CS251: Outlab: Avenger**sh** Assemble!

Please refer to general instructions and submission guidelines at the end of this document before submitting.

Avengers, earth's mightiest heroes have saved our world on multiple occasions. But even the greatest heroes need help at times. Today the chance is in your hands!

1. What year was it?

Prologue

It's been many years since the crash, Captain America had just woken up from his long deep sleep. Captain found himself alone in a modern world that he hardly recognized with no idea what he was going to do with his life. The first thing that comes to his mind, "How many years was I asleep?" Unfortunately, he couldn't remember. But he knew that the crash happened in a leap year just after 1897. Your task is to help Captain America trace out the year of his crash.

Problem

A. Write a bash script named **isLeapYear.sh**, which when given a year (a positive integer) as an argument, prints (without quotes) to stdout

a. "Leap Year!" if it is a leap year
b. "Not a leap year." otherwise
c. "Invalid argument!" if the input is anything other than a positive integer

Usage: `bash isLeapYear.sh <year>`

(Hints: logical operators, if-else-block)

B. Using the above-written script, input years from 1897 to see which year turns out to be a leap year. Write the year in YYYY format in file **yearOfCrash.txt** (will be autograded)

2. Peace in our time.

Prologue

After successfully raiding a Hydra base in Sokovia and seizing Loki's scepter it seemed like a good time for Avengers to party. But little did they know what Mr. Stark (a.k.a Iron Man) and Dr. Banner (a.k.a Hulk) were planning secretly. By the time they did, Ultron (an evil A.I accidentally created by Mr.Stark and Dr.Banner) had already destroyed Jarvis (a good A.I created by Mr.Stark). Soon he would destroy all their data too! But Mr.Stark always has a contingency plan, Offline storage! Your task is to help Mr.Stark back up his data quickly and efficiently.

Fortunately, Mr.Stark also uses Linux, and baSH comes to the rescue

Problem

A. The basic task in backup is copying files. Observe the behavior of **cp** command.

1. Does it copy and overwrite destination file if it already exists?
2. Does it overwrite even when the source and destination files are of different types but have the same name?
3. Does it copy when the destination file path has non-existent directories?
   a. i.e. say if destination file path is `/home/labuser/Desktop/CarromChase/README.md` and there is not CarromChase directory in Desktop, would cp work?
4. Write **true/false** to each of the above questions in a file **ans.txt,** one line for one question in that order (will be autograded).

B. Mr.Stark wants to avoid copying a file if it already exists at the destination, as time is of utmost importance. Write a bash script **dupe.sh** which takes two arguments: source file path and destination file path and copies source file to destination file only if destination file DOES NOT exist

1. Usage: **`bash dupe.sh <source-file-path> <destination-file-path>`**

2. Print (without quotes) to stdout
   a. "Copied <source-file-path>" if the source file is copied
   b. "Not copied <source-file-path>" otherwise

3. **Assumption:** You may assume that all the directory of <destination-file-path> exists, i.e. if <destination-file-path>=`./dest/src/lib.c`, `./dest/src/` dir already exists

(Hints: cp, file existence check, if-else-block)

C. Mr.Stark realizes that some of his source files are nested deep inside directories. Especially his java files. One example is (`./dest/src/main/java/com/jarvis/ai/Main.java`). It would take all the time in the world to create the nest of directories (src, main, java, com ...) at the destination. Your job is to create a bash script **superdupe.sh** (which is an extension of **dupe.sh)** so that it can also create a nest of directories at the destination and then copy the file.

1. **Note** that this implies the assumption in Task B is no longer valid

(Hints: mkdir, touch, cp)

D. Unfortunately, Mr.Stark has not enough time to copy all the data file by file. Write a script **dupedir.sh** which takes two arguments source directory and destination directory and copies all (regular) files inside "source directory and all nested directories under source directory" to destination dir by calling **superdupe.sh** on each file.

1. Usage: **`bash dupedir.sh <source-dir> <destination-dir>`**

2. To illustrate say:
    a. source directory (dir1) has following structure (4 files in total)
        i. `dir1/`
            1. `readme1.txt`
            2. `readme2.txt`
            3. `subdir1/`
                a. `readme3.txt`
                b. `subdir2/`
                    i. `readme4.txt`
                c. `subdir3/`

    b. And destination dir (dir2) has following structure (3 files in total)
        i. `dir2/`
            1. `readme10.txt`
            2. `readme20.txt`
            3. `readme1.txt`

    c. After executing **bash dupedir.sh dir1/ dir2/**, dir2 is supposed to have the following structure (6 files in total)
        i. `dir2/`
            1. `readme10.txt`
            2. `readme20.txt`
            3. `readme1.txt`
            4. `readme2.txt`
            5. `subdir1/`
                a. `readme3.txt`
                b. `subdir2/`
                    i. `readme4.txt`

    d. i.e. readme1.txt should not be copied again, readme10.txt and readme20.txt should be untouched, and all others should be copied

3. The source dir can have any number of nested directories

4. Empty directories in source dir need not be copied

5. **<u>Observe</u>** that all the **dupedir.sh** is idempotent. That means the repeated execution of it with same arguments would produce the same result as executing it once (in the filesystem may not be in stdout).

