Einführung in die Programmierung für Physiker

WS 2017/2018 – Marc Wagner

Francesca Cuteri: cuteri@th.physik.uni-frankfurt.de
Alessandro Sciarra: sciarra@th.physik.uni-frankfurt.de

Exercise sheet 1

To be corrected in tutorials in the week from 23/10/2017 to 27/10/2017

Exercise 1 [*Playing with the shell*]
In this exercise we will play around with some basic shell commands: `mkdir`, `touch`, `chmod`, `rmdir`, `rm`, `pwd`, `cat`. Moreover, remember that `ls` and `cd` are always your friends. When working with `Bash` or similar shells, the `<Tab>` key, which completes text, and the `<Up>` and `<Down>` arrows, which navigate through the command history, are very handy. Play a little bit with them during the exercise to find out how they can help you.

(i) Open a terminal in Linux.

(ii) Use the `passwd` command to change the password for your account. You will be first prompted for your old password, then you will be prompted twice for a replacement password. The second entry is compared against the first and both are required to match in order for the password to be changed. From your next login to your account, you will need to use the new choosen password.

(iii) Create one directory in your home, let us call it `ex`, and make inside it the following directories: `bin`, `etc`, `physics` and `src`, such that[1]

```
$ ls ex/
bin etc physics src
```

Do you need to `cd` to `ex/` to do so? Check `man mkdir` to look for some option that allows you to skip the `cd` step.

(iv) Go inside directory `ex/src/` (you can always check where you are with the `pwd` command) and make three directories: `c`, `c++` and `python`. Then go to directory `c`. We will use the command `touch` to create some files. First, execute `touch Makefile`, then `touch makefile` and last repeat `touch Makefile`. What does it happen? How many new files do you expect to have after running these three commands? Remember: the shell is case-sensitive. Can `ls` help us finding which file we created/modified last? Find at least two ways you can sort files in folders with `ls` by checking `man ls`. Finally, go to `~/ex/` and remove directory `src/` and all its content. You can use `rm` or `rmdir` to complete this task. Perhaps some option is needed (check the `man page` in case of doubt).

(v) Next, go to `~/ex/physics/` and create a new directory called `.secret.project`, and go inside it. Then, use your favorite editor (`emacs`, `xemacs`, `vi`, `vim`, `nano`, `kate`, `gedit`, etc.) to open a file called `readme.text` and write something inside it. Save the file and exit. Now, `cd` to `~/ex/physics/` and do `ls`. Where is the directory `.secret.project`, and the file `readme.text`? If you have troubles, check the `man page` of the `ls` command.

(vi) Change the working directory to `~/bin` and do `touch my.fake.program`. Can you execute it via `./my.fake.program`? Use `ls -l` to find out if and by who `my.fake.program` can be read, written, executed. You can use `chmod` to make it executable, for instance with

```
$ chmod +x my.fake.program
```

At this moment you can execute it (check the permissions in the first column of `ls -l` again). Now run `./my.fake.program`. What is the content of the file? What is the result of executing it? Put some content in the file using `echo`. For instance

[1] Here and in all the sheet, the `$` is not part of the command and has not to be typed. It indicates only the end of the *prompt* (ask your tutor if you do not know what the prompt is).

```
$ echo "Hello world!" > my.fake.program
$ echo "Goodbye." >> my.fake.program
```

What is the contents of the file now? Check it with `cat`. Note the usage of `>>` in the second command above, instead of the symbol `>` in the first command. What happens if you execute both echo lines with `>`? What is the content of the file afterwards? Now, put some new content to the file, and execute it

```
$ echo "cal" > my.fake.program
$ echo "date" >> my.fake.program
$ echo "ls" >> my.fake.program
$ ./ my.fake.program
```

Can you explain the output? Again, feel free to check the `man page` of any command whenever you need.

Exercise 2

Gnuplot, as mentioned in the lectures, is a command-based, graphical plotting program for Linux which, in addition to being very handy when one needs a quick and dirty plot, is also extremely customisable, and can, with sufficient practice, be used to generate professional looking plots.

(i) First, open Gnuplot by typing `gnuplot` in the terminal. Although a bit minimalistic, Gnuplot comes with extensive documentation through the help command.

