Linux for Beginners

ABiMS Training Module 2018

Philippe Bordron
Mark Hoebeke
Gildas Le Corguillé

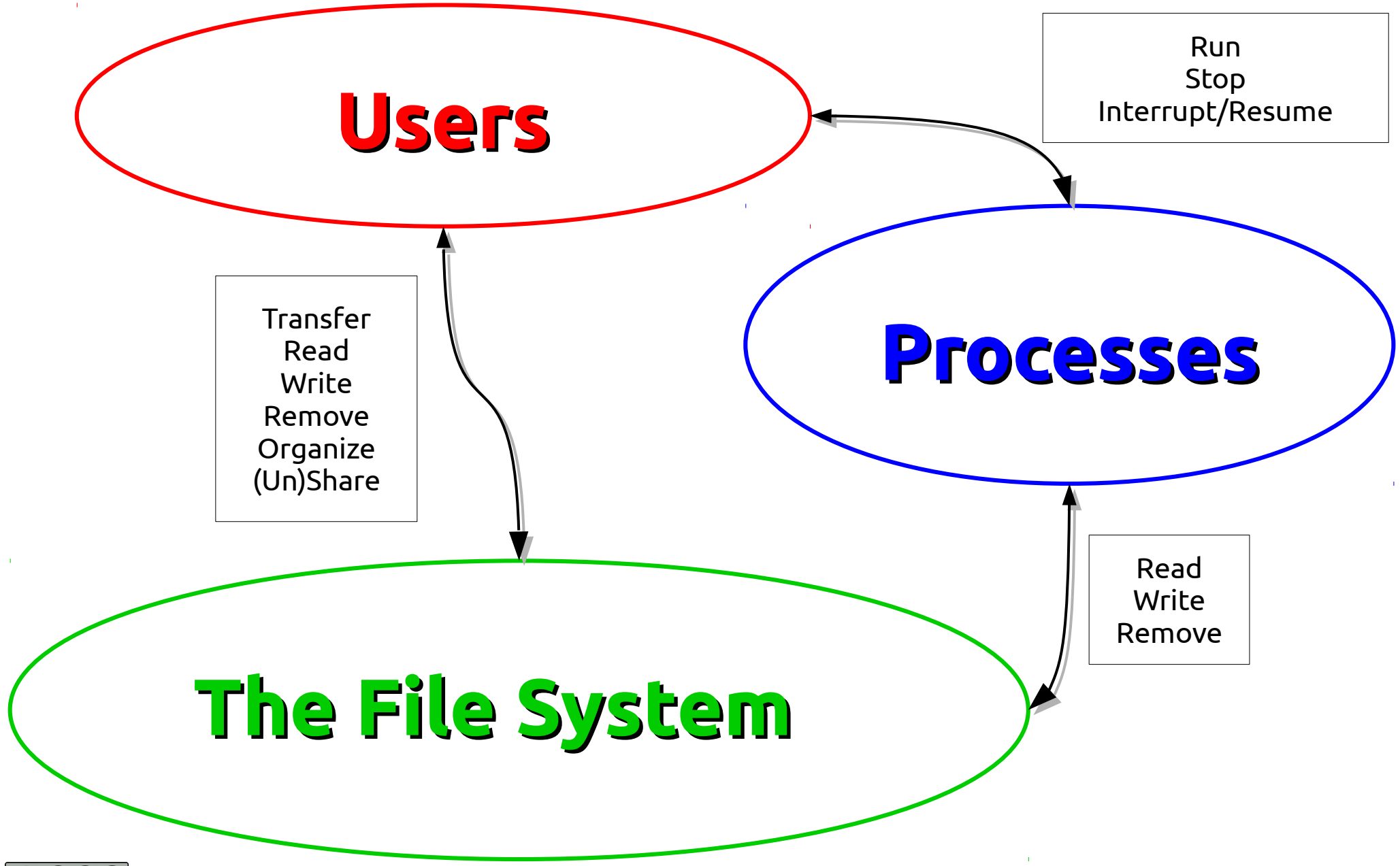
Linux for Beginners | Goals

My {colleague|competitor} told me about a highly interesting tool for analysing data quite related to mine. It's some kind of thingy running on *Lynuks* and installed somewhere on a server.

How do I:

- Connect (**myself**) to the server?
- Transfer **my files** holding the data on the server?
- Launch the **program**?
- Specify where to find the **files** with the data it will process?
- Specify where to write the **files** with the results it will generate?
- Organize all these **files** in **folders** not to get lost later on ?
- Run several **programs** "simultaneously" (or several "instances" of the same **program**) with different parameters or input data?
- Stop an instance of the **program** if it runs for too long?
- {Share|protect} **my files** containing input data or results {with my colleagues| from my competitors}?

Linux initiation | Les notions-clés

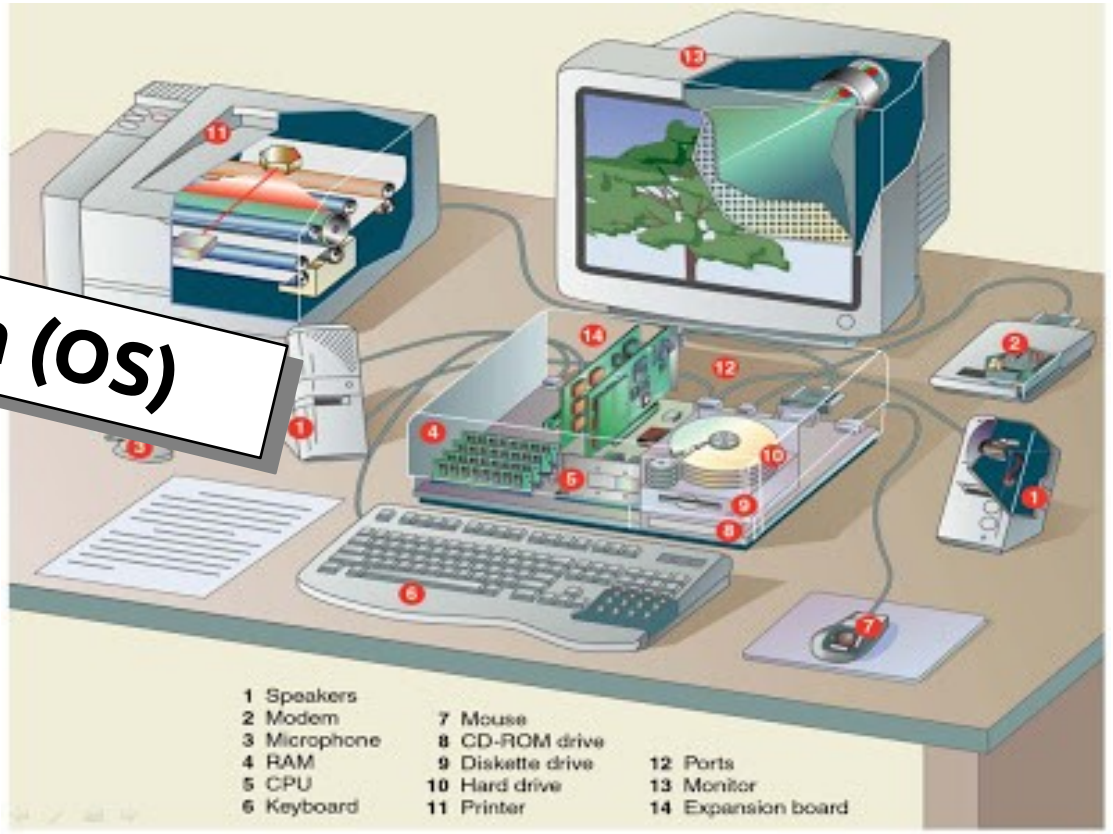


Linux for Beginners | Outline

Linux for Beginners | Outline

Purpose of an Operating System

TheOperating System (OS)



An OS is a “privileged” program run when the machine is switched on and that:

- Loads other programs in memory (RAM),
- Allocates the resources (memory, CPU time, disk space) they request,
- Handles their communications (input/output) with peripheral devices (screen, keyboard, mouse, network, printer…)
- Halts their execution
- Reclaims the allocated resources.

Why Linux ?

- Preemptive multitasking & multi-user system
  - Durability & stability with a track record back to 1994
- *Open-source* **and free (as in beer)**
  - Its code can be freely copied, modified and redistributed
- Offering a vastsoftware catalogue :
  - Office suites : LibreOffice
  - Internet tools : browsers (Firefox, Chrome), e-mail programs (Thunderbird, Evolution)
  - Multimedia : audio/video playback tools (VLC, Totem)
  - Graphics : images manipulation (Gimp), 3D modeling (Blender)
  - Software development : languages (Python, Java, C/C++…), environments (Eclipse, IDLE, PyDev, DDD)
- Scientific disciplines included :
  - Bioinformatics : blast, emboss, phylip, mafft, clustal, trimal...

Linux Distributions

A Linux distribution includes:

- Some flavour of the Linux kernel.
- A portfolio of prepackaged software
- Administration tools facilitating the installation and update of these *packages.*

The main differences between distributions are:

- On a technical level :
  - The *packaging* format used to package(!) software.
  - The tools to administer these *packages.*
- Business model wise :
  - Technical Support : community-based vs. commercial.
  - The licenses for the software they provide.

Linux for Beginners | Outline

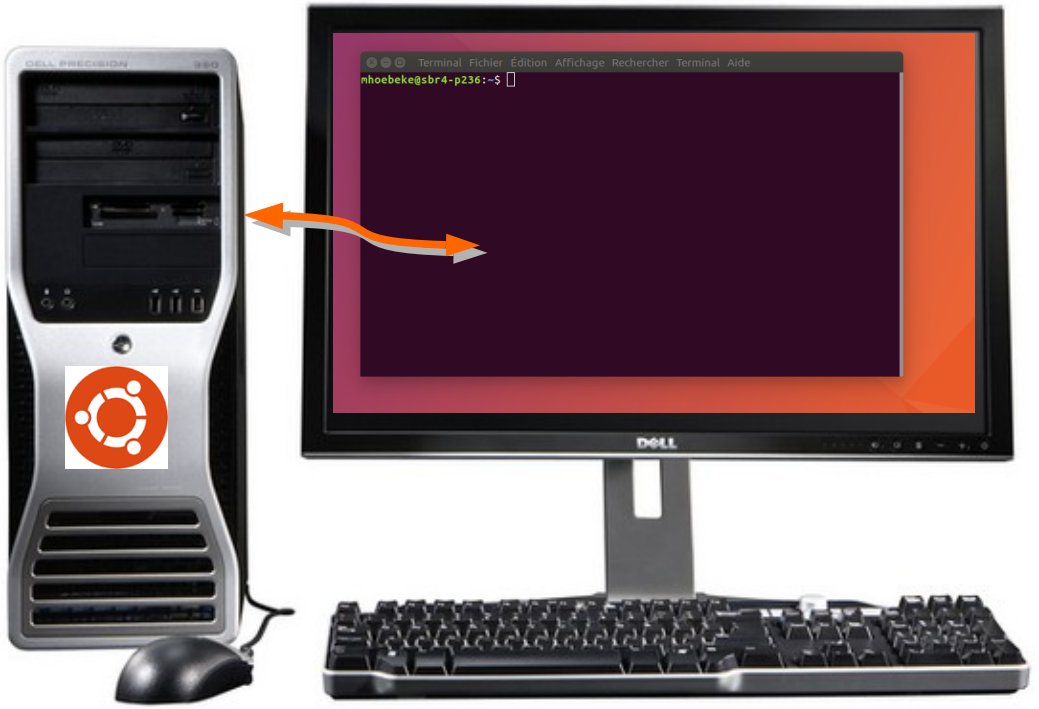
Connecting | The Terminal

The Terminal :

- A means to ”communicate”with **a machine** relying on a **command line** in the context of a **session**

Use Case 1 : Local Sessions

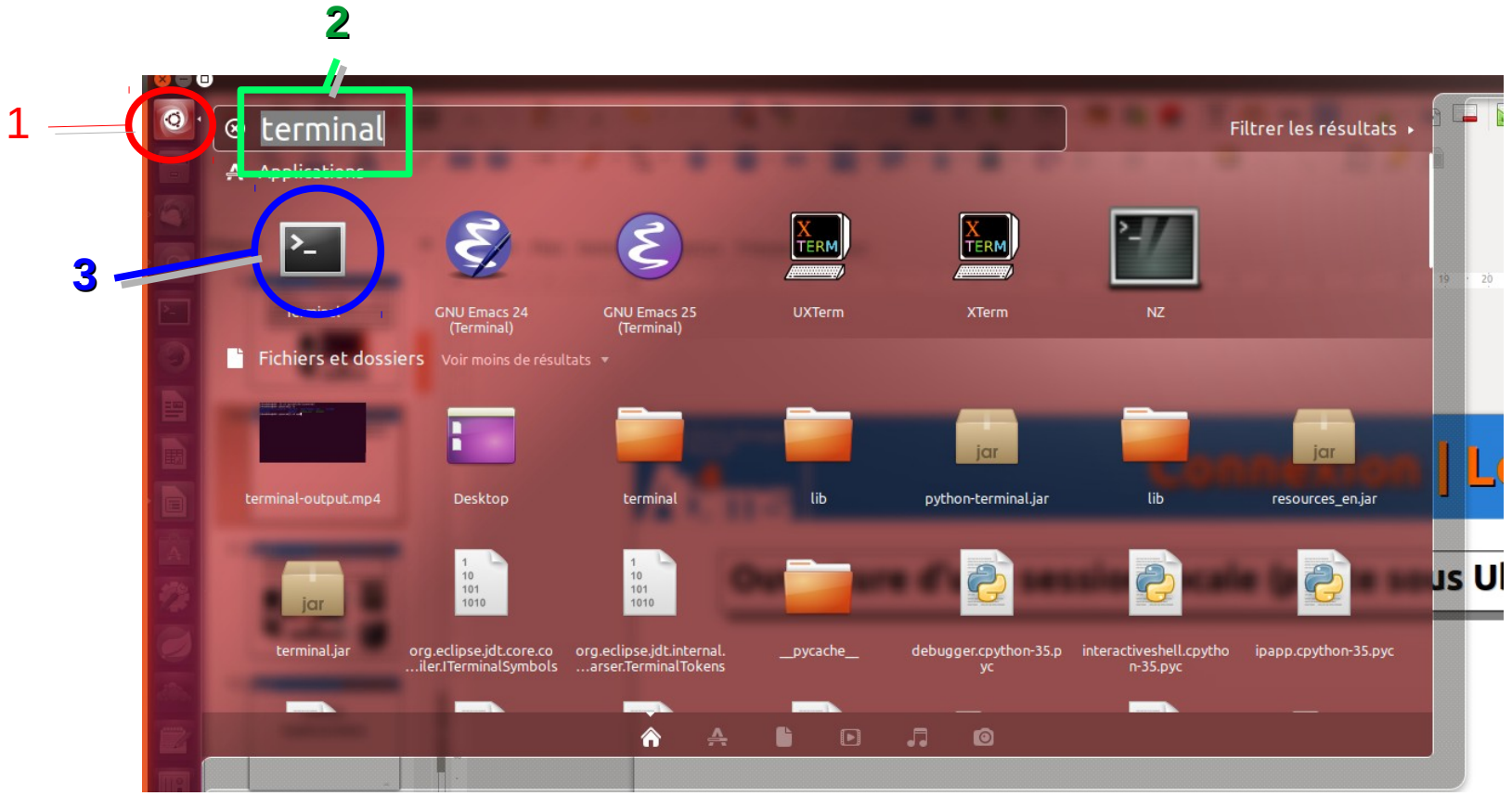
- Type commands (programs) that will run (use memory, CPU and disk space) on **your workstation**



Connecting | Local Sessions

Opening a local session(on a workstation running Ubuntu) :