(Hints: find, while loop, string manipulation)

3. Sling Ring

Prologue
A rogue sorcerer has created many portals in new york for illegal trades using a sling ring. As the protector of new york sanctum, it is Dr. Strange's responsibility to handle it. But he is busy at the moment fighting Thanos. He would like you to take care of it.

Problem
Each portal is a **symbolic link** (which is a link to a **target file/directory).** Your job is to create a script **symLinks.sh** which takes a directory and output filename as arguments and prints the details of each symbolic link inside "the argument directory and all nested directories under the argument directory" in the following format to the output file

- <symbolic-link-file-path> <target-of-symbolic-link> <md5sum-sum-of-target> <dir-or-not>
- <md5sum-sum-of-target> should be **@** if the target is a directory
- <dir-or-not> is 1 if the target is a directory, 0 otherwise
- The rows of the file must be **sorted** (w.r.t whole row)
- You can assume that <symbolic-link-file-path> does not have any whitespaces
- Usage: **`bash symLinks.sh <directory> <outputfile>`**

(Hints: find, exec, readlink, md5sum, sort, $() command)

4. Infinity Gauntlet

Prologue
It's been many years since Thanos (the mad titan) started on his quest of destroying half of the universe to bring balance to life. He has collected all the infinity stones in his Infinity Gauntlet. Still, he knows that Avengers will do anything in their power to stop him. So he wants to make sure that he has the optimum arrangement of infinity stones on his gauntlet. Your job is to tell him how many possible ways of arrangements exist.

Problem
The number of arrangements for n stones is **n factorial**. Your job is to write a program to calculate factorial of an integer.
For reference, factorial of a number

$\text{fact}(n) = n * \text{fact}(n-1)$ for $n > 0$
$\text{fact}(n) = 1$ for $n = 0$
$\text{fact}(n) =$ invalid for $n < 0$

A. Write a script **fact.sh** that takes an integer as an argument and prints out it's factorial. Do this task using functions and return statements. Be sure to set the exit status to 1 if the argument is a negative integer or a non-integer.

Illustrations

bash fact.sh 2
#output is
2

bash fact.sh -1
#output is
-
echo #?
#output is
1

Hint:
Functions and exit status
Use of return statements actually sets the exit code/status of the function call (the exit status/code is stored in the variable '?')
There is no direct way for a function to return a value in bash. We use the exit status of the function to return a value here as a hack.

Submit the file **fact.sh**.

B. Check the values output by your script for arguments 0, 1, 2, 3, 4, 5, 6, 7.. Where does it deviate from the supposed values and why?

Write your answer in a file **ans.txt**.

C. To deal with the above problem, we use something called **command substitution**. It is basically a workaround to be able to return any values whatsoever. The procedure is to write the function (say f1) such that it prints the return value to stdout; call the function (f1) in the form **$(<function> <arg1> <arg2>)**; use the struct $(..) in other commands in your script.

Illustrations

echo $(cat f1.txt)
# the $(..) is replaced by the output of 'cat f1.txt', the replacement is read by 'echo'
# and output to stdout

Using command substitution, write a bash script **fact_gen.sh** that works with all integers.

5. Broken Bifrost

Prologue
Heimdall, the guardian of Bifrost (a magical pipe bridge between earth and Asgard and other seven realms of the universe) is having a tough day. He needs to open Bifrost to two realms at once. The only problem, he doesn't know how. He needs your help to solve the crisis before the news reaches Odin (king of Asgard). Fortunately, Bifrost is built on Linux (at least that's what we want you to believe) and as usual bash comes to rescue.

