Lab #4

Bash Scripting

BINF 2111, Fall 2024

Terminology

Script	A list of programmatically-written instructions (commands) that can be carried out when ran
Variable	A named container for a particular set of bits or type of data (e.g. integer, float, string, etc.)
Array	A data structure that can store a fixed-size collection of elements of the same data type

Commands To Know

Commands are case sensitive !!

● Command

● Input (like a file or folder)

Command	Meaning	Usage
bash	Run a bash script	bash script.sh
chmod	Change permissions (access mode) of a file	chmod script.sh
whoami	Prints the current user	whoami
$USER	The environment variable that points to the current user	echo $USER
$ROOT	The environment variable that points to the root directory	echo $ROOT
date	Prints the current date and time	date
${#v}	The length of variable v	echo ${#v}

Command Breakdown - chmod

- **chmod**: change permissions of a file
  - Useful Options
    - -R Recursive, change permissions of folder and everything in it

- Octal Mode
  - Three digit number:
    - First - Owner
    - Second - Group
    - Third - Others
  - Add the values to change permissions
    - 4 Read permission
    - 2 Write permission
    - 1 Execute permission

- Usage

  4+2+1=7 (RWX)
  - `chmod 777 file.txt`

  Owner, Group, and Others get all permissions

  $\underbrace{4+2=6}$ (RW) 2+1=3 (WX)
  - `chmod 643 file.txt`

  Owner gets read and write
  Group gets read only
  Others get write and execute

Command Breakdown - chmod

- **chmod**: change permissions of a file
  - Useful Options
    - -R Recursive, change permissions of folder and everything in it
- Symbolic Mode
  - Combination of letters and operators
    - u Owner
    - g Group
    - o Others
    - a All
    - + Add permissions
    - - Remove permissions
    - r Read
    - w Write
    - x Execute
- Usage
  - `chmod a+x file.txt`

    Add execute permissions for all individuals
  - `chmod u+rw,go+r file.txt`

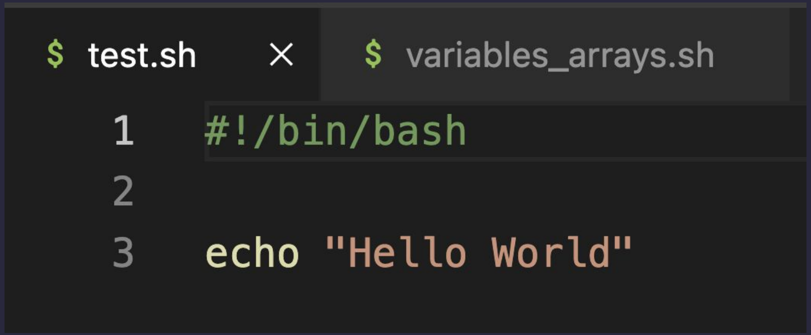
    Add read and write permissions for the owner
    Add read permissions for the group and others

Scripts

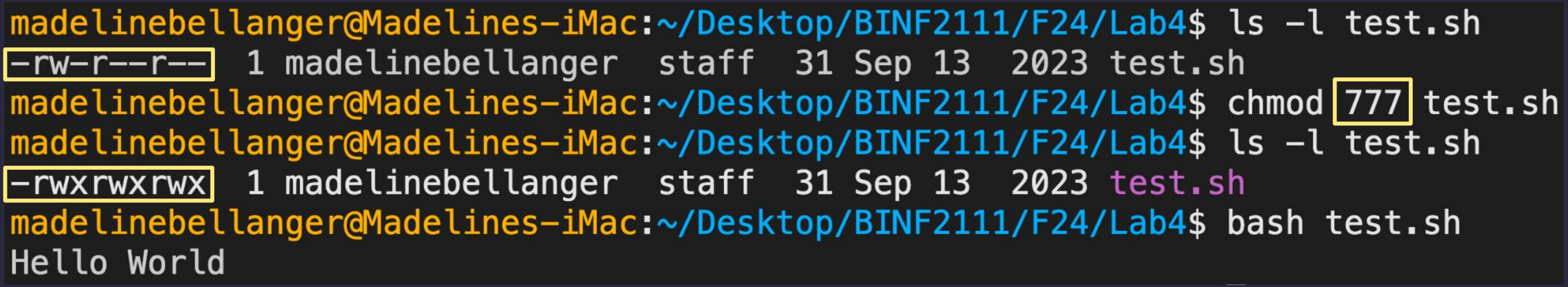
- Set Up
  - Always begin with hashbang/shebang (#!/bin/bash)

    ```
    #!/bin/bash
    ```
  - File name should end in .sh
  - Edited with text editor or IDE
    - My favorites are nano (text editor) and Visual Studio Code (IDE)
- Compiling and Running
  - "Compile" with `chmod`
    - Set execute permissions
  - Running
    - `./file.sh`
    - `bash file.sh`

