Refresher course in Computer Science

- 5 weeks ; 50 hours in total

- A lot to do (not in that order):
  - Linux basics / using a shell
  - Compile (makefile) and debugging (gdb)
  - C language
  - Fortran language
  - Unix system

- How we’ll work:
  - A few lectures,
  - More practical work,
  - Some short projects,
  - One written exam,
  - A last bigger project.

- Personal implication:
  - take good uses,
  - practice programmation,
  - read, search on the web
  - ...

- Questions are welcome!

- Documents available at
  http://www.lifl.fr/~poteaux/teaching.html
The patriot bug

In 1991, a Patriot anti-missile failed to intercept a Scud missile. 28 people were killed.

- The code worked with time increments of 0.1 s.
- But 0.1 is not representable in binary.
- In the 24-bit format used, the number stored was 0.099999904632568359375
- The error was 0.0000000953. After 100 hours = 360,000 seconds, time is wrong by 0.34s.
- In 0.34s, a Scud moves 500m
A real story (appeared at CERN)

Using a IA32/x87 FPU (provides a 80 bit double extended format):
- Use the (robust and tested) standard sort function of the STL C++ library
- to sort objects by their radius: according to $x \times x + y \times y$.
- Sometimes (rarely) segfault, infinite loop...
- Why? Because the sort algorithm works under the following naive assumption: if $A \nless B$, then, later, $A \geq B$
  - $x \times x + y \times y$ inlined and compiled differently at two points of the program,
  - computation on 64 or 80 bits, depending on register allocation
  - enough to break the assumption (horribly rarely).

No programming mistake! (thus very difficult to fix)
Another example: equality with floating point numbers

If you compile and run the following C program

```c
#include <stdio.h>
int main()
{
    int i;
    float ref,index;
    ref = 169.0 / 170.0;
    for ( i = 0; i < 250; i ++ ) {
        index = i ;
        if (ref==(index/(index+1.0))) break;
    }
    printf ("i = %d\n" , i ) ;
    return 0;
}
```

(gcc file.c ; ./a.out) you get the following printed on the screen:

```
i = 250
```
First steps on Linux

Adrien Poteaux

LIFL, Université Lille 1

Year 2014-2015

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

http://creativecommons.org/licenses/by-nc-sa/3.0/
How a processor is working?

- **User level:**
  - Calling programs
  - Function calls
  - System libraries
  - User programs

- **Kernel level**
  - Filesystem
  - Dealing with memory
  - Managing processes

- **Hardware level**
Using the hardware
- order the different executions
- dealing with the rights (security)

Communicate
- using the RAM (Random Access Memory),
- using persistent memory (hard drive),
- by specific data structures (message queue, semaphore...)
Shell?

- Not part of the OS (it is a process)
- Interface between the user and the OS
- Used to run commands / filters

- Stdin, 0: standard input
  *(default: keyboard)*

- Stdout, 1: standard output
  *(default: screen)*

- Stderr, 2: error output
  *(default: screen)*

- Other files

- One filter = one identifier *(ls)* + options *(−al)* + parameters *(/bin)*
- Each process returns a *return code* (one byte → echo $?).

adrien.poteaux@lifl.fr

Linux?
Shell parameters

space, tab separators
Enter sends the expression
\ escape character
| concatenates the input / output of 2 filters
& starts the process in background
; separates 2 expressions
() group of computation
< and << redirection of the input
> and >> redirection of the output
|| logical OR
&& logical AND
Absolute path (from the root)

```
emacs /home/poteaux/CS/slides-shell.tex
```
Absolute path (from the root)

root

- home
  - poteaux
    - file
    - file
    - CS
      - file
      - slides-shell.tex
  - bin
    - ls
    - cd
    - rm
  - directory
    - file
  - tmp
    - toto.txt

directory

emacs $HOME/CS/slides-shell.tex
Absolute path (from the root)

```
root
  └── bin
  │    ├── ls
  │    │    └── file
  │    └── cd
  │         └── file
  └── directory
       ├── file
       └── toto.txt
           ├── tmp
           └── home
                ├── file
                └── poteaux
                        ├── file
                        └── CS
                                ├── file
                                └── slides-shell.tex

emacs ~/CS/slides-shell.tex
```
Relative path (from the current directory)

```
root
  ├── home
  │   └── file
  ├── bin
  │   ├── ls
  │   └── cd
  └── directory
      ├── file
      └── toto.txt

poteaux
  └── file

CS
  └── slides-shell.tex

adrien.poteaux@lifl.fr
```