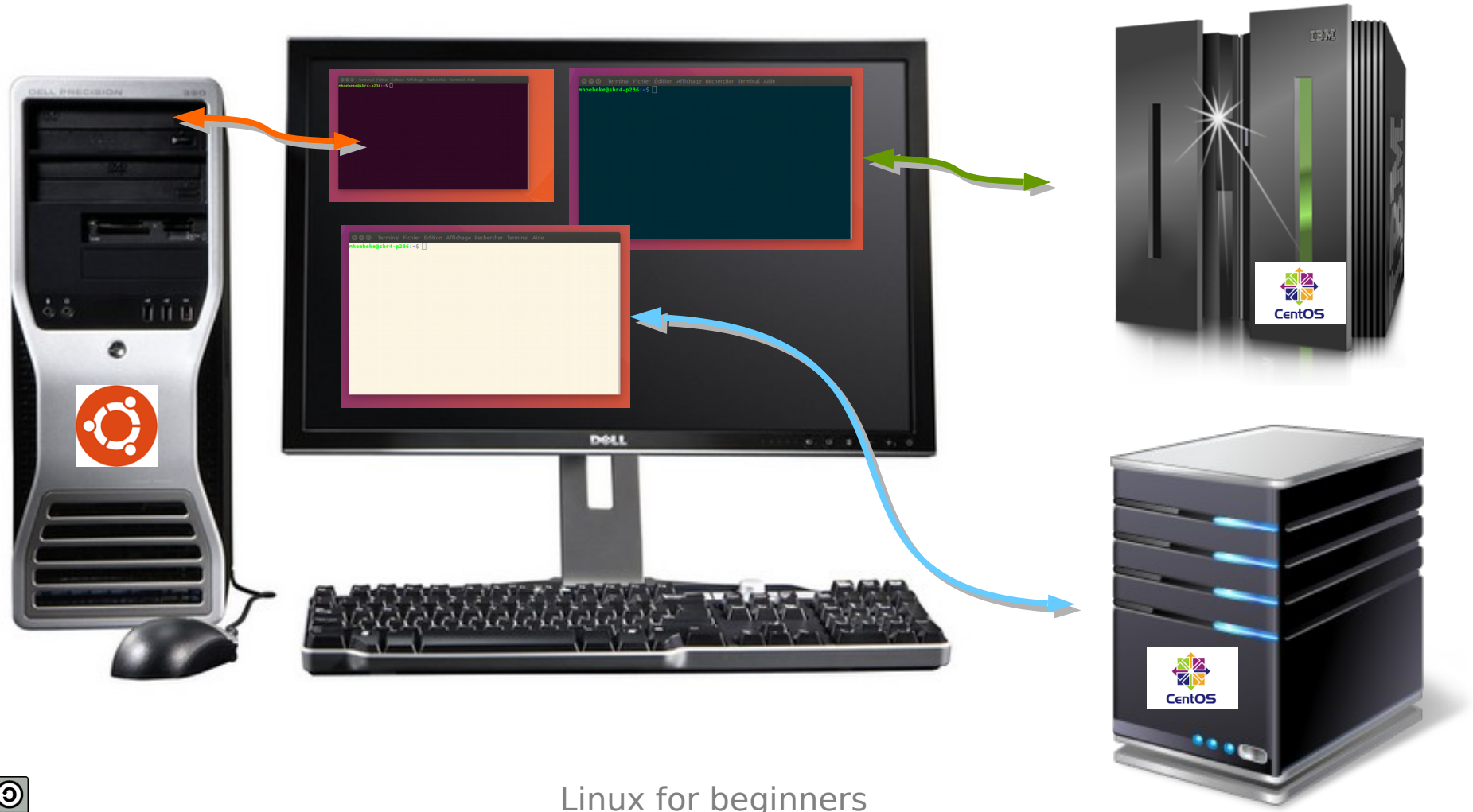
- Using the keyboard : **Ctrl+Alt+T**
- Using the mouse:



Connecting | Remote Sessions

Remote Sessions :

- To work on a machine accessible through the network
- Potentially opening additional windows on the remote machine

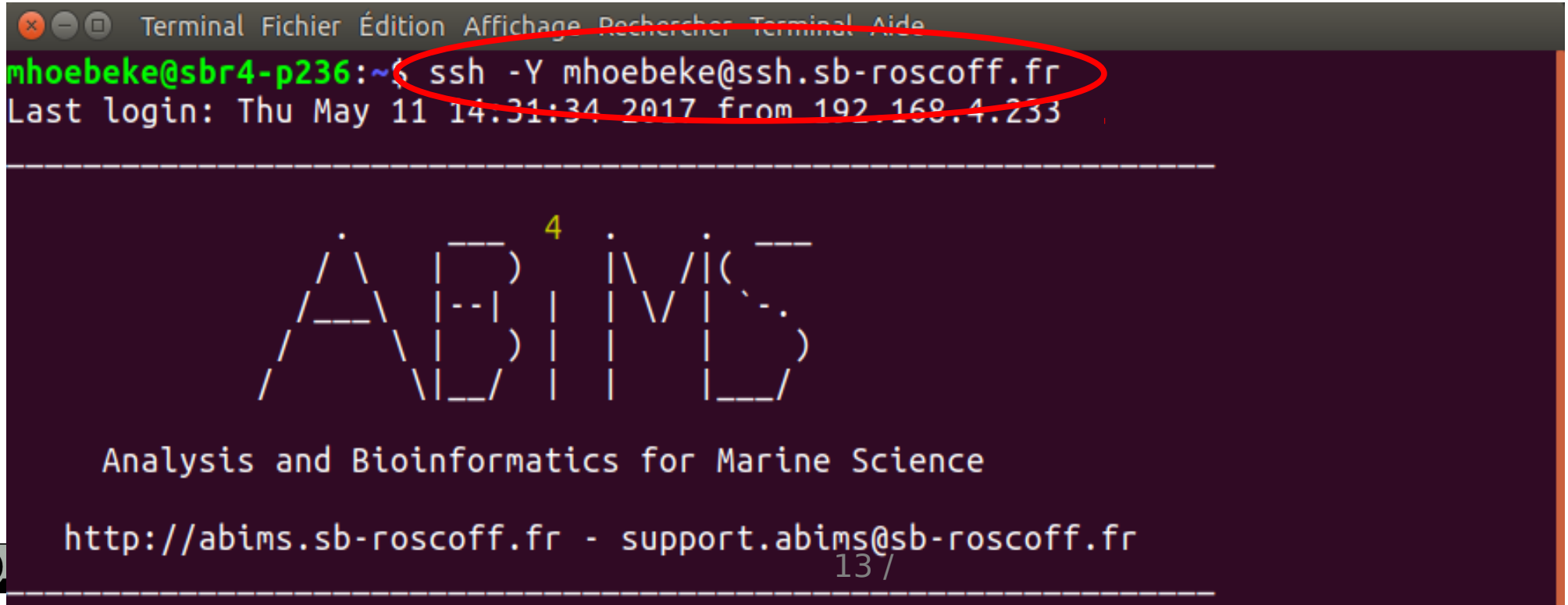


Connecting | Remote Sessions & SSH

Using the Secure *(SSH)* Shell Protocol:

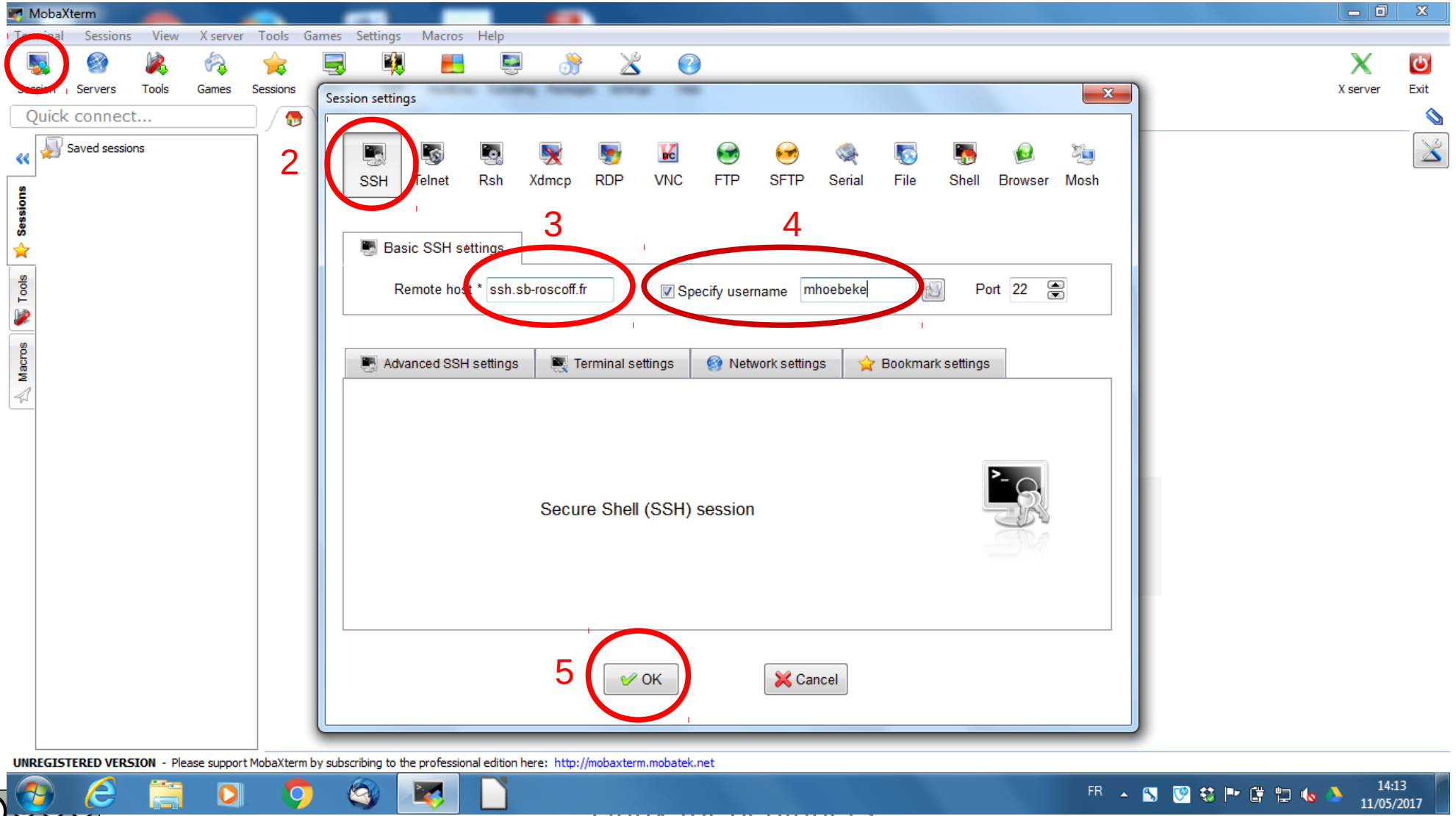
- Encrypted (secure) communications protocol opening a channel allowing information exchange between two machines.
- Requires knowledge of:
  - The name (or address) of the remote machine,
  - The user name (login) on the remote machine.