(ii) Start off with typing `help plot` which will tell you how to use the plotting function. Following the first example of the documentation, you can plot a simple curve by typing

```
plot sin(x)*x**2
```

Gnuplot has a set of standard functions you can use when plotting, some of which are summarised below:

```
exp()      log()      **
sin()      cos()      tan()
asin()     acos()     atan()
cosh()     sinh()     tanh()
```

(a) First try to plot the Gaussian distribution with 0 mean value and unit standard deviation:

$$\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \tag{1}$$

(b) Gnuplot should have been able to plot that quite nicely, but lets tidy it up a little.

(1) Give the plot labels, such as x at the x-axis and P at the y-axis
(2) Restrict the plot to only plot the ranges $x \in [-5, 5]$ and $y \in [0, \frac{1}{2}]$
(3) Plot tics along the x-axis at intervals of 1, and along the y-axis at intervals of $\frac{1}{10}$
(4) Plot special named tics at the points $x = -1.96$ and $x = 1.96$, and name them "95%"
(5) Give the plot a title, e.g. "Gaussian distribution ($\langle x \rangle = 0$, var $= 1$)"
(6) Set the sample rate for the function to make the plot smoother

All of the above tasks can be done with Gnuplots `set [option]` function. You should read the help section (by typing `help set [option]`) on some of the following options and learn how to use them. You can use the `replot` function between each call to `set` to have your changes displayed, instead of re-writing the `plot` command.

```
xlabel     xrange     xtics
title      samples    key
```

(c) Lastly, let us change Gnuplots `terminal` and `output` so that we can save the plot we have made as an image file. The `terminal` setting tells Gnuplot which kind of output it should generate, e.q. gif, png, jpg, pdf, tex and so on (you might have noticed during startup that Gnuplot set the output to wxt, so if you ever need to switch back, you do not need to close and re-open Gnuplot). The `output` setting tells Gnuplot which file to save to. Read the documentation for the two settings, and save your plot in a filetype of your own choice.

In this exercise we only covered styling the plot borders. Gnuplot offers much more functionality for which you can refer to the web[2].

(iii) Close Gnuplot for a while. If you do not know how to do it, check `help exit`.

(iv) Now let us try to plot points from, pairs of x and y coordinates, with some vertical error bars `yerrorbars`.

(a) First we need to have a file (call it `datafile`) from which Gnuplot will be able to read our data. It will look like a table having as many rows as points we want to draw and 3 columns. Let us put as first column the x coordinate of our points, the y coordinates of our points will be listed in the second column and the third column will contain our errors on y. If you like, you can manually edit the file using your bash knowledge (`echo`-ing and appending row by row) or simply by copying numbers in a text editor of your choice. The file content should look like this

```
-1.0    0.999901    0.021686
-0.8    0.410919    0.011464
-0.6    0.128373    0.043199
-0.4    0.022666    0.000070
-0.2    0.001660    0.000740
0.0    0.000232    0.001102
0.2    0.001863    0.003381
0.4    0.018073    0.002906
0.6    0.127141    0.029107
0.8    0.410828    0.016060
1.0    0.996238    0.017804
```

(b) You can also download the file `data` using the command `wget` which can get some options and expects some `url` (always check the corresponding `man` entry). The `url` you need is `https://th.physik.uni-frankfurt.de/~cuteri/data`.

(c) **Optional**.
Now you have two files (even if you will need one). Are you sure they are the same? You can check it with the `diff` command.

(d) Open Gnuplot again and let us plot our data using:

```
plot "data" using 1:2:3 with yerrorbars
```

Can you understand the argument `1:2:3` of the `using` specification? You could use it to flip x and y...

(e) Can you see a trend in those points? Can you guess a function? Try by running `replot <function>` to check if your guess was right!

(f) **For the most curious**.
How do you explain the output of the following command?

```
plot "data" using 1:($2**0.25):(0.25*$3**0.25) with yerrorbars
replot sqrt(x**2)
```

[2] Ask google or have a look to `gnuplot.sourceforge.net/demo_5.1`.

Exercise 3 [*Installing GNU/Linux*]

Having a GNU/Linux system at hand is probably a very good idea in this course. It is flexible and powerful, and its most fashionable distributions (Fedora, Debian, Ubuntu, Suse...) bring by default many tools for programming. And provide easy access to much more software through their *package managers*.

