Open Source Community

Bash Scripting

Agenda;

-What is Bash Scripting

-Running a Bash Script

-Variables

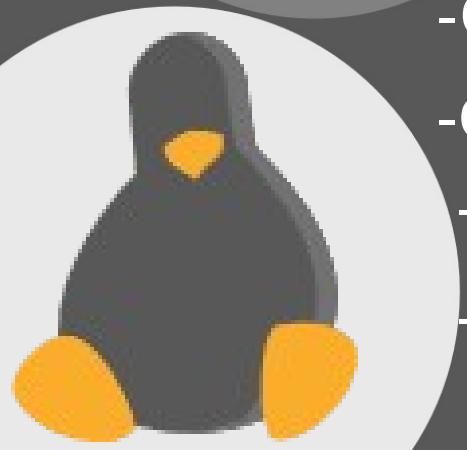
-Input and output in Bash

-Operations in Bash

-Conditionals in Bash

-Loops in Bash

-Functions



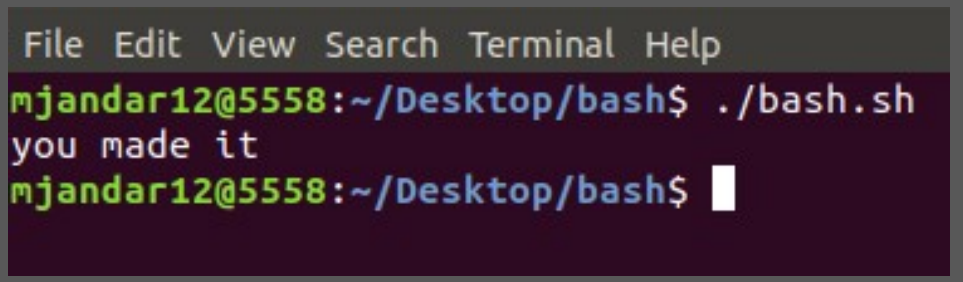
So what is a Bash script?

A Bash script is a plain text file which contains a series of commands, These commands are mixture of commands we would normally type ourselves on the command line
(such as **ls** or **cp** for example)

```
1   #!/bin/bash
2   #INPUT_SAMPLE_LIST=$1
3   cd /Volumes/PhilDrive_EMS/TestDec7/snv_postprocess/
4   ...
11  . paths.txt
12  ...
30
31  echo "Debug level set for $DEBUG_LEVEL"
32  echo "log found in scripts directory"
33  ...
50  cp $HIGH_SNP_OUT ./
51  cp $LOW_SNP_OUT ./
52  cp $GERM_SNP_OUT ./
53  #  echo "${SCRIPT_DIR}/run_somatic_mutation_analysis ${i} no_false_snp"
54  if [ $DEBUG_LEVEL -gt 0 ]
55  then
56  echo "INFO: ${SCRIPT_DIR}run_somatic_mutation_analysis.sh $SAMPLE no_false_snp
57  `basename ${LOW_SNP_OUT}` `basename ${GERM_SNP_OUT}` `basename ${HIGH_SNP_OUT}`
58  ${D_BAM_FILE} ${G_BAM_FILE}\n">${LOG}
59
60  fi
61  ${SCRIPT_DIR}run_somatic_mutation_analysis.sh
62
63  echo "End of somatic mutation analysis">> $LOG
```

Running a Bash file

You simply add **./** , sh or bash before the file name



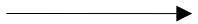
note that you need to make the bash file executable to run it, you can do that with chmod +x bashfile.sh

Variables

```
File  Edit  View  Search  Terminal  Help
#!/bin/bash

var='hallo world'

echo $var
```



```
File  Edit  View  Search  Terminal  Help
mjandar12@5558:~/Desktop/bash$ ./bash.sh
hallo world
mjandar12@5558:~/Desktop/bash$
```

- ‘ ’ Single quotes returns the value between them without processing
  echo ‘ $var ‘ prints $var
- “” Double quotes returns the value between them with processing
  echo “ $var “ prints the value of var
- The $ is used to get the value of variable

Input and output in Bash

- Input
- Input is referred to as *STDIN*.
- You can pass arguments to a bash script as input.
- To prompt input to the user, you use the command: ***read*** which is roughly equivalent to scanf() in C or input() in Python.
- Output
- Output is referred to as *STDOUT*.
- To print output to the user, you use the command: ***echo*** which is roughly equivalent to printf() in C or print() in Python.

