CSE/ISE 337: Scripting Languages

Stony Brook University

Programming Assignment #3

Fall 2020

Assignment Due: Monday, Nov 16th, 2020, by 11:59 PM (EST)

Learning Outcomes

After completion of this programming project, you should be able:

- Design and implement scripts in Bash
- Work efficiently with command-line tools in UNIX

Instructions

- Take time to read the problem descriptions carefully. The devil is in the details.
- Pay attention to the script names and error messages.
- Only display in console if explicitly asked to do so.
- Zip all files and submit the zipped distribution (**lastname_firstname.zip**). The zip file should contain:
  - prog1.sh
  - prog2.sh
  - prog3.sh
  - prog4.sh
  - prog5.sh
- Create your own test cases to thoroughly test your code.
- All scripts must take input from the command line (if any).

Problem 1 (20 points)

Imagine that we have written numerous C files (.c files) in a directory D and compiled then to object files (.o files). Now, we want to ship directory D to a client who wants to run the source code. However, we cannot release our source code to protect intellectual property. Hence, we will have to move our source files to another directory and keep the object files in the same directory. However, we need to ensure the following when we are moving source files.

1. While moving files to the destination directory, the directory structure should be maintained. For example, consider that we are moving all C files from directory *project* to a directory *project_src_bkup*. Let us say that the directory *project* has C source files in *project/*, *project/subProj1, project/subProj1/subsubProj1*, and *project/subProj2*. The destination directory *project_src_bkup* should have the corresponding source files in the same directory structure, that is, *project_src_bkup/*, *project_src_bkup/subProj1, project_src_bkup/subProj1/subsubProj1*, and *project_src_bkup/subProj2*.
2. There may be some directories in the directory that we are moving files from that contain more than 3 C files. Moving files from such directories should be done in an interactive manner. Since we don't want to lose a large number of files, all files being moved should be first displayed to the user. If the user confirms by typing 'y' or 'Y', then the files should be moved; otherwise, the files should be skipped.
3. If the destination directory already exists, then it should not be re-created. However, if it does not exist, then it should be created. Subdirectories in the destination directory may be re-created even if they exist.
4. If the source directory does not exist, then exit with status 0 and display a message "<src-dir> not found"

Task

Write a script to automate the process described above. Your script should take 2 inputs from the command line. Any temporary files created during script execution should be deleted after the script finishes execution.

Parsing the arguments
The 1st input will be the directory containing the source files and the 2nd input should be the destination directory. If either of the inputs are missing or if more than the required inputs are provided, then the script should display the message "src and dest dirs missing".

Script name
Name your shell script called *prog1.sh*

Problem 2 (20 points)

Imagine that we are given a data file, where each line is a sequence of numbers. The numbers in each line are separated by either 1 of the 3 characters – comma(,), semicolon(;) or colon(:). Further, there is no limit to the number of numbers in each line. Here is an example of a sample data file:

1;2;3;4;5
11:4:23:12
18,4,17,13,21,19,25

As can be seen from the example, you could imagine a data file to be arranged in rows and columns where the columns are separated by either comma, semicolon, or colon.

Task
Write a script that will take a data file (as described above) as input and will compute the sum of each column in the data file. The sum of each column should be written to an output file which will also be provided as input to the script. All inputs to the script will be provided from the command line. The output file should be written in the format *Col <n> : sum*, that is, each line in the output file should have the column and its corresponding sum. For example, based on the data file shown above, the output file should look like the following:

Col 1 : 30
Col 2 : 10
Col 3 : 43
Col 4 : 29
Col 5 : 26

Col 6 : 19
Col 7 : 25

Parsing the arguments
If a non-existent data file is provided as input to the script, then the script should report "<filename> not found". If the output file provided as input does not exist, then it should be created. If an output file provided as input exists, then it should be over-written with the new output of the script. You can assume that a data file will always have the format specified above. If either the data file or the output file path is not provided, then display a message "data file or output file not found"

Script name
Name your shell script *prog2.sh*

Problem 3 (20 points)

Imagine an exam with N parts. The final score of the exam takes into account all N parts. Each part has a weight W_N associated with it. If Q_N is the score a student receives in part N, then the weighted average of the exam for that student will be $\left(\sum_{i=1}^{N} Wi * Qi\right)/\left(\sum_{i=1}^{N} Wi\right)$.
Assume that you have recorded the ID of each student who took the exam along with the score that student got for each part in a file. A sample file might look as follows:

ID,Q1,Q2,Q3,Q4,Q5
101,8,6,9,4,4
102,9,9,9,10,4
103,5,6,2,4,4
104,1,2,2,1,4
105,10,10,10,9,4
106,10,10,10,10,4
107,7,7,8,9,4
108,5,6,5,6,5
109,10,9,9,4,4

We need to compute the weighted average of an exam across all students. A weighted average for the sample file (shown above) will be 5, assuming that the weights 1,2,3,4, and 5 were assigned to Q1,Q2,Q3,Q4, and Q5. You can assume that that *all scores will be integers* and the final result should be *rounded off to the nearest integer towards 0* (e.g., 5.6 should be rounded off to 5).

