CS2518 - P1 - BASH

Overview

In this practical we are going to create a number of small shell scripts (using *bash* as shell) that make up a simple data processing chain. A unix environment provides a large number of small programs that we can use to manipulate data and we can string them together using shell scripts to perform data pre-processing; i.e. preparing our data before we pass it into our main program.

The Example Scenario

Assume we have an industrial installation where we have 5 sensing devices (IoT devices) installed. These devices measure periodically temperature and pressure values and report these to a central base station in the factory (How we construct these devices and how we form a network to send this data is subject of CS4628 and we just assume here it all works). The base station stores the received data and we can request the data in form of a CSV file which looks like this:

```
ID,time_stamp,temperature,pressure
1007,1736942151,20.78,993.43
1006,1736941151,1475.69,956.26
1007,1736941737,26.31,981.03
1006,1736941852,21.29,951.38
1007,1736942818,17.02,996.44
1005,1736940596,19.28,1038.99
1007,1736943189,23.02,1034.23
1007,1736940764,29.53,977.8
1006,1736940017,18.47,1025.06
1006,1736941448,24.36,1039.96
...
```

Each row in the file represents a reading from a sensor; the first line in the file shows the name of the different fields. Each row provides 4 values: the ID of the device that provided the sensor reading (1000, 1005, 1006 or 1007), the unix time stamp when that recording was taken, the measured temperature value, the measured pressure value.

An example file is provided called *sensor_data.csv* containing 250 sensor readings. We may want to use this file as input for a program (e.g. machine learning or visualization) and we need to pre-process this file a bit. For example, we may want to have a tool to show us the content in a more human readable form. We may want to sort the file as the entries (the time stamps) are not in ascending order. We also have the issue that some temperature readings are way too high (over 1000 degrees) which is a sensor fault

and we want to exclude such readings. We could address all these issues in our main program but it is also possible to use shell scripts to do this.

Task 1: Print File

The first task is to write a script that can print the file content in a more human readable version. Write a bash script called *print_csv.sh* that performs the following functions:

(1) It takes the filename of the csv file as command line argument (we want to start the script as *./print_csv.sh sensor_data.csv*.
(2) The script tests if a file with the given name exists; It should produce an error message if the file cannot be found.
(3) If the file exists the script should read the file line by line and produce the following output:

```
...
ID: 1006, Time: 1736939424, Temp: 22.3, Pressure: 1002.04
ID: 1006, Time: 1736942926, Temp: 18.27, Pressure: 1041.86
ID: 1000, Time: 1736943016, Temp: 22.79, Pressure: 1034.93
ID: 1005, Time: 1736941918, Temp: 15.13, Pressure: 954.28
...
```

Hints:

- You need an *if* statement to test if the file used as argument exists.
- The program *tail* can be used to skip the header lines in a file (type *man tail* to get a manual for this command).
- You can read a line from input using *IFS=, read -r ID time temp pressure*. IFS sets the separator and in this example four tokens are read and placed in the variables ID time temp pressure (type *help read*).
- You can process line by line from a file

```
while read LINE; do
    echo "I read one line!"
done < inputfile
```

Task 2: Convert Time Stamp

The time here is represented as a unix time stamp. The unix time stamp is a way to track time as a running total of seconds starting at the Unix Epoch on January 1st, 1970 at UTC. A unix time stamp is the number of seconds between a particular date and the Unix Epoch[1]. As the unix time stamp is not very human readable, modify the shell script to print the time in a better readable format.

(1) We still want to start the script as *./print_csv.sh sensor_data.csv*.

[1]On January 19, the Unix Time Stamp will cease to work due to a 32-bit overflow. Before this moment millions of applications will need to either adopt a new convention for time stamps or be migrated to 64-bit systems which will buy the time stamp a "bit" more time.

(2) It should now produce the following output:

```
...
ID: 1000, Time: 2025-01-15 11:13:04, Temp: 18.97, Pressure:
    1008.58
ID: 1007, Time: 2025-01-15 11:54:50, Temp: 26.48, Pressure:
    1039.71
ID: 1007, Time: 2025-01-15 11:31:54, Temp: 23.12, Pressure:
    1009.67
...
```

Hints:

- You can use the program *date* to convert a unix time stamp into other formats (type *man date*).

Task 3: Filter Outliers

When you examine the file *sensor_data.csv* you notice that some temperature readings are unusually high (over 1000 degree). We assume these are generated by a faulty sensor and that room temperature can never be that high. Therefore we now would like to construct a shell script *./filter_csv.sh* that removes all lines in the csv files where the measured temperature is above 50 degrees. Write a bash script called *filter_csv.sh* that performs the following functions:

(1) It takes an input filename as the first command line argument and an output filename as a second (we want to start the script as *./filter_csv.sh sensor_data.csv filtered_sensor_data.csv*.
(2) The script tests if a file with the given input name exists; It should produce an error message if the file cannot be found.
(3) The script tests if an output file name was provided; It should produce an error message if this parameter is missing.
(4) The script tests if the output file already exists; It should produce an error message if this file already exists (no overwriting of the file).
(5) The script should read the input file line by line, if the line is an outlier it is ignored, otherwise it is written to the output file.

Hints:

- You can use the program *awk* to analyze each line (type *man awk*).
- You can use the program *tail* to read lines without the header line.
- You can connect tail and awk using a pipe.

Task 4: Sort

When you examine the file *sensor_data.csv* you notice that sensor readings are not in chronological order (You may use your script from Task 2 to make it easier to see). Therefore we now would like to construct a shell script *./sort_csv.sh* that sorts lines in

the csv file in ascending order using the time stamp. Write a bash script called *sort_csv.sh* that performs the following functions:

(1) It takes an input filename as the first command line argument and an output filename as a second (we want to start the script as *./sort_csv.sh sensor_data.csv sorted_sensor_data.csv*.
(2) The script tests if a file with the given input name exists; It should produce an error message if the file cannot be found.
(3) The script tests if an output file name was provided; It should produce an error message if this parameter is missing.
(4) The script tests if the output file already exists; It should produce an error message if this file already exists (no overwriting of the file).
(5) The script should read the input file and then produce a sorted output file.

Hints:

- You can use the program *sort* (type *man sort*).
- You can use the program *tail* to read lines without the header line).
- You can connect tail and sort using a pipe.

CS2518 Continuous Assessment - PART 1

Please submit the four commented scripts you created. In addition provide a brief text description of each of your four scripts. The description for each script should be short paragraph describing the core functionality.

The scripts and your description has to be submitted at the end of the semester together with questions from the other practicals (You do not have to complete all tasks within the practical slot).