**CSCI 2200, Fall 2018**

**Assignment 2**

1. When first working with a language, it's a good practice to determine what limits the language puts on arithmetic. This problem asks you to implement in **bash**, then run, the following pseudocode to determine the largest power of two that the **bash** interpreter can represent.

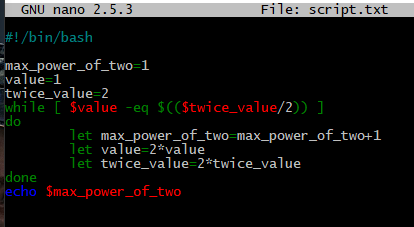
***set max\_power\_of\_two to 1 set value to 1***

***set twice\_value to 2 while value equals twice\_value/2 begin add one to max\_power\_of\_two set value to 2 \* value set twice\_value to 2 \* twice\_value***

***end***

***display max\_power\_of\_two***

After you implement it, run it, showing your code's execution. Provide a screenshot of your code.





­­

1. Write a script that prints out: ­­

*[Current user] has [number of non-hidden files in users home directory] files in his/her home directory.*

Provide a screenshot of your script along with the output received with the execution of the script

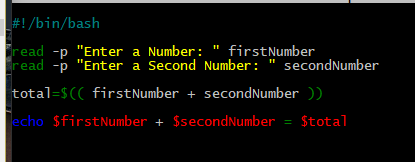


1. Write a script that prompts for input as follows: *Enter a Number:*

*Enter another number:*

The script should then add the numbers together and displ­ay the results as *[First number] + [second number] = [total]*

Provide a screenshot of your script along with the output received with the execution of the script with two numbers





1. Check *script1.sh* posted on D2L, right below Assignment 2. Then, answer each of the following questions about *script1.sh*.
   * 1. What is the meaning of the ***$#*** at lines 5 and 6?

Number of arguments.

* + 1. What is the effect of the ***<<<*** at line 9?

- rake stdin from here string

* + 1. What is the purpose of the redirect to ***/dev/null*** at line 9?

-So that no null or error instances are printed.

* + 1. What is the effect of the ***i++*** in line 17?

It is used as an incrementor.

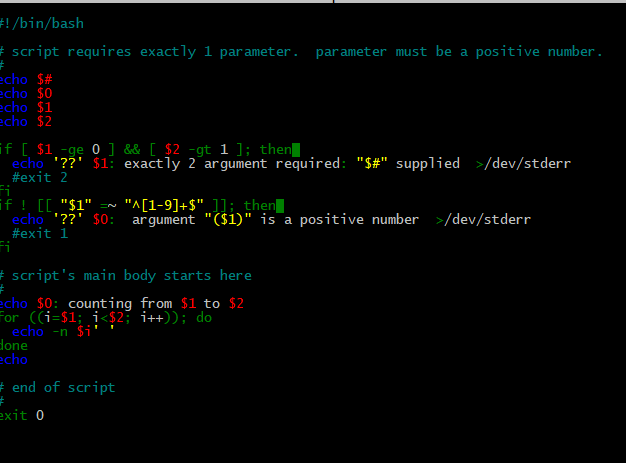
* + 1. What is the purpose of the -***n*** in line 18?

Create a new line

* + 1. What is the purpose of the singly quoted space in line 18?

