Tutorial I

October 17

Excercise 1 [*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 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.

Excercise 2 [*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) 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

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

(ii) 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.
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).

(iii) 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`. . .) 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.

(iv) Change the working directory to `~/bin` and do `touch my.fake.program`. Use `chmod` to make it executable, for instance with

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

At this moment you can execute it. Try it (`./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

```
$ 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.

Excercise 3 [*Basic gnuplot usage*] 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.

First, open gnuplot by typing `gnuplot` in the terminal. Although a bit minimalistic, gnuplot comes with extensive documentation through the `help` command. 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 sine curve by typing

```
        plot sin(x)
```

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()
```

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

$$\frac{1}{\sqrt{2\pi}}\exp\{-\tfrac{1}{2}x^2\}$$

(ii) Gnuplot should have been able to plot that quite nicely, but let's tidy it up a little.

(a) Give the plot labels, such as x at the x-axis and P at the y-axis

(b) Restrict the plot to only plot the ranges $x \in [-5, 5]$ and $y \in [0, 0.5]$

(c) Plot tics along the x-axis at intervals of 1, and along the y-axis at intervals of 0.1

(d) Plot special named tics at the points $x = -1.96$ and $x = 1.96$, and name them "95%"

(e) Give the plot a title, e.g. "Gaussian distribution ($\langle x\rangle = 0, \mathrm{var} = 1$)"

(f) Set the sample rate for the function to make the plot smoother

All of the above tasks can be done with gnuplot's `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
```

(iv) Lastly, let us change gnuplot's `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, and not the lines themselves. The lines are generally stylised along with the plot command using the command `with`, such as `plot sin(x) with [options]`. It is once again recommended to look at the documentation for this (`help plot with`), to get an overview of the options available. You could for instance try plotting the above function with points instead of lines and vary the `samples` setting to get a clear idea of what it does.

Excercise 4 [*Advanced shell*] Open a new file called `script.sh` with an editor and write inside the following code

```
#!/bin/bash
# this line is a comment: it is ignored; also blank lines are ignored

for i in $(ls /tmp); do
  echo $i | wc -c >> lengths.text
done
```

Make it executable and run it. Try to understand what happened.

(i) Make a new script, called `script2.sh`. The output of this new script, once executed, must be a file called `name_lengths.text` with so many lines as files in the `/tmp` directory, and each line must contain two entries: the first being the name of the corresponding file in `/tmp`, and the second the length of this file name (for instance, if there was a file called "house" in that directory, the corresponding line would be "house 5").

(ii) Make another script, called `script3.sh`. The output of this one, once executed, should be a file called `name_sizes.text` with so many lines as files in the `/tmp` directory, and each line must contain two entries: the first being again the name of the corresponding file in `/tmp`, and the second its size in bytes.