CS 251 Outlab 1 : Unix CLI and Bash

Please refer to the general instructions and submission guidelines at the end of this document before submitting.
General update: **DO NOT use sed or awk for any question in this outlab.**

Task 1 - Prime Curious! (15 Marks)

Primality test is an algorithm for determining whether an input number is prime or not. Among many other fields of mathematics, it is very widely used for cryptography. By now you must have done primality test ample number of times in C++ but now we will do the same problem using bash script.

Subtasks

A. Write a bash script named **isPrimeYear.sh** , which when given a year (a positive integer) as an argument, prints (*without quotes*) to stdout
   a. "Prime Year!" if the year is a prime number.
   b. "Not a prime year." otherwise.
   c. "Invalid argument!" if the input is anything other than a positive integer.

   Usage: **bash isPrimeYear.sh <year>**

   Note: You can refer this link to know how to check if an argument is a number or not.
   (Hints: logical operators, if-else-block, while loop) **(12 Marks)**

B. Modify the above-written script (done locally and need not be submitted) to find the number wof prime years we had so far i.e., from start of CE to present year ! Write the answer (just the number and nothing else) in file **numberOfPrimeYears.txt** (will be autograded) **(3 Marks)**

Note : No need to submit the modified script, only answer to subtask A needs to be put in the isPrimeYear.sh.

-

Task 2 - The Answer To Life and Everything ! (15 Marks)

Unix systems have some special files that can be found in the /dev folder.
/dev/null
/dev/random
/dev/urandom
etc.
These have great utility, you can read up here before continuing.

Subtasks

A. Write a bash script **randomNumber.sh** which when given n as an argument generates an n digit number using shell commands you've learnt (and more!) along with the file **/dev/urandom**.
   Note that your number should not have leading zeros. 007 is not an acceptable 3 digit number. (no Mr. Bond, sorry)
B. You were told that the answer to life and everything is **42.** Write a bash script **findTheAnswer.sh** which uses the script from subtask A (remember relative path) to generate 2 digit random numbers until 42 is generated, and pipeline the output to **howFarFromTruth.txt.** You must print all numbers generated in a new line, and also output 42 when it's found.
C. Using the script in part B, write a bash script **expectedDistance.sh** to find the average iterations required before 42 is generated over K attempts, where K is a command line argument.

-

Task 3 - Help the TA's! (20 Marks)

You were all given a form to fill the team/group formation for evaluating the outlabs and project. However there are some students who have been a part of multiple groups and we TA's of the class are having a tough time to find those students.

We did not take into account that we have to write a bash script to group the teams (so that all the roll numbers come under a single column) and a simple Google form/excel input doesn't really cut it for us. We are now in a soup and need your help desperately.

Write a bash script **helpGroupTheCsv.sh** which take two arguments : 'an input csv (comma-separated value) file' and a file to save the output after grouping the students as shown below.

Input File Format :

```
team1,student1,student1_rollno,student2,student2_rollno,student3,student3_rollno
team2,student1,student1_rollno,student2,student2_rollno,student3,student3_rollno
team3,student1,student1_rollno,student2,student2_rollno,student3,student3_rollno
team4,student1,student1_rollno,student2,student2_rollno,student3,student3_rollno
.
.
```

Output File Format :

```
team,student,student_rollno
team,student,student_rollno
team,student,student_rollno
.
.
```

Note that the **output must also be sorted on the column number 3** (student_rollno) to identify students enrolled in multiple teams.

Note :

1. By doing such sort, the students with the same team name may or may not be together, but we can find students who enrolled multiple times.
2. Also teams containing containing such multiple enrolled students should be displayed in case-insensitive sort order of their team names.
3. Refer the sample test case provided for more clarity.

Usage: **./helpGroupTheCsv.sh <inputfilename> <outputfilename>**

This problem statement shall be graded automatically using scripts, so please ensure your output is properly saved in the output file and is sorted based on student roll number (column 3).
(Hints: sort, cut) (**20 marks**)

-

Task 4 - Web Mining (10 marks + 15 marks)

Web mining is the application of data mining techniques to discover patterns from the World Wide Web. As the name proposes, this is information gathered by mining the web. It makes utilization of automated apparatuses to reveal and extricate data from servers and web reports, and it permits organizations to get to both organized and unstructured information from browser activities, server logs, website and link structure, page content and different sources**.**

Important tool for this question : wget

Subtasks

A. The Portable Document Format (PDF) is a file format developed (mainly by Adobe Systems) in the 1990s to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems. PDF documents are usually meant to be read by humans. But sometimes to do complex queries a computer is better. Today we'll learn how to count the number of words from PDFs.

Your goal is to write a **bash** script to get the pdf from an online link and count the number of words in it.