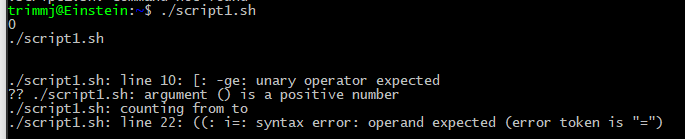
Adds a space after the variable

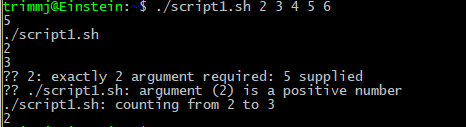
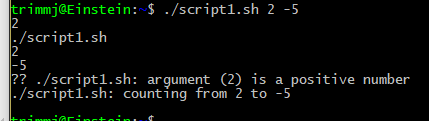
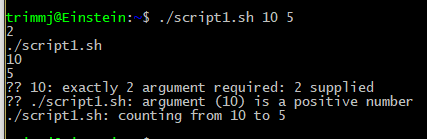
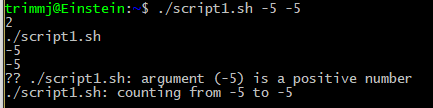
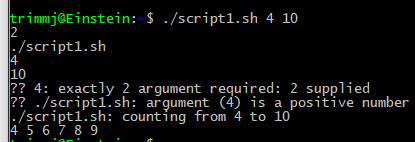
1. Modify *script1.sh* to apply the following changes:
   * 1. Allow the first parameter to be 0 as well as positive.
     2. Add a second parameter: a second positive integer that identifies the number to count up to.
     3. Modify the check at line 5 so that it checks for two parameters
     4. Add two checks before the script's main body
        1. A first that confirms that the second parameter is an integer (Hint: Use grep search for a member of the character class [0-9])
        2. A second that verifies that the first parameter is less than or equal to the second e. Exit with a 1 status code if either test fails.
     5. Modify line 16 to say that the loop is counting from the first parameter to the second parameter
     6. Modify the loop header at line 17 so that the loop counts up from the first parameter to the second parameter, inclusive



For this problem, provide a screenshot of the updated *script1.sh*, along with the output from six test runs:

* + - One where the script is invoked with too few parameters



* + - One where the script is invoked with too many parameters
    - 
    - One where the script is invoked with exactly two parameters, where the first parameter is a nonnegative integer and the second isn't.
    - 
    - One where the script is invoked with exactly two parameters, both nonnegative integers, where first integer is greater than the second
    - 
    - One where the script is invoked with exactly two parameters, both nonnegative integers, where first integer equals the second
    - 
    - One where the script is invoked with exactly two parameters, both nonnegative integers, where first integer is smaller than the second
    - 

1. This can be viewed as a three-part problem.

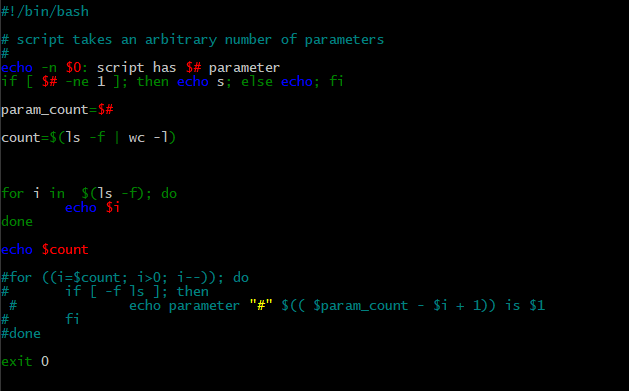
a. Modify *script2.sh* so that it returns a count of the number of command line arguments that correspond to regular files.

You can do so by

* + - * replacing the loop body's first statement with an if statement that o tests whether the current ***$1*** is a regular file o increments a counter variable if so
      * adding a statement after the loop that echoes the final count

You will, of course, need to initialize this counter variable to 0 before the loop's start and output its value at loop's end.

Also, make sure that the statement that outputs the count clearly labels the value that it’s showing.



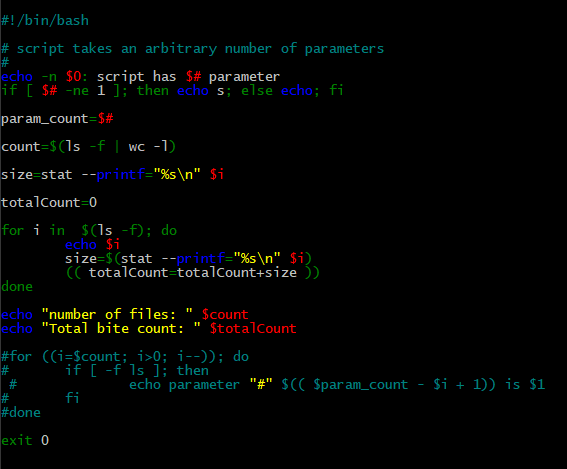
b. Modify *script2.sh* so that it returns a count of the number of command line arguments that correspond to regular files as well as the total number of bytes in all these files. You can do so by

 replacing the loop body's first statement with an if statement that o tests whether the current ***$1*** is a regular file

o increments a counter variable and total size variable if so

 adding statements after the loop that echo the final count and total size

You will, of course, need to initialize these variables to 0 before the loop's start and output its value at loop's end. Also, make sure that the statements that output the count and total clearly label the values that they’re showing.



