Bash Script

Contents

[1. Overview 1](#_Toc427244851)

[2. Formatting 1](#_Toc427244852)

[3. Variable ( a temporary store for a piece of information) 1](#_Toc427244853)

[4. Input 2](#_Toc427244854)

[5. Arithmetic 2](#_Toc427244855)

[6. If Statements 4](#_Toc427244856)

[7. Loops 5](#_Toc427244857)

# 1. Overview

* A Bash script is a plain text file which contains a series of commands.
  + Anything you can run normally on the command line can be put into a script.
  + Anything you can put into a script can be run normally on the command line.
* It is convention to give files that are Bash scripts an extension of **.sh** (myscript.sh for example) Linux is an extensionless system so a script doesn’t have to have this characteristic in order to work.

# 2. Formatting

* Shebang Line (first line of script): #!/bin/bash
* Be mindful of the presence or absence of spaces when looking at code

# 3. Variable ( a temporary store for a piece of information)

* 2 actions to perform for variables
  + **Setting** a value for a variable
    - myvariable=Hello
    - There is no space on either side of the equals(=) sign!
    - we leave out the $ sign.
  + Reading the value for a variable
    - When referring to or reading a variable we place a $ sign before the variable name.
* **Other Special Variables**
  + **$0** - The name of the Bash script.
  + **$1 - $9** - The first 9 arguments to the Bash script. (As mentioned above.)
  + **$#** - How many arguments were passed to the Bash script.
  + **$@** - All the arguments supplied to the Bash script.
  + **$?** - The exit status of the most recently run process.
  + **$$** - The process ID of the current script.
  + **$USER** - The username of the user running the script.
  + **$HOSTNAME** - The hostname of the machine the script is running on.
  + **$SECONDS** - The number of seconds since the script was started.
  + **$RANDOM** - Returns a different random number each time is it referred to.
  + **$LINENO** - Returns the current line number in the Bash script.

If you type the command **env** on the command line you will see a listing of other variables which you may also refer to.

# 4. Input

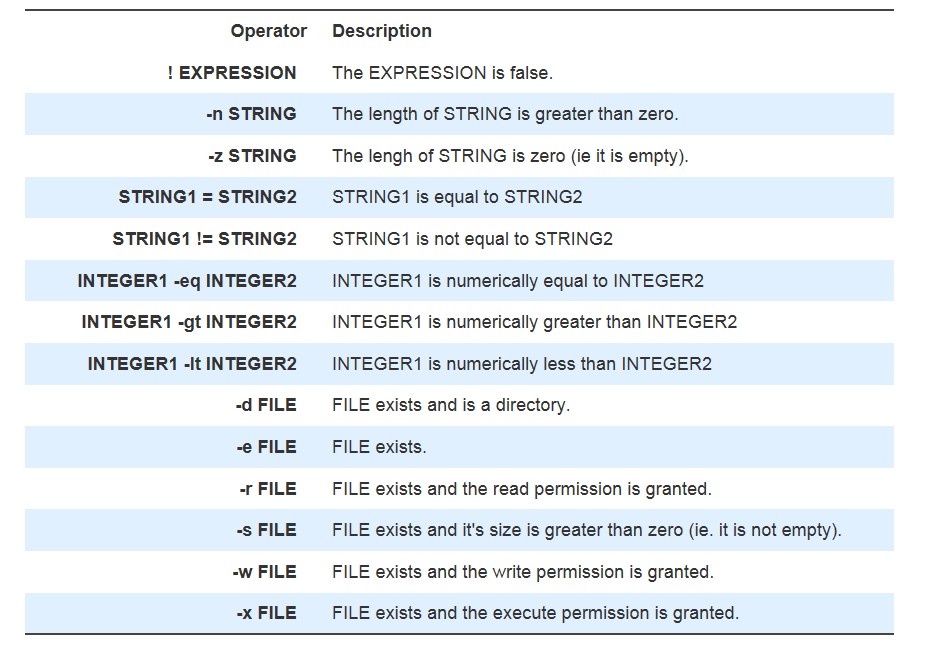
* 3 methods for getting input from the user:
  + Command line arguments
  + Read input during script execution
    - use command **read**, this command takes the input and will save it into a variable
      * read var1
      * The general mechanism is that you can supply several variable names to **read**. Read will then take your input and split it on whitespace.
  + Accept data that has been redirected into the Bash script via STDIN

