School of Computer Science, McGill University

COMP-206 Introduction to Software Systems, Winter 2022

Mini Assignment 3: Advanced Unix Utils

Due Date March 9th, 18:00 EST

This is an individual assignment. You need to solve these questions on your own.

You MUST use `mimi.cs.mcgill.ca` to create the solution to this assignment. An important objective of the course is to make students practice working completely on a remote system. Therefore, you must not use your Mac command-line, Windows command-line, nor a Linux distro installed locally on your laptop. You can access `mimi.cs.mcgill.ca` from your personal computer using **ssh** or **putty** as seen in class and in Lab A. **If we find evidence that you have been instead using your laptop, etc., to do some parts of your assignment work, you might loose all of the assignment points.** All of your solutions should be composed of commands that are executable in `mimi.cs.mcgill.ca`.

For this assignment, you will have to turn in one shell script. Instructors/TAs upon their discretion may ask you to demonstrate/explain your solution. No points are awarded for commands that do not execute at all. (Commands that execute, but provide incorrect behavior/output will be given partial marks.) All questions are graded proportionally. This means that if 40% of the question is correct, you will receive 40% of the grade.**TAs WILL NOT modify your scripts in anyways to make it work.**

Please read through the entire assignment before you start working on it. You can loose up to 3 points for not following the instructions in addition to the points lost per questions.

Lab D provides some background help for this mini assignment.

Total Points: 20

Ex. 1 — Parsing sensor data logs for analysis (20 Points)

It is often necessary to parse the output produced by various specialized software systems to extract/generate data that is of specific interest to us. In this assignment, we will use the advanced Unix utilities that we covered in class to analyze the output files from a weather monitoring process.

The data files that we will be using for this assignment is available under the directory hierarchy of `/home/2013/jdsilv2/206/m3/dataset1`. Please note that this directory may not be accessible through FileZilla, etc. It is primarily meant to be accessed from the Unix command line in mimi. These will also be the files that TAs will be using to test your scripts.

The data files are generated by a program that reads five different temperature sensors followed by three wind speed sensors and one wind direction sensor once every hour (24 readings in a given day) and records these readings. The temperature sensors are a bit sensitive instruments, and as such sometimes do not provide data. If the program was unable to read a particular temperature sensor, it will indicate the corresponding sensor's reading as `NOINF` or `MISSED SYNC STEP`. Luckily the wind sensors do not have any issues. Along with this, the program also logs various other information (such as `performing diagnostics`, etc.) which we are not concerned with (but is nevertheless present in the output file and may interfere with our data analysis).

A sample output of one of these data files is given below. (truncated for brevity). You can deduce the message formats from the data files given to you as part of this assignment. Part of this assignments objective is also to have students explore files and file systems and analyze them to find the information they need. Please note that positive temperature readings do not have an explicit `+` sign associated with them.

```
2022-01-24 00:05:25 observation line  -0.70 -1.38 -2.15 -2.67 -2.73 1 0 4 7
...
2022-01-24 03:01:28 performing diagnostics on temp. sensor: 4
2022-01-24 03:03:09 performing diagnostics on temp. sensor: 5
2022-01-24 03:07:25 observation line  -2.70 -2.88 -3.65 MISSED SYNC STEP MISSED SYNC STEP 3 4 7 7
```

```
2022-01-24 04:01:14 observation line  -3.70 -2.88 -3.65 -8.18 -5.72 8 6 7 7
2022-01-24 05:04:18 observation line [data log flushed]  -5.20 -2.88 -5.15 -8.18 -5.72 13 5 10 4
2022-01-24 06:03:08 observation line  -5.20 -2.38 -4.65 -7.18 -4.22 17 4 11 3
2022-01-24 07:03:50 active power disengaged
...
2022-01-24 16:00:08 performing diagnostics on temp. sensor: 3
2022-01-24 16:02:13 observation line [data log flushed]  2.30 4.62 NOINF -1.18 3.78 12 4 3 7
...
2022-01-24 23:01:34 performing diagnostics on temp. sensor: 4
2022-01-24 23:05:09 observation line  0.80 4.62 -0.15 MISSED SYNC STEP 3.28 5 0 7 0
```

You will be writing a shell script `wparser.bash` that would process these data files.

1. **(1 Point)** The shell script is expected to be given the name of a directory, under which it will start the search for data files whose names are of the form `weather_info_*.data` as its argument. (Do not hard code the directory name in your script).
   If the script is not invoked with the correct number of arguments, it should throw an usage message and terminate with a code of 1.

