TP1–2 · Commands and shell scripting

NUnix · L3 MIAGE · University of Rennes

Elisabeth Bouriah and Louis Ledoux 2026–2027

Use a Linux machine with Bash.

1. Getting started

Log into your Linux account and open a terminal.

1. Run `pwd`, then `echo "$HOME"`. What does each command display? If the terminal starts in your home directory, why are the results the same? Which command returns you home from any directory?
2. Create a directory named `NUnix` in your home directory with `mkdir`. Which command lets you check that it exists?
3. Enter it with `cd`. Try completing its name with **Tab**. If several names share the same beginning, press Tab twice to list the possibilities.
4. Use the **Up** and **Down** arrows to browse previous commands. Display the absolute path of your current directory again.
5. Try copying and pasting text in the terminal. On many Linux terminals, selecting text and clicking the middle mouse button pastes the selection. You can also use **Ctrl+Shift+C** and **Ctrl+Shift+V**.
6. Create a directory named `tp1` inside `NUnix`. This will be your working directory for the lab.
7. Download tp-files.zip either through your browser (save it in `~/NUnix`) or from a terminal in `~/NUnix` with `wget URL` (replace `URL` with the link address). In `~/NUnix`, run `unzip tp-files.zip`. Check that `files` and `tp1` are now two separate directories in `NUnix`.

2. The man command

`man command` displays a command's manual. Use **Space** to advance one page, **Enter** to advance one line, and **q** to quit.

Look up the following commands and options. For Bash built-ins such as `cd` and `history`, use `help cd` or `help history`.

```
man, man -k
cd, cd .., ls, ls -a, ls -l, ls -al, pwd
mkdir, rmdir, cp, cp -R, mv, rm, rm -R
touch, echo, history, history -c, who, passwd, cat
```

3. Environment variables

Store the path of the supplied files in a variable to access them from another directory:

```
export variable_name=directory_path
```

1. Enter `~/NUnix/files` and use `pwd` to find its **absolute path**. Store this path in an environment variable named `tpfiles`.
2. Return to `~/NUnix`. Use `tpfiles` to list the contents of `files`.
3. Use it to display the contents of `prog1.c`.

4. Redirections

1. Copy the contents of the directory named by `tpfiles` into `~/NUnix/tp1` using `cp`. Work in `~/NUnix/tp1` for the remaining exercises.
2. Open `prog1.c` in a text editor. What does the program do? When launching a graphical editor from the terminal, try adding `&` to run it in the background and keep using the shell.
3. Run `make` in `~/NUnix/tp1` to compile the supplied programs (a C compiler is required). Then run `./prog1` in each of these situations:

Standard input	Standard output
Keyboard	Screen
File `d1`	Screen
Keyboard	File `res1`
File `d1`	File `res2`

   When typing input, finish with `%`, the program's terminating character.

5. Shell scripts

1. Read `prog2.c`. Its standard input consists of an integer `i`, followed by a sequence of characters divided into parts by `%`:

   ```
   first part%second part%third part%
   ```

   It writes the `i`th part to standard output. Parts are numbered from 1.
2. Run `./prog2` with several inputs to understand its behaviour.
3. Write a script `select.sh`, called with three arguments:

   ```
   ./select.sh i f1 f2
   ```

   The integer `i` selects a part, `f1` is the input file containing the sequence, and `f2` is the output file. Use `prog2` to extract the requested part. You can use the supplied `d2` as input.

   Check the number of arguments, file existence and access permissions. These forms of `cat` may help:

   ```
   cat f1 f2
   cat - f
   ```

   The first concatenates two files. The second reads standard input first, then the file `f`.
4. Write `selmult`, called as `./selmult f`. It repeatedly reads an integer `i` and displays the `i`th part of file `f` using `prog2`. Stop when the user enters `0`.

6. Extracting system information

`ps` reports running processes; `ps -A` selects all processes. `find` searches the filesystem. A redirection such as `2>/dev/null` discards error messages.

1. Create `aliveprocs.txt` containing the names of all currently running processes (the command-name column). Display the file.
2. List the running processes sorted by virtual memory size. Display the result. Consult `man ps` for the available fields and their units.
3. List the running processes that do not belong to `root`, sorted by process name.
4. List process names in order of total CPU time. If the same name occurs more than once, display it only once.
5. Display the names of regular files in your home directory that have been modified today.
6. Create `read-only.txt` containing the paths of regular files on the system owned by the user running the command.

7. Text-editing program

Write a Bash program that searches the files in a given directory for a string `str` and replaces its occurrences with another string, `new_str`.

1. Accept command-line options with values, such as `./main.sh -a value`.
2. Use functions in your program.
3. Provide a help option explaining the arguments and their purpose.
4. Validate the arguments and return an error status when they are invalid.
5. Then add a choice between replacing the first occurrence and replacing all occurrences.

For example, `-h` could display help and `-p` could specify the directory. Choose and document the remaining options.

Use a directory tree such as this to exercise your program on files at several depths:

```
.
├── file.txt
├── folder1/
│   ├── file.txt
│   └── sub1/
│       ├── file.txt
│       └── subsub1/
│           └── file.txt
├── folder8/
│   └── file.txt
└── main.sh
```