OS Lab
Session 6: Bash Script

AUT – CEIT

Instructor: @MerajNouredini

What are scripts?

Scripts

- Think of a script for a play, or a movie, or a TV show. The script tells the actors what they should say and do.
- A script for a computer tells the computer what it should do or say. In the context of Bash scripts we are telling the Bash shell what it should do.

Bash Script

- Anything you can run normally on the command line can be put into a script and it will do exactly the same thing. Similarly, anything you can put into a script can also be run normally on the command line and it will do exactly the same thing.

How do they work?

- When we are at the terminal we have a Bash process running in order to give us the Bash shell. If we start a script running it doesn't actually run in that process but instead starts a new process to run inside

A Simple Bash Script file

```
1. #!/bin/bash
2. # A sample Bash script, by Ryan
3.
4. echo Hello World!
```

A Simple Bash Script file

- The Shebang (#!)
  - #!/bin/bash
- Comment (#)
- Command (echo)
- Script Formatting (spaces, indents)
  - we'll point these out as we encounter them

Variables

- Variables are one of those things that are actually quite easy to use but are also quite easy to get yourself into trouble with if you don't properly understand how they work.
- When referring to or reading a variable we place a $ sign before the variable name.
- When setting a variable we leave out the $ sign.
- Some people like to always write variable names in uppercase so they stand out. It's your preference however. They can be all uppercase, all lowercase, or a mixture.

Variables

```
1. #!/bin/bash
2. # A simple variable example
3.
4. myvariable=Hello
5.
6. anothervar=Fred
7.
8. echo $myvariable $anothervar
```

Command Line Arguments

```
1.  #!/bin/bash
2.  # A simple copy script
3.
4.  cp $1 $2
5.
6.  # Let's verify the copy worked
7.
8.  echo Details for $2
9.  ls -lh $2
```

Other Special Vars

- $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.

Quotes

- When we enclose our content in quotes we are indicating to Bash that the contents should be considered as a single item. You may use single quotes (') or double quotes (").

Command Substitution

- Command substitution allows us to take the output of a command or program (what would normally be printed to the screen) and save it as the value of a variable. To do this we place it within brackets, preceded by a $ sign.
  - myvar=$(ls /etc | wc -l)

Exporting Variables

- Remember how in the previous section we talked about scripts being run in their own process? This introduces a phenomenon known as scope which affects variables amongst other things. The idea is that variables are limited to the process they were created in. Normaly this isn't an issue but sometimes, for instance, a script may run another script as one of its commands. If we want the variable to be available to the second script then we need to export the variable.

Input

- Ask the User for Input
  - read var1
  - read -p 'Username: ' uservar
  - read -sp 'Password: ' passvar
- Remember pipes?
  - STDIN - /dev/stdin or /proc/self/fd/0
  - STDOUT - /dev/stdout or /proc/self/fd/1
  - STDERR - /dev/stderr or /proc/self/fd/2

Arithmetic

Operator	Operation
+, -, /*, /	addition, subtraction, multiply, divide
var++	Increase the variable var by 1
var--	Decrease the variable var by 1
%	Modulus (Return the remainder after division)

Arithmetic - let

```
1.  #!/bin/bash
2.  # Basic arithmetic using let
3.
4.  let a=5+4
5.  echo $a # 9
6.
7.  let "a = 5 + 4"
8.  echo $a # 9
9.
10. let a++
11. echo $a # 10
12.
13. let "a = 4 * 5"
14. echo $a # 20
15.
16. let "a = $1 + 30"
17. echo $a # 30 + first command line argument
```

Arithmetic - expr

- **expr** is similar to let except instead of saving the result to a variable it instead prints the answer.
  - expr item1 operator item2

Arithmetic - Double Parentheses

```
1.  #!/bin/bash
2.  # Basic arithmetic using double parentheses
3.
4.  a=$(( 4 + 5 ))
5.  echo $a # 9
6.
7.  a=$((3+5))
8.  echo $a # 8
9.
10. b=$(( a + 3 ))
11. echo $b # 11
12.
13. b=$(( $a + 4 ))
14. echo $b # 12
```