   ```
   $ ./wparser.bash
   Usage ./wparser.bash <weatherdatadir>
   ```

2. **(1 Point)**
   If the passed argument is not a valid directory name, it should throw an error message and terminate with code 1. For this particular situation (and only here), the error message must be send to the standard error and not the standard output.

   ```
   $ ./wparser.bash /nosuchdir
   Error! /nosuchdir is not a valid directory name
   ```

   You do not have to explicitly check if you have the permissions to access the directory or the data files.

3. **(1 Point)** Within the shell script, use an appropriate Unix command to look for data files starting under the given directory hierarchy that matches the specific file name pattern mentioned above (keep in mind that the data files might be under some arbitrary subdirectories, etc.). Do not assume specific directory hierarchies. Each data file only contains the information for that specific day. And each day has its own data file and never spread across multiple data files. No points are awarded for this question even if you miss one valid data file or include data files which does not follow the pattern given to you.

4. **(8 Points)** Write a function `extractData` in your shell script `wparser.bash`. **(-4 points)** if not written as a function. This function will perform the core processing logic associated with each file. The main script will just iteratively call this function with different files as its argument.

   This function should accept a data file as its argument. This function should then produce an output of the following format that that contains only the temperature and wind information from the sensors along with a header. (Truncated for brevity). A clean format of data like this form can be used by various information processing systems.

   ```
   Processing Data From  <path to the filename here>
   ==================================
   Year,Month,Day,Hour,TempS1,TempS2,TempS3,TempS4,TempS5,WindS1,WindS2,WindS3,WinDir
   2022,01,24,00,-0.70,-1.38,-2.15,-2.67,-2.73,1,0,4,NW
   ...
   2022,01,24,04,-3.70,-2.88,-3.65,-8.18,-5.72,8,6,7,NW
   2022,01,24,05,-5.20,-2.88,-5.15,-8.18,-5.72,13,5,10,S
   2022,01,24,06,-5.20,-2.38,-4.65,-7.18,-4.22,17,4,11,SE
   2022,01,24,07,-3.70,-2.38,-4.65,-6.68,-3.72,14,6,6,SE
   ...
   2022,01,24,21,1.80,5.12,2.85,-1.68,3.28,4,4,5,SE
   2022,01,24,22,0.80,5.12,1.35,-1.68,3.28,3,0,6,SE
   2022,01,24,23,0.80,4.62,-0.15,-1.68,3.28,5,0,7,N
   ==================================
   ```

The script is basically only including the year, month, day and hour information, followed by the temperature reported by each temperature sensor, and wind speed from wind sensors at that time. If a temperature sensor's reading is `NOINF` or `MISSED SYNC STEP` in the original data file for that particular time, the script must instead output the previous readout for that temperature sensor.
For simplicity, you can assume that the first readout for all the temperature sensors in a given day does not have any issues. The output should follow the same order of time as in the original data file.
The script also translates the wind speed direction codes (which runs from 0 through 7) to appropriate mnemonics, i.e. (`N`,`NE`,`E`,`SE`,`S`,`SW`,`W`,`NW`) respectively.

5. **(4 Points)** Immediately following the previous output produced from a data file, the script (aka inside the function to be specific) should produce the statistics as to what was the maximum as well as the minimum temperature and wind speed that was reported for a given hour. The format is given below. For this report, **it is important to ignore the temperature sensors that it was not able to read data for that hour** and consider only the sensors that were functioning and produced a valid reading in that hour.

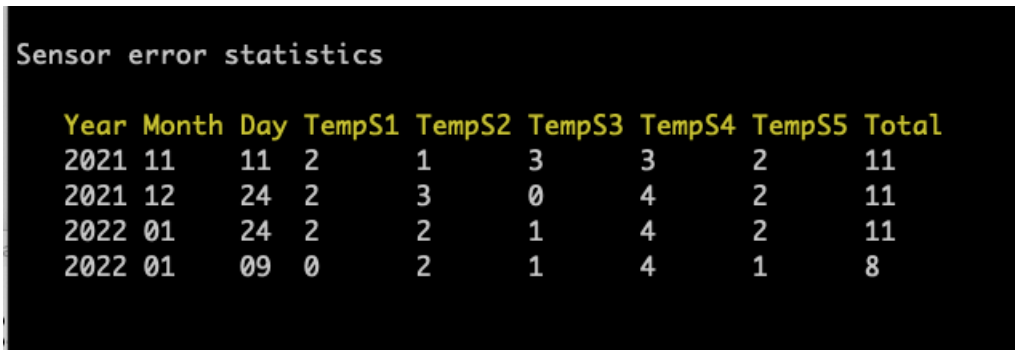
```
Observation Summary
Year,Month,Day,Hour,MaxTemp,MinTemp,MaxWS,MinWS
2022,01,24,00,-0.70,-2.73,4,0
...
2022,01,24,06,-2.38,-7.18,17,4
2022,01,24,07,-2.38,-6.68,14,6
2022,01,24,08,-2.38,-5.18,12,2
2022,01,24,09,-1.38,-3.68,13,0
...
2022,01,24,23,4.62,-0.15,7,0
==================================
```

As in the previous case, the output should follow the same order of time as in the original data file.