Using SSH from a local terminal



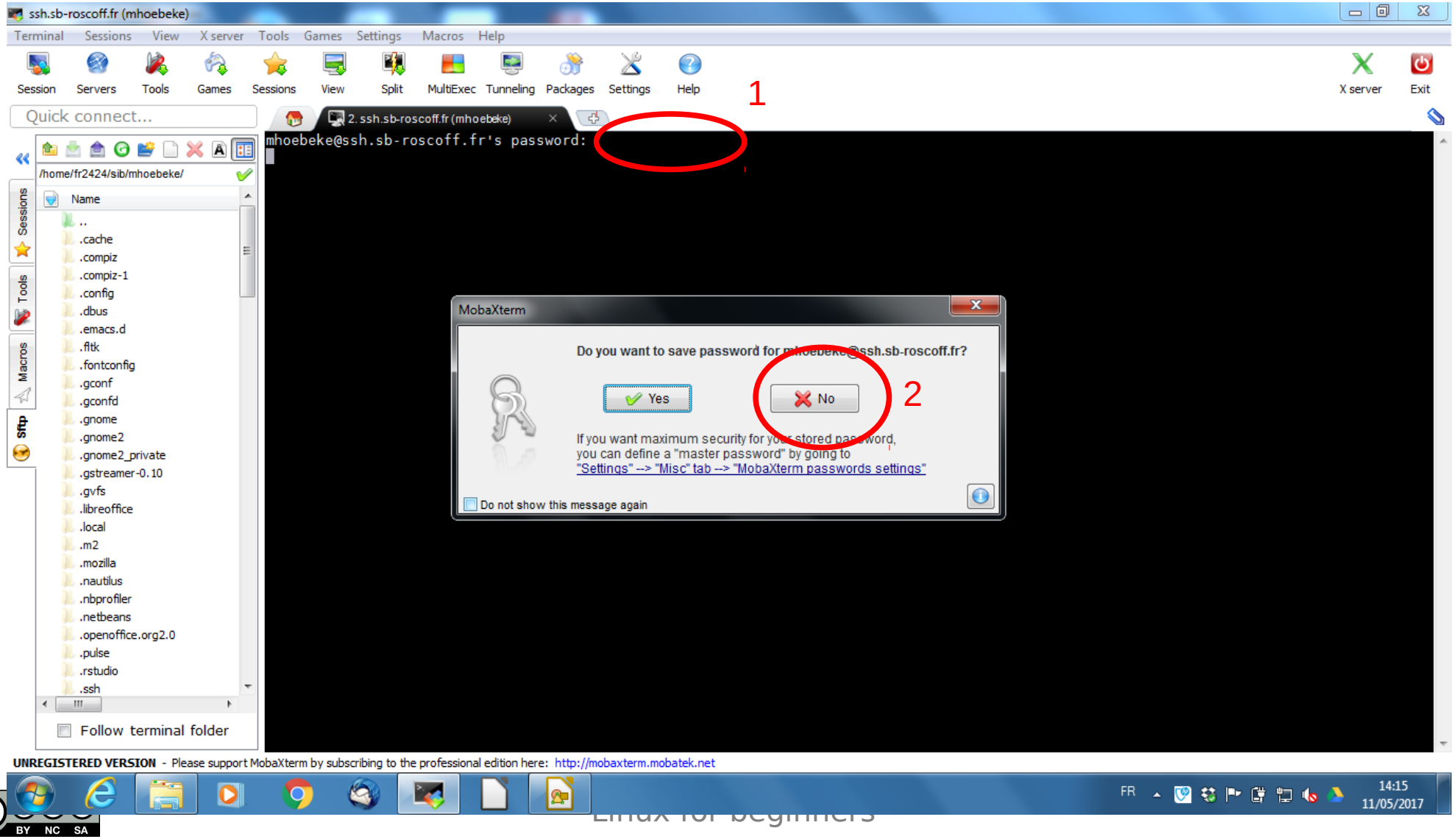
Connecting | Remote Sessions & SSH

Using SSH on a Windows workstation with MobaXTerm



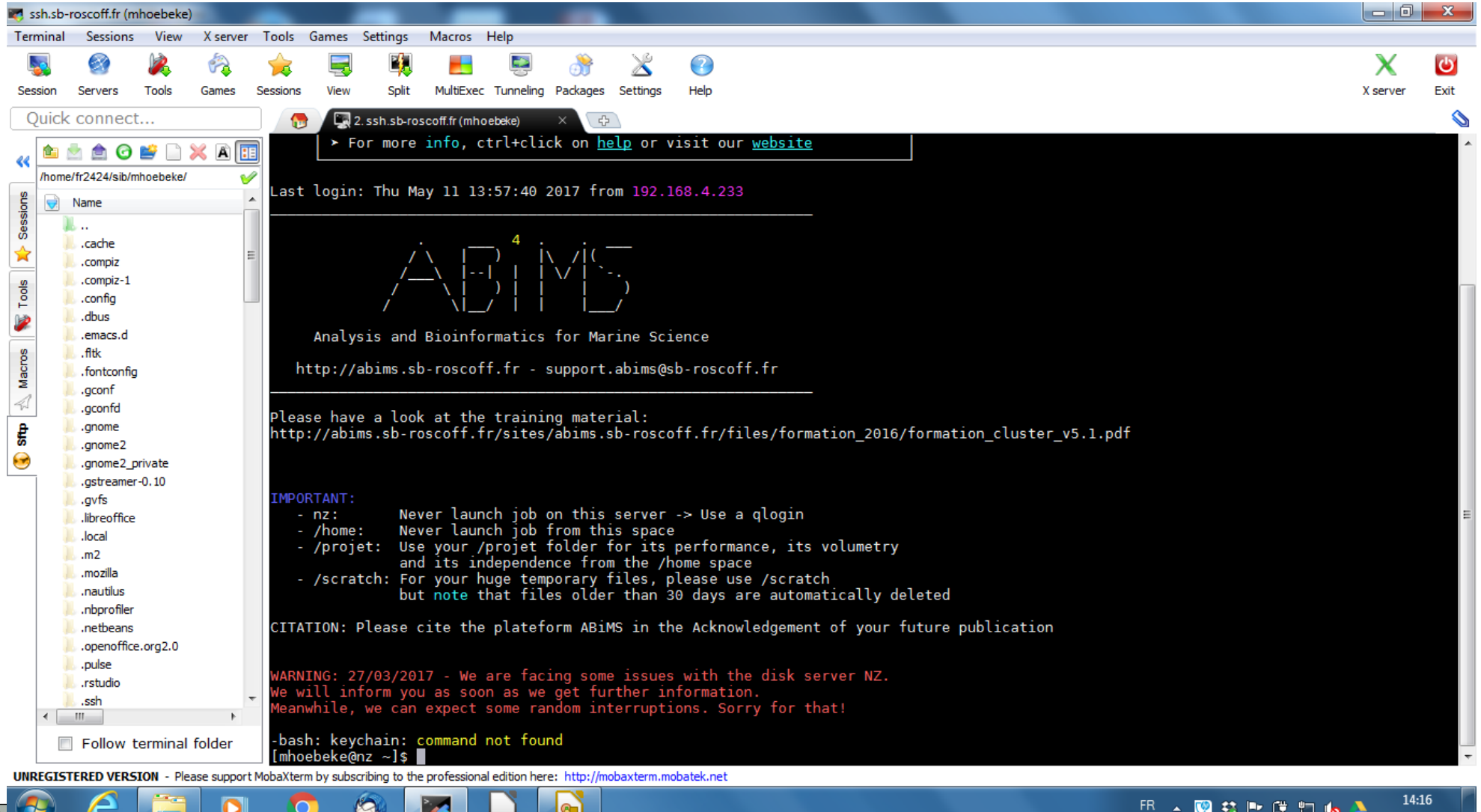
Connecting | Remote Sessions & SSH

Using SSH on a Windows workstation with MobaXTerm



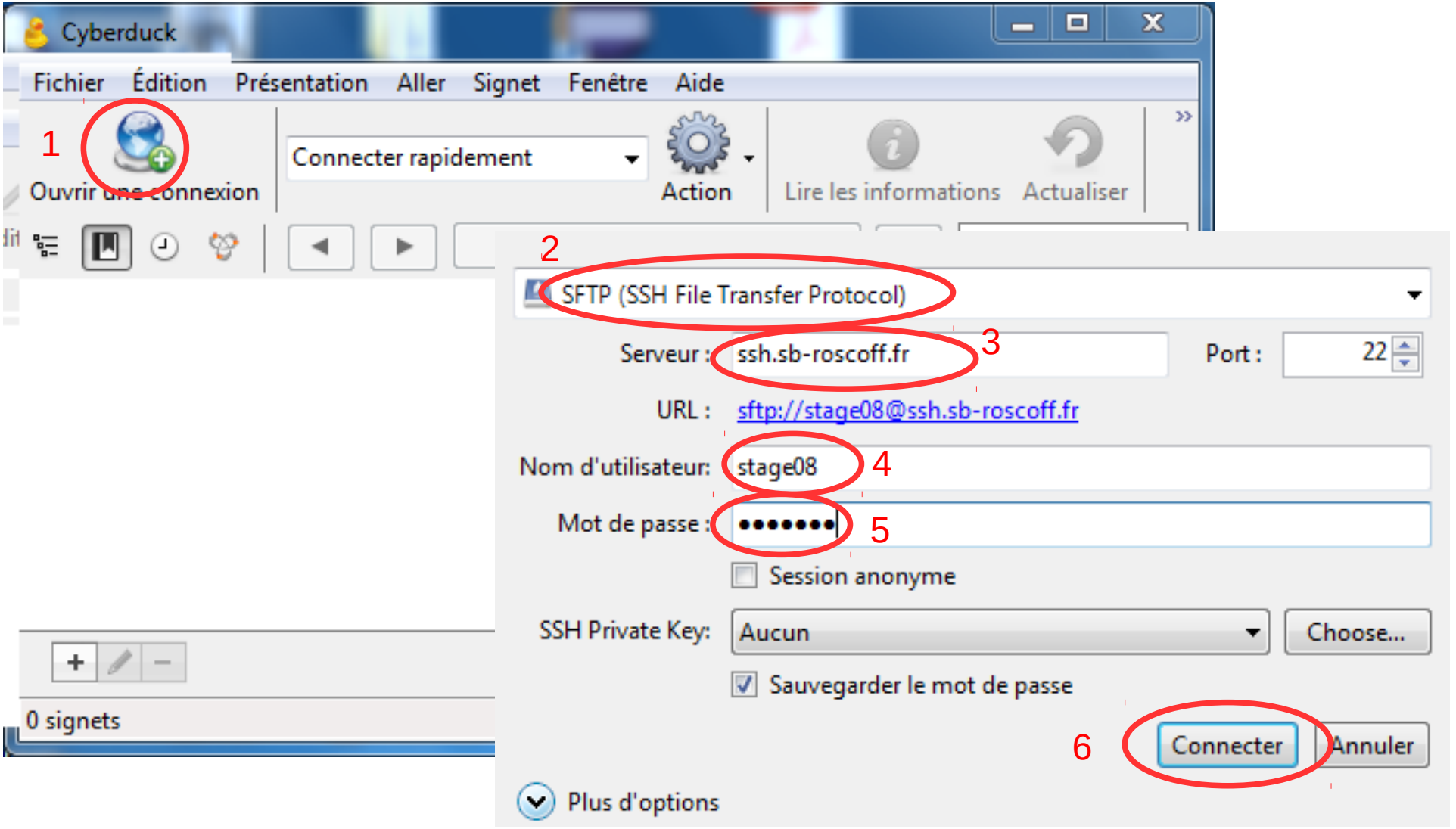
Connecting | Remote Sessions & SSH

Using SSH on a Windows workstation with MobaXTerm

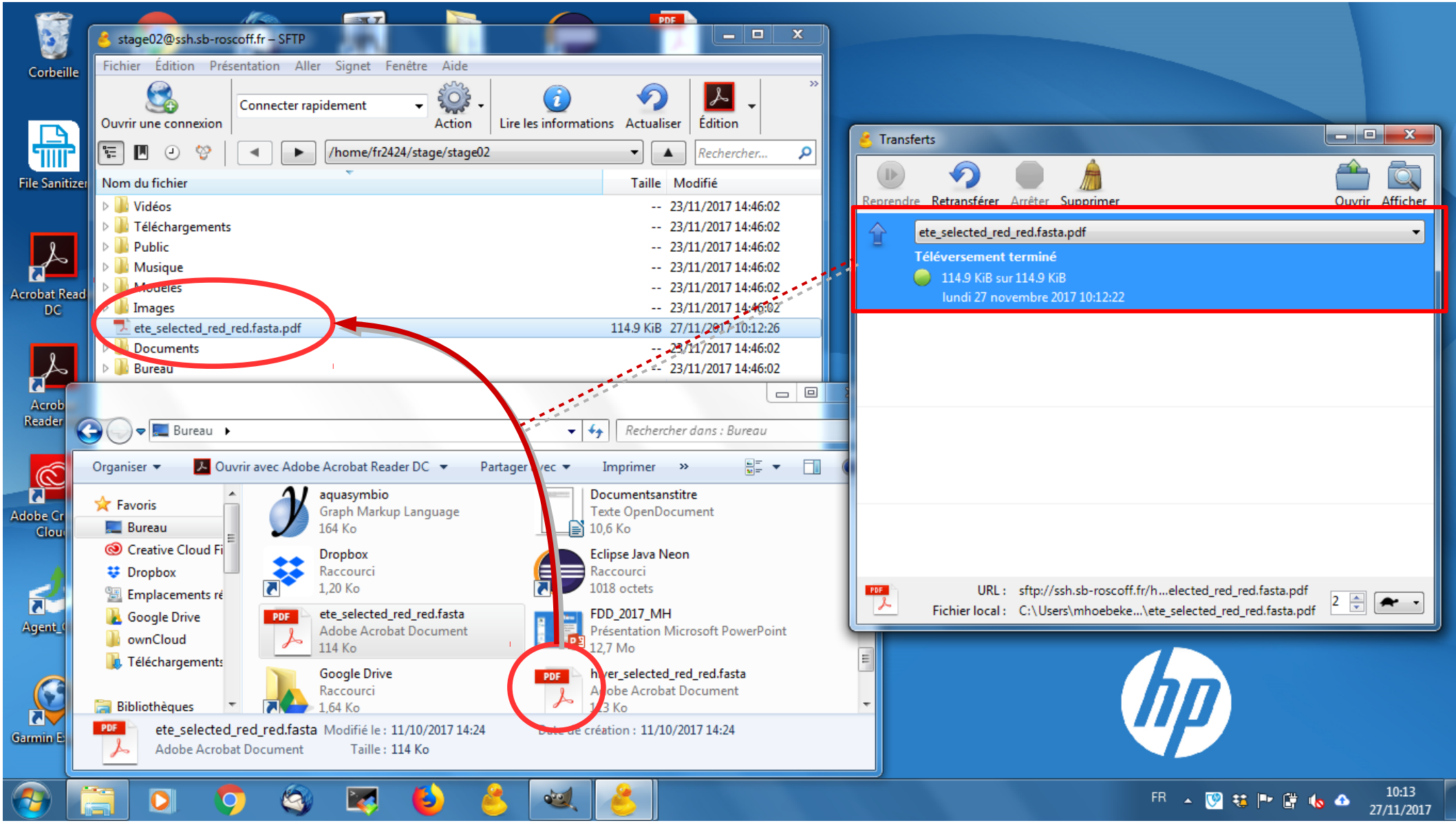


File Transfer using Cyberduck

Defining the connection parameters
Establishing the connection

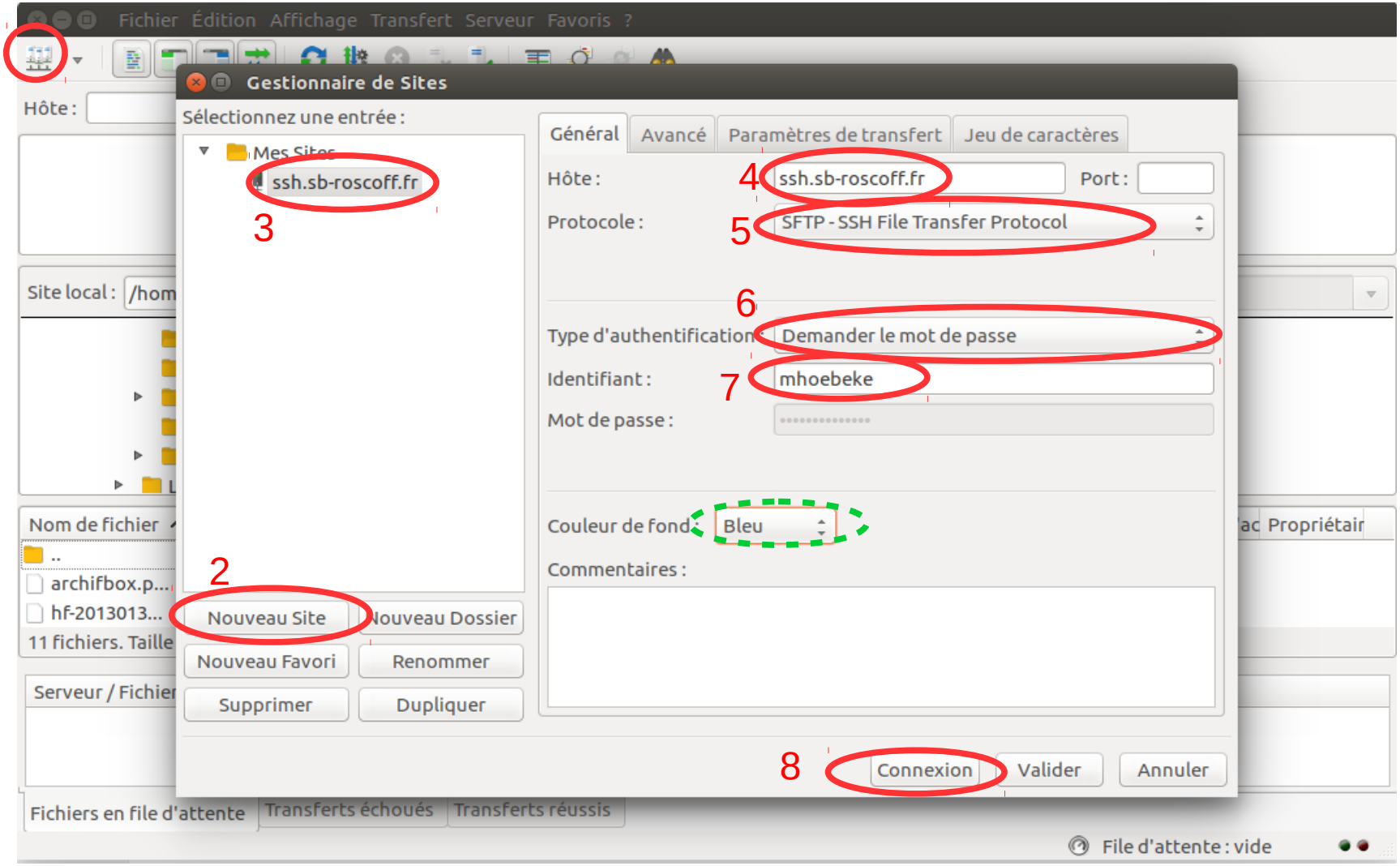


File Transfer using Cyberduck

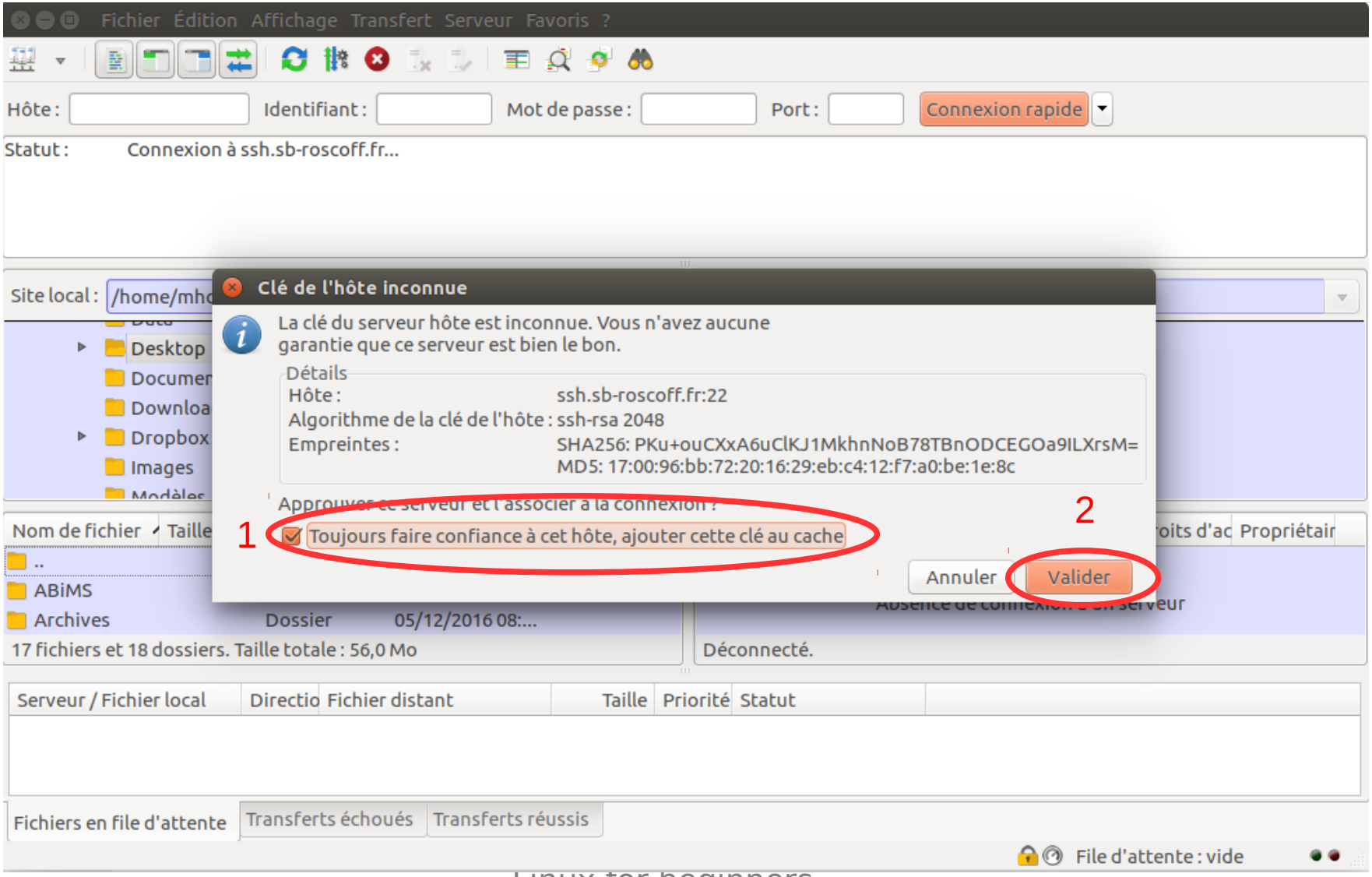


File Transfer using FileZilla

Défining the connection parameters
Establishing the connection

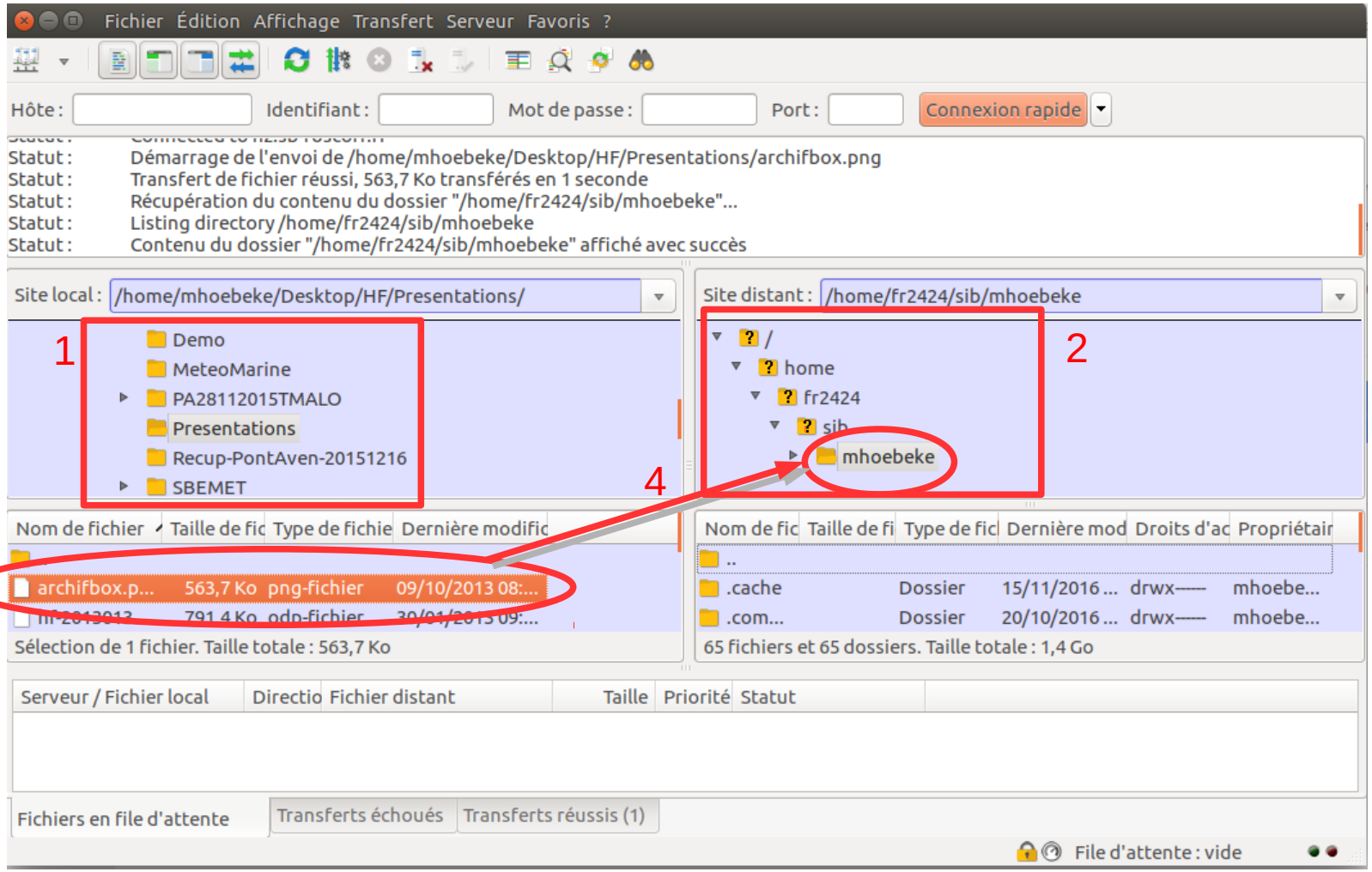


Validating the connection



File Transfer using FileZilla

Selecting the source folder
Selecting the destination folder
Selecting the file(s) to transfer
Transfer through *drag 'n drop*



Exercises

- Retrieve the dataset for today's course at :

  `https://frama.link/abimslinux2018`

- Using your **`stageXX`** account, transfer the dataset to host : **`ssh.sb-roscoff.fr`**, in folder **`/projet/fr2424/stage/stageXX`**

- Open a terminal session on host **`ssh.sb-roscoff.fr`**, and issue the following (black magic for now) commands :

```
[stage01@nz~]$ cdprojet
[stage01@nz~]$ tar zxvf Linux-Initiation.tgz
```

Linux for Beginners | Outline

Structure of the Command Line

```
[stage01@nz ~]$ head -n 20 insulin.fas  #print the first 20 lines
```

[stage01@nz ~] $ The *prompt* **:** displays the current user's login **(stage01)**, the host (or machine) name **(nz)**, the current directory (working directory) **(~)**

head The name of the program to run (first word following the prompt)

-n 20 A command *option* **(-n)** possibly with an associated value **(20).**

insulin.fas A command *argument*

 # print (...) Comments (ignored by the command)

- The space character is used to separate the various fields of the command line.
- **The character case (upper/lower) is important** (**head** is not the same as **HEAD**)
- Each command has its own set of arguments and options
- ***The Enter key (↵) is used to run the program***

The Command Line | Built-in Help

Each (self respecting) command is documented

To display a short documentation on how to use a command, it is possible to use the –**help** or **-h** options

```
[stage01@nz~]$ ls --help
Utilisation : ls [OPTION]... [FICHIER]...
Afficher des renseignements sur les FICHIERs (du répertoire actuel par
défaut).
```

To get a more detailed documentation, it is possible to use the **man** *(*i.e. ***man****ual)* command Giving it as an argument the name of the command for which to display the documentation.

```
[stage01@nz~]$ man ls
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information about the FILEs (the current directory by default).  Sort
       entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

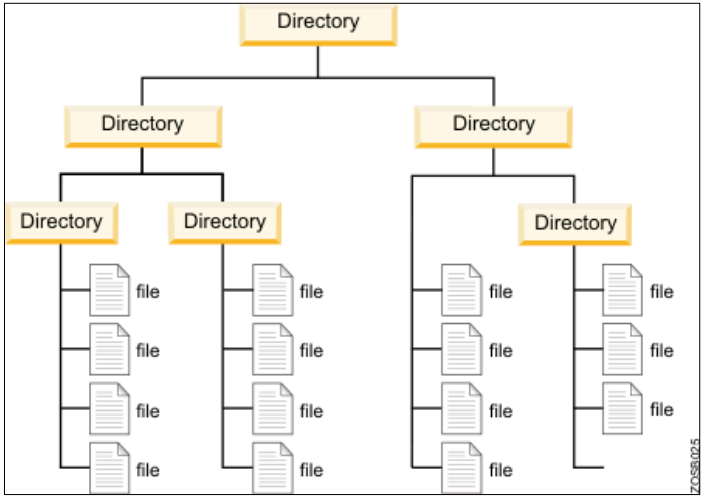
Linux for Beginners | Outline

Le système de fichiers | Utilité & concepts de base

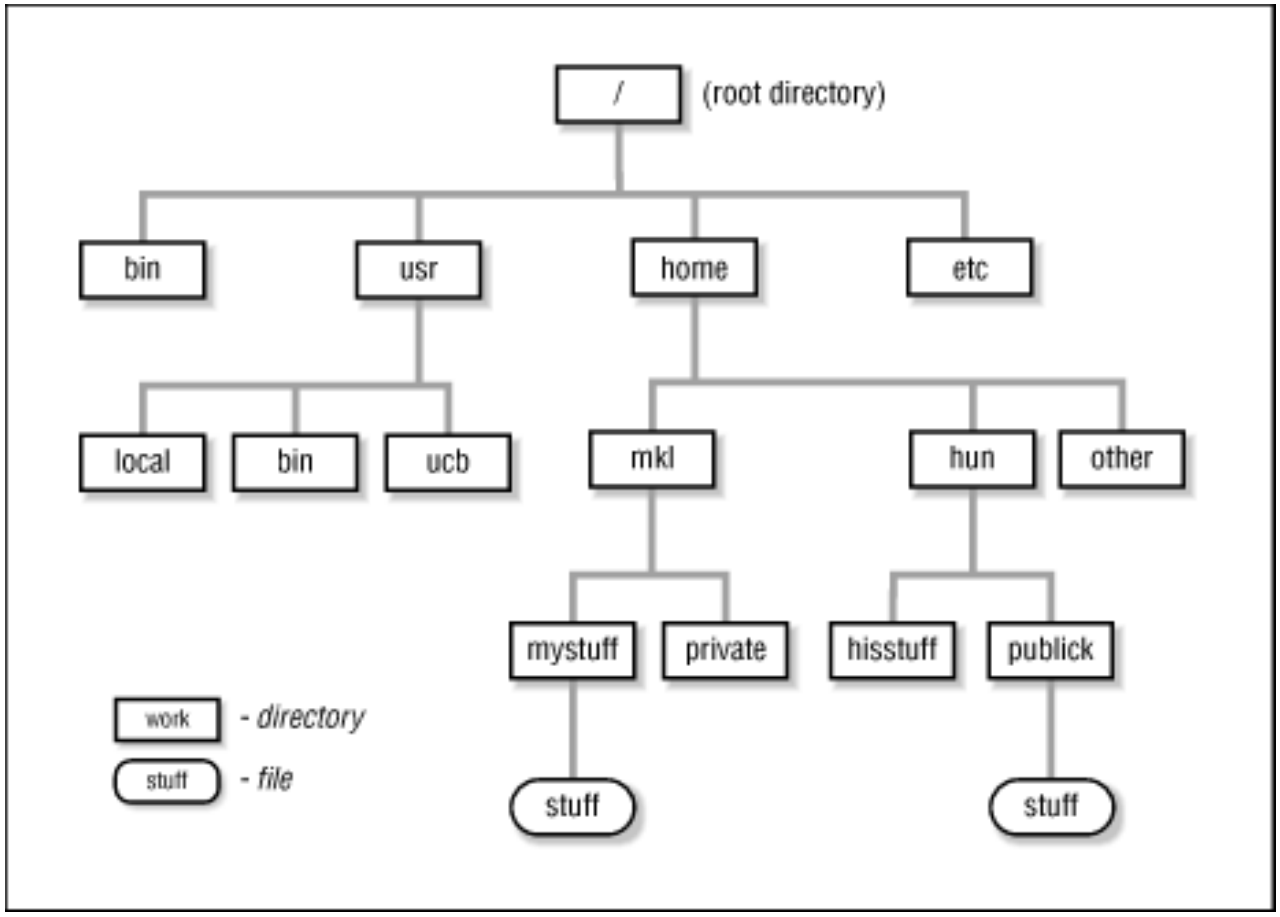
In most Operating Systems, data are stored in **files** (text, images, tables, sequences, measurement series….). Quickly, the number of files increases and it becomes necessary to organize them to avoid getting lost. This is done by grouping them in **folders or directories**. Folders can be stored in other folders, which in turn can be stored in other folders, which in turn can be stored in other folders...

