BASH FUNDAMENTALS

...

TLO KNOWLEDGE AND SKILLS

Conditions:

- Given a classroom, applicable references, and a practical exercise, the Cyber Mission Force student will demonstrate an understanding of Bash fundamentals.

Knowledge:

- Identify the fundamental programming language of Bash.
- Understand the structure of how bash scripting is written.

Skills:

- Working knowledge of basic bash commands

OBJECTIVES

- Understand fundamental programming concepts.
- Describe scripting structures
- Identify and demonstrate Bash built-in commands and variables

WHAT IS BASH?

Going back to what we learned in Linux, BASH is a type of shell, similar to the C Shell, Bash Shell, Korn Shell, etc.

A shell is a way to communicate between the user and the computer to execute commands and carry out tasks.

To determine the shell that you are in you can use the environmental variable $SHELL.

WHAT IS BASH?

```
echo $SHELL

#This command will provide the shell type along with location, in our case /bin/bash
```

If you are not already in BASH you should change to the shell.

```
bash

#This will drop you into a bash shell
```

WHAT IS A BASH SCRIPT?

A BASH script is an executable file that allows you to run a set of commands to accomplish a task in an efficient way.

In the BASH shell it is best practice to name your scripts with the .sh extension to indicate that your file is a script.

In BASH we start all our scripts with the #! Shebang and then the location of your shell.

#! Shabang

In BASH we starts all our scripts with the #! Shabang and then the location of your shell.

This allows the kernel to know which interpreter to use.

WRITING OUR FIRST SCRIPT

Open a text editor and write the following:

```
---
#!/bin/bash

echo “Welcome to Scripting!”
---
```

COMMENTS

Comments are lines that the computer will not interpret as commands.

These are helpful when trouble shooting and revising your script if necessary.

In BASH the octothorpe (#) will indicate a comment.

COMMENTS IN SCRIPTS

The following is an example of a comment, the placement of the comment does not matter except to the human reading the script.

```
----
#!/bin/bash

echo “Welcome to Scripting!”
      # Prints out the string “Welcome to Scripting” to the screen
----
```

EXECUTING SCRIPTS

By default BASH scripts will not have executable permissions.

In order to execute a script, you must either:

- Pass the script to an interpreter (in our case BASH) ‘bash ourscript.sh’
- Assign your script executable permissions using ‘chmod’.

When we execute a script, the script is run in a new shell and not in the interactive shell that you called the script from.

EXECUTING SCRIPTS

```
workstation16:~$ ls -l
workstation16:~$ bash firstscript.sh
```

this will allow us to run our script a single time without executable permissions

---

```
workstation16:~$ chmod +x firstscript.sh
workstation16:~$ ls -l
```

this changes the permissions and allows everyone to run our script.

```
workstation16:~$ ./firstscript.sh
```

CHECK ON LEARNING

1. What must you include as the first line of your script?

   A Shabang #! with the interpreter you wish to use. e.x #!/bin/bash

2. What is the purpose of adding comments to a script?

   To indicate what your script does for troubleshooting or future reference

3. Why can we not immediately execute a script once it's been written?

   Your script must have executable permissions

BASH BUILT-IN COMMANDS

The BASH shell has many commands that are considered built-in (also known as "internal commands") that are native to the shell.

These commands will not be required to open a new process or run a program.

- declare
- echo
- let
- local
- read
- type
- source
- export

BUILT-IN COMMANDS

declare - allows you to give attributes to variables. -i for integer and -A for associative array

echo – outputs the given arguments to the screen. -e allows for backslash interpretation

let – allows arithmetic operations to be performed on variables

local – creates a local variable for a function

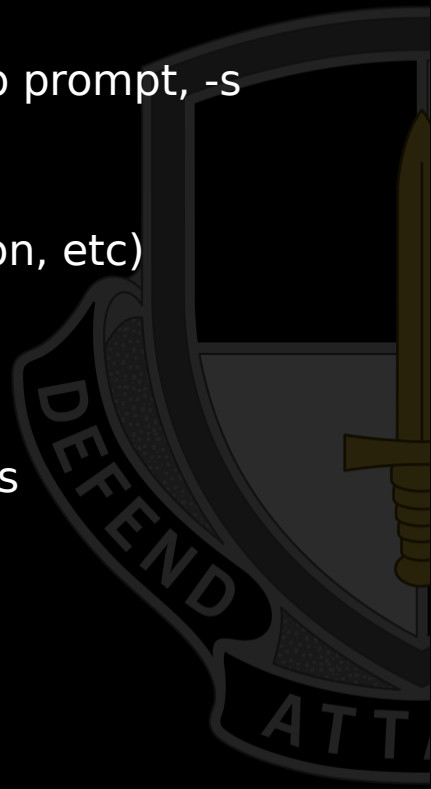
BUILT-IN COMMANDS

read – reads input from standard input and assigns as a variable. -p prompt, -s silent, -r backslash not an escape

type – tells how commands are interpreted (i.e builtin, alias, function, etc)

source – read and execute file, can be used for variables

export – allows us to pass variables and functions to child processes



CHECK ON LEARNING

1. What command can you use to ask for user input?

   read

2. What command allows you to specify your variable type?

   declare

3. What commands would you use in relation to variables?

   source, export, declare, local

BASH VARIABLES

Variables are containers for data that will allow you to easily reference and take action on that data stored inside.

