# Master ROC – Bash & C
# Introduction to Commande line under GNU/Linux

Sami Taktak
sami.taktak@cnam.fr

September 2023

During the laboratory sessions we will use the GNU/Linux operating system.
Start by login under OpenSuse GNU/Linux.

## Part 1 : Introduction

### a) Standard Input / Outputs

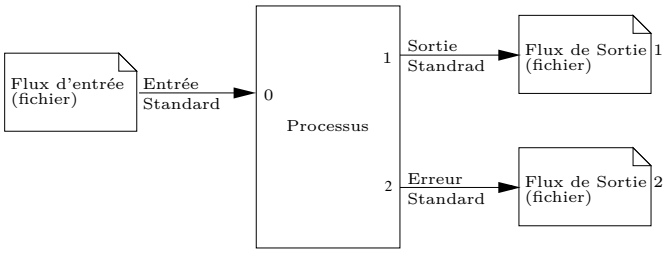


Figure 1: Standard Input / Outputs

To each process is associated a standard input, a standard output, and a standard error output. By default, process read and write on these input/output.

For example, the command **ls** write on its standard output the list of files contained in the current directory:

```
$ ls

Archives              Documents             public_html
bin                   Downloads             Téléchargements
bonjour.txt           mbox                  test.txt
Bureau                Music                 tmp
Desktop               Pictures              Videos
```

**cat** command dump the content of a file on its standard output:

```
$ cat bonjour.txt
Bonjour le monde !
```

The command **echo** allow to display a string on its standard output:

```
$ echo 'Bonjour le monde !'
Bonjour le monde !
```

By default, standard input and outputs of a program started from the command line is redirected to the terminal from which its has been started. In the previous examples, the standard outputs of those commands where redirected to the terminal. The standard inputs were redirected to the keyboard.

## b) Input/Output Redirection

It is often convenient to redirect input/output of a program, for example to save its output into a file.

For example, to save the list of files contained within the current working directory, one could use the operator '**>**' to redirect the standard output of the command `ls` to a file:

```
$ ls > liste-fichiers.txt
$ cat liste-fichiers.txt
Archives     Documents  public_html
bin          Downloads  Téléchargements
bonjour.txt  mbox       test.txt
Bureau       Music      tmp
Desktop      Pictures   Videos
```

The shell provides the following rediction operators:

| Operator | Description |
|---|---|
| `< file_name` | redirection of the standard input from `file_name` |
| `> file_name` | redirection of the standard output to `file_name`. Any previous content in the file is overwriten |
| `>> file_name` | redirection of the standard input to `file_name` in append mode |
| `2>` or `2>> file_name` | redirection of the standard error output to `file_name` in overwrite mode (`2>`) or append mode (`2>>`) |
| `&> file_name` | redirect both standard output and standard error output |

Table 1: Redirection Operators

## Exercises

1. What does the following commandes do ?
   (a) `$ cat f1`
   (b) `$ cat f1 > s1`
   (c) `$ cat f1 f2`
   (d) `$ cat f1 f2 > s2`
2. Wath is the difference between this two command sequences ?

(a) `$ cat f1 f2 > s2`

(b)
```
$ cat f1 > s2
$ cat f2 >> s2
```

3. Give the command which allow to save the directory list of `~/rep1` in `~/liste_rep1.txt`;

# Part 2 : Advanced Command Usage

## a) Chaining Commands

It is often useful to send the output of a command to another command. Try the following command:

```
$ ls -l /usr/bin
```

The output of that command is very long and it is not convenient to read it on a terminal. It would be more useful to display it one page after the other. So, we might want to display it using a pager like `less`.

One way to do it, is to same the output of that command into a file and then, open the resulting file using a pager.

```
$ ls -l /usr/bin > ls-output.txt
$ less ls-ouput.txt
```

Other way it to directly redirect the output of the `ls` command to a pager. Using the pipe operator '`|`', we can send the output of a command to the input of another command:

```
$ ls -l /usr/bin | less
```

### Exercises

Try those commands within a shell.

## b) Text Related Commands

For example, the following command allow to display the first 10 lines of the output of **`ls -l /usr/bin`**:

```
$ ls -l /usr/bin | head -n 10

-rwxr-xr-x 1 root   root          39 Jun 14 18:50 7z
-rwxr-xr-x 1 root   root          40 Jun 14 18:50 7za
-rwxr-xr-x 1 root   root      106520 May 16 13:48 a2p
lrwxrwxrwx 1 root   root          52 Sep 17 06:21 a2ping -> ../share/texliv
-rwxr-xr-x 1 root   root         883 Feb 23  2007 a5booklet
lrwxrwxrwx 1 root   root          54 Sep 17 06:21 a5toa4 -> ../share/texliv
lrwxrwxrwx 1 root   root          25 Sep 21 01:15 aclocal -> /etc/alternati
-rwxr-xr-x 1 root   root       36792 Aug 15 12:37 aclocal-1.15
-rwxr-xr-x 1 root   root       18872 Jul  2 14:31 aconnect
-rwxr-xr-x 1 root   root       19760 Nov  5  2013 acpi
```

| Commande | Arguments | Description |
|---|---|---|
| head | -n x | only display the first x lines |
| tail | -n x | only display the last x lines |
| tr | 'c' 'b' | replace all occurrences of character 'c' by 'b' |
| tr | -s 'c' | replace multiple occurrences o character 'c' by a single occurrence |
| cut | -f x | extract column number x (default column separator is the tabulation character) |
| sort | *(none)* | sort lines alphabetically |
| uniq | *(none)* | delete multiple occurrence of a line |