Filesystem 6/20
Relative path (from the current directory)

cd
t มี panic CS/slides-shell.tex &
cd CS ;emacs slides-shell.tex
Relative path (from the current directory)

```
root
  ↓
home
  ↓
poteaux
  ↓
CS
  ↓
file
  ↓
slides-shell.tex

bin
  ↓
ls
  ↓

bin
  ↓
rm
  ↓
file
  ↓
directory

 tmp
  ↓
toto.txt
```

```
cd /directory
/bin/rm file
```
root

home

bin

directory

tmp

poteaux

file

CS

file

slides-shell.tex

cd /directory

rm file

(try echo $PATH)
Redirections

```
cd; ls CS > /tmp/toto.txt
```
Redirections

```
cd ; ls CS > /tmp/toto.txt
ls CS >> /tmp/toto.txt
```
A graph organisation

```
root
  ├── bin
  │    └── ls
  │    └── cd
  │    └── rm
  └── home
       └── poteaux
            └── file
            └── link
            └── cs
                 └── file
                 └── slides-shell.tex
  └── directory
       └── file
       └── toto.txt
```

```
cd ; emacs ../poteaux/./link
```

similar to

```
emacs /directory/file
```
Rights on a file

- Encoded on 10 bits.
- First bit = type of the file
  (d for directory, l for link, - for a classical file...)
- 3 groups of 3 bits (user, group, others):
  - r if the file is readable,
  - w if one can modify it,
  - x if one can execute it.
- option -l of ls:
  ```
  pot@computer:shell$ ls -l slides-shell.tex
  -rw-r--r-- 1 pot pot 17352 sept. 6 14:55 slides-shell.tex
  ```
- chmod: change a file's rights.
Variables

- **Definition:** \$ FOO="Hello world"
- **Evaluation:** use the prefix \$

  \$ echo FOO  \$ echo $FOO
  FOO              Hello world

- **Everything is a chain character**
  - quotes ' ' \(\rightarrow\) disable evaluation,
  - quotation mark " " \(\rightarrow\) evaluation then character chain,
  - backquotes ` ` \(\rightarrow\) chain evaluated as a command.

  \$ echo 'F00'
  F00

  \$ echo "echo 'F00'"
  echo 'Hello world'

  \$ BAR="anything you want" ;%
  echo $BAR
  anything you want

  \$ BAR=`anything you want`
  anything: Command not found.
Command-line editing.

- ↑ and ↓ ⇒ scroll through your bash history
- Ctrl + r followed by a word ⇒ searches in the history,
- Ctrl + l ⇒ clears the terminal (clear command),
- Ctrl + d sends the end of file (EOF)
- Shift + PgUp and Shift + PgDown ⇒ scroll up / down,
- Ctrl + a ⇒ beginning of the line (Home),
- Ctrl + e ⇒ end of the line (End),
- Ctrl + u deletes every character before the cursor,
- Ctrl + k deletes every character after the cursor,
- Ctrl + w deletes the first word on the left,
- Alt + d deletes the first word on the right,
- Ctrl + y paste deleted text (via Ctrl + u, Ctrl + k, Ctrl + w or Alt + d).
- the Ctrl key *amplifies* editing functions
Regular expressions

- ? is any character (b?l ↔ bol or bal or bwl or b3l or ...)
- brackets [ ] specifies a set of characters
  - dupon[dt] is either dupond or dupont (no other choice).
  - dupon[d-t] is dupon plus any character between d and t.
  - dupon[^dt] is dupon plus any character except d and t.
- * is 0, 1 or any number of any character.
  - ls *.c lists every file ending by .c in the current directory.
Process informations

- A lot of informations stored
  - PID (process identifier),
  - PPID (father),
  - state (running, zombie, stopped...)
  - many others...

- Several commands to get informations: top, ps, jobs...