The File System | The Directory Tree

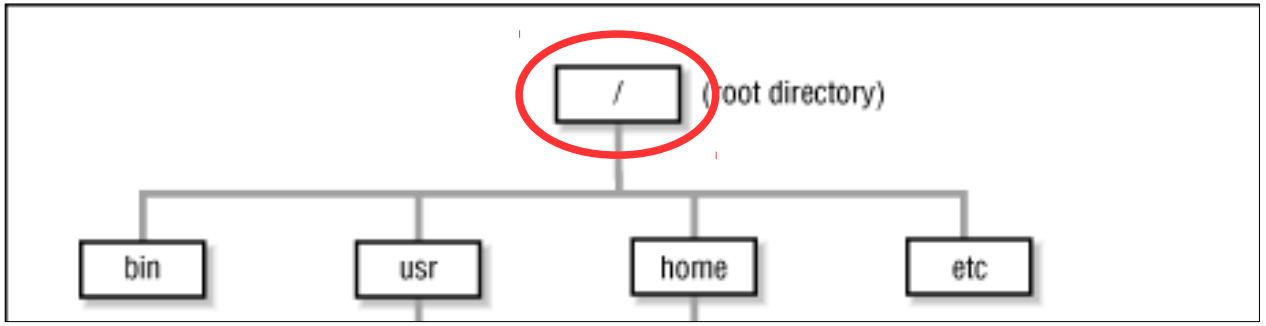
The Concept



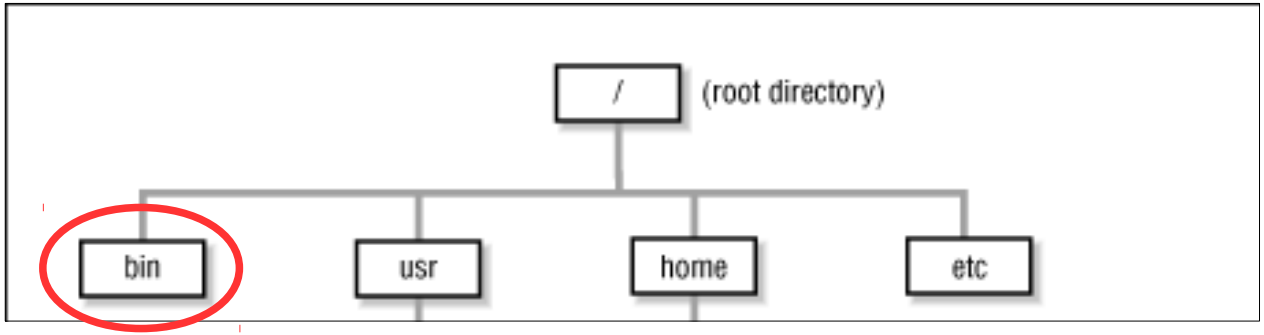
The way it has been implemented in Linux (Unix)



The File System | Some important directories

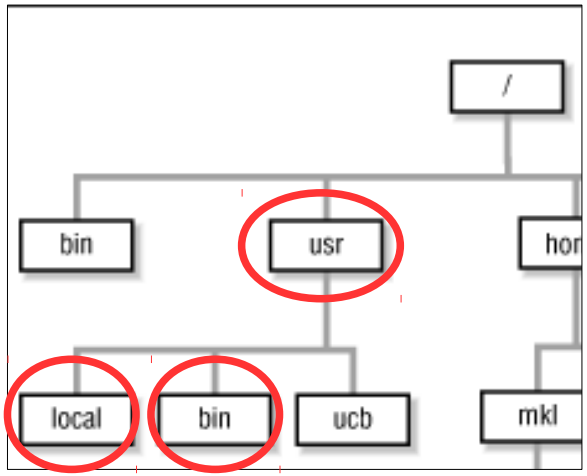


Slash* or *root directory : the **unique** (top level) entry point for the whole file system. A *path* leading to a file or a directory can always be specified starting from the root directory



/bin : directory containing the major part of system commands.

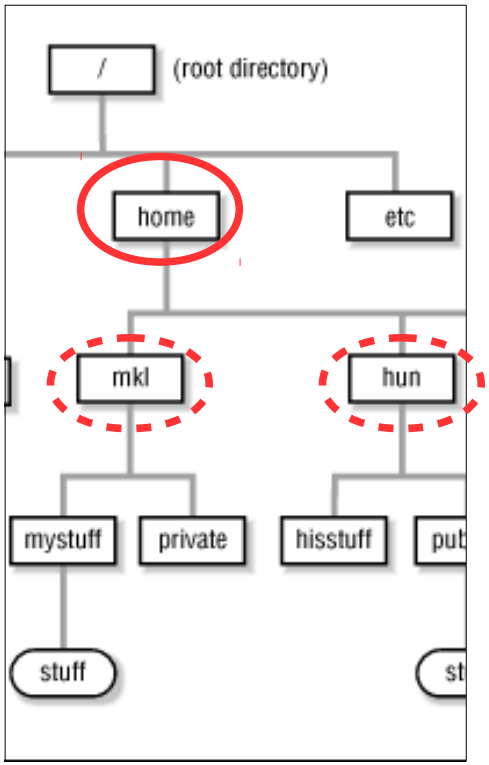
The File System | Some important directories



/usr : directory containing sub-directories with "user" commands.

/usr/bin : directory with frequently used commands (not as important to run the system as those in ***/bin***).

/usr/local : directory with subdirectories containing tools installed on this specific machine (in particular ***/usr/local/bin***)



/home : top level directory of the user data directory tree.

Its organization depends on the number of users having an account on the machine :
- few users (on workstations) : each users subdirectory is located directly in ***/home***.
- many users : user subdirectories are located in sub(sub(sub))folders.

At the SBR : the organisation matches the research unit and team layout, as in : **/home/fr2424/sib/mhoebeke.** The user directories are accessible on all (Linux) machines of the campus.

The File System | Useful directories

/tmp : a directory where anyone can read and write files (but, only a file's creator has the rights to remove her own stuff). Handy to share data with colleagues **on the same machine** (but beware the volume of data).

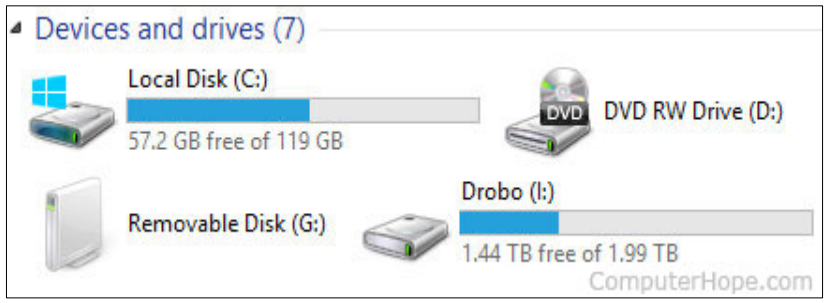
SBR Specifics

/projet : toplevel directory of a tree mimicking the ***/home*** *(unit/team/user)* directory tree and destined to hold project data than need to be backed-up (initial data sets, final results).

/scratch : toplevel directory of a directory tree mimicking the ***/home*** *(unit/team/user)* tree destined to hold "work" data such as those generated by ongoing computations, and not needing backups. "Old" files are automatically removed after a certain inactivity delay.

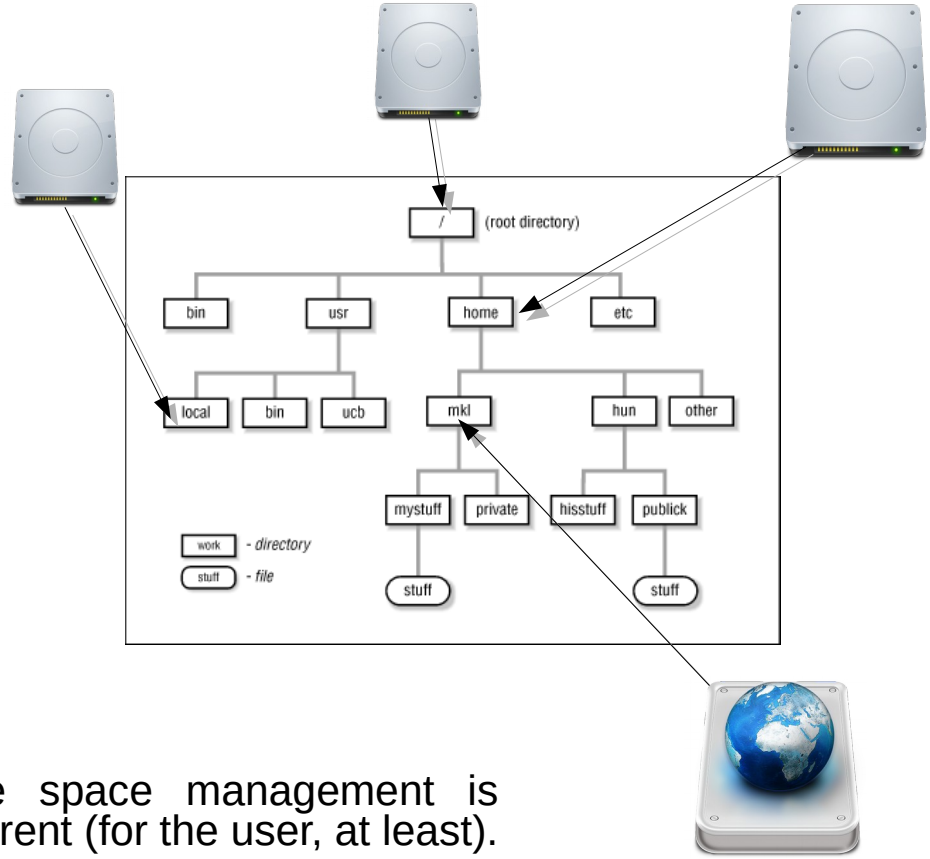
The File Systems | So where are the actual disks ?

Windows-like systems define a single letter for each storage device (hard disk, DVD reader/writer, USB key, nerwork drive), generating a **non-unique top level directory tree.**.



In order to use a file, one has to know on which drive (letter) it is physically located. Disk space extension by adding new disks leads to the definition of new letters. Moving data between disks may entail application reconfiguration.

UNIX/Linux like systems associate all devices (not only storage devices), both local and remote, to directories in the ***single rooted directory*** **tree,** through so-called *mount points.*



Storage space management is transparent (for the user, at least).

Navigating in the File System

Command execution takes place in the context of a **session,** defining at all times a **current user** (the user running the command) and a **current directory** or ***working directory*** (the directory where the command has been typed in).

When starting a new session, the current directory is always the **home directory.**

Who am I (who is the current user) ?

