Program 1 – CS 344

This assignment asks you to write a bash shell script to compute statistics. The purpose is to get you familiar with the Unix shell, shell programming, Unix utilities, standard input, output, and error, pipelines, process ids, exit values, and signals.

What you're going to submit is your script, called stats.

Overview

In this assignment you will write a bash shell script to calculate mean averages and medians from an input file of numbers. This is the sort of calculation I might do when figuring out the grades for this course. The input file will have whole number values separated by tabs, and each line of this file will have the same number of values. (For example, each row might be the scores of a student on assignments.) Your script should be able to calculate the mean and median across the rows (like I might do to calculate an individual student's course grade) or down the columns (like I might do to find the average score on an assignment).

You will probably need commands like these, so please read up on them: while, cat, read, expr, cut, head, tail, wc, and sort.

Your script must be called "stats". The general format of the stats command is

```
stats {-rows|-cols} [input_file]
```

Note that when things are in curly braces separated by a vertical bar, it means you should choose one of the things; here for example, you must choose either -rows or -cols. The option -rows calculates the mean and median across the rows; the option -cols calculates the mean and median down the columns. When things are in square braces it means they are optional; you can include them or not, as you choose. If you specify an input_file the data is read from that file; otherwise, it is read from standard input.

Here is a sample run of what your script might return, using an input file called test_file (this particular one can be downloaded here, note that in Windows, the newline characters may not display as newlines. Move this to your UNIX account, without opening and saving it in Windows, and then cat it out: you'll see the newlines there):

```
% cat test_file

 1       1       1       1       1
 9       3       4       5       5
 6       7       8       9       7
 3       6       8       9       1
 3       4       2       1       4
 6       4       4       7       7

% stats -rows test_file
 Average Median
```

```
 1       1
 5       5
 7       7
 5       6
 3       3
 6       6

% cat test_file | stats –c
 Averages:
 5       4       5       5       4
 Medians:
 6       4       4       7       5

% echo $?
 0

% stats
 Usage: stats {-rows|-cols} [file]

% stats -r test_file nya-nya-nya
 Usage: stats {-rows|-cols} [file]

% stats -both test_file
 Usage: stats {-rows|-cols} [file]

% chmod -r test_file

% stats -columns test_file
 stats: cannot read test_file

% stats -columns no_such_file
 stats: cannot read no_such_file

% echo $?
 1
```

Specifications

You must check for the right number and format of arguments to stats. You should allow users to abbreviate -rows and -cols; any word beginning with a lowercase r is taken to be rows and any word beginning with a lowercase c is taken to be cols. So, for example, you would get mean averages and medians across the rows with -r, -rowwise and -rumplestiltskin, but not -Rows. If the command has too many or two few arguments or if the arguments of the wrong format you should output an error message to standard error. You should also output an error message to standard error if the input file is specified, but it is not readable.

You must output the statistics to standard output in the format shown above. Be sure all error messages are sent to standard error and the statistics are sent to standard output. If the input file is empty, print an error and exit. If there are any errors of any kind, the exit value should be 1; if the stats program runs successfully, then the exit value should be 0.

Your stats program should be able to handle files with any reasonable number of rows or columns. You can assume that each row will be less than 1000 bytes long (because Unix utilities assume that input lines will not be too long), but don't make any assumptions about the number of rows. Think about where in your program the size of the input file matters. You can assume that all rows will have the same number of values; you do not have to do any error checking on this.

You will probably need to use temporary files. For this assignment, the temporary files should be put in the current working directory. (A more standard place for temporary files is in /tmp but don't do that for this assignment; it makes grading easier if they are in the current directory.) *Be sure the temporary file uses the process id as part of its name, so that there will not be conflicts if the stats program is running more than once.* Be sure you remove any temporary files when your stats program is done. You should also use the trap command to catch interrupt, hangup, and terminate signals to remove the temporary files if the stats program is terminated unexpectedly.

All values and results are and must be whole numbers. You may use the expr command to do your calculations, or any other bash shell scripting method, but you may not do the calculations by dropping into another language, like awk, perl, python, or any other language. You may certainly use these other languages for all other parts of the assignment. Note that expr only works with whole numbers. When you calculate the average you must round to the nearest whole number, where half values round up (i.e. 7.5 rounds up to 8). This is the most common form of rounding. You can learn more about rounding methods here (see Half Round Up):

http://www.mathsisfun.com/numbers/rounding-methods.html (Links to an external site.)Links to an external site.

To calculate the median, sort the values and take the middle value. For example, the median of 97, 90, and 83 is 90. The median of 97, 90, 83, and 54 is still 90 - when there are an even number of values, choose the larger of the two middle values.

Your script, stats, must be entirely contained in that one file. Do not split this assignment into multiple files or programs.

Note that it's ok to return an error from stats if the input file is empty

To make it easy to see how you're doing, you can download the actual grading script here:

p1gradingscript

This script is the very one that will be used to assign your script a grade. To use the script, just place it in the same directory as your stats script and run it like this:

```
$ p1gradingscript
```

When we run your script for grading, we will do this to put your results into a file we can examine more easily:

```
$ p1gradingscript > p1results.username
```

To compare yours to a perfect solution, you can download here a completely correct run of my stats script that shows what you should get if everything is working correctly:

p1cleantestscript

The p1gradingscript itself is a good resource for seeing how some of the more complex shell scripting commands work, too.