Introduction to Scientific Computing

Dr Hanno Rein

Last updated: September 8, 2014

Contents

1 Linux

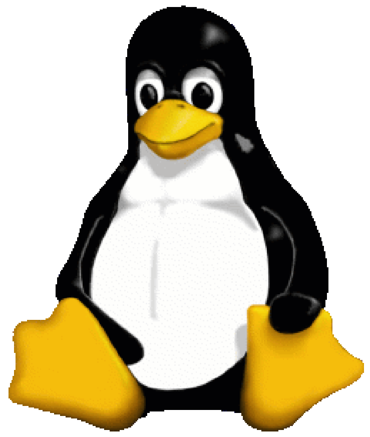


Figure 1: Tux, the official Linux mascot.

Linux is the leading operating system on servers and supercomputers. That's why we are using it in this course. If you go into science and do any sort of computationally intense task, you will almost certainly come accross Linux.

We'll start with a little history lesson of Linux and go back in time more than 40 years. At that time, computers were as big as buildings. Writing software for these monstrosities was very different from writing software today. You first had to understand how the machine works in every single detail. The final software product would only be useful for the one machine you wrote it for.

In 1969 developers at Bell Labs came up with a solution that they called UNIX. The idea was to simplify the software development process. Therfore UNIX was suppoed to be simple, elegant and most importantly, software was supposed to be reusable. Now, when you write a piece of software, you can use a higher level interface to the hardware and you don't need to understand every single register of the computer. And once the program is finished, it will also run without any or only minor changes on another computer that supports UNIX.

In the early 1990s personal computers did not have access to UNIX. They ran MS DOS or Windows 3.1. A computer science student at the University of Helsinki, Linus Turvalds, wanted to have the same advantages that UNIX brought to big computer mainframes to the personal computer. So he started hacking. He came up with Linux, a UNIX-like operating system. But unlike UNIX, which is proprietary, Linux is free and open source. Thus, everyone is allowed to use it free of charge and because the source code is available to everyone, one can even make change to the operating system. Linux follows the same POSIX standards[1] that UNIX uses and shares therefore many of the advantages of UNIX.

If you have not used Linux before, you might find some concepts different from other operating systems such as Windows and MacOS. By default, Linux doesn't come with a graphical user interface. Especially on computer clusters, all user interaction is done via a shell, which we will discuss in the following subsection.

1.1 Shell

A shell, also referred to as a command-line interface, is a text based user interace to access the operating system's services.

[1]It can be treated as POSIX compliant but has never been formally certified.

You will need to get familiar with using the Linux shell to run your programs. We'll go through the most important tasks in the class, but you might want to read up on some more details.

When you have sucessfully logged into a computer, you'll see the command line interface on the screen. This is a prompt waiting for your input:

```
rein001 ~ $
```

Code 1: Shell prompt (bash).

The precise format can vary from machine to machine, but often it follows the following rules:

- The first part is the computer name. Here it is `rein001`.
- Next comes the directory you are currently in. The tilde (∼) indicates that you are in your home directory.
- Then comes a dollar symbol (some people prefer a percent sign or a hash symbol).
- After that, a courser is blinking, waiting for your input.

Let's try and enter our first command: `date`. Press enter to execute the command.

```
rein001 ~ $ date
Sun Jul 13 15:44:16 EDT 2014
rein001 ~ $
```

Code 2: Shell after executing the date command.

As you can see, the command does what you'd expect. It outputs the current date and time on the screen. After the program has finished, we are back at the shell, exactly where we started from.

The general structure of a command is always the same: `command -o2 --option2=value argument1 argument2`. First is the command you want to execute, followed by short and long options as well as arguments. This will become very clear after we go through a few examples.

There are a few tricks that make working with the shell very efficient. For example, suppose we want to execute the same command again. We could just enter it again. However, if the command get very long, this can be cumbersome. If you press the arrow up button, you can cycle through the previously executed commands. When you've found the one you want to reuse, just press enter. You can also edit the command before you execute it.

Now, let us do something slightly more complicted. Most of the time you'll be working in your home direcotry which is typically located at the location `/home/yourloginname/`. To find out where in the filesytem you currently are, use the `pwd` command

```
rein001 ~ $ pwd
/home/rein
```

Code 3: Shell after executing the pwd command.

