Exercise Sheet 5

Bash & co.

29 September 2023

Exercise 1

Some tasks for which Awk can provide good support

Go through the few slides discussed in the lecture about `awk`, focusing on the examples there. Once prepared an input file via

```
paste <(shuf -n 20 /usr/share/dict/ngerman) <(shuf -i 1-5000 -n 20)
```

achieve the following tasks using a `awk` command. You can assume that the file contains a word in the first column and an integer number in the second one.

1. Print every third line.
2. Display only lines for which the number on the second column is smaller than 1000.
3. Calculate the average of the second column.
4. Print lines starting by a vowel.
5. Print the first column word if the second column contains a number larger than 3000.

Exercise 2

A taste of awk and sed

Awk and sed are two incredible tools, but it is not automatically true that they are the *best tools* to solve your problem. In this exercise they can be used, but it might be that you find your way without using them. This is true in general, indeed.

Here you can find a chunk of the standard output of a LQCD simulation, in particular measurements lines, only. Each measure reports in square brackets the simulation step (the so-called trajectory) at which it has been carried out. Unfortunately, there have been some I/O problems on the cluster where this file was generated and there are holes in the simulation. Observables of interest for the purpose of the exercise are `PLAQUETTE`, `fRECTANGLE` and `POLYAKOV`. Have a look to the file and then, playing in the terminal, figure out how to answer to the following questions.

1. How many times each observable has been measured?
2. Which is the first and last trajectory for each observable?
3. Where did the I/O problems occur for each observables?

You could set up a (long) command which would print

```
PLAQUETTE: Measures=2311 tr=[2000-4999] Holes=[2056-2287 3202-3433 4349-4579]
fRECTANGLE: Measures=2312 tr=[2000-4999] Holes=[2056-2286 3202-3433 4349-4579]
POLYAKOV: Measures=2311 tr=[2000-4998] Holes=[2056-2286 3202-3433 4349-4579]
```

Now that you understood the basic idea, write a Bash script in order to extract the observables in a file with 4 columns: the trajectory number and the three observables (`PLAQUETTE`, `fRECTANGLE` and `POLYAKOV`). Pay attention to the trajectories for which you do not have all observables! In such a cases, you should discard the whole trajectory (nothing to be added to the output data file) and print a warning to the user.

NOTE: In the whole exercise, you might dislike having to process several time the file, since it might be slow. Well, what does *slow* mean? How does the time you might save compare to the time you need to come up with a solution to avoid a file processing? Do you remember Donald Knuth's words? *"Premature Optimisation Is the Root of All Evil"*.

Exercise 3

Plotting data in LaTeX from a bash script

As final exercise of this course we would like to write a last script, which might let you discover some interesting feature for your work as physicist. We assume you have a LaTeX distribution installed on the machine where you'll work. If this is not the case, just add the following `pdflatex` function to your script to mimic it:

```
function pdflatex()
{
    echo "$@"; sleep 5
    touch 'Plot.'{log,aux,pdf}
}
```

You will loose the graphical aspect of this exercise, but you will still be able to work on it.

As you might know, LaTeX has a wonderful package, `PGF`, which is provided with a math engine and on which lots of packages are based (e.g. `TikZ`, `forest`, `genealogytree`). Here we will use `pgfplots`, which allows to make nice-looking plots of e.g. data. Our goal is to create a bash script which plots the data in a given file, using one column as x-values and another one for the y-values. The minimal user requirements are the following, but with a bit of creativity you can implement much more.

1. The input data filename must be given on the command line.
2. It should be possible to specify the x and y columns to be plotted, using some default values.
3. The script should create the plot as pdf file and give informative output to the user.
4. An external `.tex` source file should be used, which could also be compiled by hand if needed.

The flow should be clear. The command line options are parsed, the `.tex` file is used according to the user directives and it is finally compiled. **The first column in the data file is the number 0, the second is the number 1, etc.**

The way to hand over information from bash to LaTeX world is not part of the exercise and the short story[a] is that you should use this command

```
pdflatex -interaction=batchmode \
"\def\filename{...} \def\xColumn{...} \def\yColumn{...} \input{Plot.tex}"
```

in your bash script to compile the `.tex` file (the `...` should be the properly filled by your bash script). Here you find the LaTeX source code to be used (you can have a look to it, although not necessary).

Now that you know how to hand over information to the LaTeX compiler, you can focus on your bash script. In particular, be sure to have examined and dealt with the following aspects.

- As you know, LaTeX produces a bunch of additional files and it would be nice to remove them to leave the present folder in a tidy state.
- The clean-up should be always done, unless you have some reason not to do it. For example, how does you script behave when pressing `CTRL-C` during execution? It might be interesting to use the `-halt-on-error` of pdflatex to make it better handle an interruption.
- Which strategy did you use to do error handling? What happens when the LaTeX compilation fails? Trigger a failure by hand to test your code.

You can produce an input data file using `awk` via

```
awk 'BEGIN{for(i=0; i<10; i++){print i"\t "rand()"\t"2*i"\t"10*rand()}}'
```

and try to plot e.g. the fourth column as function of the third one.

[a] **For the curious reader:** In the LaTeX code, the macro can be defined only `\ifundefined` to provide a default value to compile the `.tex` file by hand. Invoking the `pdflatex` compiler as above, it is possible to define some macros in advance which are then handed-over to the LaTeX source code from outside. The `batchmode` of the `pdflatex` compiler will suppress all but a handful of declarative lines, without stopping, even in the case of a syntax error.