BASH variables are stored as character strings.

Variables are only defined within your current shell session.

In order to create a variable you simply give the variable name and set it equal to the value you want it to contain. To access the data within our variable we must use the $.

BASH VARIABLES SCRIPT

```
---
#!/bin/bash

string="This is a variable containing a long string"
character=a
integer=1234

echo $string
echo $character
echo $integer
---
```

BASH VARIABLES

```
workstation16:~$ my_var=One
```

```
#!/bin/bash

echo "\"$my_var\" is the value of my variable"

my_var=One

echo "\"$my_var\" is the value of my variable"
```

EXPORT/SOURCE

If we want variables from our current interactive shell to be accessible to scripts we must use the export command.

```
workstation16:~$ export my_var
workstation16:~$ ./ourscript.sh
```

If we want the variable changes that take place within a script we must source our script when we run it.

```
workstation16:~$ source ./ourscript.sh
workstation16:~$ echo $my_var
```

DECLARE

BASH allows us to assign variables as a specific type using the declare built-in command.

```
workstation16:~$ declare -i my_var
```

CHECK ON LEARNING

1. True or False. BASH handles all variables as strings by default.

   True

2. What must you use when you call a variable?

   $

BASH FEATURES

When writing BASH scripts we have many features that we can include to make our scripts more user friendly.

- If-Then statements
- Conditional statements
- Comparison Operators
  - String comparison
  - Integer comparison
- Loops
  - For loop
  - While loop

IF-THEN STATEMENTS

If-Then statements allow you to write a block of commands that will complete based on the evaluation of a comparison or a test.

This will allow you to determine what code is run based on something being true or false.

WRITING IF-THEN STATEMENTS

In BASH to create an If-Then statement you first create your If giving the condition that must be met.

After your if statement you have a Then statement telling the commands you wish to be executed.

In BASH you must include fi at the end of your statement to signal the end of your code.

WRITING IF-THEN STATEMENTS

```
----
#!/bin/bash

if [ conditional ]
then
        commands to execute
fi
----
```

```
----
#!/bin/bash

a=$true

if $a
then
        echo “variable a is true”
fi
----
```

COMPARISON OPERATORS

Comparison operators allow us to compare string and integers to determine an outcome.

Typically these will be pair with If-Then statements to indicate a specific outcome based on what the comparison is.

COMPARISON OPERATORS (STRING)

When writing a comparison statement we start with brackets [] and then the two things we wish to compare with an operator.

If we use double brackets [[ ]] we can use pattern matching which we'll cover a little later.

- == equal to
- != not equal to
- < less than
- > greater than
- -z string is null
- -n string is not null
- -f file exists
- -s file size is not zero
- -d file is a directory

COMPARISON OPERATORS (INTERGER)

When writing a comparison statement we start with brackets [] and then the two things we wish to compare with an operator.

- -eq equal to
- -ne not equal to
- -lt less than
- -gt greater than
- -le less than or equal to
- -ge greater than or equal to

LOOPS

A loop will allow a set of commands to be run over and over until broken.

Loops will be broken based on the condition that you set or if there are no more items to iterate on.

LOOPS (FOR)

A for loop will repeat based on the statement given.

For loops begin with the word for, are given an amount of times to repeat, and then told to do a set of commands.

For loops must end with the word done.

LOOPS (FOR)

```
----
#!/bin/bash

for i in condition
do
commands
done
----
```

```
----
#!/bin/bash

for i in 1 2 3 4
do
echo $i
done
----
```

```
----
#!/bin/bash

iffile = ifcon.sh
filecon = $(cat $iffile)

for i in $filecon
do
echo $i
done
----
```

LOOPS (WHILE)

A while loop will allow a set of commands to be run continuously until the specified condition evaluates to false.

While loops will start with the word while and then a condition.

After the condition this loop contains the word do to indicate the commands that will be run and then done indicating the end of the block of commands that will be run every repetition.

LOOPS (WHILE)

```
----
#!/bin/bash

While condition
do
commands
done
----
```

```
----
#!/bin/bash

declare -i a
a=1

while [ $a -lt 6 ]
do
echo $a
a=”$a+1”
done
----
```

LOOPS (WHILE)

While Loop Script Demo

LOOPS (UNTIL)

An until loop will allow a set of commands to be run continuously until the specified condition evaluates to true.

Until loops will start with the word until and then a condition.

After the condition this loop contains the word do to indicate the commands that will be run and then done indicating the end of the block of commands that will be run every repetition.

FUNCTIONS

Functions allow you to write a set of commands that can be executed by calling the name of the function.

This allows you to create smaller logical subsections to preform their task when called on.

Within scripting, programmers will use functions to perform repetitive tasks to reduce the amount of code that must be written.

FUNCTIONS

```
---
#!/bin/bash

FunctionName () {
    commands
}
---
```

```
---
#!/bin/bash

hello () {
    echo “Welcome to Scripting”
}
---
```

FUNCTIONS

1. What is the difference between a for loop and a while loop?

   A For loop will iterate over a range whereas a while loop will loop until a condition is false

2. How are if-then statements written?

   By giving a test to be evaluated as true or false and executing code accordingly.

3. True or False. Conditional statements allow you to avoid writing repetitive tasks.

   False

FUNCTIONS

1. What must you include at the end of an If-Then statement?

   fi

2. What surrounds the code block in a loop?

   do; done