# operating systems lab - week 5: exercise

Nicolo Colombo

October 27, 2020

This lab is about the UNIX shell and `bash` programming. In the first part, you will be practising with the main features of the UNIX shell. In the second part, you will write and run a `sh` program. In the end, you will write a C program that simulates the behavior of the shell and executes your `sh` script in an interactive way. Please try to complete all sections of this lab sheet before attempting this week's Moodle quiz

lab quiz - week 5

as some of the quiz questions may require you to run, comment or modify the programs you are asked to write.

## Set up

Depending on your OS, use the following instructions to connect to `linux.cim.rhuk.ac.uk`:

**Unix** Open the terminal and run

```
ssh yyyyxxx@linux.cim.rhul.ac.uk
```

where `yyyyxxx` is your college username, and enter your password to access the teaching server.

**Windows** Launch the Windows SSH client `puTTY` [1], enter the following

```
linux.cim.rhul.ac.uk
```

in the empty field *Host Name (or IP address)* and click on *Open*. The client opens a new window where you are required to enter your college user name `yyyyxxx` and password.

Once logged in, you should be able to see the content of and navigate in your home directory using the standard UNIX commands, e.g. `ls`, `cd`, `cp`. Go to the `CS2850labs` directory[2] and run the command

```
$mkdir week5
```

to create a new sub-directory called `week5`. We suggest you save and compile all programs you write for this exercise in this directory.

Use a command-line text editor, e.g. emacs, nano, or vim,

to open, edit and save your programs. The advantage of command-line editors is that they can be used in a non-graphical SSH session. [3] You can create a new C file or open an existing C file, `file_name.c`, by running the command

```
$editorName file_name.c
```

where, e.g. `editorName` is `vim`

We suggest you save separate files for all single parts of this exercise and follow the name suggestions given in each section. [4]

Compile your C code by running

[1] `puTTY` should be installed on all department's machines. If you work on your own Windows machine you can download it at download puTTY and install it as explained.

[2] The parent directory you have created for the first week's lab

[3] If you do not want to open and close the editor every time you modify and save your code you can open a new SSH session and use two shell windows simultaneously.

[4] This is mainly because some of the Moodle quiz questions may refer to single pieces of code through the suggested file names.

```
$clang -o file_name file_name.c
```

and run the corresponding binary files `file_name` through

```
$./file_name
```

For debugging, we suggest you use the free debugging tools of valgrind, which is already installed on the teaching server `linux.cim.rhul.ac.uk`. To check your code, you just need to run the command

```
$valgrind ./file_name
```

and have a look at the messages printed on the terminal.

# 1 The UNIX shell

In this section, you will get familiar with the most common commands of the UNIX shell. You will write a simple `bash` script that uses `grep` for extracting specific lines of a text file.

## 1.1 Background processes

If you are not already familiar with file management commands such as

`cd`, `ls`, `mkdir`, `cp`, `mv`

please read the corresponding man page using `man <command>` [5] and try them on your home directory. You should also know how to open and edit a file with a command-line editor, e.g. emacs, nano, or vim. So far, you opened and closeed the editor every time you needed to return to the shell, e.g. for recompiling your files. The *ampersand* operator "`&`" allows you to keep the editor open in background while you enter other command in the terminal. Use your favorite editor, `editor_name`, to create a file, `students.txt`, copy of this file students.txt into it, save, and exit. Reopen `students.txt` in background by entering the command

```
$editor_name students.txt &
```

Now the shell outputs the "job number" and PID (process id) of the process running the editor but does not open the usual editor window. Try the following commands:

- `ps`, to see all running processes and corresponding PID
- `fg process_name`, e.g. `fg vim`, to open the editor window
- `ctrl-z` in the editor window, to return to the terminal
- `kill PID`, to terminate the process from the terminal
- `ps`, to show the list of active processes in this shell [6]
- `killall process_name`, to kill all matching processes

**Note** If, after trying to kill a process through `kill PID` you are still seeing it, try to run

```
kill -SIGKILL PID
```

or

```
kill -9 PID
```

to send a SIGKILL signal.

**Tip** Use tab-completion to save yourself typing out entire directory names: after typing the first few characters of a directory or file, hit the "tab" key to let the shell complete the name. If there's more than one match, you can press tab twice to see a list of matches.