Write a script, **pdfWordCounter.sh** (Usage: **./pdfWordCounter.sh <URL> <Word>**) which

a. Downloads the PDF
b. Converts the pdf file into text format and prints the number of occurrences of the given word (case insensitive and should not match with a substring) to stdout using **pdftotext** command and **grep** command.
   Eg. break does not match with breakfast
c. Deletes the pdf file that was generated
d. **Note** that your script should only generate one pdf file during execution and then delete it. A text file should not be created even temporarily while doing **b** step. Only output on the terminal should be the number of occurrences of the given word.
   (Hints: grep, pdftotext)

(10 marks)

B. Write a script **downloadContents.sh** that performs the following tasks
   Usage: **./downloadContents.sh <url> <directory_path>**

   a. Given a url as 1st argument, recursive download its contents and follow all links **only** along those webpages that exist on the same domain (example on facebook.com, you won't follow a link to an external advertisement from, say, Spicejet).
      Convert all links in the html files that point to files you have just downloaded, whether they are relative or absolute. (Hint: wget is extremely powerful, see here)
      Download all files in a folder whose path is given as 2nd argument to the script. **(5 marks)**
   b. Use the **tree** command to store the directories content in JSON format (read more about JSON here). Store this object in a file called **urlReport.json (2 marks)**
   c. Print the md5sum of this newly made file. **(2 marks)**
   d. Find the number of '{' (open curly braces) in this file and print this on a new line. **(1 mark)**
   e. If there is a process with this process id (answer from part d) in the computer, print the name of this process (as mentioned in the COMMAND column of ps aux output), else if there's no such process print "No such process" without quotes (print on new line). Use the ps command (with the write arguments). ps aux for e.g. lists all processes in your system. **(5 marks)**
      A sample download has been provided in testcases tar (on moodle).
      Last updated - 15:22 4/08

   Output Format : (everything on a new line and no spaces anywhere)

      <md5sum of the json file>
      <number of { >
      <name of process>

(Hints: md5sum, grep, ps)

(15 marks)

-

Task 5 - Solve the mystery! (25 marks)

Prologue

Sherlock Holmes and Dr.Watson are trying to solve a murder mystery. After examining the evidence keenly, Holmes remembers that a similar case happened a few years ago. So he gives Watson a few keywords and asks him to **search** the past records to get any clue regarding the case. As there are many files in the database, Watson cannot manually possibly search in all the files. So as an expert in bash, you have got a chance to show your expertise and impress him.

Problem

You have been provided with the past records, **Data** (folder present in **Task 5**) having the information about the cases dealt in the past by Holmes. Your goal is to write a bash script **recursiveSearch.sh** which takes as arguments the keywords to be searched in the text files present in Data folder and print the relative path of the file, line number and the text in the line **containing all the given keywords** to stdout.

Sub-tasks

(5 marks)

A. Print the usage of the script (**Usage: ./recursiveSearch.sh <words-to-search>**) if no argument is provided, set the exit status to 1 and exit.

An Illustration is as follows:

$: **./recursiveSearch.sh** (No arguments provided)

Usage: ./recursiveSearch.sh <words-to-search>

$: **echo $?**

1

(20 marks)

B. The script should find the lines containing **all the keywords** (your search must be case insensitive) from all the text files present in **Data/** folder **(**note that you have to recursively search the files present in the subdirectories of Data/ as well) and stdout the **path of the file** **(**relative to the current directory), **line number** and **text in the line** and represent it as following:

path of the file:line number:**text in the line** (no space before and after the : symbol)

(Note: If **and** is the only keyword then a line containing **hand** should not be printed, i.e., a line should be printed only if it explicitly contains **and** as a word in it.)
(hint: grep)

An illustration is as follows:

$: **./recursiveSearch.sh "watson" "Holmes"**
Data/folder1/folder3/file2.txt:88:"Pray take a seat," said Holmes. "This is my friend and colleague, Dr. Watson, who is occasionally good enough to help me in my cases. Whom have I the honour to address?" (This is one line)
Data/folder2/file5.txt:88:"Pray take a seat," said Holmes. "This is my friend and colleague, Dr. Watson, who is occasionally good enough to help me in my cases. Whom have I the honour to address?" (This is one line)

(Note : The above output is actually just two lines despite the apparent wrap due to large text)

Test cases

- There are four test cases testcase1.sh - testcase4.sh in Task5 directory provided.
- Put your **recursiveSearch.sh** in Task5 directory provided and execute each of them to test your script.
- The difference between expected output and your output will be displayed when you execute a test case
- You have to put your **recursiveSearch.sh** according to submission guidelines after testing

Note

- You can assume that script **recursiveSearch.sh** and **Data** folder will be in **same** directory while we execute you script. Your script should **not** generate any **temporary files** to read/write during the execution.
- Don't worry about the order in which the lines display in the output, we will anyway sort the output while checking.
- We will run your script against hidden test cases.

-

General Instructions

- Make sure you know what you write, you might be asked to explain your code at a later point in time.
- The submission will be graded automatically, so stick to the naming conventions strictly.
- The deadline for this lab is **Tuesday, 6th August, 11:55 PM.**

Submission Instructions

After creating your directory, package it into a tarball
`<rollno1>-<rollno2>-<rollno3>-outlab1.tar.gz` in ascending order.
The roll numbers should be in ascending order (hint: sort)
Submit once only per team from the moodle account of smallest roll number.
The directory structure should be as follows (nothing more nothing less)

```
<rollno1>-<rollno2>-<rollno3>-outlab1/
├── Task1
│   ├── isPrimeYear.sh
│   └── numberOfPrimeYears.txt
├── Task2
│   ├── A
│   │   └── randomNumber.sh
│   ├── B
│   │   └── findTheAnswer.sh
│   └── C
│       └── expectedDistance.sh
├── Task3
│   └── helpGroupTheCsv.sh
│
├── Task4
│   ├── A
│   │   └── pdfWordCounter.sh
│   └── B
│       └── downloadContents.sh
└── Task5
    └── recursiveSearch.sh
```