# 5. Arithmetic

* **Let** *(make a variable equal to an expression)*
  + let a=5+4
    - This is the basic format. Note that if we don't put quotes around the expression then it must be written with no spaces.
  + let “a = 5 + 4”
    - This time we have used quotes which allow us to space out the expression to make it more readable.
  + let a++
  + let “a = 4 \* 5”
  + let “a = $1 + 30 “
* **expr** *(print out the result of the expression)*
  + expr 5 + 4
    - This is the basic format. Note that there must be spaces between the items and no quotes.
  + expr "5 + 4"
    - If we do put quotes around the expression then the expression will not be evaluated but printed instead.
  + expr 5+4
    - If we do not put spaces between the items of the expression then the expression will not be evaluated but printed instead.
  + expr 5 \\* $1
    - Some characters have a special meaning to Bash so we must escape them (put a backslash in front of) to remove their special meaning.
  + expr 11 % 2
  + a=$( expr 10 - 3 )
    - This time we're using expr within command substitution in order to save the result to the variable **a**.
  + echo $a # 7
* **$(( expression))** *(return the result of the expression, double parentheses is the preferred method.)* 
  + a=$(( 4 + 5 ))
    - This is the basic format. As you can see we may space it out nicely for readability without the need for quotes.
  + a=$((3+5))
    - As you can see, it works just the same if we take spacing out.
  + b=$(( a + 3 ))
    - We may include variables without the preceding $ sign.
  + b=$(( $a + 4 ))
    - Variables can be included with the $ sign if you prefer.
  + (( b++ ))
  + (( b += 3 ))
  + a=$(( 4 \* 5 ))
    - Unlike other methods, when we do multiplication we don't need to escape the **\*** sign (to \\* )
* **${#var}** *(return the length of the variable var)*
  + a='Hello World'| echo ${#a} *# 11*

# 6. If Statements

* **basic if statement**
  + if [ <some test> ]  
    then  
     <commands>  
    elif [ <some test> ]   
    then  
     <different commands>  
    else  
     <other commands>  
    fi
* **Test**
  + The square brackets ( [ ] ) in the **if** statement above are actually a reference to the command **test**.
* Boolean Operations
  + Multiple conditions require boolean operators:
    - **and** - &&
    - **or** - ||



* + When we refer to FILE above we are actually meaning a [path](http://ryanstutorials.net/linuxtutorial/navigation.php). Remember that a path may be absolute or relative and may refer to a file or a directory.
  + Case Statement
    - case <variable> in  
      <pattern 1>)  
       <commands>  
       ;;  
      <pattern 2>)  
       <other commands>  
       ;;  
      esac

# 7. Loops

* **While-do-done Loops**

while [ <some test> ]  
do  
 <commands>  
done

* **Until-do-done Loops**

until [ <some test> ]  
do  
 <commands>  
done

* **for Loop**

for var in <list>  
do  
 <commands>  
done

* **Break** statement
  + The **break** statement tells Bash to leave the loop straight away.
* **Continue** statement
  + The **continue** statement tells Bash to stop running through this iteration of the loop and begin the next iteration.
* **Select do done**
  + select var in <list>  
    do  
     <commands>  
    done
  + Display a simple menu system for selecting items from a list.

# 8. Functions

- basic format:

function\_name () {  
 <commands>  
}

* The function definition ( the actual function itself) must appear in the script before any calls to the function.
* Passing Arguments
  + We supply the arguments directly after the function name. Within the function they are accessible as **$1, $2, etc**.
* Return Values
  + We use the keyword **return** to indicate a return status.
* **local <name>=<value>**

Create a local variable within a function