- One example:

  ```
  $ ps -ef | grep emacs
  poteaux  4223  3442  0 13:56 pts/0  00:00:31 emacs slides-shell.tex
  poteaux  9157  3442  0 15:39 pts/0  00:00:00 grep emacs
  
  $ kill 4223 would kill emacs.
  ```
Background and foreground

- **Foreground work**
  - command
  - command &

- **Background work**
  - fg %job
  - bg %job
  - stop %job
  - kill %job

- **Stopped work**
  - fg %job

- **Ended**
  - Control-c
  - Control-z
Use the good tools!

- The key command: `man`. Uses the `less` viewer:
  - `h` will display the available commands,
  - `q` will quit the viewer,
  - `/` followed by an expression will search the next apparition of this expression in the document, move to the first results, and highlight the others,
  - `n` moves to the next apparition of the searched expression,
  - `N` moves to the previous apparition of the searched expression,

- Use a *programming* text editor (emacs, vi). First contact is frightening but worths it!
  - Print the refcard!
  - Learn some shortcuts! (try not to use the mouse)

- Read more (links on the webpage) ! Practice!
  - *An Introduction to the Linux Command Shell For Beginners*,
  - *Unix Programming Tools* (sections 4 and 5 at the moment).
Scripts and control structures

- Basic features on shell
Script: short definition

- File containing shell commands.
- Interpret the file:
  - `$ . foo` (in the current shell),
  - `$ sh foo` (in another shell),
  - ...
- Run the file directly:
  - Need to be executable (`chmod`)
  - *Magic number:* `#!/bin/sh`
  - interpreted in a another shell

**Remark:** `/bin/sh` is just a link to the shell interpreter you are using. It can be for instance `bash`, `dash` or `csh`. You can see which version you are using via the command `$ ls -l /bin/sh`
Shell parameters

- number, special character or name (sequence of alphanumeric characters that is not a number or a special character)
- Special and positions parameters (=arguments of the command):
  - 0: current command name;
  - #: number of position parameters;
  - *, @: all the position parameters;
  - 1 to 9: the 9 first position parameters;
  - x: the position parameter x(>9);
  - $: the pid (process identifier) of the current command;
  - _: the last used parameter;
  - -: the flags (options) of the current command;
  - ?: the exit-status of the last command runned.
    - everything ok: ? = 0,
    - something anormal: ? ≠ 0;
  - !: the pid of the last run process in background
- The shift command shifts the numbered parameters (1 is lost and # is updated)
The test function

- Checking files type or making compareasons,
- Either test EXPRESSION or [ EXPRESSION ]
- [ -f FILE ], [ -d FILE ],...
- [ $A -ge 2 ], [ $A -eq $B ],...

Have a look at man test!
Some control structures

- **Simple condition:**
  ```
  if instruction-test
      then instruction
  elif other_instruction-test
      then other_instruction
  else last_instruction
  fi
  ```

- **Multiple conditions:** the function `case`
  ```
  case expr in
      regular_expression_1) instructions;;
      regular_expression_n) instructions;;
  esac
  ```

  Once a condition has been satisfied, the other conditions are not executed (this is different in C).
Some control structures

- Numerated iteration:
  For var in expr do
    instruction
  done

  - If expr is empty, the instruction is not executed,
  - in expr can be omitted; by default, this will be in "$@".

- Conditionnal iteration
  while instruction-test do
    instruction
  done

- Inverse conditionnal iteration
  until instruction-test do
    instruction
  done
A few details

- Position parameters = arguments of the command

- new line terminates the command:

```bash
if [ $# -ne 2 ] ; then echo "pb" ; fi ⇐⇒ if [ $# -ne 2 ]
then echo "pb"
fi
```

A function follows the following syntax:

```bash
name_of_the_function()
instructions
redirection of an exit status
```

An example of a basic script in the lecture notes (section 1.8.2).
How to send projects

- One archive by email: adrien.poteaux@lifl.fr,
  - `tar czf`; cf `man tar`
  - `tp1-surname.tar.gz`

- It should contain:
  - The requested files (clean code - indentation, comments are welcome - put your names again there),
  - A makefile!
  - Potentially a README (any general comment you want to make...),
  - No unnecessary file (backup files, intermediate compilations files, executables that are generated by your code...).