Task

Write a script that takes as input a score file in the format shown above as the 1st argument along with a sequence of arguments for weights. *The script should display the weighted average rounded off to the nearest integer towards 0 on the console*.

Parsing the arguments

The 1st argument will always be the data file with the score and ID information. The 1st weight argument should be considered as the weight for the first part of the exam, the 2nd weight argument as the weight of the 2nd part, and so on and so forth up to N. If the no. of weight arguments provided is less than N, then assume that the remaining parts of the exam have weight 1. If the no. of weight arguments provided is more than N, then ignore the additional weight arguments. The first N weight arguments should be considered the weight for the N parts of the exam.

If an input data file is not provided as 1st argument, then display an error message "missing data file".

Script name

Name your shell script *prog3.sh*

Examples

Assuming that the sample data file shown above is called *data.txt*, then the script can be invoked as follows:

```
$ ./prog3.sh data.txt
```

>> should display weighted average of all students who took the exam, where every weight is 1.

$ ./prog3.sh data.txt 1 2 3
>> should display weighted average of all students who took the exam, where Q1 has weight Q2 has weight 2, Q3 has weight 3, the remaining weights are 1.
$ ./prog3.sh data.txt 1 2 3 4 5
>> should display weighted average of all students who took the exam, where Q1-Q5 are assigned weights 1-5 respectively.

$ ./prog3.sh data.txt 1 2 3 4 5 9 10
>> should display weighted average of all students who took the exam, where Q1-Q5 are assigned weights 1-5 respectively.

Problem 4 (20 points)

Imagine a scenario where you have the scores of each student in an exam and you are supposed to assign an appropriate letter grade based on the score.
Further, assume that you have collected the scores of each student in a file. Each such file will contain the student ID and a breakdown of the scores for each question in the exam. Here is an example of a sample file for a student:

ID,Q1,Q2,Q3,Q4,Q5
102,9,9,9,10,10

Each student will have a corresponding file in the format shown above. You can assume that an exam will always have 5 questions worth 10 points each.

Task
Write a script that will take a directory of score files as input, compute the score received by a student as a percentage, and display the student ID and her corresponding letter-grade in the console in the format ID:<letter-grade>. The table below shows the percentage range and the corresponding letter grade:

Percentage Score Range	Letter Grade
93-100	A
80-92	B
65-79	C
< 65	D

Parsing the arguments
The scores directory is the only argument to the script. If it is not provided, then display a message "score directory missing". If the argument provided is not a valid directory, then display "<dirname> is not a directory".

Script name
Name your shell script *prog4.sh*

Example
Assume that the scores directory provided as input has the following files:

```
==> data/prob4-score1.txt <==
ID,Q1,Q2,Q3,Q4,Q5
101,8,6,9,4,10

==> data/prob4-score2.txt <==
ID,Q1,Q2,Q3,Q4,Q5
102,9,9,9,10,10

==> data/prob4-score3.txt <==
ID,Q1,Q2,Q3,Q4,Q5
103,5,6,2,4,6

==> data/prob4-score4.txt <==
ID,Q1,Q2,Q3,Q4,Q5
101,10,10,10,10,10
```

The script should display the following in the console:

```
101:C
102:A
103:D
101:A
```

Problem 5 (20 points)

An automated spell checker is a piece of software that can automatically detect spelling mistakes in a file. Imagine that you have to develop a specific kind of spell checker that can be used to verify the spelling of 4 letter words. Your spell checker will rely on a dictionary that contains all possible four letter words.

Task
Write a script that takes a file as input and a dictionary file as input and displays all 4-letter words that have been spelt incorrectly in the console. Assume that the dictionary file is a newline separated list of all possible 4 letter words, that is, each word will be on a newline.

Parsing the arguments
The 1st argument is the file whose spelling needs to be checked and the 2nd argument is a dictionary. If neither is provided, then display a message "input file and dictionary missing". If an invalid filename is provided as 1st argument, then display a message "<filename> is not a file". Same for the 2nd argument.

Script name
Name your shell script *prog5.sh*

Example
Assume that we want to check the spelling of 4-letter words of a file called *prob5-sample.txt*. This file has the following text:

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.
The Python interpreter and the extensive standard library are freely available in source or binary forc for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

In the above text the word *forc* should have been spelt as *form*. Hence, on executing the script as follows:

$ ./prog5.sh prob5-sample.txt dictionary.txt

should display *forc* in the console.