```
[stage01@nz ~]$ whoami
stage01
```

Where am I (what is the current directory) ?

```
[stage01@nz ~]$ pwd    # print working directory
/home/fr2424/stage/stage01
```

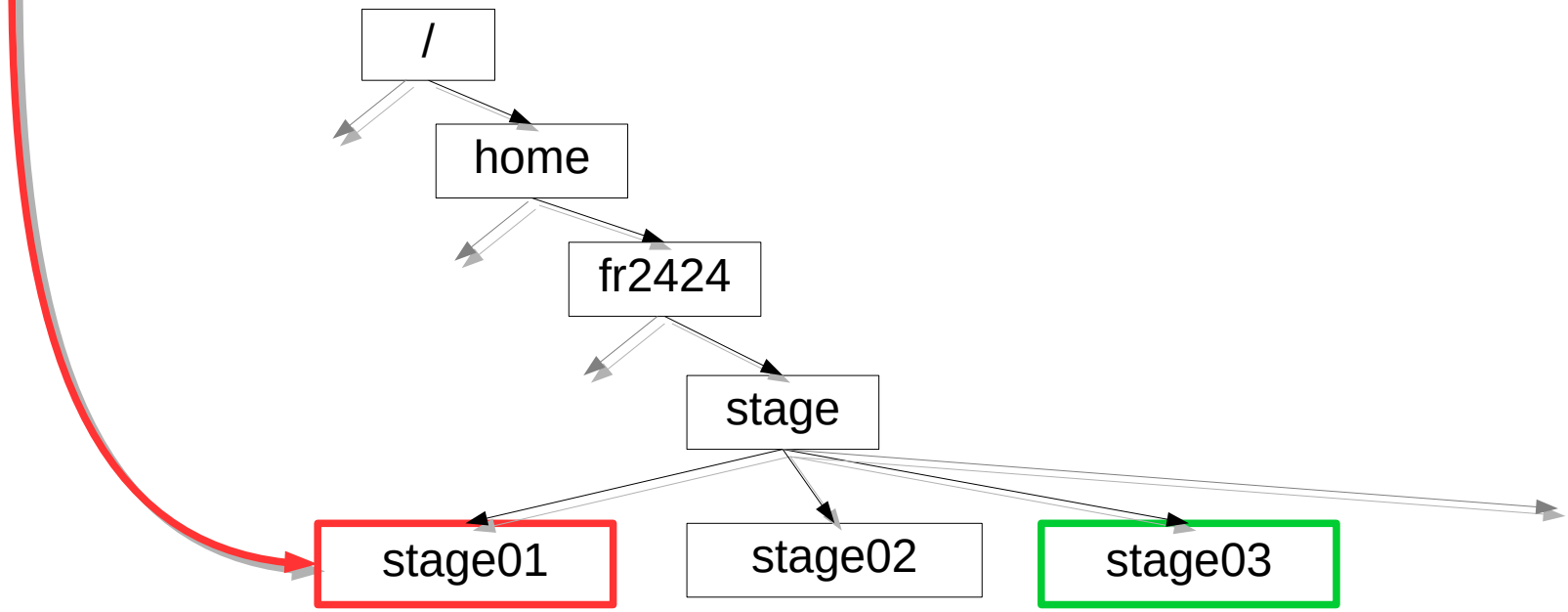
What can be found “here” (what are the contents of the current directory) ?

```
[stage01@nz ~]$ ls    # list (contents of working directory)
Bureau  Documents  Images  Modèles  Musique  Public
Téléchargements  Vidéos
```

Navigating in the File System

To change the current directory (a.k.a “move around” in the directory tree)

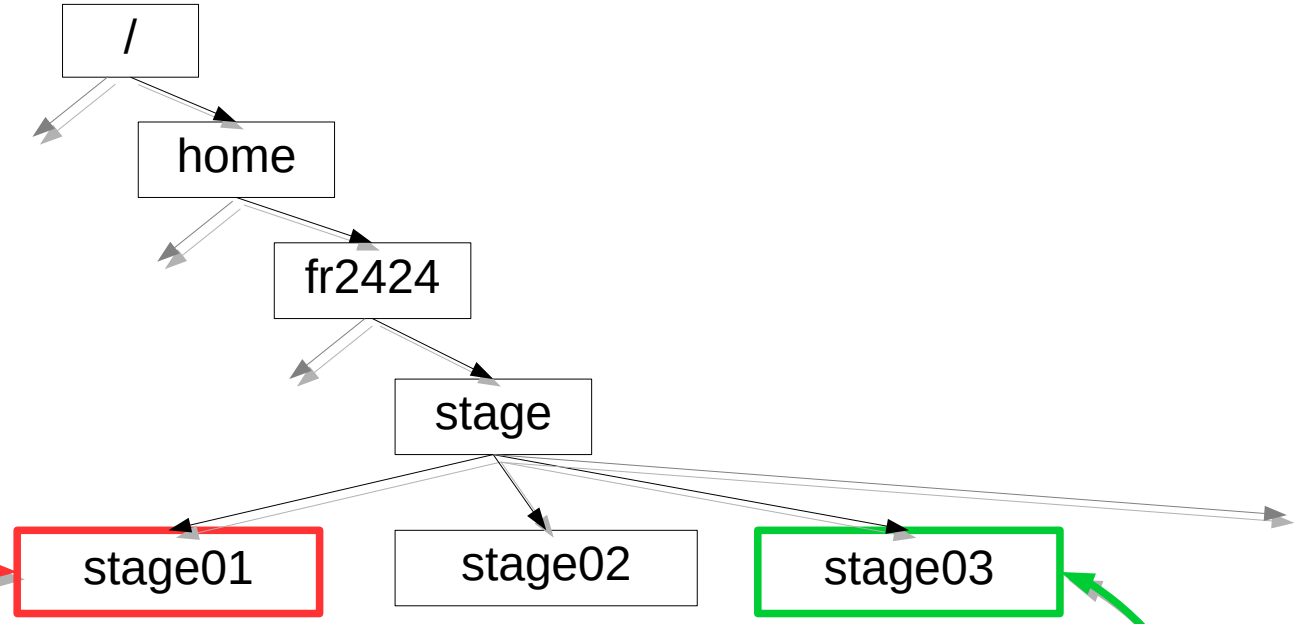
```
[stage01@nz ~]$ pwd    # print working directory
/home/fr2424/stage/stage01
```



Navigating in the File System

To change the current directory (a.k.a “move around” in the directory tree)

```
[stage01@nz ~]$ pwd    # print working directory
/home/fr2424/stage/stage01
```

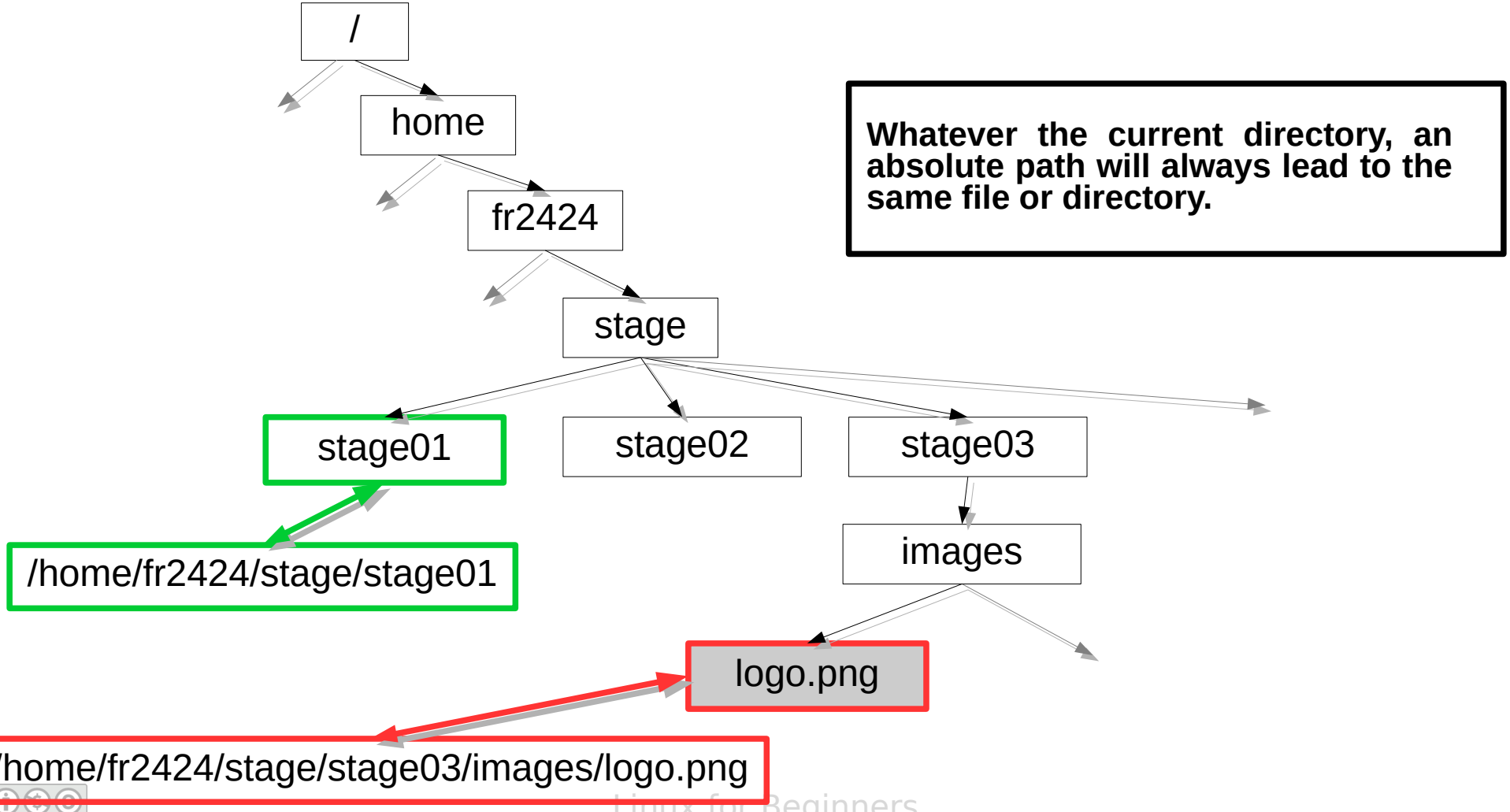



