Technical programming tutorials

April 5, 2016

Chapter 1

A beginner's crash course on Unix

1.1 Setting up your system

As the course is about programming, you need to set up your programming environment. You will need a number of programs and libraries, all of which are readily available free of charge on the web. Due to restrictions and limitations of some operating systems, it is recommended that you either install `Linux` on your computer, or that you install it on a virtual machine (for instance using `VirtualBox`), within your operating system of choice. The following notes will assume that you are running `Ubuntu Linux` (version 14.04 or greater).

The web is crowded with guides on how to install `Linux` on your system or on a virtual machine, hence we will not cover it here. Step-by-step instructions can be found at, e.g., [1] and ready to use `VirtualBox` images are available at different locations. [2]

You will need a text editor to write your code. You are free to use the one that better suits you. Possible choices are `gedit` and `kate` (basic) or `vim` and `emacs` (more advanced).

For what concerns the `C++` part of the course, you will need to install

- `g++` (version >= 4.7);
- `gdb`;
- `valgrind`;
- `make`;
- `cmake`;
- `git`;
- `LAPACK`, along with its C interface `LAPACKE`;
- `BLAS`, along with its C interface `CBlas`;
- `Eigen`;
- `Boost`;

Your Python environment must include

- `Python` (version 2.7, optionally install also Python >=3);
- `matplotlib`;

- `NumPy`;
- `SciPy`;
- `iPython` (enhanced interactive Python shell, optional).

1.1.1 Installing the required packages

To install the required software, run **[NOTE:** Test commands**]**

```
sudo apt-get install git gcc gdb valgrind make cmake liblapack-dev \
                     liblapacke-dev libblas-dev libatlas-base-dev \
                     libeigen3-dev libboost-all-dev
sudo apt-get install python python-numpy python-scipy python-matplotlib
```

Optionally, you may also install the `python3` stack with

```
sudo apt-get install python3 python3-numpy python3-scipy \
                     python3-matplotlib
```

Finally, for the interactive Python shell, run

```
sudo apt-get install ipython
```

and/or `ipython3`.

The command `sudo` allows you to perform system administrator tasks as an unprivileged user, and requires that you input the user's password.

1.2 The Unix shell

This section is meant to be a survival kit for those who have never opened up a shell (or terminal, or console, or command line). The shell is a textual interface that allows the user to input commands and read their output.

Unix command are typically entered in the form

```
command [OPTION]... [INPUT]...
```

Options are usually passed to programs in the form flags that start by a `-` sign (flags, short options) or by a `--` sign (arguments, long options).

1.2.1 Paths and the `cd` and `pwd` commands

The `cd` command is used to move from one folder to another, and indeed it is an acronym for *change directory*. The `pwd` (*print working directory*) command prints the directory the shell is currently open on. . In Unix, the files and folders are organised in a tree-like structure, where the top level entity is called *root*, and indicated with `/`. Files are located by means of *paths*: absolute paths stem from the root, all the way to the file, e.g.,

```
cd /home/user/Desktop
```

would change to the desktop folder of the user `user`. Relative paths stem from the current directory and point to the file, as in

```
cd ../Documents
```

which switches to the documents folder from another folder on the same level. The leading `..` is an example of special path specifiers:

`.` This directory;

`..` The parent directory;

`~` The home folder of the current user;

`-` The previous path (only used alone);

`*` The wildcard, used to specify many paths with a common part. For instance `*.txt` match all files and folders that end by ".txt".

1.2.2 Other useful commands

The following is a non-exhaustive list of useful commands

`man` Followed by another command name, opens its manual page.

`ls` Lists the files and folders in the current directory.

`cat` Followed by one or more files, prints them to standard output in the same order.

`more` Followed by a file, prints it to screen, one page at a time.

`less` Like `more`, and much else (*"less is more"*).

`head` Print the first few lines of one or more files.

`tail` Print the last few lines of one or more files.

1.2.3 Pipes and redirects

Many Unix programs are said to be *filters*. This means that the output of a program can be directly fed to another one. This behaviour is achieved by means of the pipe operator `|`. In

```
command1 | command2
```

the output of `command1` is fed (piped) as input to `command2`.

It is also often useful to redirect the standard output of programs into files:

```
ls > myfile.txt
```

writes the output of `ls` into the file `myfile.txt`. If the file already exists, it is overwritten. If one instead wants to append the output to an existing file, one can substitute `>` by the append operator `>>`.

1.2.4 Process control

Running programs are called processes, and can run in the *foreground* or in the *background*. The difference is that a foreground process blocks the shell that launched it until it terminates. Instead, background processes return the control to the shell, where other commands can then be executed whilst the background process is still running.