This first exercise is a reminder of what was said in the lecture. The simplest GNU/Linux installation is probably done on a *clean* computer - without any other operating system. But it is usually possible to make GNU/Linux coexisting with other operating systems. Many distributions have online documentation to help you in such a task.

Another option is to use some kind of virtualization software. In that case, you install some program like VirtualBox or Xen (or many others) in your current operating system. The general idea is that those programs simulate computers, therefore you can create a new virtual computer inside the virtualization program and install whatever operating system you like without affecting the installation of the current real operating system.

You are free to choose the method you like more.

Exercise 4 [*Read my mind*]

In this exercise we want to implement a very easy game to challenge the pc. We want the pc pick up a number and we want to guess it until we discover it. Our program, then, should give us a feedback about our answer and wait for a new input, exiting once the correct answer has been given. Let us discover together something more about the shell and the `bash` language. For each command used in the following, you can use the linux manual to get more information on it (google is also a good friend in this).

(i) If you open a file, write inside some bash code, save and close it, then you can execute with the `bash` command. Let us do it. Move yourself to a suitable folder, create a new file called `readMyMind.bash`, and write in it the following code.

```
echo ""
echo "Which number am I thinking? [0,32767]"
```

Save and exit, then run it giving the following command.

```
bash readMyMind.bash
```

This will be the beginning of our code.

(ii) Next element to learn is how to let the computer choose its number. It should be randomly (we do not want to cheat). Try to write `echo $RANDOM` on the command line and press return. Do it again. And again. It looks like what we need. Nice!

(iii) Now we need to store this random number somewhere in our script. It is possible to put it in a *variable* which we will call `answer`. And the operation of storing some content in it is done via the `=` operator, which must not be separated by spaces. Something like

```
answer=$RANDOM
```

added to your script will do the job (without spaces around the `=` sign!).

(iv) We should also be able to give a number to the program. Or, said differently, we should let our program wait for a number that should be stored then in a variable. This is done via the `read` command. Writing something like `read myAnswer` will make the program stop and wait for an input. Once we type something on the keyboard and we press return, what we typed will be stored in the `myAnswer` variable and the program will continue.

(v) Since we will hardly win at the first attempt, we need some kind of repetition in our code, a so-called *loop*. Here we will use a `loop`, whose syntax reads

```
while ...; do
    ...
done
```

and we have to fill out the dots. Next to `while`, we should put a condition, which, if satisfied, allows the content of the loop to be executed. A way to read the loop may be: "While the condition is true, do what is written inside". The condition in our script will be

```
while read myAnswer; do
    ...
done
```

(vi) A last thing we need is how to compare numbers. We need three cases.

(a) The number we guessed is *equal* to the answer: we won!

(b) The number we guessed is *less than* the answer: we should guess a larger one.

(c) The number we guessed is *greater than* the answer: we should guess a smaller one.

In words we would say: "If my number is equal to the answer, I won; otherwise if it is larger I need to guess a smaller one; otherwise I need to guess a larger one". In bash this becomes

```
if [ $myAnswer -eq $answer ]; then
    echo ""
    echo "You won!"
    echo ""
    break
elif [ $myAnswer -gt $answer ]; then
    echo -n "Too large! Try again: "
else
    echo -n "Too small! Try again: "
fi
```

Some remarks:

(a) with the `$`-symbol, the content of a variable is obtained;

(b) `-eq` checks if what is left it equal to what is right;

(c) `-gt` checks if what is left it greater than what is right;

(d) blank spaces around the brackets enclosing the if condition are mandatory;

(e) what does the option `-n` of `echo` do?

All together your file `readMyMind.bash` should contain something like the following code.

```
echo ""
echo "Which number am I thinking? [0,32767]"
answer=$RANDOM
echo -n "Try to guess: "
while read myAnswer; do
    if [ $myAnswer -eq $answer ]; then
        echo ""
        echo "YOU WON!"
        echo ""
        break
    elif [ $myAnswer -gt $answer ]; then
        echo -n "Too large! Try again: "
    else
        echo -n "Too small! Try again: "
    fi
done
```

Try to run it and... have fun!!