```
[stage01@nz ~]$ cd /home/fr2424/stage/stage03 # change dir
[stage01@nz ~]$ pwd
/home/fr2424/stage/stage03
```

Absolute *paths*

Referring to files and directories located in the file system, as command arguments or options, is done through **paths.**

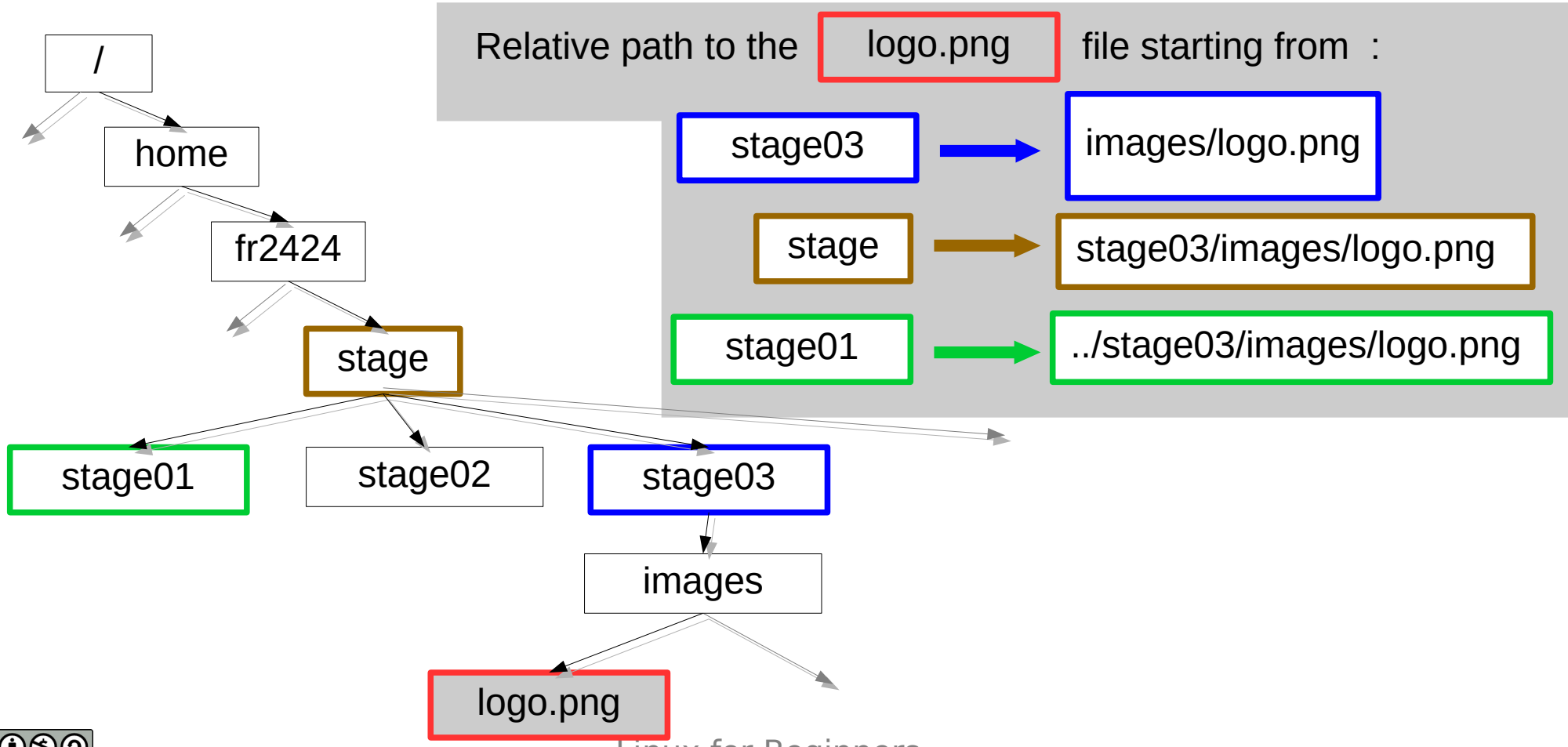
Absolute paths are built starting from the root directory and adding the subdirectories one by one separated by a slash character (/), until the desired file or directory is reached.



Whatever the current directory, an absolute path will always lead to the same file or directory.

Relative *paths*

Relative paths are built **with the current directory as starting point** and traversing the directory tree upwards or downwards, until the desired directory or file is reached. The successive path components are separated with *slash (/)* characters, and :
- On each **upward step** in the tree, two dots **(..)** are added to the path.
- On each **downward step** in the tree, the name of the directory is added to the path.



A (provisional) conclusion

A shortcut to refer to the current directory :

- The dot (“.” character) always refers to the current directory

```
[stage01@nz ~]$ pwd
/home/fr2424/stage/stage01
[stage01@nz ~]$ cd . # ???
[stage01@nz ~]$ pwd
/home/fr2424/stage/stage01
```

Use case : run a command file located in the current directory : `./mycommand`

A shortcut for the home directory :

- The tilde (“~” character) always refers to the home directory of the current user

```
[stage01@nz ~]$ pwd
/home/fr2424/stage/stage03
[stage01@nz ~]$ cd ~ # change to home dir
[stage01@nz ~]$ pwd
/home/fr2424/stage/stage01
```

Commands for manipulating files & directories

Listing the contents of a directory: `ls`

```
[stage11@nz ~]$ ls                    # no arguments : current dir
Bureau  Documents  Images  Modèles  Musique  Public  Téléchargements
Vidéos
[stage11@nz ~]$ ls /tmp/Linux-Initiation # absolute dir path
acteur.csv  cours  insulin.fas  insulin_vs_nt.blast  tmp
[stage11@nz ~]$ ls .. # relative dir path (parent dir)
common                stage02  stage1   stage17  stage24  stage31  stage6
common.linux-avance   stage03  stage10  stage18  stage25  stage32  stage7
(...)
```

"Hidden" files (and directories)

By default, **`ls`** *does not* display files having names starting with a dot. The **`-a`** (all) option needs to be added for them to be included (the **`la`** shortcut can also be used instead of **`ls`**).

```
[stage11@nz ~]$ ls -a                  # also show hidden files
.              Bureau    Documents      .kde      Public              .zshrc
..             .cache    .emacs         .local    Téléchargements
.bash_logout   .compiz   .gconf         Modèles   Vidéos
(...)
```

Listing files matching a particular pattern

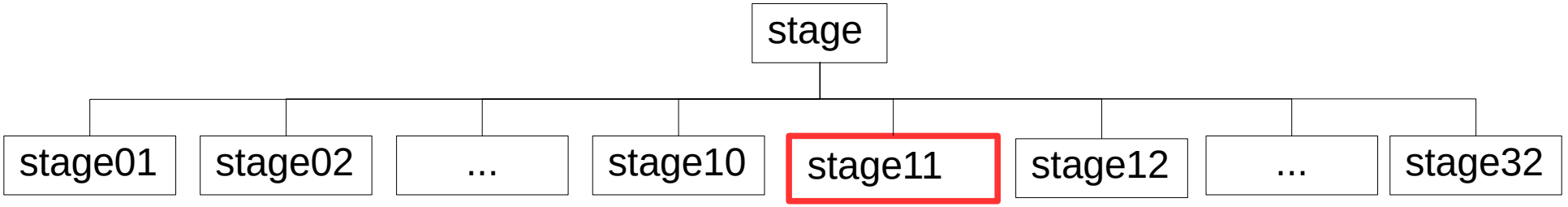
Using the * character in an argument of **ls** restricts the list to the files and directories whose names match the pattern formed by the argument:

- **image*** : all files starting with the letters *image* (**image-001**, **images-des-vacances**, **imagettes**)
- ***seq*** : all files having the letters *seq* in their names (**sequences**, **mes-sequences**, **maiseqoidon**)
- ***** : each and every file (no restriction)

```
[stage11@nz ~]$ ls Linux-Initiation/*ins*
Linux-Initiation/insulin.fas  Linux-Initiation/insulin_vs_nt.blast
```

Using *autocompletion*

To avoid to have to type in long filenames it is possible to use the **[TAB]** key. Pressing the **[TAB]** key once launches a file name (or directory) lookup to determine which ones start with what has already been typed.
-If there is a single match, it will be added to the command line,
-If there are several matches another press of the **[TAB]** key will list them all.




```
[stage11@nz ~]$ ls ../sta[TAB]
[stage11@nz ~]$ ls ../stage
[stage11@nz ~]$ ls ../stage[TAB]
stage01/ stage07/ stage12/ stage18/ stage23/ stage29/ stage34/ stage7/
stage02/ stage08/ stage13/ stage19/ stage24/ stage3/  stage35/ stage8/
stage03/ stage09/ stage14/ stage2/  stage25/ stage30/ stage36/ stage9/
(...)
```

Displaying the contents of a whole directory (sub)tree with a single command

Through the **ls** command, to which the **-R** (recursive) option is added:

```
[stage11@nz ~]$ ls -R Linux-Initiation
Linux-Initiation/:
acteur.csv  cours  insulin.fas  insulin_vs_nt.blast  tmp
Linux-Initiation/cours:
Linux-Initiation/tmp:
```

With the **tree** command:

```
[stage11@nz ~]$ tree Linux-Initiation
Linux-Initiation
├── acteur.csv
├── cours
├── insulin.fas
├── insulin_vs_nt.blast
└── tmp

2 directories, 3 files
```

Organising data by creating subdirectories

With the **`mkdir`** (*make directory)* command. By default, only the last directory of the path given as argument is created:

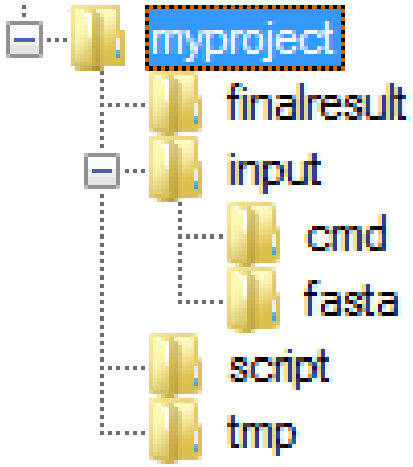
```
[stage11@nz ~]$ mkdir Linux-Initiation/tmp/essais
[stage11@nz ~]$ ls -R Linux-Initiation
Linux-Initiation/:
acteur.csv  cours  insulin.fas  insulin_vs_nt.blast  tmp
Linux-Initiation/cours:
Linux-Initiation/tmp:
Linux-Initiation/tmp/essais:
```

The **`-p`** option enables the creation of a whole subtree in a single step:

```
[stage11@nz ~]$ mkdir -p Linux-Initiation/exercices/ex1/data
[stage11@nz ~]$ ls -R Linux-Initiation/
(...)
Linux-Initiation/exercices:
ex1
Linux-Initiation/exercices/ex1:
data
Linux-Initiation/exercices/ex1/data:
```

Exercises

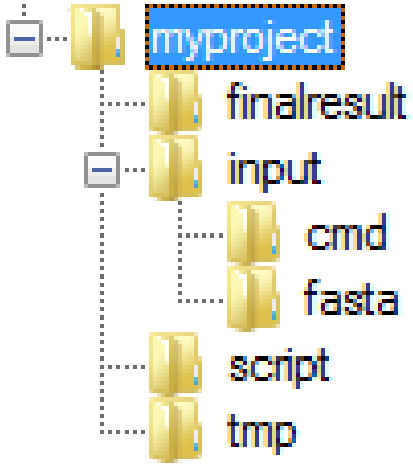
- Create the following directory structure in your project directory:



- Check it has been correctly created by displaying it :

Exercises

- Create the following directory structure in your project directory:



Solution

```
$ cdprojet
$ mkdir myproject
$ cd myproject
$ mkdir finalresult input script tmp
$ cd input
$ mkdir cmd fasta
```

- Check it has been correctly created by displaying it :

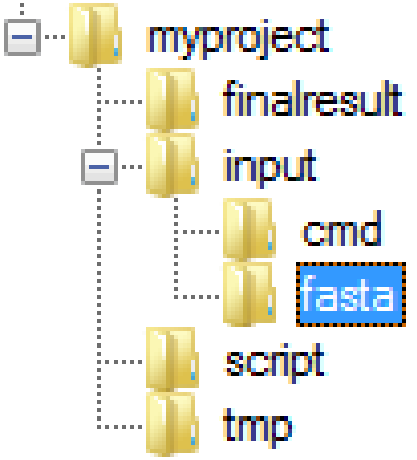
Solution

```
$ tree
$ tree -L 1
$ tree -L 2
```

Exercises

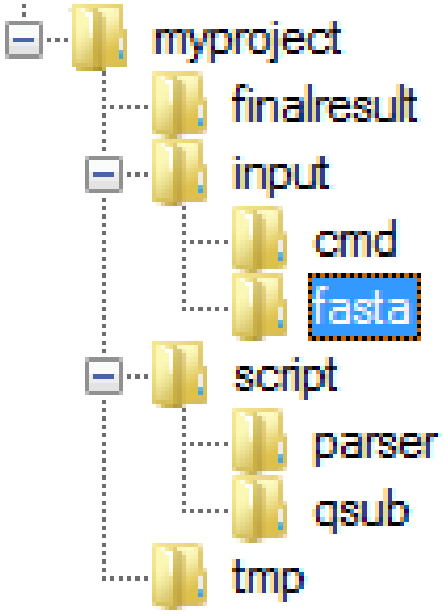
Using a single command line for each of the following items :

- Return to your project directory



- Change to the **`fasta`** directory

- Create a **`parser`** directory in the **`script`** directory



Using a single command line for each of the following items :

- Return to your project *directory*

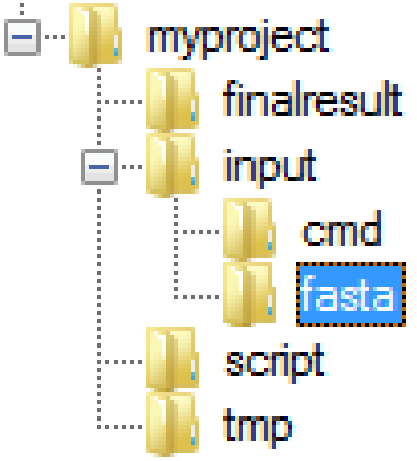
Solution

```
$ cdprojet
```

- Change to the **`fasta`** directory

Solution

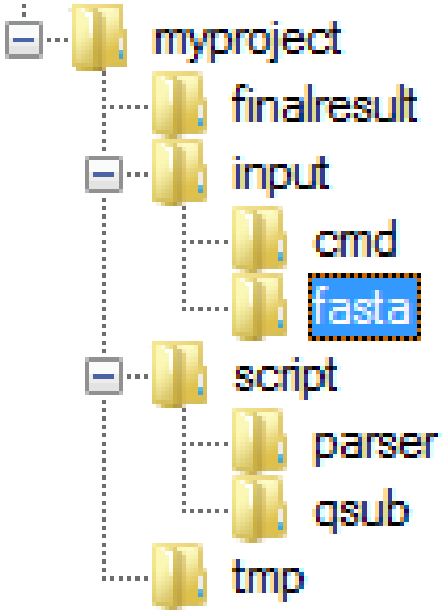
```
$ cd myproject/input/fasta
```



- Create a **`parser`** directory in the **`script`** directory

Solution

```
$ mkdir ../../script/parser
```



Copying data

The **cp** *(copy)* command copies one or more files, and even whole directory trees. It is used as follows : **cp** ***SRC DEST*** where **SRC** is the path to the already existing data (the source) and **DEST** is the path the the destination location.

Ex. 1 : copying a single file

```
[stage11@nz ~]$ cp acteur.csv acteur_bak.csv
```

Ex. 2 : copying a single file in another directory

```
[stage11@nz ~]$ cp acteur.csv tmp # keep same filename in DEST
[stage11@nz ~]$ cp acteur.csv tmp/stars.csv # change filename
```

Ex. 3 : copying a set of files matchin a pattern to another directory

```
[stage11@nz ~]$ cp insulin* tmp
[stage11@nz ~]$ ls tmp
insulin.fas  insulin_vs_nt.blast
```

Ex. 4 : copying a complete directory structure using the **-r** *(recursive)* option

```
[stage11@nz ~]$ cp -r ../stage10/exercices/solutions  .
```

Move or rename data

The **mv** *(move)* command, depending on its arguments, either renames or moves one or more files, and possibily whole directories.
It is used as follows: **mv *SRC DEST*** where **SRC** is the path to the already existing data (the source data) and **DEST** either the new name for the file or the directory in which it has to be moved.

Ex. 1 : renaming a single file

