Laboratory 1

1. **Attendance requirements (see course website)**
2. **Practical assignments grading (see course website)**
3. **Connection to campus/exam servers:**
    a. `ssh -p 8937 username@www.scs.ubbcluj.ro # from home`
    b. `ssh username@linux.scs.ubbcluj.ro # from campus`
    c. `ssh username@172.30.0.9 # from campus`
4. **The Command-line: Why?**
    a. It is the most powerful and flexible way to get things done in an operating system, provided that you know what you are doing
    b. Hundreds of programs already written that can do things for you already and can work and interact with each other (practically a Lego set of programs). Using them properly yields solutions fast.
    c. No real production program is run from Eclipse, IntelliJ, Code Blocks, Borland, ..., but from the command line
    d. Many times, the command line is all you have available to interact with an OS (especially in the cloud)
    e. Very cool, although ugly: cryptic one-liner for what would require lots of lines of C code. What are the top ten most frequent names of our students?

```
grep -E /home/scs /etc/passwd | awk -F: '{print $5}' | grep -E -v "^ex_|\." | sed -E "s/ /\n/g" | grep -E -i "[a-z]{3,}" | tr "A-Z" "a-z" | sort | uniq -c | sort -n -r | head -10
          83 andrei
          71 alexandru
          53 mihai
          47 maria
          41 daniel
          34 bogdan
          32 vlad
          31 andreea
          31 alexandra
          31 adrian
```

- The command line is a program that allows us to run other programs
- Students probably ran programs before by clicking on icons or clicking "Run" in a development environment like Borland. The command line is just another way of running programs, and it is the primary way of running commands in production environments.
- Can be found in any OS:
    - Linux: Sh, Bash, Ksh, …
    - Windows: Cmd, Powershell, …
    - MacOS: Terminal (actually Bash)
- We will use Linux, and the command line is case sensitive: `mkdir` is not the same thing as `MkDir`, nor is `hello.c` the same as `Hello.C`

5. **Tricks and shortcuts:**

The mouse is really neat for copy/paste. Everything you select in the console is already copied to the clipboard. Right-click (Windows) or middle-click (Linux) will paste.

- There are several keyboard shortcuts, which although identical to those used in Windows, do totally different things
  - `Tab` - Autocomplete
  - `Up/Down arrow` - navigate through the history of commands
  - `Ctrl-C` - Stop the currently running program; really useful when you have an infinite loop
  - `Ctrl-S` - Lock the console. You will be tempted to save your work with this, and then get confused.
  - `Ctrl-Q` - Unlock console. This is the antidote for `Ctrl-S`. Note that everything you typed while the console was locked will show up after you press this.
  - `Ctrl-Z` - Suspend the execution of the current program. Do not use this to "undo" anything
  - `Ctrl-A` - Jump to the beginning of the line
  - `Ctrl-E` - Jump to the end of the line
  - `Ctrl-F` - Move forward on character
  - `Ctrl-B` - Move backwards on character
  - `Ctrl-D` - End of file when providing program input. When pressed on the first position of the command line, ends the connection (similar to running `exit`).
  - `Ctrl-R` - Search through the history of commands
  - `Ctrl-K` - Cut the text from the current position to the end of the line
  - `Ctrl-Y` - Paste what was cut with `Ctrl-K`

6. Commands and paths:

A command is a program, any program
We will only use commands that work in the console, meaning the interface is exclusively text
To run a command, type its name followed by whatever arguments are necessary, everything separated by space.

Commands:

- o list the content of the current directory run `ls`
- To see the content of file `/etc/passwd`, use `cat /etc/passwd`. Here, `/etc/passwd` is an argument
- To see the content of the current directory, run `ls -l`. Here, `-l` is an argument too
- To see the content of the current directory, with all the details, and including the hidden files, run `ls -l -a` or `ls -l --all`. Here, `-a` and `--all` have the same effect, one being the short form, and the other being the long-form.
- To create a directory name `abc`, run `mkdir abc`
- To display the content of the current directory, with all the details but without the annoying colors, run `ls -l --color=never`. Here, `--color=never` is an argument with value. Sometimes, the equal sign is not necessary, but always consult the manual (command `man`) or the `--help` option (eg `ls --help`)
- To do the same thing above, for the directory `/etc`, run `ls -l --color=never /etc`

Paths:

- UNIX file system has a single root, unlike Windows which has a root for every drive mounted (ie C:, D:, etc)
- The UNIX file system root is `/`, and all drives are mounted as directories, somewhere in the file system
- The UNIX file separator is /, unlike Windows, where the separator is \
- Every user has a home directory, which is the current directory when you connect over SSH. Run command `pwd` to find the path to your current directory.