To execute a program in the background, a `&` sign is appended to the command string, e.g.,

```
command --option input.file &
```

A program running in the background is assigned a number. It can be recalled in the foreground by issuing the command `fg` followed by the same number. A program running in the foreground may be stopped by issuing `CTRL+Z` in the same shell. A number is assigned to the stopped process, which can be used to move it to the background by issuing the `bg` command.

It is sometimes useful to terminate a foreground program before its execution has completed. To do this, issue `CTRL+C`.

1.3 The Unix way

Over the years, Unix programmers developed a series of standard practices one should keep in mind. [3] We mention a few:

Return 0: Programs that completed successfully their task should return the integer 0 to the shell that called them, and any non-zero value in case of an error. This is to make possible to automate error handling.

K.I.S.S.: *"Keep it simple, stupid!"* Write programs that are as simple as the problem at hand allows. Simple programs are easier to write, read, and less bug prone.

Rule of Repair If your program fails, make it fail loudly. This makes sure that you discover errors early on, and not the day before you turn in an assignment.

Rule of Silence Avoid writing programs that produce tons of useless output. Most Unix programs do not write anything at all when they run successfully. This said, producing some extra output while developing or debugging is perfectly OK.

Precocious Optimisation Optimised code is harder to read, maintain and develop. It is usually better to first have a non-optimised working version of a program before optimising it.

Chapter 2

Python: first steps

Python is a high-level object-oriented scripting language. It is an interpreted language, meaning that one needs not compile the code: the code is compiled on the fly by the interpreter whenever needed. It is designed from the ground up to encourage concise and tidy code, thanks to its broad standard library and to blocks of code being delimited based on their indentation.

The Python standard library is organised in *packages* which contain one or more *modules*. The standard library contains many packages, and it is well documented. The documentation can be browsed online [4] or in an interactive session by using the `help()` function.

A note on versions - Two versions of Python are available: `python2` (simply `python` in `Ubuntu`) and `python3`. The former is currently in maintenance-only mode, while the latter is actively developed. Most of the enhancements of `python3` have been backported in `python2`, and for the purposes of this course, there are two main differences:

- the keyword `print` of `python2` has become a function, `print()`;
- In `python3`, the division operator `/` is always intended as float division. Integer division must be explicitly called with the `//` operator.

2.1 PEP 8

Writing tidy code makes it more readable, understandable and maintainable. Python has got a list of best practices, commonly called `PEP 8` [5], that you are warmly encouraged to follow. In particular,

- do not use lines longer than 80 columns;
- use 4 spaces for indentation (no tabs!);
- leave spaces around operators (`a + b` is ok, `a+b` is not ok).

Adhering to `PEP 8` can also be beneficial when writing in other programming languages, such as `C++` throughout this course.

2.2 Hello Python world!

Following tradition, let us write the *Hello world!* program in Python.

```
print("Hello world!")
```

To execute it, write this one-line program in a file, say `hello.py`, save it[1], and then run from a terminal in the same folder,

```
python hello.py
```

As a result, the program should print "Hello world!" to screen and terminate. The function `print()` causes Python to print its argument to standard output. The text enclosed in single (`'`) or double (`"`) quotes is a Python *string*.

An alternative way to run a Python program is to use a *shebang*. The shebang is the first line of a script and starts by `#!`. It instructs the shell on which interpreter to use to run the script. Let's write a new program, `hello2.py`, which reads

```
#!/usr/bin/env python
print("Hello world!")
```

We can now make this file executable by issuing

```
chmod +x hello2.py
```

At this point the file hello2.py can be run directly by issuing

```
./hello2.py
```

[1]You may also use the interactive Python interpreter, invoking the command `python` or `ipython` in a shell, and entering the one line program.

Chapter 3

C++: first steps

C++ is a low level programming language. Unlike Python, it is a compiled language, meaning that an intermediate step must be added between writing a program and running it. By means of a *compiler*, the code must be translated into *machine code* that the computer is able to execute.

3.1 Hello C++ world

Again, we are going to write the most basic program in C++. Write the following code in the file `hello.cpp`:

```
#include <iostream>

int main() {
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

As we anticipated, we must now compile the program. This is done from the command line by invoking the compiler `g++`,

```
g++ hello.cpp
```

which will generate the file `a.out`. We can now run our program by issuing

```
./a.out
```

For how simple it is, this example already contains quite a bit of technicalities for which we remand to the textbooks. [6]

Bibliography

[1] N. C. Institute, "Install Ubuntu on Oracle VirtualBox," (2016).

[2] Virtualboxes.org, "Ubuntu images," (2016), mind the version!

[3] Wikipedia, "Unix philosophy," (2016).

[4] Python Software Foundation, "Python documentation," (2016), DuckDuckGo users may search directly into Python documentation by using the !python bang syntax.

[5] G. van Rossum, "Style guide for Python code," (2001).

[6] A. Koenig and B. Moo, "Accelerated c++. c++ in-depth series," (2000).