```
[stage11@nz ~]$ mv acteur.csv liste_acteurs.csv
```

Ex. 2 : moving a single file to an **(already existing)** directory

```
[stage11@nz ~]$ mv acteur.csv tmp # keep same filename in DEST
[stage11@nz ~]$ mv acteur.csv tmp/stars.csv # change filename
```

Ex. 3 : moving a set of files matching a pattern to another **(existing)** directory

```
[stage11@nz ~]$ mv insulin* tmp
```

Ex. 4 : moving a complete directory structure

```
[stage11@nz ~]$ mv tmp/work/last_stage/output ./finalresults
```

- If **./finalresults** already exists, the **output** y directory will be moved into it.
- if **./finalresults** doesn't exist, it will be created and will contain all the data previously located in **output**

Deleting data

The **rm** *(remove)* command deletes th file(s) whose path(s) are given as argument.

What is deleted with rm cannot be restored.

(jamais, never, jamas, nie, nooit, gwech ebet, någonsin, никогда, 曾經)

Ex. 1 : removing a single file

```
[stage11@nz ~]$ rm acteur_bak.csv
```

Ex. 2 : removing a set of files matching a pattern

```
[stage11@nz ~]$ rm insulin*
```

Ex. 3 : removing a complete directory structure using the **-r** *(recursive)* option

```
[stage11@nz ~]$ rm -r Linux-Initiation/tmp
```

Ex. 4 : **Armageddon** : **forced** (**-f**) removal of a whole directory structure

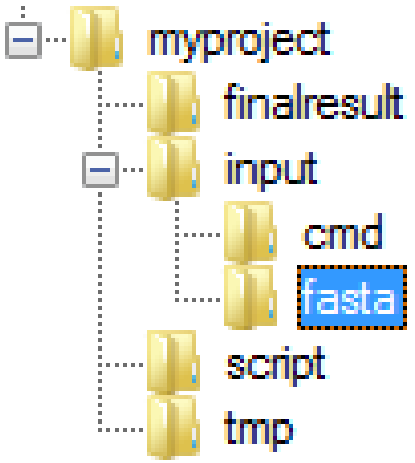
```
[stage11@nz ~]$ rm -rf ~/tmp/worthless_files
```

Special case : removal of an empty directory with the **rmdir** command

```
[stage11@nz ~]$ rmdir ~/tmp/empty_directory
```

Copy the **`insulin.fas`** file in the **`fasta`** directory

- Go to your project directory



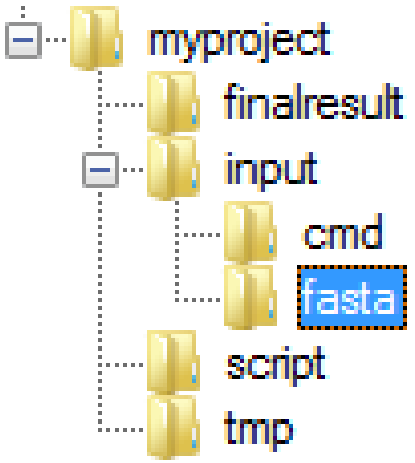
- Copy the file to its destination directory

Copy the **insulin.fas** file in the **fasta** directory

- Go to your project directory

Solution

```
$ cdprojet
```



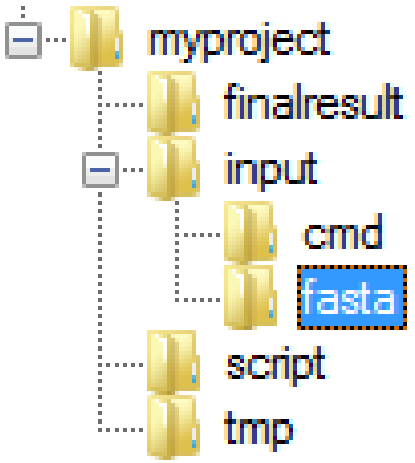
- Copy the file to its destination directory

Solution

```
$ cp Linux-initiation/insulin.fas myproject/input/fasta
```

After making **`myproject/finalresult`** your current directory

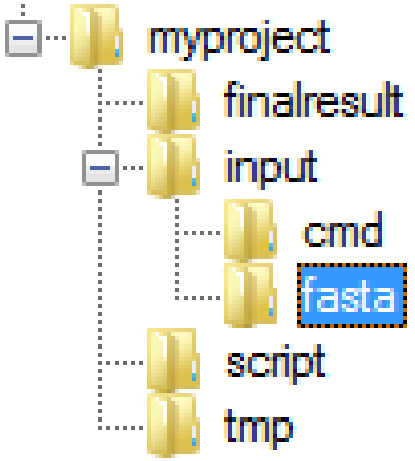
- Move the **`insulin.fas`** file from the **`input/fasta`** directory to the **`tmp`** directory



- Remove the **`tmp`** directory and all its contents

After making **myproject/finalresult** your current directory

- Move the **insulin.fas** file from the **input/fasta** directory to the **tmp** directory



Solution

```
$ cdprojet
$ cd myproject/finalresult
$ mv ../input/fasta/insulin.fas ../tmp
```

- Remove the **tmp** directory and all its contents

Solution

```
$ cdprojet
$ rm -r myproject/tmp
```

Linux for Beginners | Outline

A few words about file names (1)

Linux is **very** permissive about valid characters in file names (space characters, accents…). It's safer to avoid using them widely. Some recommendations:

Uppercase & lowercase characters ; digits ; dash ; underscore ; dot.

Characters with accents or other diacritical signs

Space characters or other punctuation marks

The "case of the space" character : it can be *despecialized* with the *backslash* (\) or the double quotes (")

```
[n00b@nz ~]$ mkdir nouveau dossier # creates 2 dirs, :(
[n00b@nz ~]$ ls .
./:
nouveau
dossier
```

```
[tux@nz ~]$ mkdir nouveau\ dossier # creates 1 dir, gg
[tux@nz ~]$ mkdir "nouveau dossier 2" # id.
[tux@nz ~]$ ls .
./:
nouveau dossier
Nouveau dossier 2
```

A few words about file names (2)

Linux doesn't put any requirements on file name extensions (`.txt`, `.csv`, `.pdf`, `.html`, etc.). Any extension can be given to any type of file.

IT IS STRONGLY RECOMMENDED TO REMAIN CONSISTENT

Linux uses other recipes to determine the nature of a file's contents (cf. the `file` command).

Determining the nature of a file (1)

The **`file`** command displays a hypothesis about a file's nature. It examines the beginning of the file and compares this *fingerprint* to an internal "database" of fingerprints.

Ex. 1 & 2 : Application specific files.

```
[stage11@nz ~]$ file Linux-Initiation-2017.pdf
Linux-Initiation-2017.pdf: PDF document, version 1.4
```

```
[stage11@nz ~]$ file Linux-Initiation-2017.pptx
Linux-Initiation-2017.pptx: Microsoft PowerPoint 2007+
```

Ex. 3 & 4 : Compressed archive files.

```
[stage11@nz ~]$ file Linux-Initiation-supports.zip
Linux-Initiation-2017-supports.zip: Zip archive data, at
least v2.0 to extract
```

```
[stage11@nz ~]$ file Linux-Initiation-supports.tar.gz
Linux-Initiation-2017-supports.tar.gz: gzip compressed data,
last modified: Sat May 7 23:56:36 2017, from Unix
```

Determining the nature of a file (2)

Ex. 5 & 6 : Executable files (binary commands or series of commands in a text file)

```
[stage11@nz ~]$ file /usr/bin/file
/usr/bin/file: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux
2.6.32, BuildID[sha1]=a4f09f32eb214a3f9435484fa01d54c939bcf30c, stripped
```

```
[stage11@nz ~]$ file monscript.sh # textfile with commands
monscript.sh: POSIX shell script, ASCII text executable
```

Ex. 7 & 8 : Text files

```
[stage11@nz ~]$ file insulin.fas
acteur.csv: ASCII text, with CRLF line terminators
```

```
[stage11@nz ~]$ file acteur.csv
acteur.csv: ASCII text, with CRLF line terminators
```

Ex. 9 : Files with data in a format unknown to the **`file`** command.

```
[stage11@nz ~]$ file random.dat
random.dat: data
```

Examining the contents of a (text) file

The **cat** command displays the entire contents of a file.

```
[stage11@nz ~]$ cat acteur.csv
First Name;Last Name;Age
Chuck;Norris;70
Sylvester;Stallone;64
Steven;Seagal;59
```

The **head** command displays the first (10) lines of a file. **head -*n*** displays the ***n*** first lines.

```
[stage11@nz ~]$ head -2 acteur.csv
First Name;Last Name;Age
Chuck;Norris;70
```

The **tail** command displays the (10) last lines of a file. **tail -*n*** displays the ***n*** last ones.

```
[stage11@nz ~]$ tail -2 acteur.csv
Sylvester;Stallone;64
Steven;Seagal;59
```

Manipulating File Contents

Interactively examining the contents of a (text) file

The **more** command displays the contents of a file “one page at a time”. The space bar moves from the current page to the next; and “q” is used to quit.

```
[stage11@nz ~]$ more insulin.fas
>gi|163659904|ref|NM_000618.3| Homo sapiens insulin-like growth factor
1 (somatomedin C) (IGF1), transcript variant 4, mRNA
TTTTGTAGATAAATGTGAGGATTTTCTCTAAATCCCTCTTCTGTTTGCTAAATCTCACTGTCACTGCTAA
(…)
--More-- 4%
```

The **less** command also displays the contents of a file “one page at a time”. The spacebar moves from the current page to the next; and “q” is used to quit. The ⬆ and ⬇arrows allow to move back and forth in the file.

```
[stage11@nz ~]$ less insulin.fas
>gi|163659904|ref|NM_000618.3| Homo sapiens insulin-like growth factor
1 (somatomedin C) (IGF1), transcript variant 4, mRNA
TTTTGTAGATAAATGTGAGGATTTTCTCTAAATCCCTCTTCTGTTTGCTAAATCTCACTGTCACTGCTAA
(…)
insulin.fas
```

Both **more** and **less** allow to search for a text string in the file. This is done by typing slash (**/**) followed by the text to search for, and then typing **Enter**↵ Ex : **/variant**

Searching for information in a file (1)

The **grep** command takes two arguments : a ***pattern*** and a ***file name*** ; it displays every line of the file containing the pattern.

```
[stage11@nz ~]$ grep transcript insulin.fas
>gi|163659904|ref|NM_000618.3| Homo sapiens insulin-like growth factor 1 (somatomedin C) (IGF1),
transcript variant 4, mRNA
>gi|163659900|ref|NM_001111284.1| Homo sapiens insulin-like growth factor 1 (somatomedin C) (IGF1),
transcript variant 2, mRNA
>gi|163659895|ref|NM_001111276.1| Mus musculus insulin-like growth factor 1 (Igf1), transcript
variant 5, mRNA
>gi|163659893|ref|NM_001111275.1| Mus musculus insulin-like growth factor 1 (Igf1), transcript
variant 4, mRNA
>gi|163659891|ref|NM_010512.4| Mus musculus insulin-like growth factor 1 (Igf1), transcript variant
1, mRNA
```

By default, **grep** is case sensitive. To override this behaviour, the **-i** (ignorecase) option can be used.

```
[stage11@nz ~]$ grep TRANSCRIPT insulin.fas # returns nothing
[stage11@nz ~]$ grep -i TRANSCRIPT insulin.fas
>gi|163659904|ref|NM_000618.3| Homo sapiens insulin-like growth factor 1 (somatomedin C) (IGF1),
transcript variant 4, mRNA
>gi|163659900|ref|NM_001111284.1| Homo sapiens insulin-like growth factor 1 (somatomedin C) (IGF1),
transcript variant 2, mRNA
(...)
```

Searching for information in a file (2)

grep also allows to count the lines matching the pattern with the **-c** (count) option.

```
[stage11@nz ~]$ grep -c transcript insulin.fas
5
```

As is the case for almost every command, options for **grep** can be combined.

```
[stage11@nz ~]$ grep -c -i TRANSCRIPT insulin.fas
5
```

By default, **grep** can also display the lines ***not containing*** the pattern, thanks to the **-v** (in**v**ert) option.

```
[stage11@nz ~]$ grep -v -c -i TRANSCRIPT insulin.fas
511
```

grep can be used to search for a pattern in all the files of a directory tree, with the **-r** (recursive) option. In this configuration, the second argument has to be the name of a directory. The lines with the information about the patterns are then prefixed with the filename to which they belong.

```
[stage11@nz ~]$ grep -r -c -i TRANSCRIPT .
./insulin_vs_nt.blast:144
./acteur.csv:0
./insulin.fas:5
```

Exercises

Find two ways of displaying the first line of the **`acteur.csv`** file (using two different commands)

Find two ways to display the last three lines of the **`acteur.csv`** file (using two different commands) [Granted, one is quite tricky.]

Exercises

Find two ways of displaying the first line of the **`acteur.csv`** file (using two different commands)

Solution

```
$ head -1 acteur.csv
$ grep First  acteur.csv
```

Find two ways to display the last three lines of the **`acteur.csv`** file (using two different commands) [Granted, one is quite tricky.]

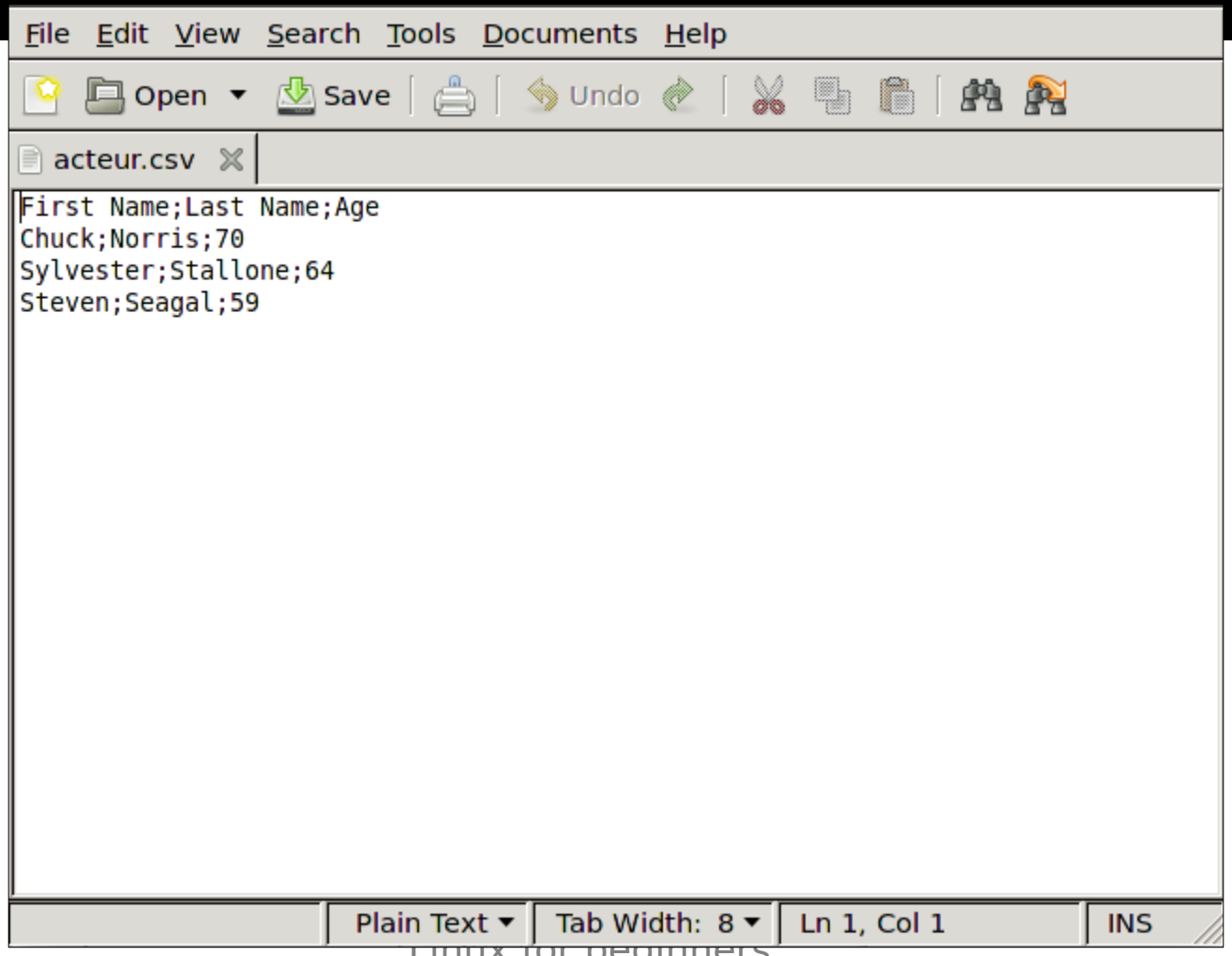
Solution

```
$ tail -3 acteur.csv
$ grep -v First  acteur.csv
```

Changing the contents of a (text) file

The **gedit** command opens a window with a text editor.