To find out what files are in this directory use the `ls` command. If you only want to get the names of the files and direcotries, just use `ls`. With the option `-l`, you get more information.

```
rein001 ~ $ ls
directory1  directory2  file1.txt  file2.txt
rein001 ~ $ ls -l
total 8
drwx------ 2 rein rein 4096 Jul 13 16:01
   directory1
drwx------ 2 rein rein 4096 Jul 13 16:01
   directory2
-rw------- 1 rein rein    0 Jul 13 16:01 file1.
   txt
-rw------- 1 rein rein    0 Jul 13 16:01 file2.
   txt
```

Code 4: Shell after executing the ls command.

In the detailed output, the first symbols indicate the permissions of the file, then the username and the user group who owns the file, then the size and the last time it was modified.

Let's create a new directory. This is done using the `mkdir` command

```
rein001 ~ $ mkdir directory3
rein001 ~ $ ls
directory1  directory2  directory3  file1.txt
   file2.txt
```

Code 5: Shell after executing the mkdir command.

As you can see, we have sucessfully created a new directory. Let's go into that directory. This is done with the `cd` command.

```
rein001 ~ $ cd directory3/
rein001 ~/directory3 $
```

Code 6: Shell after executing the cd command.

If we want to go one level up, we type `cd ..`.

```
rein001 ~/directory3 $ cd ..
rein001 ~ $
```

Code 7: Shell after executing the cd command.

To delete a file or a directory, use the `rm` command followed by the directory or file you want to delete. If you are deleting a directory, you need to use the option `-r`.

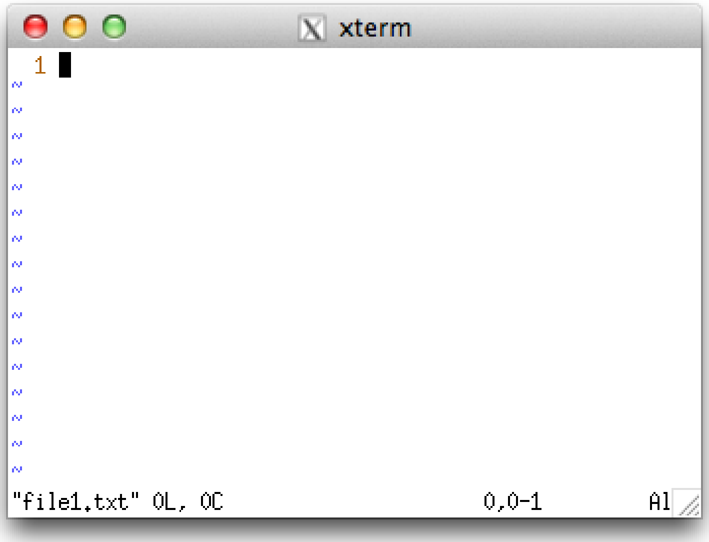
```
rein001 ~ $ rm -r directory3/
rein001 ~ $ ls
directory1  directory2  file1.txt  file2.txt
```

Code 8: Shell after executing the rm command.

1.2 Text editor vi

So far, we have only moved around the file system, created a directory and executed a built-in command (`date`). Let us now do what something more creative: editing a text file. There are many text editors available. If you do a survey of the favourite text editor among programmers, you'll get many different answers. I will be using the `vi` (or `vim`) editor. You can use any text editor you like, but I encourage you to use vi too.

To start editing a (either new or already exisiting) file in the current direcotry, type `vi filename`, for example `vi file1.txt`. The file suffix `.txt` indicates that we are working with a text file (but you can choose any suffix you want). Then, vi starts and shows you a screen like in the following screenshot.



```
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"file1.txt" 0L, 0C                    0,0-1          Al
```

Figure 2: Empty vi screen.

When vi starts, you cannot immediately edit the text, as you are in `normal mode`. In this mode, the editor interprets every keystroke as a command. To enter insert mode, press `i`. Now you can insert text, as well as edit already existing text just as you're used to. To exit insert mode press the escape button on your keyboard. Now you're back in normal mode.

If you want to save the changes to the file, first make sure you are in normal mode by pressing escape. Then, press `:w` followed by enter. The bottom line in vi will indicate that the file has been saved. To exit, again, first make sure you are in normal mode by pressing escape. Then, press `:q` followed by enter. You can also execute the two commands at once by using `:wq`.

2 Version Control System (git)

Git is a revision control system. It helps with maintaining source code of large programs. Git was initially invented by Linus Torvald for the Linux kernel. In the last few years, git has become the industry standard for version control.

