOOP (with arguments)

```
for x in I like ops235
do
 echo "The argument is: $x"
done
```

Output:

```
The argument is: I
The argument is: like
The argument is: ops235
```

Comment:

Each argument is stored as the variable x and repeated in command until all arguments were used

FOR LOOP (positional parameters)

```
set ops235 is fun
for x
do
   echo "argument is: $x"
done
```

Output:

```
argument is: ops235
argument is: is
 argument is: fun
```

Comment:

Each positional parameter (eg. $1, $2, etc) is stored as the variable x and the command is repeated until all were used.

Determinant Loops (for loop)

You can also use a for loop in a more traditional method like used with the C, C++, or Java pgramming languages. To make the for loop work this way, it required mathematical operations which we already discussed.

FOR LOOP (traditional method using mathematical operations)

```
for ((x = 1; x <= 3; x++))
do
  echo "The number is: $x"
done
```

Output:

```
The number is 1
The number is 2
The number is 3
```

Comment:

The variable x is initialized with a value of 1. The command will repeat as long as the value is equal or less then 3. X++ is a shortcut for x=x+1 which means to advance the value of x by 1 at the end of each loop. Note (( )) allows for spaces between = and you can use <, >, =>, =< symbols!

Indeterminant Loops (while, until)

In-determinant loops repeat based on unknown conditions. Unlike determinant loops, you may not have a pre-determined (known) number of times it will loop. An example would be for error checking to force the user to keep entering data until it is correct. You can use pipeline commands (using grep or egrep) for this purpose. The until statement repeats "**until** test condition is true". The while statement repeats only "**while** test condition is true"

Example:

```
read -p "enter a whole number: " num
until echo $num | grep -q "^[0-9][0-9]*$"
do
 read -p "Incorrect. Please enter WHOLE NUMBER: " num
done
```

Output:

```
enter a whole number: x
Incorrect. Please enter WHOLE NUMBER: 2X9
Incorrect. Please enter WHOLE NUMBER: 43
```

Indeterminant Loops (while, until)

The conditional statement && runs next command if the previous command or test is true.
The Conditional statement || runs next command if the previous command or test is false.

Example:

```
read -p "pick a number between 1 and 10: " num
while [ $num -lt 1 ] || [ $num -gt 10 ]
do
 read -p "Incorrect. Please pick number between 1 and 10: " num
done
```

Output:

```
pick a number between 1 and 10: 14
Incorrect. Please pick number between 1 and 10: -2
Incorrect. Please pick number between 1 and 10: 6
```