**BASH SCRIPTING**

**Introduction**

In the terminal, use: *whereis bash*

This will tell you the location of bash which you will use at the top of the file:

Test.sh

#! /bin/bash

# This is a comment, the below will output ‘hello world’ in the terminal

Echo “hello world”

Name=”Ali Issaee” # notice no spaces

Echo $name # This can also be within double quotes

**Read user input**

Test.sh

Echo “name: “

Read name # this will prompt the user to enter their name and will store the result in the ‘name’ variable

Echo “Your name is $name”

Use the -p flag for the user to write on the same line: *read -p name*

Use the -ps flag for the user not to see the input, similar to password

You do not have to set the *read* keyword to a variable, instead, you can just use the built in variable $REPLY to get the users response, e.g.:

Echo “name: “

Read

Echo “Your name is $REPLY”

Echo “Your friends names are: “

Read -a names # -a flag is for array

Echo “your friends names are ${names[@]} # [index], [@] prints all in array

**Passing arguments to a script**

Test.sh

Echo $1 $2 $3 ‘> $1 $2 $3’

# Or you can store in an array:

Args=(“$@”)

Echo args@

Echo args# # This displays the count (total elements in an array)

Terminal

./test.sh Ali Reza Issaee

Output

Ali Reza Issaee > $1 $2 $3

**If statement**

Test.sh

Count=10

If [ $count -eq 10 ] # notice spaces

Then

# code

Fi # end if statement

|  |  |
| --- | --- |
| **Comparison operators** | **Explanation** |
| == OR -eq | Equal to |
| != OR -ne | Not equal to |
| > OR -gt | Greater than |
| >= OR -ge | Greater than or equal to |
| < OR -lt | Lesser than |
| <= OR -le | Lesser than or equal to |

|  |  |
| --- | --- |
| **String operators** | **Explanation** |
| == | Equals |
| != | Not equal |
| < | Less alphabetically |
| > | More alphabetically |
| -z | Is null |

When you use < <= > >= with integers, wrap the *if* condition like so: (( #condition ))

With strings, use: [[ #condition ]]

**If else**

Test.sh

If [[ *condition* ]]

Then

#statement

Else

#statement

Fi

Test.sh (example 2)

If (( #condition ))

Then

#statement

Elif [ #condition ]

Then

#statement

Else

#statement

Fi

**File test operators**

Echo -e “Enter the name of the file: \c”

The -e flag is used in conjunction with \c (at the end of the string) in order for the user to enter the file name on the same line.

If [ -e $filename ]

Then

Echo “$filename file exists”

Else

Echo “the file, $filename, does not exist”

Fi

The -e flag checks whether a file exists

The -d flag checks whether a directory exists

**How to append output to the end of a file**

Test.sh

If [ -w $filename ] # -w = file write permissions

Then

Echo “Type some text to append to the file”

Cat >> $filename

Else

Echo “You do not have write permissions”

Fi

// cat is the editor to use to append output to the file

// the ‘>>’ is the operators which appends text to the end of the tile

// ‘>’ will overwrite the file with the users input

**Logical AND & OR**

If [ #condition1 ] && [ #condition2]

OR if [ #condition1 -a #condition2] # -a alternative AND operator

The OR (||) operator is used in the same way, instead of the -a flag, use the -o flag if using the second variant.

**Arithmetic**

Test.sh

Echo $(( 1+1 ))

#alternative

Num1=2

Num2=5

Echo $(expr $num1 \\* $num2) # *expr* is keyword. You only need to escape (\) the multiplication operator

\* Note – use ‘bc’ command in terminal to access basic calculator – you can also use it in your script \*

Echo “20.5 \* 3” | bc

If you use this way, and you are using division, use:

Echo “scale=2; 20.5/5” | bc # where scale=2 is the no. of decimal places

Without *scale=2*, the result will be an integer, i.e. 0 decimal places

**The case statement**

The case statement is similar to the *switch* statement

Test.sh

$vehicle=car

Case $vehicle in

“car”)

Echo “car selected”;; #2 semicolons

“van”)

Echo “van selected”;;

)

Echo “unknown”

Esac # *case* backwards to end statement

**Arrays**

Bash supports single dimensional arrays

Names=(“Ali”, “Lauren”)

Echo ${names[@]} #[] = index, @ returns all elements in array

Echo “${!names[@]}” #using ! – will return the indexes (not values)

Echo “${#names[@]}” # to get the total number in array, use #

To add more arrays, use: names[2]=”Sha”

To remove element from array: unset names[1]

**While loop**

N=1

While [ $n -le 10 ]

Do

Echo $n

N=$(( n+1)) # can also use: (( ++n ))

Done

**Sleep command**

Use the keyword *sleep*, followed by the number of seconds you want the bash script to sleep for, i.e. pause the script the moment it reaches the sleep command: *sleep 2*