[5] You can also use online resources such as this online tutorial

[6] You can use this to obtain the PID of a given process and terminate it using `kill <PID>`.

## 1.2 Other commands for seeing a file content

To see the content of `students.txt` without opening a text editor, you can also use

`cat`, `more`, `head`, or `tail`

Check the Unix `man`, i.e. by entering

```
$man command
```

whit $\texttt{command} \in \{\texttt{cat}, \texttt{more}, \texttt{head}, or \texttt{tail}\}$, for the correct usage of all of them. For each command, try also to understand what you can do through their more advanced options.

## 1.3 Sorting the lines of a text file

You can sort the line of an input text file according to a specified criterion with the command `sort` Try the following commands:

$$\texttt{sort students.txt}, \quad \texttt{sort} \;\; - \texttt{r students.txt}, \quad \texttt{sort} - \texttt{t}/ - \texttt{k2 students.txt}$$

What are the corresponding criteria for sorting the entries? How does the option `-t` work?

For extracting info from a file without inspecting it directly you can use `wc` with various options. Check the `man` page of `wc` and use that command to print the line count for `students.txt`.

Finally, check the `man` page for `cut` and use it to filter the information printed out from `students.txt`. Can you figure out how to show only the name of the students?

## 1.4 I/O Redirection

Normally, command-line programs print to *standard output*, which is connected to the terminal by default. I/O redirection commands

$$>, \quad <, \quad >>, \quad |$$

allow you to read and write data to disk or to communicate between commands (processes) by connecting their standard input and standard output streams.

- $x{>}y$ redirects the output of $x$ to file $y$
- $x{>>}y$ redirects the output of $x$ on file $y$ without overwriting the file.
- $x{<}y$ redirects file $y$ into command $x$.
- $x|y$ connects the standard output of command $x$ to the standard input of command $y$.

Try to understand how the redirection operators work in practice by combining two or three of the UNIX commands mentioned in the pevious sections.

Anser the follwoing questions:

1. What is printed in the file `lsOut.txt` after running `ls > lsOut.txt`?
2. What happens if you run `ls -l >> lsOut.txt` three times?
3. What is the difference between running `wc students.txt` and `wc < students.txt`?
4. Why the output of `ls | wc -l students.txt` consists of a single line?
5. Why do the commands

   ```
   sort students.txt | head -5 | sort -r,   sort -r students.txt
   ```

   produce the same output?
6. Can you predict the output of `wc -l students.txt | wc -l`?

**Optional** Combine `cut` and `sort` and the I/O redirection commands to print on a new file, `names.txt`, the student names (only their names) sorted alphabetically by first name.

### 1.5 grep

To quickly inspect and filter text files you can also use grep, which allow you to print all file lines that match a pattern. In particular, grep becomes a very powerful tool when its argument is a *regular expression.* See wild cards listfor a list of the wild cards you can use to build regular expressions in UNIX. For example, what is the difference between the output of `grep Candice students.txt` and `grep Ca[no] students.txt` ?

Check the man page of grep and answer the following questions:

1. How can you combine grep and wc to find the number of students taking CS1860?
2. How can return the lines of students who are *not* taking CS1890?

## 2 sh scripts

In this section, you will see how UNIX command-line instructions can be combined into basic shell scripts and run from a user C program.

### 2.1 Variables and inputs

Copy the following script into a new file, myGrep.sh,

```
#!/bin/sh
#myGrep.sh
IN=$2
OUT="out.txt"
PATTERN=$1
if test -f "$IN" ; then
        grep $PATTERN $IN > $OUT
        head -10 $OUT
fi
```

and use `ls -l` to check its permissions. If you do not have the right to execute it, add it by entering

```
chmod u+x myGrep.sh
```

and then run the script by entering

```
$./myGrep.sh pattern file_name
```

where file_name = students.txt and pattern = Al. Can you write a single-line combination of the UNIX commands that produces the same output on the terminal?[7]

### 2.2 ID filter

In a new file, select.sh, write a more refined version of myGrep.sh that accepts two integer parameters, startID and endID such that startID $\leq$ endID, and prints on the terminal the lines of students.txt corresponding to students whose student ID is included in the range [startID, endID], i.e. all lines starting with ID$x$ such that startID $\leq x \leq$ endID. The input file can be fixed and does not need to be passed as a parameter, i.e. you can write

```
IN="student.txt"
```

instead of IN=$1 as in myGrep.sh.

