CS 251 Quiz 1 (Bash & Unix)

Please refer to submission guidelines at the end of this document before submitting

Task 1 - PDF Scraping

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 scrape data from PDFs and extract useful information.
Here is the PDF available online containing information regarding second year courses offered to the CSE batch of 1999-2003. Your goal is to write a **bash** script to scrape the PDF and extract the details like Course Code, Course Name, Total Credits and the Instructor.

Sub-tasks(Sample output files are available in p1/)

(2 marks)

A. Write a script, **pdfScraper.sh** (Usage: **./pdfScraper.sh <URL>**) which
   a. Downloads the PDF
   b. Converts the pdf file into text format and prints text to stdout using **pdftotext** command
   c. Deletes the pdf file that was generated
   d. **Note** that you 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.

(8 marks)

B. Write a script, **courseExtractor.sh** (Usage: **./courseExtractor.sh <URL>**) which extracts the **Course Code**, **Course Name**, **Total Credits** and the **Instructor** of respective courses from the given URL and prints to stdout in the following format:

   Course Code | Course Name | Total Credits | Instructor
   CS xyz | <Course_Name> | <Credits> | <Name_of_the_course_instructor>
   CS abc | <Course_Name> | <Credits> | <Name_of_the_course_instructor>
   ……
   ……

   Note
   - ❏ Use " | " as the delimiter
   - ❏ ALL the **Course Codes** present in the PDF should be stdout in the format shown (in order same as in PDF)
   - ❏ Run the code as '**./courseExtractor.sh <URL>**' and it should stdout the required data
   - ❏ You script should only stdout the required output and delete all the intermediate files generated (including the PDF and the .txt file)
   - ❏ Hidden test case PDFs might have multiple pages but will have the same format

P.S. - This is a work of fiction. Course codes, names, description, instructor names and any other incidents are either the products of the author's imagination or used in a fictitious manner. Any resemblance to actual persons in the input data, living or dead, or actual events is purely coincidental.

Task 2 - Cryptography

Prologue

A conspiracy is being planned to assassinate Caesar by his own Senate Members. His only chance of staying alive is to decode '**TFEXIRKLCRKZFEJ WFI KYVV YRKY JRMVU TRVJRI**', which is known to be a form of **Caesar Cipher**. But he doesn't has the expertise to decode it. So now the onus of saving him is on you, for you are known to be the best mathematician in the Roman Republic. It's time for you to prove your valor.

Problem

The **Caesar Cipher** shifts all the letters in a piece of text by a certain number of places. The key for this cipher is a letter which represents the number of place for the shift. So, for example, a key D (4th letter) means "shift 3 places" and a key M (13th letter) means "shift 12 places". Note that a key A means "do not shift" and a key Z can either mean "shift 25 places" or "shift one place backwards".
For example, the word "CAESAR" with a shift P becomes "RPTHPG".
Caesar himself used 3 as a key for protecting messages of military significance however the key to this cipher is unknown and hence Caesar has come to you for help.

Sub-tasks(Sample output files are available in p2/)

(8 marks)

A. Write a script **saveCaesar.sh** to find the key of the given cipher. The structure of your script should be as following:
    a. Input the cipher to the script as a string (with spaces preserved) using the **read** command
    b. Increment the characters of the input string one-by-one maintaining a **key** variable alongside and look for meaningful words in the stdout output (Hint: use '*tr*' command)
    c. stdout for saveCaesar.sh should look like: (Also have a look at the sample input-output available in folder p2/)
    A <decoded_text_with_0_shifts>
    B <decoded_text_with_1_shift>
    C <decoded_text_with_2_shifts>
    ...
    Z <decoded_text_with_25_shifts>
    d. Note down the words which you find meaningful and the corresponding **key** value and save it in **decodedCipher.txt** in the following format: (a two line output)
    key : <key>
    code : <decoded_cipher>

(2 marks)

B. Having saved Caesar now you plan to retaliate back to decimate your enemies. So write a script **retaliation.sh** for sending a protected message '**KILL ALL**' to your military forces which can have variable key. The script structure is as following:
    a. The script takes the **key** value as a command line argument
    b. The message to be encoded is again taken from stdin using the **read** command (similar to the A. part)
    c. The output should simply be the encoded message encrypted using the key value and print that encrypted message to stdout

Task 3 - Sherlock

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 **p3/**) 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

(2 marks)

- 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

(8 marks)

- 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 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 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 also be printed, as **hand** also contains **and** in it. i.e., you only need to find keyword as substring of the line)
  (hint: check grep command)

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
- Put your **recursiveSearch.sh** in p3 dir 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 a hidden test cases.

Submission Guidelines

- If you have any assumptions you feel you need to make, please mention them in **assumptions.txt** (You can't just make any assumption and proceed to get full credit. It's better if you ask TAs before you make a non-trivial assumption)
- After creating your directory, package it into a tarball (**<rollno>-quiz1.tar.gz**) and submit it on Moodle
- The tar should contain the following files in the following directory structure:

```
<rollno>-quiz1/
|___Task1/
|______A/
|_________pdfScraper.sh
|______B/
|_________courseExtractor.sh
|___Task2/
|______A/
|_________saveCaesar.sh
|_________decodedCipher.txt
|______B/
|_________retaliation.sh
|___Task3/
|______recursiveSearch.sh
|___assumptions.txt
```