There are many reasons why you should always use a version control system, when you develop software. For example, it acts as a backup. If you ever make a change that you later regret, you can turn back time. A version control system also let's you see which changes you made over time. In a nutshell, git keeps a complete history of whatever you have done within the resository.

We will use git in this course to coordinate assignments. You have to learn how to use git in order to submit the assignments. We will only use the most basic functionality of git, so only a few command are needed, which we will discuss now.

2.1 Setting up a repository

Note that if you work on the linux machine provided for this course, you do not need to setup a git repository. This has been done for you. If you wish to use git on your own computer, you can download git for free at `http://git-scm.com`.

Suppose you want to use git to track changes in the direcotry `direcotry3`. Simply go to that directory and tpye `git init`.

```
rein001 ~ $ cd directory3/
rein001 ~/directory3 $ git init
Initialized empty Git repository in
/home/rein/directory3/.git/
```

Code 9: Shell after executing the git init command.

This will initialize git and setup a (hidden) folder with the name `.git` (you can ignore the folder).

2.2 Adding files and committing changes

You have setup a repository. But before you can keep track of files, you need to add them to the repository. This is done with the `git add` command.

```
rein001 ~/directory3 $ vi test.txt
rein001 ~/directory3 $ git add test.txt
```

Code 10: Shell after executing the git add command.

The file is now monitored by git. You can check the status of the file by using the `git status` command.

To make an entry in the repository (this is called a commit) you now type `git commit -a`. A text editor will open and ask you for a commit message. This commit message should summarize what you have done. For example, it could say *"My first git repository"*, or something like *"I finally got it working"*. Note that if you are using the vi editor, you enter the editor in normal mode. First press `i` to enter into insert mode, then type your commit message. Exit the editor by pressing the escape key, followed by `:wq`.

Now the changes have been commit into the local repository. You can verify that by typing `git log` as well as `git status`.

```
rein001 ~/directory3 $ git commit -a
[master (root-commit) d5f9e43] First commit
1 file changed, 1 insertion(+)
create mode 100644 test.txt
rein001 ~/directory3 $ git status
# On branch master
nothing to commit (working directory clean)
rein001 ~/directory3 $ git log
commit d5f9e439091078d56232f19b93bd2a51314996c1
Author: Hanno Rein <hanno@hanno-rein.de>
Date:   Sun Jul 13 20:52:43 2014 -0400

      First commit
rein001 ~/directory3 $
```

Code 11: Shell after adding and commit files to git.

The next time you edit the files and want to commit the changes, just use `git commit -a`. You only need to use `git add` when you want to add new files.

So far you we have only manipulated our repositry on the local machine. The power of git becomes apparent when multiple repositories on different machines can be kept in sync. We will make use of this feature for the assignments.

3 Assignments

Assignments for this course work differently than for any other course. As the successful submission of assignments is mandatory, it is key that you understand how to do that. We'll go over it now.

3.1 Registration

Before you can submit assignments, you need to register for the course. To do that, visit `http://rein001.utsc.utoronto.ca`. This setup is free and should only take a few minutes. Please make sure you are using your **full name**. It should have the exact same format as on university documents. When asked for an e-mail address, you have to use your UofT e-mail address. Other e-mail addresses will not be accepted.

After your account has been approved by an instructor, you will receive an e-mail with details on your account and how to submit assignments.

3.2 Remote access using SSH

You are encouraged to do the assignments on the linux machine which is solely provided for this course. You can login remotely to this machine either from your own laptop or from the computer lab. To do that, you need to have an SSH client installed. On MacOS and Linux, this comes with the operating system by default. On Windows, you need to install one by yourself. Putty is the recommended choice. It can be found at `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`.

The hostname of the linux machine is `rein001.utsc.utoronto.ca`. You need to login with your username and password provided in the automated e-mail you should have received after the registration.

3.3 Using git to submit assignments

After you have logged in to `rein001.utsc.utoronto.ca`, you are in your home directory. You'll see a sub-directory called assignments. Move into that directory. This is your main git repository for assignments. You'll find the assignments in this directory. Before you start, make sure you have the latest version of the upstream repository containing the assignments. To do this, type `git pull`. This will pull all changes into your repository and merge them. Unless you have edited the assignment files themselves, this is a fast-forward merge.

```
hannorein@rein001:~$ ls
assignments
hannorein@rein001:~$ cd assignments/
hannorein@rein001:~/assignments$ ls
FirstAssignment.txt  README.md
hannorein@rein001:~/assignments$ git pull
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (7/7), done.
From github.com:PSCB57/assignments
 * branch            master     -> FETCH_HEAD
Updating 7719bf1..3f541ab
Fast-forward
 README.md                    |    2 +-
 assignment1/assignment.txt  |   10 ++++++++++
 1 file changed, 11 insertions(+)
 create mode 100644 assignment1/assignment.txt
```

Code 12: Pulling new assignments.

