Computer Science & Software Engineering

CITS2003/CITS4407 OPEN SOURCE TOOLS AND SCRIPTING

Home
Weekly Schedule
Labs
Resources

Interesting Things
HelpOSTS
Lecture/Lab Times
CITS2003
Lecture/Lab Times
CITS4407
Unit Outline

Assignment 2 2023

Submission deadline: 11:59pm, Monday 22 May
Value: 20% of CITS2003/CITS4407.
To be done individually.

This assignment will involve creating two Bash Shell scripts, which will use Unix tools, e.g. Sed, Awk, and/or calls to other Bash Shell scripts. The top level scripts are to be called `preprocess` and `breaches_per_month`. Please make sure you use the specified names, as those are the names which the testing software will use to test your script. If you choose to use Git (see below), please also include the .git folder.

You need to package the scripts (and .git folder) into a single submission consisting of a directory that has been compressed with zip or tar, and submit the zip/tar file via *cssubmit*. No other method of submission will be accepted.

Revisiting Kaggle Catalogue of US Cybersecurity Breaches

This assignment will make use of the Catalogue of US Cybersecurity Breaches data that you first looked at in Assignment 1, although it will be a different version of that data-file. There are two parts to this assignment. The first will involve data-cleaning, which is a hugely important first step in almost every data analysis task. In the second part you will use cleaned data to analyse the distribution of incidents across months.

Data Cleaning

The Bash Shell script at the front of the data-cleaning task must be called **`preprocess`**, and will be given the name of a data-file as its only argument. The primary data-file you will be using is Cyber_Security_Breaches_noym.tsv, though other data-files will also be tested. You can assume the interpretation of the fields will not vary.

`preprocess` needs to achieve the following aims:

- Perform the usual anti-bugging/sanity-checking of the fields, particularly those which are structured, i.e. not free text.
- Add a month column (containing numbers 1..12) and a year field (containing 4-digit years), in that order, after the final column (currently Type_of_Breach). The data for these should be taken from the fourth (Date_of_Breach) column of the input data-file, and if there is a range of dates – indicated by a pair of dates separated by a hyphen – please use the first of these (the start date). Please bear in in mind that this is US data, so therefore uses the US date format: Month/Day/Year.
- The Type of Breach column has the primary breach-type, sometimes followed by alternative interpretations, separated from the primary breach by a comma or

slash "/". To make the breach-type more uniform remove everything in field after the first comma or slash.
- The Location of Breached Information and the Summary columns can be dropped.
- Any rows with erroneous data should be dropped.
- New: The output of `preprocess` should be sent to standard output.

To give you something to work toward, and as input for the second part of the assignment, the cleaned file corresponding to Cyber_Security_Breaches_noym.tsv is Cyber_Security_Breaches_clean.tsv (although I could have use any file name).

Data Analysis

You undertook some analyses of the Cyber Breaches data in Assignment 1; there are many more that could be done, e.g. to answer the question, "Is the nature of the breaches changing over time, and if so, how?"

The particular analysis you are asked to do for this assignment is to see if there is a pattern to the breaches across months, for the several years that are covered by the data-file. The Bash Shell script at the head of this program must be called **`breaches_per_month`**; it too takes a single argument: the name of the (cleaned!) data-file. The program should compute the total number of incidents per month for each of the months, and then compute the median and median absolute deviation (MAD) across the 12 months. (Median and MAD are more robust measure of centrality and spread than mean and standard-deviation when you cannot be sure that the data is normally distributed.)

The values for the median and MAD should be printed to standard output. When you have the median and MAD, you are to print a table of the months, where, against each month you list the number of incidents and then either "++", if the count is 1 median-absolute-deviation above the median (or more); "--", if the count is 1 median-absolute-deviation below the median (or less). If the count is within 1 median-absolute-deviation of the median don't add anything. For example, the count for January may look like:

```
Jan     100     --
```

or

```
Jan         300     ++
```

or

```
Jan         200
```

Testing

As with Assignment 1, 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. Of the 20 marks, 15 will be awarded based on how the programs deal with different types of input, both input that conforms to expectations and error state input that anti-bugging should catch. However, beyond that, error messages need to be as informative as possible. 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. You can assume that cleaned data-file submitted to `breaches_per_month` is, in fact, clean.

Of the remaining 5 marks, there will be 1 mark for be for appropriate use of Git. This means you must have multiple commits at different times with appropriate/relevant commit messages. One commit at the start and one at the end of the project is not sufficient.

The final 4 marks 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:

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

Hints

- The median and MAD computations require a prior sorting step. Standard Awk does not have an inbuilt sort function, but Gawk does: `asort`, which you may care to investigate.
- Before you get to work on `preprocess`, I recommend you look at the range of ways the dates have been expressed in the data-file, even though all use the US date format.

Department of Computer Science & Software Engineering
The University of Western Australia
Last modified: 16 May 2023
Modified By: Michael Wise

THE UNIVERSITY OF WESTERN AUSTRALIA