Introduction to the operating system LINUX/UNIX

Peter Reinhardt
Laboratoire de Chimie Théorique, Sorbonne Université et CNRS UMR 7616,
4 place Jussieu, 75252 Paris Cedex 05, France

A computer does not necessarily run under Windows, notably in a context where computational power is needed for calculations and not for treatment of text or graphics. We give some elements to be able to work with LINUX.

Some notions of UNIX

UNIX serves to tell a computer what you would like it to do for you. We speak thus of an operating system which has, in the case of UNIX, the advantage to be present at any size of a computer, from a tiny Raspberry Pi to the world's largest super computers. Standard tasks are the creation of folders (directories), handling of files, creation of input data for scientific programs, printing of results, archiving and transfer of data, and graphical treatment of the results.

Linux distributions come today with graphical interface and icons as on a normal PC. But behind any command via an icon there is a UNIX command which can be launched via a terminal without any need for a graphical interface.

UNIX commands are rarely intuitive, even for experts of the English language. In these few pages we will encounter some important commands. Anyway, once lost in the jungle of possible commands, one can always try the line **man** ⟨**command**⟩ for recalling the details of the functionality of a command. By the way, there are today a number of "dialects" of Linux/Unix as every constructor of computers has developed his own Unix.

A command line has always the syntax

```
command −options arguments
```

1. Help!!

If the screen shows a mess of lost characters – output of several programs running in parallel for instance, you wipe the screen with ⟨$CTRL$⟩ `l`.

A running program which goes astray can be stopped immediately with ⟨$CTRL$⟩ `c`.

A. Managing files

On a hard disk files are organized in directories with an arborescence.

We "walk" around with the command **cd** (i.e. change directory). Either with respect to the spot where we are (the name can be found with **pwd** as "present work directory") via **cd ..** or **cd my-sub-directory**, or with an absolute path via **cd /home/my_login/my_directory/my_sub_directory**. The content of a directory is listed with the command **ls** which allows many options: for example **ls −l** shows details of every entry, **ls −lt** lists the files in chronological order, **ls −F** adds file types etc. . **mkdir**

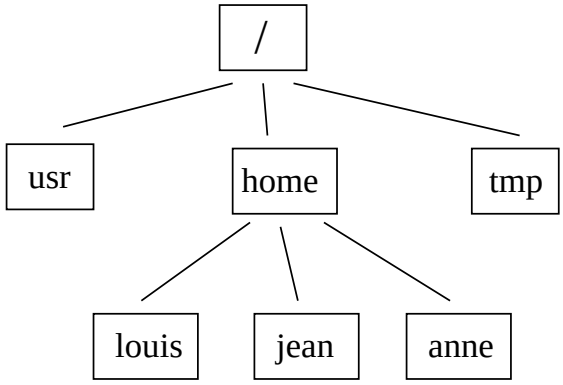

FIG. 1: Example of a UNIX arborescence.

my_sub directory creates a new directory, **rm my_file** erases "my_file" and **rm -r my_sub directory** deletes the whole sub directory.

Avoid white spaces in file or directory names. In the same sense accents can cause trouble for UNIX !

For copying a file we use **cp**. Copying entire directories is done with the option **cp −R**. We displace or rename a file or directory with the command **mv**.

B. File editors

Experts use **emacs** or **vi**. We may resort to **gedit**, **nano** or **nedit** which run on separate windows, but after a while one needs something more powerful.

Let us have a look at **vi** – a good introduction is obtained with the command **vimtutor**.

1. Editing within the page

We walk around in a page with the arrows ←, ↓, ↑, → or, if they do not work, with the letters "h", "j", "k" et "l". The beginning of a line is attained with "0" (zero) and the end with "$". "G" finds us at the end of a file. "$\langle CTRL \rangle$ f" advances one page and "$\langle CTRL \rangle$ b" goes back one page. The lowercase "x" erases the character under the cursor, "D" deletes all characters until the end of the line. "P" re-inserts the deleted characters before the cursor and "p" inserts them behind the cursor. "dd" deletes one line, "10dd" deletes ten lines.

This seems very complicated, but a few sessions using the editor shows the efficiency of **vi** (at least on an English keyboard).

2. The insertion/replacement mode

By typing "i" we access the insertion mode, and we can enter text. Please avoid the arrows as they work differently on different flavors of "vi". If an error occurs, we can use "backspace" (⟵). Quitting the insertion mode is done with the $\langle ESC \rangle$ key.

"r" replaces a single character with another one; "R" replaces some text, and "s" one single character with an arbitrary text. Insertions and replacements are as well terminated with the $\langle ESC \rangle$ key.

One adds something to a line at the end by "A" or before the beginning with "I", and one creates a new line with "o".

3. Explicit commands

These commands are accessible via ":", "/" or "?": on the footer of the page appear a ":" (or "/" or "?") as command line. ":wq" saves the file you are editing (there is no automatic backup!)and leaves the **vi**. ":r file" includes *file* into the file you are working on. "/" allows to forward search for a pattern, and "?" is the backward search command. ":q" quits the **vi** and ":q!" quits immediately without saving changes, ":w! name" saves the content to a file `name` without paying attention whether the file exists already or not.

C. Execution of a program and process control

To nay action, UNIX assigns a process with a unique process number. The command **ps x** shows all the processes of a user. A UNIX command is normally run interactively, but can be sent to the background by adding a "&" to the end of the command If a command does not what you wanted it to do, or simply does nothing, you can kill it immediately with $\langle CTRL \rangle$ c. If the command is already in the background you have to find its process number with **ps x** and kill it explicitly with **kill −9 process_number**.

Imagine you created a command which needs some data, and prints the result of the screen. We may furnish the data one by one until the necessary data are transmitted. And the result arrives on the screen. This may be useful once, but in the long run we would like to prepare the data before, and save the result to a file.

So we use "<" et ">", the former for providing the data, and the latter for redirecting the output to a file. : **command < data_file > output_file**. Nothing is now printed to the screen — the command terminates quietly (at least that is what we expect). And we can consult the output with **vi**. By the way, the file content can be listed with the command **cat** without edition of the file.

Another way to proceed is to use a "pip" and the command "| **tee**" which splits the output stream to the screen and to a file: **command < data_file | tee output_file**.

D. Recap of some UNIX commands

- **cd** : change directory
- **ls** : list the content of a directory, with the options
  - –l : show details
  - –t : in chronological order
  - –R : recursively with sub-directories
  - –a : listing as well hidden files or directories, with names starting with a point.
  - –F : add information on the file or directory types.
- **mkdir** : create a directory
- **mv** : move or rename a file or directory
- **cp** : copy a file or directory
- **rm** : erase a file ; **rm −r** : erase a directory
- **du** : list disk use of the present directory ; options
  - –s : show only the global use
  - –h : show in readable format (k, M, G for kilo, mega or giga bytes)
- **df** : show disk usage, good for verifying whether you saturated a disk.
- **ps x** : lists *your* active processes.
- **top** : lists the most active processes at the moment.
- **kill −9** : kill one of your processes.
- **nedit** : simple file editor
- **vi** : the more efficient file editor
- **cat** : list the content of a file
- **tee** : split the output to screen for a copy to a file ; to be used with the pipe |, for example **ls −l | tee content_of_my_directory**.
- $*$ replace any string; ? replaces one single character
- $<$ redirection of an entry (read data from a file **program** $<$ **data**)
  $>$ redirection of the output to a file (e.g. **program** $<$ **data** $>$ **output**).