c. Modify *script2.sh* so that it returns a count of the number of command line arguments that correspond to regular files; the total number of bytes in all these files; and, if at least one regular file was located, the average number of bytes in each file. You can do so by

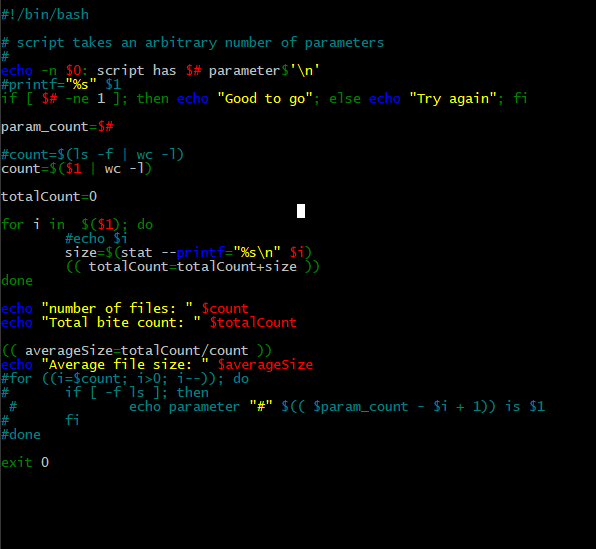
 replacing the loop body's first statement with an if statement that o tests whether the current ***$1*** is a regular file

o increments a counter variable and total size variable if so

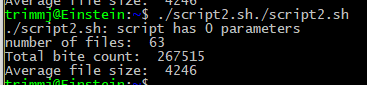
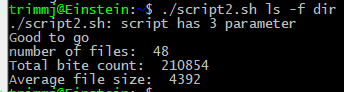
* + - adding statements after the loop that echo the final count, the total size, and, if the final count is nonzero, the average number of bytes per file.

You will, of course, need to initialize the counter and total size variables to 0 before the loop's start and output its value at loop's end. Don't sweat the roundoff error; truncating division will be adequate for computing average file size.

Finally, make sure that the statements that output the count and total clearly label the values that they’re showing.



For this problem from among **a-c** that you choose to work, provide a screenshot of your sample script (clearly label your answer), along with the output from three test runs:

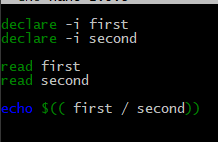
* + - One where the script is invoked with no arguments
    - 
    - One where the script is invoked with multiple arguments, at least one of which corresponds to a regular file
    - 
    - One where the script is invoked with multiple arguments, at least one of which fails to correspond to a regular file



1. This can be viewed as a four-part problem.

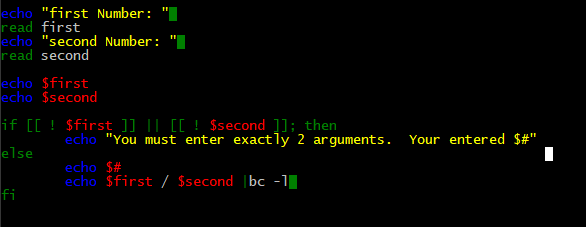
* 1. Write a script (*a.sh*) that accepts two arguments on the command line. It should then divide the first number by the second (you may use integer or floating point arithmetic for this problem) number and display the results as:

[First number] / [second number] = [answer]

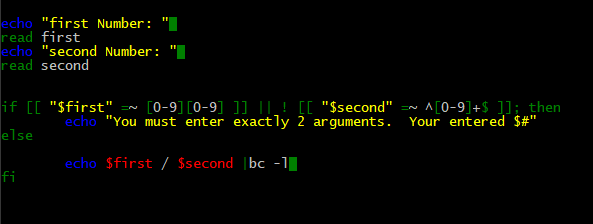


* 1. Copy your *a.sh* and name it *b.sh*. Modify the script so that it checks to make sure the user has entered 2 arguments. (Use an if statement).
     + If the user has not entered two arguments, the script should display the message:

