**Introduction to Bash Scripting**

Bash (or shell) scripting is a great way to automate repetitive tasks and can save you a ton of time as a developer. Bash scripts execute within a Bash shell interpreter terminal. Any command you can run in your terminal can be run in a Bash script. When you have a command or set of commands that you will be using frequently, consider writing a Bash script to perform it.

There are some conventions to follow to ensure that your computer is able to find and execute your Bash scripts. The beginning of your script file should start with #!/bin/bash on its own line. This tells the computer which type of interpreter to use for the script. When saving the script file, it is good practice to place commonly used scripts in the ~/bin/ directory.

The script files also need to have the “execute” permission to allow them to be run. To add this permission to a file with filename: script.sh use:

chmod +x script.sh

Your terminal runs a file every time it is opened to load its configuration. On Linux style shells, this is ~/.bashrc and on OSX, this is ~/.bash\_profile. To ensure that scripts in ~/bin/ are available, you must add this directory to your PATH within your configuration file:

* PATH=~/bin:$PATH

Now any scripts in the ~/bin directory can be run from anywhere by typing the filename.

**Instructions**

**1.**

In the code editor, at the top of the **script.sh** file, add a line to specify that the script should be run using bash.

Hint

To make it known that the **script.sh** file is a Bash script, include the following line at the very top of the file.

#!/bin/bash

The file editor is the section that has the **script.sh** file name at the very top. You can also distinguish it from the terminal due to the line numbers included on its left side.

The terminal, which you’ll interact with in a later step, has a blinking cursor and has a tab titled: **bash**.

**2.**

Add an echo bash command to **script.sh** to make the script print “Hello Codecademy!” when the script is run.

Hint

Include the following line in the terminal:

echo "Hello Codecademy!"

Also, “Codecademy” only has 1 “a”. It’s “Codecademy” not “CodeAcademy”!

**3.**

In the provided terminal, type out the command to run the script.

Hint

In the terminal, type out the command ./script.sh, it should look like:

$ ./script.sh

You do not need to type out the $. The $ is automatically included in the terminal and is used to show that the provided code is a terminal command.

After the command has been properly typed out, press enter to run the script.

# Variables

Within bash scripts (or the terminal for that matter), variables are declared by setting the variable name equal to another value. For example, to set the variable greeting to “Hello”, you would use the following syntax:

greeting="Hello"

Note that there is no space between the variable name, the equals sign, or “Hello”.

To access the value of a variable, we use the variable name prepended with a dollar sign ($). In the previous example, if we want to print the variable value to the screen, we use the following syntax:

echo $greeting

**Instructions**

**1.**

Create a variable phrase on the line before the current echo statement. Set the phrase equal to the string “Hello to you!”.

Hint

phrase="Hello to you!"

**2.**

Update the echo statement to make use of the variable instead of the fixed string.

Hint

echo $phrase

**3.**

Run the script in the terminal.

Hint

Use ./script.sh to run the script.

**LEARN BASH SCRIPTING**

**Conditionals**

When bash scripting, you can use conditionals to control which set of commands within the script run. Use if to start the conditional, followed by the condition in square brackets ([ ]). then begins the code that will run if the condition is met. else begins the code that will run if the condition is not met. Lastly, the conditional is closed with a backwards if, fi.

A complete conditional in a bash script uses the following syntax:

if [ $index -lt 5 ]

then

echo $index

else

echo 5

fi

Bash scripts use a specific list of operators for comparison. Here we used -lt which is “less than”. The result of this conditional is that if $index is less than 5, it will print to the screen. If it is 5 or greater, “5” will be printed to the screen.

Here is the list of comparison operators for numbers you can use within bash scripts:

* Equal: -eq
* Not equal: -ne
* Less than or equal: -le
* Less than: -lt
* Greater than or equal: -ge
* Greater than: -gt
* Is null: -z

When comparing strings, it is best practice to put the variable into quotes ("). This prevents errors if the variable is null or contains spaces. The common operators for comparing strings are:

* Equal: ==
* Not equal: !=

For example, to compare if the variables foo and bar contain the same string:

if [ "$foo" == "$bar"]

**Instructions**

**1.**

We’ve added two different greetings and a variable to store how many times the user has been greeted before. If this variable is less than 1, we want to use first\_greeting. Otherwise, we want to use later\_greeting.

Add a line setting up the if conditional. Use the -lt operator.

Hint

if [ $greeting\_occasion -lt 1 ]

**2.**

Use echo to print $first\_greeting if the conditional is met and $later\_greeting otherwise. Be sure to close the if statement.

Hint

Use then, else and fi.

**3.**

Run the script in the terminal. Try adjusting the value of $greeting\_occasion to test your if statement.

Hint

Use ./script.sh to run the script.

#!/bin/bash

first\_greeting="Nice to meet you!"

later\_greeting="How are you?"

greeting\_occasion=1

if [ $greeting\_occasion -lt 1 ]

then

echo $first\_greeting

else

echo $later\_greeting

fi

# Loops

There are 3 different ways to loop within a bash script: for, while and until.

A for loop is used to iterate through a list and execute an action at each step. For example, if we had a list of words stored in a variable paragraph, we could use the following syntax to print each one:

for word in $paragraph

do

echo $word

done

Note that word is being “defined” at the top of the for loop so there is no $ prepended. Remember that we prepend the $ when accessing the value of the variable. So, when accessing the variable within the do block, we use $word as usual.