**Read files**

While read p

Do

Echo $p

Done < hello.sh # file to read

# Or you can use the below:

Cat hello.sh | while read p

Do

Echo $p

Done

**Until loop**

The *until* loop is similar to the *while* loop, but it uses the *until* keyword instead of *while.* The loop will continue **until** the condition is met:

While [ $n -ge 10 ] # this will stop at 10

Until [ $n -ge 10 ] # this will stop until it reaches 10, i.e. will stop at 9

**For loop**

For i in {1..10} # loop 1-10 times. Use {1...2..10} to increment by 2

Do

Echo $i

Done

# Or you can use the below:

For (( i=0; i<5; i++ ))

Do

Echo $i

Done

**Using for loop to execute commands**

e.g.1

For command ls pwd date # where ls, pwd and data are the commands to run in order

Do

Echo “….. $command is about to be executed”

$command # execute command

Done

E.g.2

For item in \* # where \* means everything

Do

If [ -d $item ] # if item is a directory

Then

Echo $item

Fi

Done

The above example prints out all the directories

**Select loop**

The select loop is good for dealing with menus

Select name in mark john tom

Do

Echo “$name selected”

Done

The output in the terminal will look like:

1. Mark
2. John
3. Tom

// At this point user inputs either 1,2 or 3. i.e. if user types 3

Tom selected

You can also use cases within this loop to output different things dependent on the users selected input.

**Break & continue**

Use the keyword *break* to exit the loop completely

Use the keyword *continue* to move on to the next iteration

**Functions**

To create a function:

Function hello() {

Echo “Hello World”

}

# Or…

Hello() {

Echo “Hello World”

}

In order to invoke a function, use: *Hello*

To pass arguments:

Function hello() {

Echo $1 $2

}

Hello hello world #output will be hello world (the first ‘hello’ is the function call)

**Local variables**

Global and local variables work similar to JavaScript but with one difference:

Function print() {

Name=”Ali”

Echo $name

}

Name=”lauren”

Print #invoke function

Echo $name #output will be Ali

The reason for the above is that the function with the same *name* variable as the global variable *name,* overwrites the global variable. In order to avoid this happening, in the function you can declare your variables as so: *local name=”Ali”*

The *local* keyword before the variable declaration will ensure that this variable is only used within the lexical/function scope.

**Function examples**

Useage() {

Echo “you need to provide a filename”

}

Function is\_file\_exists() {

Local file=$1 #argument

[[ -f $file ]] && return 0 || return 1 # see below for explanation \*\*

}

\*\* This is not a ternary operator! This has 3 conditions, the first: [[ -f $file ]] which checks to see if a file exists and will result in 1 or 0 (i.e. truthy or falsey). If truthy, then the code will ignore the condition following the && and go straight to the code after the || sign. Therefore if the file exists, 1 will be returned (i.e. true).

[[ $# -eq 0 ]] && useage # if number of arguments passed === 0, run the *useage* function

If ( is\_file\_exists “$1” )

Then

Echo “File found”

Else

Echo “File not found”

Fi

**Readonly command**

These are similar to constants:

Name=”Ali”

Readonly name

The name variable can no longer be mutated.

When creating a readonly function, use the -f flag:

Hello() {}

Readonly -f hello

**Debugging**

To debug a bash script, before running the script, use: *bash -x*

e.g. bash -x ./scriptname.sh

This will give you more information if errors occur

Another method is to do the following:

Test.sh

Set -x # start debugging here

# some code here

Set +x # to turn debugging off

# more code which you don’t want to debug

**Test**

Part 1

1. How do you locate where bash is stored on PC
2. Create a basic ‘hello world’ script with comments
3. Run the script and change to executable if required
4. Read user input, store in a variable and echo it out
5. What built in variable can you use instead of storing user input in a variable
6. How can you hide input from user input (like password)
7. Pass arguments to a script and echo it/them out

Part 2

1. Create if else if else statement
2. List the comparison operators and use in if statement
3. What are the different ways to wrap conditions in an if statement
4. Check if file exists
5. Check if a directory exists
6. Get user to append text to the end of a file
7. Get user to overwrite an existing file with their input
8. Use logical && and || in if statement

Part 3

1. Perform basic arithmetic operations
2. Give an example of the case statement
3. Create an array and output all elements
4. Create more elements in an array after declaration
5. Count the number of elements in an array
6. Get the indexes from an array element

Part 4

1. Give an example of a:
   1. While loop
   2. For loop
   3. Until loop
   4. Select loop
2. Give example of the *sleep* command
3. What do *break* and *continue* do
4. Use the for loop to execute commands (e.g. ls pwd…)

Part 5

1. Create and invoke a function with and without arguments
2. How can you make variables in functions strictly local
3. Create a function which checks if a file exists. If no arguments are passed to the script, alert the user.
4. Explain: *[[ -f $filename ]] && return 0 || return 1*
5. How can you create a constant
6. List the 2 different ways to debug a bash script