MODULE 6 :
Shell Scripting

Shell Scripting in Unix

A shell is a program constructed of shell commands (`$shell`, `$path`, `ls`, `pwd`, `mkdir...`)

- Shell is an environment for user interaction, but it is not a part of kernel.
- Shell is just like as BAT files in MS-DOS.
- By default, Bash shell is default shell for Linux.

Shell Scripting in Unix(contd.)

To find your default shell in Unix OS, type the following in your terminal

```
 echo $SHELL
/bin/bash
```

To check what are the different types of shells supported, `cat` the contents of `/etc/shells` file.

Structure of a Script

`#!/bin/bash`	– It defines which shell will be used to run the script.
`#comments`	– Comments can be made by using # symbol in a script.
`chmod +x script.sh`	– To tell the linux that file is executable.
`./script.sh`	– To execute the script.

Variables in Shell Scripts

In shell scripts you can use variables, they are just *symbolic name to a chunk of memory to which we assign values*.

The variable assignment can be done with the following syntax

```
VARIABLE=value
```

Notes:

- There should be **no space** around the "`=`" sign.
- If there is a space, `VARIABLE = value` won't work.
- Also Variables are **case-sensitive**.

Consider the example below:

Variables in Shell Scripts(contd.)

Consider the example below:

```
#!/bin/sh

MY_MESSAGE="Hello World"

echo $MY_MESSAGE
```

Shell BUILTINS Commands

`export` : The export command marks an environment variable to be exported.

`read` : Use to get input (data from user) from keyboard and store (data) to variable.

`exit` : The exit command lets you quit the shell where it's run.

Other shell Builtins are…

```
alias, bg, break, cd, logout, popd, printf, pushd,
pwd, read, ulimit, umask, help, history, set, help,
history, if, jobs, kill, fg, exec, exit, continue,
declare, shopt, source, wait, test, while and many
more….
```

Shell Arithmetic

Syntax:

- `expr op1 math-operator op2`

Examples:

- $ expr 1 + 3
- $ expr 2 – 1
- $ expr 10 / 2
- $ expr 20 % 3
- $ expr 10 * 3
- $ echo `expr 6 + 3`

Shell Metacharacters

Shell Metacharacters with files listing examples:

`$ ls *`	will show all files
`$ ls a*`	will show all files whose first name is starting with letter 'a'
`$ ls *.c`	will show all files having extension .c
`$ ls ut*.c`	will show files having extension .c but file name must begin with 'ut'
`$ ls ?`	will show all files whose names are 1 character long
`$ ls fo?`	shows files names which are 3 character long and begin with ‘fo’
`$ ls [abc]*`	will show all files beginning with letters a,b,c

Relational Operators in Shell

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

Arithmetic Operators in Shell

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	\`expr $a + $b\` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	\`expr $a - $b\` will give -10
* (Multiplication)	Multiplies values on either side of the operator	\`expr $a \\* $b\` will give 200
/ (Division)	Divides left hand operand by right hand operand	\`expr $b / $a\` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	\`expr $b % $a\` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

Decision Making Constructs in Shell

Unix Shell supports conditional statements which are used to perform different actions based on different conditions. We will now understand two decision-making statements here -

- The if...else statement
- The case...esac statement

Decision Making Constructs in Shell (contd.)

The if...else statements

If else statements are useful decision-making statements which can be used to select an option from a given set of options. Unix Shell supports following forms of if...else statement -

- if...fi statement
- if...else...fi statement
- if...elif...else...fi statement

Decision Making Constructs in Shell (contd.)

if...else...fi example

```
$ vim myscript.sh

#!/bin/bash

 read choice

if [ $choice -gt 0 ]; then
    echo "$choice number is positive"
else
    echo "$ choice number is negative"
fi
```

Nested if...else...fi example

```
$ vi nestedif.sh

#!/bin/bash

echo "1. Unix (Sun Os)"
echo "2. Linux (Red Hat)"
echo -n "Select your os choice [1 or 2]? "
read osch

    if [ $osch -eq 1 ] ; then
         echo "You Pick up Unix (Sun Os)"
    else
           if [ $osch -eq 2 ] ; then
                 echo "You Pick up Linux (Red Hat)"
           else
                 echo "What you don't like Unix/Linux OS."
           fi
    fi

```

Decision Making Constructs in Shell (contd.)

The case...esac Statement

You can use multiple **if...elif** statements to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Unix Shell supports case...esac statement which handles exactly this situation, and it does so more efficiently than repeated **if...elif** statements.

There is only one form of case...esac statement which has been described in detail here -

- case...esac statement

Decision Making Constructs in Shell (contd.)

case....esac example

```
$ vi case-esac.sh

#!/bin/bash

    echo "Enter Number in between 1 to 4"
    read var

     case $var in
       1) echo “One”;;
       2) echo “Two”;;
       3) echo “Three”;;
       4) echo “Four”;;
       *) echo "Sorry, it is bigger than Four";;
    esac
```

Loop Constructs in Shell

Bash supports the following loop constructs -

- for loop
- while loop

Note that in each and every loop:

- First, the variable used in loop condition must be initialized, then execution of the loop begins.
- A test (condition) is made at the beginning of each iteration.
- The body of loop ends with a statement that modifies the value of the test (condition) variable.

Other Commands

`sleep` : It is used to delay for a fixed amount of time during the execution of any script. When the coder needs to pause the execution of any command for the particular purpose then this command is used with the particular time value. You can set the delay amount by **seconds** (s), **minutes** (m), **hours** (h) and **days** (d).

`script` : Make a typescript of a terminal session.

`eval` : execute arguments as a shell command.

`exec` : command in Linux is used to execute a command from the bash itself.

Thank You!