Script Example



```bash
#!/bin/bash

echo "Hello World"
```



```
madelinebellanger@Madelines-iMac:~/Desktop/BINF2111/F24/Lab4$ ls -l test.sh
-rw-r--r--  1 madelinebellanger  staff  31 Sep 13  2023 test.sh
madelinebellanger@Madelines-iMac:~/Desktop/BINF2111/F24/Lab4$ chmod 777 test.sh
madelinebellanger@Madelines-iMac:~/Desktop/BINF2111/F24/Lab4$ ls -l test.sh
-rwxrwxrwx  1 madelinebellanger  staff  31 Sep 13  2023 test.sh
madelinebellanger@Madelines-iMac:~/Desktop/BINF2111/F24/Lab4$ bash test.sh
Hello World
```

Variables

- Naming Conventions
  - Should not start with numbers
  - Should not contain
    - Periods, colons, dashes
- Assigning Variables
  - Assigned with **equals sign**
  - Strings belong in **quotation marks**
- Displaying Variables
  - Referenced with **dollar sign**
  - Can use echo or printf

Code:

```
# Assigning variables
echo "Assigning variables..."
string_v="variable"
string_v2="This is also a variable"

int_v=76
float_v=12.471

echo "Done!"
echo    # Print an empty line


# Printing variables
echo "Printing variables: "
echo $string_v, $int_v
printf "$string_v2 \n$float_v\n"
```

Output:

```
Assigning variables...
Done!

Printing variables:
variable, 76
This is also a variable
12.471
```

Variables - Commands and Math

- Variables can also contain commands!
  - Command should be inside of dollar sign and parentheses
  - $(command here)
- You can also do math (but only whole numbers)!
  - Length of a string
    - Found with a number sign before the string variable contained in curly braces
    - ${#string_var}

Code:

```
# Using a variable to reference a command
echo "Command as a variable: "
helloworld=$(echo "Hello World")
echo $helloworld
echo    # Print an empty line



# Finding the length of a string variable
echo "Math solutions as variables: "
string_length=${#string_v}
echo "String length is $string_length characters"
```

Output:

```
Command as a variable:
Hello World

Math solutions as variables:
String length is 8 characters
```

Variables - Math

- You can also do math (but only whole numbers)!
  - Length of a string
    - Found with a number sign before the string variable contained in curly braces
    - ${#string_var}
  - Adding and Subtracting
    - Math should be inside of dollar sign and two sets of parentheses
    - $((math here))

Code:

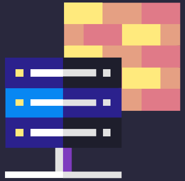
```
# Finding the length of a string variable
echo "Math solutions as variables: "
string_length=${#string_v}
echo "String length is $string_length characters"
echo    # Print an empty line


# Adding the length of two variables together
math=$(($string_length+${#string_v2}))
echo "The length of both strings added together is $math"
```

Output:

```
Math solutions as variables:
String length is 8 characters

The length of both strings added together is 31
```

Arrays - Creation and Elements

- Contained in a set of parentheses
- Finding elements
  - First element is found at array[0]
  - Range of elements are found with colons
    - array[@]:2:5 (3rd through 6th element)
  - Get all elements with @
    - array[@]



Element Indexing

```
 0       1    2      3      4    5      6
array=("this" "is" "an" "item" "in" "an" "array")
```

Code:

```
echo "The first element is: ${array[0]}"
echo "The third through sixth elements are: ${array[@]:2:5}"
echo "All elements in the array are: ${array[@]}"
```

Output:

```
The first element is: this
The third through sixth elements are: an item in an array
All elements in the array are: this is an item in an array
```

Arrays - Deleting Elements

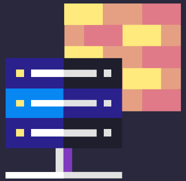
- Deleting elements
  - `unset 'array[4]'` * will delete the element within the array *
  - `${array[@]/"item"}`
  - `${array[@]/it*/}`

Code:

```
echo "Delete Item Method #1"
echo ${array[@]/"item"}
echo    # Print an empty line

echo "Delete Item Method #2"
echo ${array[@]/it*/}
echo    # Print an empty line

echo "Delete Item Method #3"
unset 'array[3]'
echo ${array[@]}
```

Output:

```
Delete Item Method #1
this is an in an array

Delete Item Method #2
this is an in an array

Delete Item Method #3
this is an in an array
```

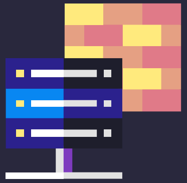


Arrays - Adding Elements

- Adding elements
  - `array=("${array[@]}" "new_item")`
  - `array+=('new_item')`

Code:

```
echo "Add Item Method #1"
array=("${array[@]}" "new item")
echo ${array[@]}
unset 'array[6]'
echo    # Print an empty line

echo "Add Item Method #2"
array+=('new item')
echo ${array[@]}
```



Output:

```
Add Item Method #1
this is an in an array new item

Add Item Method #2
this is an in an array new item
```