Problem
Your job is do the following operations
1. Copy an old file into a new file using pipe operator
2. Compute the md5 hash of the old file
3. Write the computed checksum to a file so that the new file can be verified
Steps 1 and 2 have to be done simultaneously

Why
The old file might get corrupted (especially if it is large) in between copying the file and computing the checksum which results in the inconsistent checksum, hence the md5sum computation and the copy must be done simultaneously

Your script should do the following:

A. make a copy of the old file
B. compute md5sum of the new file and verify that it matches with that of the old file (after copying) (using **md5sum -c newfile.md5**)

the script should work with relative paths
newfile.md5 should be located in the directory of **newfile**
Your script should be named **verifiable_copy.sh**
Ideally, your code should have a few lines for (A) and one line for (B)

e.g.,
bash verifiable_copy.sh old_file newfile

Challenge: Try and write all of the code for (A) in a single line just using pipes and redirections
(Hints: tee, pipes, md5, string manipulations, sed)
You might have to read about sed a bit as it is not taught in class

6. God of Mischief!

Prologue
Loki (god of mischief) is known for his notorious traps. Thor is fed up and wants to teach Loki a lesson. He wants to build a master trap to outsmart Loki. But unfortunately he doesn't know much about traps. He wants your expertise on them to create one.

Problem

A. He wants to lure Loki using Tesseract. Write an AWK script **lure.awk** that takes **tesseract.csv** (which has columns named **material**, **value** and **energy** among others) as input and does the following operations

1. Select rows in which **energy** entries are greater than or equal to 2000
2. Group those rows by the **material** column
3. Then add the **value** entries in each group
4. Finally, print **material** and **value** column entries to stdout

Note: The output should be sorted wrt material.

tesseract.csv is available in Data folder
An Illustration is as follows:

```
$: awk -f lure.awk tesseract.csv (Command for execution)
Material: T1, 7792
Material: T2, 35483
Material: T3, 20871
Material: T4, 15954
Material: T5, 17146
```

(will be checked with hidden test cases as well)

B. Signals are software interrupts to tell the operating system that an important event has occurred and it has to be handled. Trap command is used for trapping these signals and handling them. The format of the trap command is as follows
trap 'command/function_name' signals.
For example **trap 'echo trap found' SIGINT** prints **trap found** when ^C is pressed.

Let us explore more about signals and traps in the following questions.

1. Does the command **kill -l** display all the supported signals by the system?

2. Can all the signals present in the system be trapped using the trap command?

3. Does trap **exit** from the process after handling a signal?

4. If the command listed for the trap is **null**, will the signal be ignored when received?

5. To reset a trap to its default, can we use either of the commands
   trap signals or **trap - signals**?

Write **true/false** to each of the above questions in a file **ans.txt,** one line for one question in that order (will be autograded).

C. Write a bash script **trap.sh** which does the following

1. Starts execution only after ^C is pressed
2. Once ^C is pressed it uses the above **lure.awk** to get the required data from **tesseract.csv**
3. Then instead of printing to stdout pipe the output to file named **loki.csv**

tesseract.csv is available in Data folder
An Illustration is as follows:

```
$: bash trap.sh tesseract.csv          (Command for execution)
^C                                     (loki.csv should be created after pressing ^C)
$: cat loki.csv
Material: T1, 7792
Material: T2, 35483
Material: T3, 20871
Material: T4, 15954
Material: T5, 17146
```

(Hints: use trap command, while loop, sleep)

General Instructions

- Make sure you know what you write, you might be asked to explain your code thoroughly at a later point in time
- Grading is done automatically (we grade bash scripts with bash scripts), so please make sure you stick to naming instructions
    - Especially some of the tasks require you to answer in txt files, make sure you only write what is asked, nothing more nothing less
- The deadline for this lab is on **Sun 29th July 11:55 PM.**

Submission Guidelines

If you have any assumptions you feel you need to make, kindly mention them in an **assumptions.txt** file
After creating your directory, package it into a tarball (**`<rollno>-outlab2.tar.gz`**) and submit it on Moodle.

- `<rollno>-outlab2/`
    - `Task1/`
        - `A/`
            - `isLeapYear.sh`
        - `B/`
            - `yearOfCrash.txt`
    - `Task2/`
        - `A/`
            - `ans.txt`
        - `B/`
            - `dupe.sh`
        - `C/`
            - `superdupe.sh`
        - `D/`
            - `dupedir.sh`
    - `Task3/`
        - `symLinks.sh`
    - `Task4/`
        - `A/`
            - `fact.sh`
        - `B/`
            - `ans.txt`
        - `C/`
            - `fact_gen.sh`
    - `Task5/`
        - `verifiable_copy.sh`
    - `Task6/`
        - `A/`
            - `lure.awk`
        - `B/`
            - `ans.txt`
        - `C/`
            - `trap.sh`
    - `assumptions.txt`