6. **(5 Points)** Once the script is done producing the above two statistics for each data file, we want the script to report on the health of the temperature sensors across all those days (data files). The logic for this task is to be written as part of the main script itself (i.e., not inside the `extractData` function). For this purpose, we will have to count the number of times that each temperature sensor reported an error for each day. If a temperature sensor did not report an error, indicate with the value 0. The last field in each line is the total number of temperature sensor errors on that day (sum of the individual sensor errors). The output should be sorted such that the dates with the larger number of (total) errors on the top (descending order). If two (or more) dates have the same number of errors, then order their lines in the output in the chronological order of dates. (I.e. Jan 31 is before Feb 1, etc. if they both have same number of errors.)
This report, however, should be stored on to a file called `sensorstats.html` in the current directory. The format of this HTML file is very simple. You can `vi` the example format given to you to understand the format that you need to produce. If the file already exists, overwrite it.

```
HTML>
<BODY>
<H2>Sensor error statistics</H2>
<TABLE>
<TR><TH>Year</TH>...<TH>Day</TH><TH>TempS1</TH>...<TH>TempS5</TH><TH>Total</TH><TR>
<TR><TD>2021</TD><TD>11</TD><TD>11</TD>...<TD>3</TD><TD>2</TD><TD>11</TD></TR>
...
</TABLE>
</BODY>
</HTML>
```

You can also use the `lynx` command line browser available in mimi to look at the html file, which will be displayed like below (colors may be different/absent depending on your terminal software and is not relevant).



```
Sensor error statistics

   Year Month Day TempS1 TempS2 TempS3 TempS4 TempS5 Total
   2021 11    11  2      1      3      3      2      11
   2021 12    24  2      3      0      4      2      11
   2022 01    24  2      2      1      4      2      11
   2022 01    09  0      2      1      4      1      8
```

ADDITIONAL RESTRICTIONS

- You must write a reasonable amount of comments (to understand the logic) in your script. You can lose up to **-2 points** for not writing comments.
- Follow the sample output format that is given to you for the valid invocation. It does not take much effort to implement them. Not following it can result in a deduction of **-2 points** or more.
- The script **MUST NOT** create any temporary/intermediate files to do its work. Use the techniques already covered from previous assignments and labs to pass output of one command/utility to another. Violations would result in a deduction of **-3 points**.
- Any error messages from your program should be as a result of an explicit `echo` command in your script. Any error messages from commands/utilities used by your script should be handled by the script itself and not reported to the user. Violating this would result in **-2 points** deduction per occurrence.
- Your script should run correctly irrespective of any valid date/time in the data file and should not depend on the values being only for specific year, month, etc. (**-3 points** deduction).
- Your submission should be a single script (file), specifically, do not put `awk` commands, etc., in a separate file. (**-2 points deduction**).
- For the data files in the test directory given to you for testing, your script should run under 5 seconds (clock time), not "hang", etc.. Scripts that take longer than this may not get graded or maybe graded only for the outputs produced in that time. To give some perspective, a simple, unoptimized implementation of this solution runs well under 0.1 seconds. This could result in 0 or very low points depending on how far your script progressed.
- DO NOT Edit/Save files in your local laptop, not even editing comments in the scripts. This can interfere with the fileformat and it might not run on `mimi` when TAs try to execute them. This will result in a 0. No Exemptions !!. TAs will not modify your script to make it work.

WHAT TO HAND IN

Upload your script `wparser.bash` to MyCourses under the **mini 3** folder. Re-submissions are allowed. You are responsible to ensure that you have uploaded the correct files to the correct assignment/course. There are no exemptions. **NO Emailing of submissions**. If it is not in MyCourses in the correct submission folder, it does not get graded.

Late penalty is -20% per day. Even if you are late only by a few minutes it will be rounded up to a day. Maximum of 2 late days are allowed.

ASSUMPTIONS

- You may assume that any files and directories that your script needs to access will have the necessary permissions for it to execute the tasks outlined in the assignment.
- There will be at least one valid data file inside the directory hierarchy.
- When you are processing multiple data files, you can process them in any order.