Operations in Bash

- You can do 6 basic arithmetic operators in Bash:
  - a + b addition (a plus b)
  - a - b subtracting (a minus b)
  - a * b multiplication (a times b)
  - a / b integer division (a divided by b)
  - a % b modules (the integer remainder of a divided by b)
  - a ** b exponentiation (a to the power of b)
- Arithmetic can be done using the expression: $((expression))
  - Example: a=$((5 - 3 + $b))
  - Which means: variable a is equal to the value of $() the expression (5 - 3 + $b)

Conditionals in BASH

- Start a condition with if [[condition]]
- The next line contains then which is roughly equivalent to ‘{‘
- Write the commands that will execute if the condition is true.
- End your condition with fi which is roughly equivalent to ‘}’
  - Or start an elif [[condition]], with then in the line after it.
    - Write the commands that will execute if the elif condition is true.
    - End your conditionals with fi
  - Or start an else, with **NO** then in the line after it.
    - Write the commands that will execute if the else condition is true.
    - End your conditionals with fi

Conditional Operators

String Operators

Operator	Description
-z string	True if the length of string is zero
-n string	True if the length of string is non-zero
string1 == string2 or string1 = string2	True if the strings are equal; a single **=** should be used with the test command for POSIX conformance. When used with the **[[** command, this performs pattern matching as described above (compound commands).
string1 != string2	True if the strings are not equal
string1 < string2	True if string1 sorts before string2 lexicographically (refers to locale-specific sorting sequences for all alphanumeric and special characters)
string1 > string2	True if string1 sorts after string2 lexicographically

File Operators

Operator	Description
-a filename	True if the file exists; it can be empty or have some content but, so long as it exists, this will be true
-b filename	True if the file exists and is a block special file such as a hard drive like **/dev/sda** or **/dev/sda1**
-c filename	True if the file exists and is a character special file such as a TTY device like **/dev/TTY1**
-d filename	True if the file exists and is a directory
-e filename	True if the file exists; this is the same as **-a** above

Conditional Operators

Numeric comparison operators

Operator	Description
arg1 -eq arg2	True if arg1 equals arg2
arg1 -ne arg2	True if arg1 is not equal to arg2
arg1 -lt arg2	True if arg1 is less than arg2
arg1 -le arg2	True if arg1 is less than or equal to arg2
arg1 -gt arg2	True if arg1 is greater than arg2
arg1 -ge arg2	True if arg1 is greater than or equal to arg2

Refer to this link for more

Loops in Bash

Just like any programming language bash script has loops (**for**, **while** and **until**)

For loops

```
#!/bin/bash

for i in `seq 1 $1`
do
        echo $i
done
```

seq 1 $1 means sequence from 1 to the value of the first argument, ‘$1’ can be replaced with any other value.

While loops

```
#!/bin/bash

x=1
while [[ $x -lt 11 ]]
do
    echo $x
    let x+=1
done
```

You can you use *break* and *continue* to stop and continue the loop

How does it spreads?

-It can be infected by contact with an infected person

-it can be infected by touching surfaces that are not disinfected

Functions

- General function syntax :

```
#!/bin/bash

function NAME #Function Definition
{

    #DoThings
}
Name #Function call
```

Alternatively :

```
#!/bin/bash

NAME() #Function Definition
{
    #DoThings
}
Name #Function call
```

Passing arguments:

To use the arguments as variables, you can access their values by using $X where X is the order of the argument passed to the function.

```
#!/bin/bash

function add
{
    echo $(($1 + $2))
}

add 3 5
```

Thank you