Now, there is one directory for each assignment, labelled `assignment1`, `assignment2`, etc. Go into the assignment you want to work on and read the assignment, for example with `cat` or `vi`.

```
hannorein@rein001:~/assignments$ cd assignment1/
hannorein@rein001:~/assignments/assignment1$ cat
   assignment.txt
Assignment 1
------------

1) Create a new file with the name 'hello.txt' in
    the directory 'assignment1'.

2) In the first line of the file, write your name
   . Write your UofT e-mail address in the second
    line. Write your student number in the third
   line.

3) Save the file, add and commit it to the git
   repository.
```

Code 13: Reading an assignment.

Do not edit the assignment file. This will complicate future merges.

Now, work on the assignment. Make sure you stay in the right folder. When done, add the files you want to submit to git and commit the changes.

```
hannorein@rein001:~/assignments/assignment1$ git
   add hello.txt
hannorein@rein001:~/assignments/assignment1$ git
   commit -a
[master a4b10e2] My first assignment.
 0 files changed
 create mode 100644 assignment1/hello.txt
```

Code 14: Adding and commit files for an assignment.

That's it. Your instructor will be able to see your submission now.

4 Computers

A computer is a machine which can perform a set of calculations. The purpose of this course is to give you the ability to tell the computer which calculation to perfom in order to solve a given problem.

This section of the course can only give you an extremely limited overview. I encourage you to read up a bit more on computer architecture, especially if you plan to dive deeper into the subject.

4.1 How computers work

A computer consists of two main components, the memory and the processor. The memory stores commands and all sorts of data. Memory comes in the form of hard drives, RAM, a L1 caches or USB sticks. Some of these are fast, others are portable or very cheap. For us, the distinction between them is not very important. The processor, also called CPU, reads commands from the memory and executes them. Here are some examples of possible instructions:

- beep
- add regsiter a and register b and store the value in register c
- move memory content at address 123 to register a
- skip the next instruction
- go back 10 instructions

We could just write these instructions by hand and enter them one by one into the memory. This would involve a lot of repetitive work, reading lots of manuals and would take a long time. And indeed, just 40 years ago, this was the only way to get any *programming* done. Today, the computers are fast enough so that we can add a layer (or two) of abstraction between this very basic level of commands (called machine code) and what we actually have to write in our programms.

4.2 An example: Fibonacci numbers

Let's construct a computer. This is a very simple example, but it illustrates the ideas that are used in more complex systems.

We'll construct a computer with 16 memory locations. We call those registers and give them a unique number, from 0 to 15. Our computer has only seven commands:

- ADD r1 r2 r3
  This command adds the value stored in register r1 to the value stored in r2 and stores the result in r3.
- SUB r1 r2 r3
  This command substracts the value stored in register r2 from the value in r1 and stores the result in r3.
- IF r1 r2
  This command only executes the next command if the value in register r1 is bigger than the value in register r2.
- COPY r1 r2
  This command copies the value in register r1 to register r2.
- SET s1 r1
  This command sets the value in register r1 to the number s1 (s1 is just a number, not the address of a register).
- JUMP r1
  This command moves the instruction pointer by the value of r1.
- PRINT r1
  This command prints the value in register r1 on the screen.

Note that the instruction counter is increases by one at every command.

We will not use this set of commands to calculate the Fibonacci numbers. This is a series of number that occurs frequently in nature. The first few numbers in the sequence are:

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$

They can be calculated by the recursion formula

$$F_n = F_{n-1} + F_{n-2}$$

i.e. the next number is the sum of the previous two numbers. The first two numbers are set two one. Let's now implement this in our Assembler language

```
To be completed in tutorial.
```

Code 15: Simple assembler program to calculate Fibonacci numbers.

As a comparison, here is the same program in python

```
To be completed in tutorial.
```

Code 16: Python program to calculate the Fibonacci numbers.

End of lecture 1.