- You do not have to account for hidden files and directories.
- A "correct" filename will always contain valid data (no need to look for badly formatted data).
- The entries in the data files follow the order of time.
- Although the exact minute/second in which the information is recorded by the program is not very accurate, you can rely that each hour will have only one instance of it reading the sensors and that there are no missing entries.
- You can assume that the first instance of sensor readout in a file does not have any temperature sensor errors.
- You can assume that all five temperature sensors will not error out at the same time.
- You can assume that the valid values of temperatures are in the range `100.00` to `100.00`, wind speeds are within `0` to `200` and wind direction is in the range `0` to `7`, all inclusive.

HINTS

This is a high-level outline to get you started with the first part of the output format in case you feel stuck. You are not obliged to follow it.

- Use the `grep` command to extract only the lines of interest from the data file.
- Can you find a way to use `sed` to reformat this data, remove/transform some of the unnecessary information, etc., to make it a consistent number of attributes so that in the next step,
- You can use `awk` to extract only the fields of interest (with some additional "data cleaning" that `sed` may not have done).
- In class we saw how to declare `awk` variables and use them. Figure out a way to apply that approach in keeping track of the previous value of each sensor so that you can use it if the current value is not valid reading.

For the last part, (again, not obliged, there could be other ways to implement this).

- Use `awk` to perform most of your data extracting / formatting tasks for individual data files.
- Can you pipe the output of a `for` loop to another command?
- How to use `sort` command to work with a non-space delimiter?
- Figure out how `sort` command can be used to impose reverse sorting on one field (keys) followed by regular order on some other fields.
- This link provides a simple introduction to the concept of HTML tables.
- You will find it easier if you produce the data you need to format as HTML in a comma (or space, etc. - simple delimiters) separated manner first, verify the correctness of the data and the order, etc., AND THEN think about the logic to transform that into an HTML table format.

COMMANDS ALLOWED

You may use any option provided by the following commands, even the ones that have not been discussed in class. Depending on your solution approach, you may not have to use all of these commands.

```
$( )       $(( ))   $[ ]    [[ ]]     awk
basename   bc       break   case      cd
continue   date     diff    dirname   echo
exit       export   expr    find      for
function   grep     if      ls        pwd
read       sed      set     shift     sort
tar        while
```

You can also use redirection, logical and/or/negation/comparison/math operators and any check operators (such as checking if something is a file) as required with the `$[[ ]]` operator. You can also use shell variables and use/manipulate them as needed.
You may not use any commands that are not listed here or explicitly mentioned in the assignment description. **Using commands not allowed will result in 2 points deduction per such command.**

TESTING

There is no tester associated with this assignment. An file is provided that contains all the expected output for the valid invocation in `wparser.bash.out.txt`. Please refer to this if you have questions on the format and also to compare your output with the solution.

Once you have done your basic verification, you can test your script in the following manner against the test directory hierarchy that contains the actual data files.

```
$ ./wparser.bash /home/2013/jdsilv2/206/m3/dataset1
```

This is how TAs will be testing your submission and against a similar directory / data files setup. The output examples provided with the assignment corresponds to the above directory.

If you need an additional test case, you can try your script against another directory given below.

```
$ ./wparser.bash /home/2013/jdsilv2/206/m3/dataset2
```

QUESTIONS?

Remember to do step-by-step logic development. Do not write a very complex set of commands and pipeline them in the beginning. Take a single example data file and see if your `grep`, `sed`, etc., is doing the transformation you had in your mind. Do not start off by writing a very complicated logic that looks like `grep .... | sed .... | grep .. | awk ... | sed ...` ! You will not know where you logic has a problem. Always ensure the first set of commands is doing what you want them to do, before you add the logic to the next set of commands!

If you have questions, post them on the Ed discussion board and tag it under mini 3, but do not post major parts of the assignment code. Though small parts of code are acceptable, we do not want you sharing your solutions (or large parts of them) on the discussion board. If your question cannot be answered without sharing significant amounts of code, please make a private question on the discussion board or utilize TA/Instructors office hours.

Please remember that TA support is limited to giving any necessary clarification about the nature of the question or providing general advice on how to go about identifying the problem in your code. You are expected to know how to develop a high level logic, look up some syntax/options and most importantly, debug your own code. Lab C covers a lot of useful debugging techniques. We are not testing your TA's programming skills, but yours. Do not go to office hours to get your assignment "done" by the TAs.

Emailing TAs and Instructors for assignment clarifications, etc., is not allowed. TAs and instructors may convert private posts to public if they are not personal in nature and the broader student community can benefit from the information (to avoid repeat questions). Also check the pinned post "Mini 3 General Clarifications" before you post a new question. If it is already discussed there, it will not get a response. You can email your TA only if you need clarification on the assignment grade feedback that you received.