# Lab 5: Shell Scripting with cdialog and awk

Benedict Reuschling

February 21, 2019

## 1 `awk`-Programming

In this exercise, we will use the `capitals.txt` provided with this lab to process it using `awk` and shell scripts. The goal is to learn how to create quick `awk`-scripts that process data from a file and format it or do calculations on the fly. Refer to `awk(1)` to get help on parameters and general use.

The following exercises are intended to be solved in order. You can build on your previous scripts and enhance them further, but make sure to keep a copy around in case you have to start from scratch.

- Check the top lines of the file using the `head` program. The first line is a comment describing the actual content of each column. Write an `awk` script that outputs a descriptive text (column headers) at the beginning and then prints the file contents itself. When all the contents are printed, let `awk` print a separate line stating `end of file`.

- The next iteration of your script should output the column headers again after 30 lines have been printed. Write a function for printing the column headers and call it each time after 30 lines.

- Use `awk`'s built-in variables to print a summary after the `end of file`. Output the filename and the total number of lines in the file.

- In awk(1), you will find (among many other things) how to specify a custom field separator instead of the default one (space). Use it to remove the separators (`,`) between the columns and to print the file contents in tabular format.

- Now, we will concentrate more on the actual data within the file. Use `awk` combined with a shell script (when needed) to answer the following questions:
  - What is the total number of citizens in all capitals?
  - What is the name of the city with the biggest/smallest population?
  - Which continent has the most capitals?
  - How many citizens do the capitals have for each continent?

# 2 Plotting Data

Gnuplot[1] is a terminal program to plot data in various ways. It can be used interactively or using a configuration file. We will use the latter to extract data from a CSV file that will serve as our data source (poor man's database).

First, install `gnuplot` using `pkg`. Familiarize yourself with the basic gnuplot syntax. You don't have to learn much to be able to create basic graphs that can visualize your data. Once you are more versed, you can create more fancy graphs (but this is beyond what we need to achieve for this lab exercise). Here are two tutorial links that should get you started (of course, you can also find other tutorials on the Internet):

- `http://people.duke.edu/~hpgavin/gnuplot.html`
- `http://www.ibm.com/developerworks/aix/library/au-gnuplot/`

Create a basic configuration file that can be used by Gnuplot to create a basic graph. Once that works, look at the data in `population.csv` (provided with this lab), which is world population data[2]. The file format is as follows:

```
Country Name,Country Code,Year,Value
Arab World,ARB,1960,93485943
...
```

The goal for this lab exercise is to visualize the population growth for a country over time. A shell script asks the user for the country code (i.e. ARB) and range in years (i.e. 1965 till 1980). You decide whether the program should use a `dialog`-based interface or simple command line interactions only. The program extracts the data from `population.csv` for the specified region and year range and puts it into a new file. Make sure to properly format that file so that Gnuplot can process it (i.e. replace commas with spaces). Use Gnuplot to plot the selected data as a PNG file based on your configuration file and the following requirements:

- Both the y- and x-axes should be labelled properly
- The graph should have a proper title
- Of course, the plot should correspond to the selected data for that country
- Use proper error checking within your script to check for invalid inputs

**Note:** don't get too fancy with the graph output. An ugly graph with the proper data is (arguably) more valuable than a pretty plot with the wrong data.

[1] Interestingly, Gnuplot was **not** developed by the GNU project or the FSF. More about this here: `http://www.gnuplot.info/faq/faq.html`

[2] Source: `http://data.okfn.org/data/core/population`

# 3 Shell Scripting

Susan Sunshine is looking forward to her 18. birthday on January 17, 2018. Among other benefits, she is then legally allowed to donate blood in Germany without requiring her parents explicit permission. She already informed herself about the procedure and learned the following facts:

- Healthy people can donate up to six times per year.
- Between two donations, a resting time of two months is required, recommended are three. The minimum allowed distance between two full blood donations in Germany (where Susan lives) is 56 days (or eight weeks).

Write a script that calculates how many blood donations Susan can do until her 70th birthday, provided that she is healthy each time and can make the apppointment. Calculate the dates for each blood donation considering the above facts regarding donations per year and resting time. Show the results in a table with the calculated values in this form:

```
Donation No.    Date of Donation    Age
           1    17.01.2018          18
           2    ...                 ...
```

You can find more information about how to generate and use timestamps in the man page for `date`. For example, May 11, 1970, 06:57:35 is represented by the timestamp 11253455.

Write a second program so that Susan receives a (fictional) amount of 5 Euro for each donation. How much total money does Susan get this way before she is not allowed to donate anymore? Add another column to your output that shows the current amount of money for each donation as well as a total at the end:

```
Donation No.   Date of Donation    Age     Euro received    Total amount
1              17.01.2019          18      5                5
2              ...                 ...     5                10
```

In this second program, instead of displaying the results right away, display a progress bar (`cdialog -gauge`) by stopping one second for each line of output so that the user knows how much longer she can donate. The progress bar at 0% represents her first donation at age 18 and reaches 100% at her last donation (before being too old). Then the results are displayed with the total amount of money she earned overall.