Computer Science & Software Engineering

CITS4407 OPEN SOURCE TOOLS AND SCRIPTING

Assignment 1 2023

Submission deadline: 11:59pm, Monday 24 April 2023.
Value: 20% of CITS4407.
To be done individually.

This assignment will involve creating a Shell script, which will use Unix tools and/or call other Shell scripts. The top-level script has been given a name. Please make sure you use the specified name, as that is the name which the testing software will use to test your script.

You need to package the script or scripts into a single submission consisting of a zip file, and submit the zip file via *cssubmit*. No other method of submission will be accepted.

Kaggle and the Catalogue of US Cybersecurity Breaches

Kaggle is a remarkable web-based, data science resource which contains a a huge number of different data sets and tutorials on tools. (Highly recommended.) One particular data set is the Catalogue of Cyber Security Breaches (US data, so the states listed are the two letter US state codes).

I have downloaded for you the file and done some data cleaning, e.g. filling in missing values in the year column from the date of breach column. You can find the tidied file as a tab-separated, i.e. `.tsv`, file Cyber_Security_Breaches.tsv. This is similar to the the more familar `.csv` file, except that a <TAB> character is used to separate items in a line, rather than comma, which means it is much easier to deal with data that has embedded commas.

Before you go any further, I suggest you download `Cyber_Security_Breaches.tsv` and have a look at the data found in each column. (Actually, this is good general advice: Each time you start work with a new data-set, have a look at the data, in part or whole, first, to get an idea of what is expected in each column. What usually happens next is a data-cleaning step, but that has been done for you here.)

The top level program must be called **`cyber_breaches`**, and should expect two arguments each time it is called: the name of the `.csv` data file being used for this analysis, followed by a command.

cyber_breaches commands

There are four sorts of commands, one of which must appear as the second argument for each call to `cyber_breaches`.

- `maxstate`
  The program should report the code for the state that has the largest number of incidents across all years, and the corresponding count. If there are more than one such state, just report one of them

- `maxyear`
  Report the year with the greatest number of incidences across all the states, and the corresponding count. If there are more than one such year, just report one of them

- A two letter state code
  For the named state, eport the year with the maximum number of incidents, and the count. (If more than one, any one of them.)

- A four digit year
  For the named year, report the state with the maximum number of incidents for that year, and the count. (If more than one, any one of them.)

Some sample queries

Here is a sample session:

```
% cyber_breaches Cyber_Security_Breaches.tsv maxstate
State with greatest number of incidents is: CA with count 113

% cyber_breaches Cyber_Security_Breaches.tsv maxyear
Year with greatest number of incidents is: 2013 with count 252

% cyber_breaches Cyber_Security_Breaches.tsv 2010
State with greatest number of incidents for 2010 is in CA with count 15

% cyber_breaches Cyber_Security_Breaches.tsv TX
Year with greatest number of incidents for TX is in 2013 with count 17

% cyber_breaches Cyber_Security_Breaches.tsv maxnear
The max commands are either maxstate or maxyear
```

Your submission will be tested automatically against a range of seen, and unseen example. However, a human marker will be assessing you program's outputs for the range of tests, which means that the output format your program uses does not much matter for the auto-testing. However, do be aware of readability of your code and the program outputs. (See below for discussion of Style.)

Marking criteria

The program will be marked out of 20. Marking of programs will primarily be on the basis of how the programs deal with different types of input, both input that conforms to expectations - similar to the examples - and error state input that anti-bugging should catch. For example, typos, in this case of a file name, are common. Your program should deal gracefully with all error states. First of all, by catching error inputs, you ensure that ridiculous output does not result from erroneous input. In short, stop silly things from happening. However, beyond that, error messages need to be as informative as possible, so users know what went wrong. You therefore need to consider the ways users inputs may not conform to what your system is expecting and add testing to catch those issues.

The remaining 20% will be for style/maintainability. Programs are written as much for human as for computers. As such, it is important that your code be readable and mantainable. Similarly, outputs should aim to be informative (but ever verbose).

Style Rubric

Much of this has been discussed in classes, but includes comments, meaningful variable names for significant variables (i.e. not throw away variables such as loop variables), and sensible anti-bugging. It also includes making sure your program removes any temporary files that were created along the way.

For the style/maintainability mark, the rubric is:

$0 \le x < 1$	Gibberish, impossible to understand
$1 \le x < 2$	Style is really poor, but can see where the train of thought may be heading
$2 \le x < 3$	Style is acceptable with some lapses
$3 \le x < 4$	Style is good or very good, with small lapses
4	Excellent style, really easy to read and follow

Note: Automated testing is being used so that all submitted programs are being tested the same way. Sometimes it happens that there is one mistake in a program that means that no tests are passed. If the marker is able to spot the cause and fix it readily, then they are allowed to do that and your - now fixed - program will score whatever it scores from the tests, minus 2 marks, because other students will not have had the benefit of marker intervention. That's way better than getting zero for the run-time tess, right? (On the other hand, if the bug is too hard to fix, the marker needs to move on to other submissions.)

Department of Computer Science & Software Engineering
The University of Western Australia
Last modified: 20 March 2023
Modified By: Michael Wise