Shell scripting

BY

Stephen Kanyerezi

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS

Shell scripting

What is a shell script

What is a shell

- Commandline interpreter

Examples of shells in Unix/Linux

- Bourne Again Shell (bash)
- Bourne shell
- C shell
- Korn shell

How do I know which shell I am using

- echo $SHELL

It is good practice to start with a shebang (#!) while writing your scripts

Whereas the system interpretes lines preceded with a # as comments, it is not the case for a shebang. The shebang is followed by the path of the interpreter to be used

Since a script is just a collection of command instructions read line by line, let us try out a simple script with the commands we have learnt so far

Variables

- A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data

Variable Types

- Local variables
- Environment variables
- Shell Variables

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS

Variable names

The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_)

- Given the above statement, can you give examples of valid and invalid names

Take note that since environment variables are in **UPPER CASE,** make sure not to have your newly created variables in the same case. Can you give a reason why this is so?

There are those characters that have a special meaning to the shell. It is not good to use them in variable names. If you insist, then go on and quote them. Some of these may include *, !, -, $, etc.

We have learnt what variables are and how to name them, but how do we create our own

Use the syntax below
<variable_name>=<value of the variable>

Let us create our now;

I have create a variable, but how do i reference or call it out and use it downstream

Special variables

$0 - The filename of the current script.

$n - These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on)

$# - The number of arguments supplied to a script

$? - The exit status of the last command executed

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS

Operators

Arithmetic operators: +, -, *, /, %,=,==,!=

Relational operators (numericals): -eq, -ne, -gt, -lt, -le, -ge

Relational operators (strings): =, !=, -z, -n

File test operators: -e, -s, -d, -f, -w, -x, -r

Decision making

- If else statements
- Case esac statements

If else structure

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
 fi
```

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
else
   Statement(s) to be executed if expression is not true
 fi
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
  fi
```

Case esac

```
case word in
   pattern1)
      Statement(s) to be executed if pattern1 matches
      ;;
   pattern2)
      Statement(s) to be executed if pattern2 matches
      ;;
   pattern3)
      Statement(s) to be executed if pattern3 matches
      ;;
   *)
     Default condition to be executed
     ;;
  esac
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS

Loops

Loops are a programming element that repeat a portion of code a set number of times until the desired process is complete

- While loop
- Until loop
- For loop

Control of loops

- Break
- Continue

Substitution
The shell performs substitution when it encounters an expression that contains one or more special characters

In command substitution, he shell performs a given set of commands and then substitutes their output in the place of the commands

- Backticks ``
- $()

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS

Shell quoting

This is a way of dealing with characters that have special meaning to the shell, i.e, make them lose this meaning and be interpreted as literal values. Unix Shell provides various metacharacters which have special meaning while using them in any Shell Script and causes termination of a word unless quoted

There are different quoting mechanisms which include;

- Single quote: All special characters between these quotes lose their special meaning
- Double quote: Most special characters between these quotes lose their special meaning with these exceptions - $, `, \$, \’, \”,\\
- Backslash: Any character immediately following the backslash loses its special meaning
- Back quote: Anything in between back quotes would be treated as a command and would be executed