Within bash scripting until and while are very similar. while loops keep looping while the provided condition is true whereas until loops loop until the condition is true. Conditions are established the same way as they are within an if block, between square brackets. If we want to print the index variable as long as it is less than 5, we would use the following while loop:

while [ $index -lt 5 ]

do

echo $index

index=$((index + 1))

done

Note that arithmetic in bash scripting uses the $((...)) syntax and within the brackets the variable name is not prepended with a $.

The same loop could also be written as an until loop as follows:

until [ $index -eq 5 ]

do

echo $index

index=$((index + 1))

done

**Instructions**

**1.**

Start by opening a while loop directly after the greeting\_occasion variable. We want to loop while $greeting\_occasion is less than 3.

Hint

Remember, the while syntax is:

while [ $index -lt 5 ]

**2.**

Move the existing if...fi block inside of the while loop. Remember to use the do and done keywords. Increment greeting\_occasion on each loop.

Hint

Use this syntax to increment a variable:

index=$((index + 1))

**3.**

Run the script. You should see the first greeting as “Nice to meet you!” followed by two “How are you?” greetings.

Hint

Use ./script.sh to run the script.

#!/bin/bash

first\_greeting="Nice to meet you!"

later\_greeting="How are you?"

greeting\_occasion=0

while [ $greeting\_occasion -lt 3 ]

do

  if [ $greeting\_occasion -lt 1 ]

  then

    echo $first\_greeting

  else

    echo $later\_greeting

  fi

  greeting\_occasion=$((greeting\_occasion + 1))

done

# Inputs

To make bash scripts more useful, we need to be able to access data external to the bash script file itself. The first way to do this is by prompting the user for input. For this, we use the read syntax. To ask the user for input and save it to the number variable, we would use the following code:

echo "Guess a number"

read number

echo "You guessed $number"

Another way to access external data is to have the user add input arguments when they run your script. These arguments are entered after the script name and are separated by spaces. For example:

saycolors red green blue

Within the script, these are accessed using $1, $2, etc, where $1 is the first argument (here, “red”) and so on. Note that these are 1 indexed.

If your script needs to accept an indefinite number of input arguments, you can iterate over them using the "$@" syntax. For our saycolors example, we could print each color using:

for color in "$@"

do

echo $color

done

Lastly, we can access external files to our script. You can assign a set of files to a variable name using standard bash pattern matching using regular expressions. For example, to get all files in a directory, you can use the \* character:

files=/some/directory/\*

You can then iterate through each file and do something. Here, lets just print the full path and filename:

for file in $files

do

echo $file

done

**Instructions**

**1.**

Let’s ask the user how many times the program should greet them.

Just before the while loop, print the following to the terminal: “How many times should I greet?”

Hint

Use the echo command.

**2.**

Get user input and assign it to the variable greeting\_limit. Replace the limit of 3 in the while loop with $greeting\_limit.

Hint

You can assign a user’s input to a variable foo as follows:

read foo

**3.**

Run the script and try out different greeting\_limits.

Hint

Use ./script.sh to run the script.

#!/bin/bash

first\_greeting="Nice to meet you!"

later\_greeting="How are you?"

greeting\_occasion=0

echo "How many times should I greet?"

read greeting\_limit

while [ $greeting\_occasion -lt $greeting\_limit ]

do

  if [ $greeting\_occasion -lt 1 ]

  then

    echo $first\_greeting

  else

    echo $later\_greeting

  fi

  greeting\_occasion=$((greeting\_occasion + 1))

done

# Aliases

You can set up aliases for your bash scripts within your .bashrc or .bash\_profile file to allow calling your scripts without the full filename. For example, if we have our saycolors.sh script, we can alias it to the word saycolors using the following syntax:

alias saycolors='./saycolors.sh'

You can even add standard input arguments to your alias. For example, if we always want “green” to be included as the first input to saycolors, we could modify our alias to:

alias saycolors='./saycolors.sh "green"'

**Instructions**

**1.**

Our script is updated to take an argument for the number of times the user wants to be greeted:

./script.sh 5 #greets 5 times

Let’s create an alias so that when you type greet3 in the terminal, our script greets you three times.

In your own environment, you could add this alias to your **~/.bashrc** to make the alias active every time the terminal is started.

Here, just make the alias in the command line.

Hint

Your command should look like the following:

alias greet3="./script.sh 3"

**2.**

Test out your new alias!

Hint

Use the alias by typing greet3 in the terminal.

**Review**

Take a minute to review what you’ve learned about bash scripting.

* Any command that can be run in the terminal can be run in a bash script.
* Variables are assigned using an equals sign with no space (greeting="hello").
* Variables are accessed using a dollar sign (echo $greeting).
* Conditionals use if, then, else, fi syntax.
* Three types of loops can be used: for, while, and until.
* Bash scripts use a unique set of comparison operators:
  + Equal: -eq
  + Not equal: -ne
  + Less than or equal: -le
  + Less than: -lt
  + Greater than or equal: -ge
  + Greater than: -gt
  + Is null: -z
* Input arguments can be passed to a bash script after the script name, separated by spaces (myScript.sh “hello” “how are you”).
* Input can be requested from the script user with the read keyword.
* Aliases can be created in the .bashrc or .bash\_profile using the alias keyword.

**Instructions**

Our completed script is in the code editor. Feel free to edit it to make it your own.

Some ideas:

* ask the user for different greetings
* add more than two greetings
* add more conditions to adjust the greetings over time