**Example** A run of your program should produce an output analog to

```
cim-ts-node-03$ ./select.sh 1181 1185
1181/Kiera Croslin/CS1801/CS1820/CS1890
1182/Kenny Mcclelland/CS1801/CS1820/CS1830
1183/Ilse Wheat/CS1801/CS1820/CS1830
1184/Gregorio Melia/CS1801/CS1820/CS1830
1185/Londa Stacker/CS1801/CS1820/CS1830
```

[7] You may not need to create a temporary file out.txt

or

```
cim-ts-node-03$ ./select.sh 1181
Usage: ./select.sh [startID] [endID]
first ID=1001
last ID=1202
```

**Hint** You can use the following script, after replacing all `TODO`'s with the opportune variables name, expressions or statements

```
#!/bin/sh
# select.sh
IN="students.txt"
START=$1
END=$2
FIRST=‘TODO | cut -d / -f 1‘
LAST=‘TODO | cut -d / -f 1‘
if [ TODO -ne 2 ]; then
    echo "Usage: $0 [startID] [endID]"
    echo "first ID=$FIRST"
    echo "last ID=$LAST"
else
        LOOP=$START
        while [ TODO ]
        do
                if TODO; then
                        grep $LOOP $IN
                fi
                LOOP=TODO
        done
fi
```

## 2.3 Interactive ID filter

Write a C program that executes `select.sh` in an interactive way. When started, the C program should print a prompt message and wait for the user to enter an input. The input should be a pair of integers, `startID` and `endID`, as for `select.sh` described in Section 2.2. The pair of integer is passed to a *loader* that execute the script `select.sh` with parameters `startID` and `endID`. The program terminates when the user enter the string `quit`.

You can run any system command from a C program by calling

```
int execv (char *filename, char *argv[])
```

where:

- `filename` is the name of the program to be execute, e.g. `ls`, `sort`, or `select.sh`
- `argv` is an array of strings that you can use to provide arguments to `filename`. The last element of `argv` must be a null pointer, `NULL` and, by convention, its first element should be the file name of the program to execute, i.e. `filename`

See this online manual page for more details about the `exec` family of functions. In particular, note that the `execv` function replaces whatever is written after its call with the code written in `filename`, executes that code[8], and exits.

[8] with the command-line arguments specified in `argv`

**Hint** Your C program may have the following structure

```
int main(){
        int MAX = 20;
        char s[MAX];
        char *argv[4];
        argv[0] = "select.sh";
        argv[3] = NULL;
        while(1){
                printf("enter ID range or quit to exit: ");
                readLine(s, MAX);
                if(strcmp(s, "quit")){
                        separateInputs(s, argv);
                        if (!fork()){
                                executeCommand(argv);
                        }
                        wait(NULL);
                }
                else{
                        return 0;
                }
        }

}
```

where:

- `int readLine(char *s, int MAX)` saves the user's input into a buffer, `s`, declared in `main` as
  `char s[MAX]`
- `int strcmp(char *s1, char *s2)` is implemented in `string.h`
- `int separateInput(char *s, char **t)` split the input string where it finds a space, i.e. a `' '` character, and saves the pointer to the beginning of each substring into `argv[1]` and `argv[2]`
- `int executeCommands(char **argv)` calls `execv` with first argument `argv[0]` and second argument `argv` and return 0

**Example** The behaviour of the program should be similar to the following

```
cim-ts-node-03$ ./a.out
enter ID range or quit to exit: 1001 1003
1001/Elliot Gorton/CS1801/CS1820
1002/Adolfo Sechrest/CS1801/CS1820/CS1890/CS1840/CS1860
1003/Angle Klimas/CS1801/CS1820/CS1830/CS1840/CS1860
enter ID range or quit to exit: 1003 1004
1003/Angle Klimas/CS1801/CS1820/CS1830/CS1840/CS1860
1004/Ryann Moak/CS1801/CS1820/CS1840/CS1840/CS1860
enter ID range or quit to exit: quit
```