Table 2: Opérateurs de redirection

The following command allow to remove multiple occurrence of white space in the output of **ls**:

```
$ ls -l /usr/bin/ | head -n 10 | tr -s ' '

-rwxr-xr-x 1 root root 39 Jun 14 18:50 7z
-rwxr-xr-x 1 root root 40 Jun 14 18:50 7za
-rwxr-xr-x 1 root root 106520 May 16 13:48 a2p
lrwxrwxrwx 1 root root 52 Sep 17 06:21 a2ping -> ../share/texlive/texmf-dis
-rwxr-xr-x 1 root root 883 Feb 23 2007 a5booklet
lrwxrwxrwx 1 root root 54 Sep 17 06:21 a5toa4 -> ../share/texlive/texmf-dis
lrwxrwxrwx 1 root root 25 Sep 21 01:15 aclocal -> /etc/alternatives/aclocal
-rwxr-xr-x 1 root root 36792 Aug 15 12:37 aclocal-1.15
-rwxr-xr-x 1 root root 18872 Jul 2 14:31 aconnect
-rwxr-xr-x 1 root root 19760 Nov 5 2013 acpi
```

### Exercises

1. Write the command which allow to replace spaces by tabulation '\t' in the output of **ls**;
2. Write the command which allow to extract group names of files in **/usr/bin**;
3. Write the command which allow to sort alphabetically the group names of files in **/usr/bin**;
4. Write the command which allow to give the list of groups which have a file in **/usr/bin**;
5. Could we avoid the use of the command given in 1 ?

## Part 3 : Jobs Control

When we run a command inside a terminal, the terminal "hangs" until the command ends. That's happen when the command is run on the *foreground*. When started, a command can be sent to the *background* by using the operator **&**.

```
$ emacs &
[1] 23456
```

The command started `emacs` in the background, allowing the user to continue to use the terminal while `emacs` is running.

The number give under square brackets is the jobs number.

We can reference any job by prefixing its number with the symbol `%`.

| Commande | Paramètre | Description |
|---|---|---|
| `bg` | `%job` | send the job to the background |
| `fg` | `%job` | send the job to the foreground |
| `kill` | `%job` | send a signal to the job |
| `jobs` | | list the running jobs |

```
$ jobs
[1]+ Stopped man bash
[2] Running emacs
```

## a) Keyboard Shortcuts

There are shortcuts dedicated to job managements:

| Touches | Description |
|---|---|
| `Ctrl - Z` | Stop the job and send it to the background |
| `Ctrl - Q` | Resume the jobs and send it to the foreground |
| `Ctrl - C` | Send interruption signal to the job |

## b) Environment Variables

Environment variables are couples of (key, value) which are transmitted to any new command. Those variables impacts how commands behave, like specifying which language to use.

The current list of environment variables in a shell could be displayed using the command `env`:

```
$ env
TERM=xterm
SHELL=/bin/bash
USER=taktaks
PATH=/home/taktaks/bin:/usr/local/bin:/usr/bin:/bin
DESKTOP_SESSION=lightdm-xsession
PWD=/home/taktaks/
LANG=en_US.UTF-8
GDM_LANG=en_US.utf8
GDMSESSION=lightdm-xsession
XTERM_LOCALE=en_US.UTF-8
XTERM_VERSION=XTerm(325)
HOME=/home/taktaks
LANGUAGE=en_US:en
XDG_SESSION_DESKTOP=lightdm-xsession
LOGNAME=taktaks
DISPLAY=:0
XAUTHORITY=/home/taktaks/.Xauthority
OLDPWD=/home/taktaks/tmp
_=/usr/bin/env
```

Within this list of variables, we can recognise the following ones:

| Variable | Description |
|---|---|
| TERM | Terminal type |
| SHELL | Shell name |
| USER | User name |
| PATH | Paths where to look for commands |
| LANG | Set language |

To display the content of a single variable, one can use the **echo** command. We reference a variable by prefixing its name with the symbol '$' and enclosing its name within curly brackets (curly brackets might be avoided):

```
$ echo ${PATH}
PATH=/home/taktaks/bin:/usr/local/bin:/usr/bin:/bin
```

To set a value:

```
$ mavar=3; echo ${mavar}
3
```

This set a variable, but do not add it to the environment. It will not be passed to subsequent commands.

We need specify `export` to add a variable to the environment:

```
$ mavar=Hello; echo ${mavar}
Hello
$ env | grep mavar
$ bash
$ env | grep mavar
$ exit
$ export mavar
$ env | grep mavar
mavar=hello
$ bash
$ env | grep mavar
mavar=hello
```

## Exercises

1. Define a new variable `test`, start a new shell a check if that variable is in its environment;
2. Quit the shell, export the variable to the environment, and restart a new shell. Check if the variable is present in its environment
3. Modify the `PATH` variable so it include the current working directory.