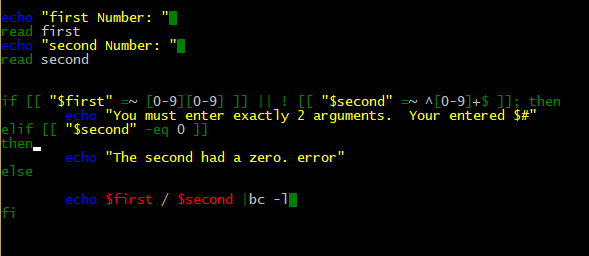
“*You must enter exactly 2 arguments. You entered [numberofargs].”* (where [numberofargs] is the number of arguments entered.)

* + - If the user entered two arguments, the script should then divide the first number by the second number as in the previous problem.
    - 

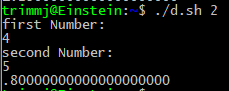
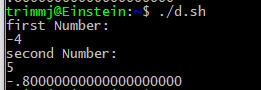
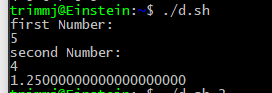
* 1. Once again copy the previous script (*b.sh*) and rename it (*c.sh*) for this problem. Hint: For this problem experiment with the following command (giving numeric or non-numeric values): echo $1 | grep -v '[^0-9]'
     + After checking that the user has entered two arguments (in the previous problem), the new script should check if the first argument is a single digit number.
     + If the first argument is not a single (or multiple) digit number, the script should give an error message - “*The first argument you entered was not a number”.*
     + If the first argument is a number, the script should check if the second argument is a number.
     + If the second argument is not a number, the script should give an error message - *“The second argument you entered was not a number”.*
     + If the second argument is a number, the script should then proceed and divide the first number by the second numbers together.



* 1. For this problem, copy *c.sh* and rename it *d.sh*. The script should have all the previous error checking. Add commands to check and see if the second number is 0 (zero). If the second number is 0 then the user should receive an error message. If the second number is not zero, then the script should divide the first number by the second number and display the results as in the previous problems.



Provide a screenshot of your script, along with the output from the below test runs:

* + - One where the script is invoked with too few parameters
    - **I don’t know how to invoke script d.sh with too few parameters. The way I wrote is was to read from the terminal….**
    - One where the script is invoked with too many parameters
    - 
    - One where the script is invoked with exactly two parameters, where the first parameter is a nonnegative integer and the second isn't.
    - 
    - One where the script is invoked with exactly two parameters, both nonnegative integers.
    - 

1. Create a script that uses list-based **for loop** and lists the files in the current directory. Show the script.



1. (BONUS) Use a nested **bash** **while** loop and a nested bash **until** loop to generate subtraction tables for values from 0-9.
   * 1. Generate the one table using (only) nested **while**  loops, with no other looping constructs
     2. Generate the other table using (only) nested **until** loops, with no other looping constructs

For both tables, allow 4 spaces per column. You can print a single number in four spaces using **bash'**s **printf** built-in.

For example, **printf "%s " 14** prints the string '**14** ' without a trailing newline **printf "%4s" 14** prints the string ' **14**' without a trailing newline. **printf "%4s\n" 14** prints the string ' **14**' followed by a newline.\

Hint:

The output for both problems is

**0 1 2 3 4 5 6 7 8 9**

* + - * 1. **0 0 0 0 0 0 0 0 0 0**
        2. **0 1 2 3 4 5 6 7 8 9**
        3. **0 2 4 6 8 10 12 14 16 18**
        4. **0 3 6 9 12 15 18 21 24 27**
        5. **0 4 8 12 16 20 24 28 32 36**
        6. **0 5 10 15 20 25 30 35 40 45**
        7. **0 6 12 18 24 30 36 42 48 54**
        8. **0 7 14 21 28 35 42 49 56 63**
        9. **0 8 16 24 32 40 48 56 64 72**
        10. **0 9 18 27 36 45 54 63 72 81**

Use an extra iteration with a special value and an "if" test to generate the column headers.