Go through the ideas above explaining the command structure:

- Space is separator
- First word is the command
- Next words are arguments
  - Values: ls /etc
  - Options
    - Short form: `ls -l`
    - Long form: `ls --all`
    - Short form with value: `cut -d : -f 1,2,3 /etc/passwd`
    - Long form with value: `cut --delimiter=: --fields=1,2,3 /etc/passwd`
    - Combined short forms: `ps -e -f` is equivalent with `ps -ef`

Chaining commands with pipe:

- A pipe | takes the output of the command before it and passes it as input to the command after it
- If you want to display the first 5 lines, in alphabetical order, of a file you need to first sort it, and then take the first 5 lines. That means redirecting the output of the command `sort`, to the input of the command `head`:
  `sort a.txt | head -n 5`
- If you now look again at the very long command in section 3, you will notice it makes heavy use of connecting commands through pipe. This is a very popular practice in command line usage.

7. THE MANUAL:

a. If you need to learn about a command or C function or many other things, you can use the built-in manual pages. To read about command `ls`, run command `man ls`, to read about C function `pthread_create` run `man pthread_create`

b. The manual will open in the pager, you need to know how to navigate and exit the pager
   i. Page Down: `SPACE` (you can also use the `PgDn` key, but in some rare and weird cases suspends the command)
   ii. Page Up: `b` (you can also use the `PgUp` key, but with the same risk as above)
   iii. Search: `/`
   iv. Exit: `q`

c. Structure
   i. Synopsis
   ii. For C functions, list of headers to be included
   iii. Arguments
   iv. Return value
   v. See also

d. Finding the manual page you need
    i. `apropos ls`
    ii. `whatis ls`
e. Manual sections
    i. `man open`
    ii. `man 2 open`
f. Bash built-in commands manual pages
    i. Some built-in Bash commands do not have their own manual pages, but rather they appear in the bash manual page
    ii. This is not applicable if the proper manual pages are installed
    iii. However, if they are not installed, run `man bash` and scroll a lot to find details about `cd`, `read`, `shift`, `fg`, `bg`, `jobs`, ...

Graduation Exam:
https://www.cs.ubbcluj.ro/wp-content/uploads/tematica-licenta-informatica-engleza-2021.pdf

8. **Editor:**

Configure your editor. You can do this for Vi/Vim. Emacs, Nano, Joe and Micro, but the example below is for Vim. If you want to configure other editors, search their documentation for the config file name and location, and the values you need to write to achieve the same things as below.

a. Create/edit file `~/.vimrc` and write in it the lines below, to enable syntax coloring, tab size of 4 and tab insertion as spaces

```
syntax on
set tabstop=4
set expandtab
```

b. Explain `~` as alias for the home directory
c. Explain that files having their name starting with `.` are hidden (`ls -a` was mentioned above)

9. **C Programming:**
    a. Discuss the main function declaration and establish the form below as the standard main function declaration:
        i. Returns `int`
        ii. Takes arguments with which we can access the command line arguments
        iii. Returns `0` if all goes well (the truth value of the return/exit value will be discussed later on)
    b. Compile it with `gcc -Wall -g -o hello hello.c` and explain why we want each argument and insist that they be used all the time
    c. Run the program as `/home/scs/an1/gr211/abir1234/hello` (the students will need to first run `pwd` to get their absolute path)
    d. Run it again as `./hello` and explain `.` and `..`

```c
// Print all arguments received from the command line
#include <stdio.h>
int main(int argc, char **argv) {
     int i;
     for(i = 0; i < argc; i++) {
          printf("%s\n", argv[i]);
     }
     return 0;
}
```

```c
// Print a greeting message to a name given as an argument
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
     if(argc < 2) {
          printf("Please provide at least one argument\n");
          exit(1);
     }
     printf("Hello, %s\n", argv[1]);
     return 0;
}
```