Length of Var

- ${#variable}

```
1.  #!/bin/bash
2.  # Show the length of a variable.
3.
4.  a='Hello World'
5.  echo ${#a} # 11
```

If Statements

- if [<some test>]

  then

  <commands>

  fi

```
1.  #!/bin/bash
2.  # Basic if statement
3.
4.  if [ $1 -gt 100 ]
5.  then
6.     echo Hey that\'s a large number.
7.     pwd
8.  fi
9.
10. date
```

If Statements

Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The lengh of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2

If Statements

-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and it's size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

If Statements

```
1.  #!/bin/bash
2.  # elif statements
3.
4.  if [ $1 -ge 18 ]
5.  then
6.    echo You may go to the party.
7.  elif [ $2 == 'yes' ]
8.  then
9.    echo You may go to the party but be back before midnight.
10. else
11.   echo You may not go to the party.
12. fi
```

If Statements

```
1.  #!/bin/bash
2.  # and example
3.
4.  if [ -r $1 ] && [ -s $1 ]
5.  then
6.     echo This file is useful.
7.  fi
```

```
1.  #!/bin/bash
2.  # or example
3.
4.  if [ $USER == 'bob' ] || [ $USER == 'andy' ]
5.  then
6.     ls -alh
7.  else
8.     ls
9.  fi
```

Case Statements

```
1.  #!/bin/bash
2.  # case example
3.
4.  case $1 in
5.     start)
6.        echo starting
7.        ;;
8.     stop)
9.        echo stoping
10.       ;;
11.    restart)
12.       echo restarting
13.       ;;
14.    *)
15.       echo don\'t know
16.       ;;
17. esac
```

Loops

```
1.  #!/bin/bash
2.  # Basic while loop
3.
4.  counter=1
5.  while [ $counter -le 10 ]
6.  do
7.     echo $counter
8.     ((counter++))
9.  done
10.
11. echo All done
```

```
1.  #!/bin/bash
2.  # Basic until loop
3.
4.  counter=1
5.  until [ $counter -gt 10 ]
6.  do
7.     echo $counter
8.     ((counter++))
9.  done
10.
11. echo All done
```

Loops

```
1.  #!/bin/bash
2.  # Basic for loop
3.
4.  names='Stan Kyle Cartman'
5.
6.  for name in $names
7.  do
8.     echo $name
9.  done
10.
11. echo All done
```

```
1.  #!/bin/bash
2.  # Basic range in for loop
3.
4.  for value in {1..5}
5.  do
6.     echo $value
7.  done
8.
9.  echo All done
```

select

```
1.  #!/bin/bash
2.  # A simple menu system
3.
4.  names='Kyle Cartman Stan Quit'
5.
6.  PS3='Select character: '
7.
8.  select name in $names
9.  do
10.    if [ $name == 'Quit' ]
11.    then
12.       break
13.    fi
14.    echo Hello $name
15. done
16.
17. echo Bye
```

Functions

```
1.  #!/bin/bash
2.  # Setting a return status for a function
3.
4.  print_something () {
5.     echo Hello $1
6.     return 5
7.  }
8.
9.  print_something Mars
10. print_something Jupiter
11. echo The previous function has a return value of $?
```

Variable Scope

- Scope refers to which parts of a script can see which variables. By default a variable is global. This means that it is visible everywhere in the script.
- We may also create a variable as a local variable.
- When we create a local variable within a function, it is only visible within that function.
- **Let’s see a code!**

Overriding Commands

```
1. #!/bin/bash
2. # Create a wrapper around the command ls
3.
4. ls () {
5.    command ls -lh
6. }
7.
8. ls
```

Class Activity

- Create a simple script which will print the numbers 1 - 10 (each on a separate line) and whether they are even or odd.

Reference

- https://ryanstutorials.net/bash-scripting-tutorial/

Questions?