```
[stage11@nz ~]$ gedit acteur.csv
```



Changing the contents of a (text) file

The **nano** command opens a text editor in the window of the active session.

```
[stage11@nz ~]$ nano acteur.csv
```

```
  GNU nano 2.7.4                  Fichier : acteur.csv

First Name;Last Name;Age
Chuck;Norris;70
Sylvester;Stallone;64
Steven;Seagal;59

                [ Lecture de 5 lignes (converties du format DOS) ]
^G Aide       ^O Écrire     ^W Chercher   ^K Couper     ^J Justifier  ^C Pos. cur.
^X Quitter    ^R Lire fich. ^\ Remplacer  ^U Coller     ^T Orthograp. ^_ Aller lig.
```

File management & Archiving

It is possible to determine the file size using the **`-l (-h)`** options of the **`ls`** command. The file size is displayed in the fifth column.

```
[stage08@nz ~]$ ls -l insulin_vs_nt.blast
-rw-r--r-- 1 stage08 stage 30025889 May 03 22:42 insulin_vs_nt.blast
[stage08@nz ~]$ ls -l -h  insulin_vs_nt.blast
-rw-r--r-- 1 stage08 stage 29M May 03 22:42 insulin_vs_nt.blast
```

The **`wc`** (word count) command also displays information about the size of a file : line, word and character count. Using the **`-l`** option restricts the output to the number of lines.

```
[stage08@nz ~]$ wc insulin_vs_nt.blast
622756  2377511 30025889 insulin_vs_nt.blast
[stage08@nz ~]$ wc -l insulin_vs_nt.blast
622756  insulin_vs_nt.blast
```

File management & Archiving

To determine the size of a directory (and all its contents), the **du** *(disk usage)* command is used, preferably with the **-h** *(human readable) option*.

```
[stage08@nz ~]$ du -h .
64.0K   ./tmp
4.0K    ./cours
29M     .
```

Adding the **-s** (summary) option displays the total volume occupied by the directory (and its contents).

```
[stage08@nz ~]$ du -s -h .
29M     .
```

File management & Archiving

To determine how much disk space is available on a mount point (a disk partition), the **df** command is used (which also comes with an **-h** option).

```
[stage08@nz ~]$ df -h
Filesystem                      Size  Used Avail Use% Mounted on
/dev/sda7                      1008M  750M  208M  79% /
(…)
/dev/sda1                       504M  152M  327M  32% /boot
(…)
/dev/sda2                        32G  541M   30G   2% /tmp
/dev/sda5                        16G  9.5G  5.5G  64% /usr
/dev/mapper/VolGroup00-LogVol00 4.0G  3.0G  795M  80% /usr/local
(…)
brazil:/home/umr7139/mma        247G  210G   25G  90% /home/umr7139/mmaucture
brazil:/home/umr7139/tccd       591G  538G   24G  96% /home/umr7139/tccd
brazil:/home/umr7144/abice      1.2T  440G  636G  41% /home/umr7144/abice
(…)
```

When specifying a directory as argument, **df** displays information about the mounted file system containing the directory.

```
[stage08@nz ~]$ df -h /home/fr2424/sib/mhoebeke
Filesystem            Size  Used Avail Use% Mounted on
brazil:/home/fr2424/sib  2.0T  1.6T  242G  88% /home/fr2424/sib
(…)
```

File management & Archiving

Linux makes available different commands for compressing files, like **gzip** (older) or **bzip2** (better compression, a little less portable between systems). By default **gzip** replaces the file whose name is given as argument with its compressed version and adds **.gz** to the file name (**.bz2** for **bzip2**).

```
[stage08@nz ~]$ gzip insulin_vs_nt.blast
[stage08@nz ~]$ ls -l -h insulin_vs_nt.blast.gz
-rw-r--r-- 1 stage08 stage 5.0M May 13 20:42 insulin_vs_nt.blast.gz
```

```
[stage08@nz ~]$ bzip2 insulin_vs_nt.blast
[stage08@nz ~]$ ls -l -h insulin_vs_nt.blast.gz
-rw-r--r-- 1 stage08 stage 3.2M May 13 20:42 insulin_vs_nt.blast.bz2
```

Decompression is achieved with **gunzip** (for **.gz** files) or **bunzip2** (for **.bz2** files).

```
[stage08@nz ~]$ gunzip insulin_vs_nt.blast.gz
```

```
[stage08@nz ~]$ bunzip2 insulin_vs_nt.blast.bz2
```

The compression ratio depends on the file contents : better for text files, quite low for already compressed data (images, sounds, videos).

File management & Archiving

Creating an archive containing several files is possible with the **tar** command.

Ex 1 : Creation of an archive with the contents of the **Linux-Initiation** directory: the **-c** option stands for “creation”, the **-f** option allows to specify the file name of the archive file name just afterwards. The final argument is the name of the directory from which to build the archive.

```
[stage08@nz ~]$ tar -cf Linux-Initiation.tar Linux-Initiation
```

tar doesn’t modify the directory to archive in any way.

Ex 2 : Extraction of all the files of the previously created archive: the **-x** option stands for “eXtraction”, the **-f** option has the same meaning as above. The **-v** option activates the “verbose mode” displaying each file name as it is extracted.

```
[stage08@nz ~]$ tar -xvf Linux-Initiation.tar
Linux-Initiation/
Linux-Initiation/tmp/
(...)
```

Ex 3 : Listing the contents of an archive : the **-t** option (“toc”) displays the list of files in an archive.

```
[stage08@nz ~]$ tar -tf Linux-Initiation.tar
Linux-Initiation/
(...)
```

Ex 4 : Extraction of a single file : the name of the file to be extracted is added as argument.

```
[stage08@nz ~]$ tar -xvf Linux-Initiation.tar Linux-Initiation/insulin.fas
Linux-Initiation/insulin.fas
```

File management & Archiving

The **tar** command can be asked to carry out “on the fly” (de-)compression. To use **gzip** for this, the **-z** option is added. To use **bzip2**, the **-j** option can be used.

Ex1. : On the fly compression/decompression with **gzip**.

```
[stage08@nz ~]$ tar -czf Linux-Initiation.tar.gz Linux-Initiation
[stage08@nz ~]$ file Linux-Initiation.tar.gz
Linux-Initiation.tar.gz: gzip compressed data (…)
[stage08@nz ~]$ tar -xzf Linux-Initiation.tar.gz
```

Ex1. : On the fly compression/decompression with **bzip2**.

```
[stage08@nz ~]$ tar -cjf Linux-Initiation.tar.bz2 Linux-Initiation
[stage08@nz ~]$ file Linux-Initiation.tar.bz2
Linux-Initiation.tar.bz2: bzip2 compressed data (…)
[stage08@nz ~]$ tar -xjf Linux-Initiation.tar.bz2
```

Using shortcuts : symbolic links

It is often handy to be able to access a set of files in a single directory, even if they are originally scattered in several directories. And to do so **without copying any data.**

Use case :
-One of my directories, **`allsequences`** qcontains a myriad of sequence files related to various organisms.
-The file names hint to the organism to which the sequences they contain belong (**`human_seq*.fasta`, `mouse_seq*.fasta`, `ecto_seq*.fasta`,** etc.)
-I wish to apply routines to these sequences whose parameters may depend on the organism. And I wish to group de result files in organism specific (a.k.a separate) directories (**`process_human/`, `process_mouse/`, `process_ecto/`,** etc.)
-But I want to avoid copying the sequence data in these **`process_*/`** directories.

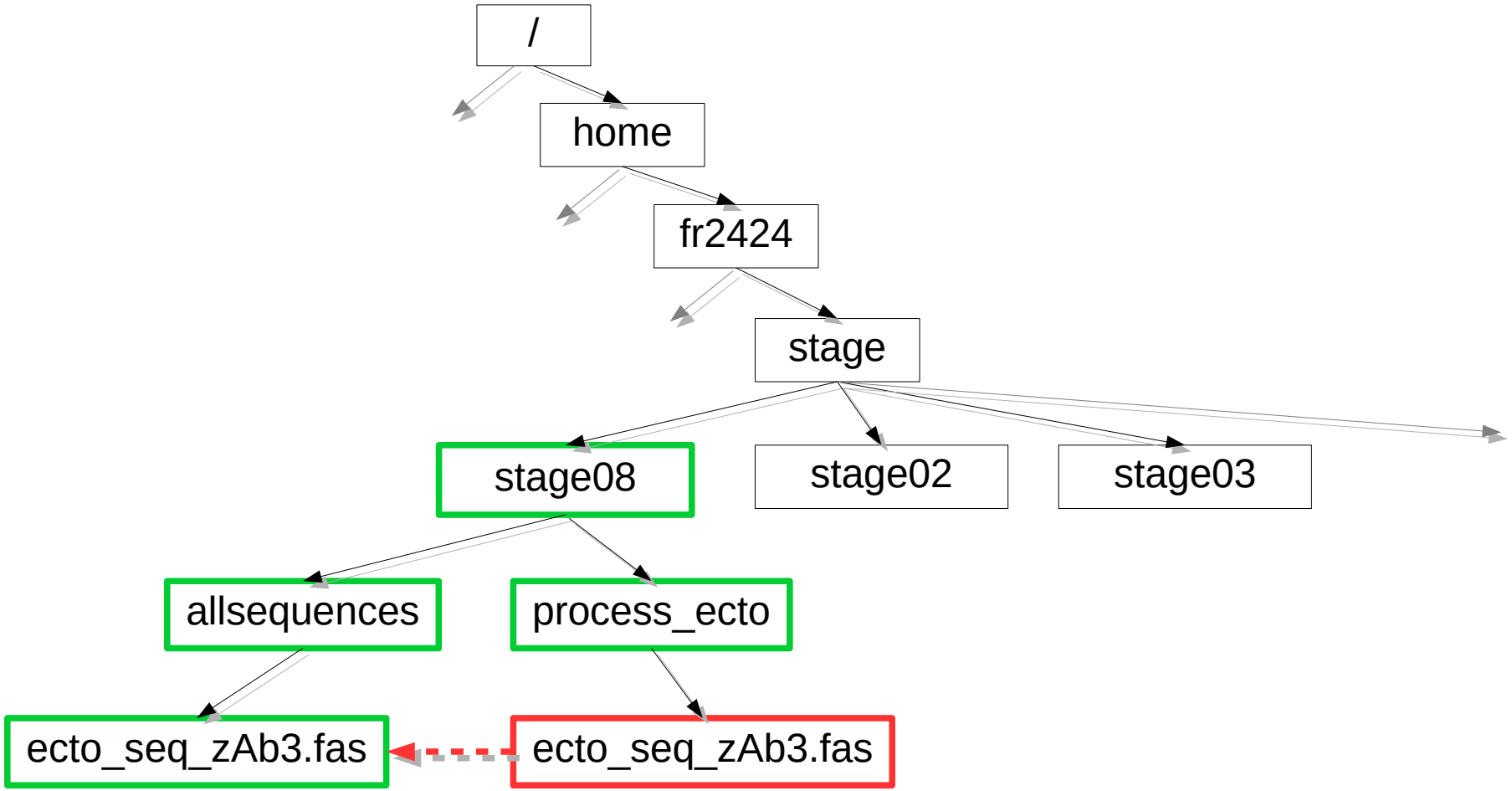
A solution relies on the creation, in the **`process_*/`** directories, of shortcuts to each sequence file belonging to a given organism, using the **`ln -s`** *(link, symbolic)* command.

Ex. 1 : Creating a symbolic link for a single file : the first argument is the name of the existing file (or directory), and the second argument the name of the shortcut to create, or the directory in which to create a shortcut with the same name.

```
[stage11@nz ~]$ ln -s allsequences/ecto_seq_zAb3.fas process_ecto/
[stage11@nz ~]$ ls -l process_ecto/
lrwxrwxrwx 1 stage08 stage 30 May 05 08:44 ecto_seq_zAb3.fas ->
allsequences/ecto_seq_zAb3.fas
```

Using shortcuts : symbolic links

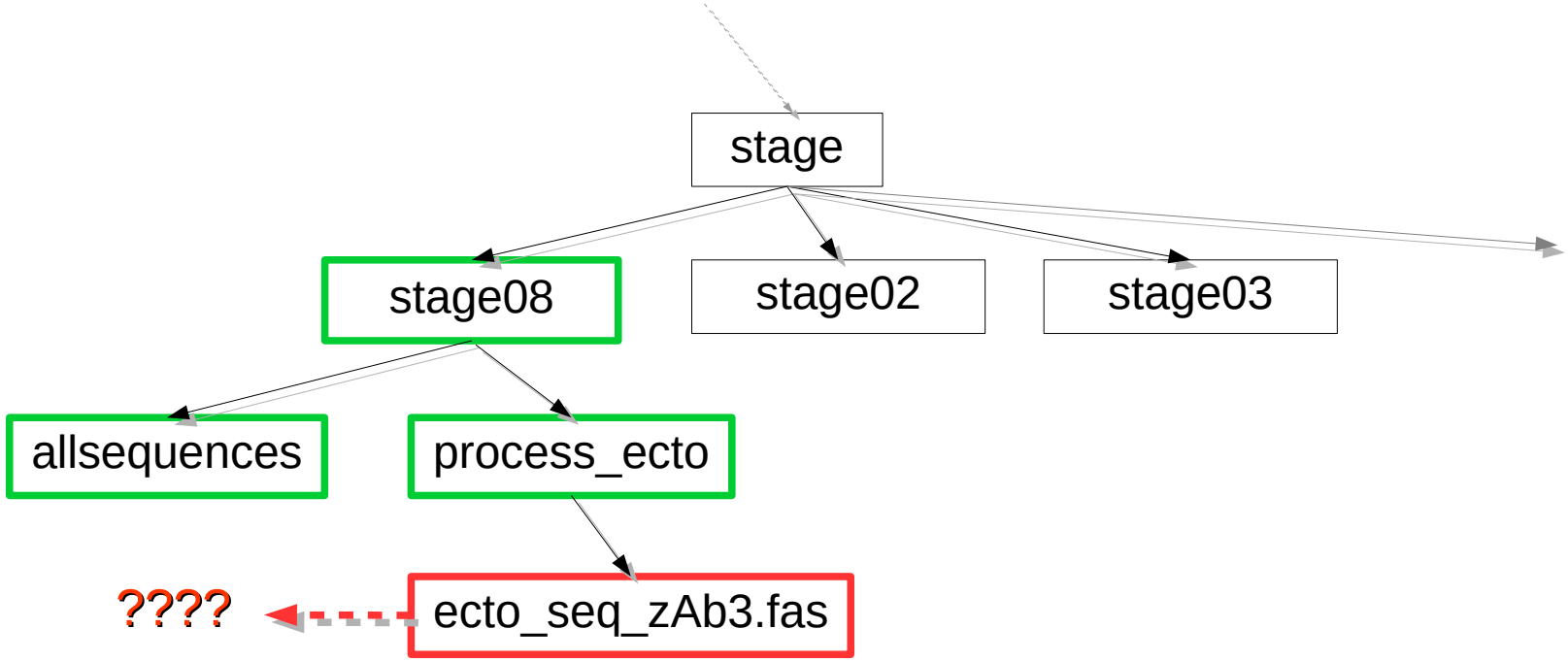
The symbolic link creates a new entry in the file system “pointing” to an already existing entry.



Using shortcuts : symbolic links

Deleting the original file makes the shortcut unusable!

```
[stage11@nz ~]$ rm -f allsequences/ecto_seq_zAb3.fas
[stage11@nz ~]$ cat process_ecto/ecto_seq_zAb3.fas
cat: process_ecto/ecto_seq_zAb3.fas: No such file or directory
```

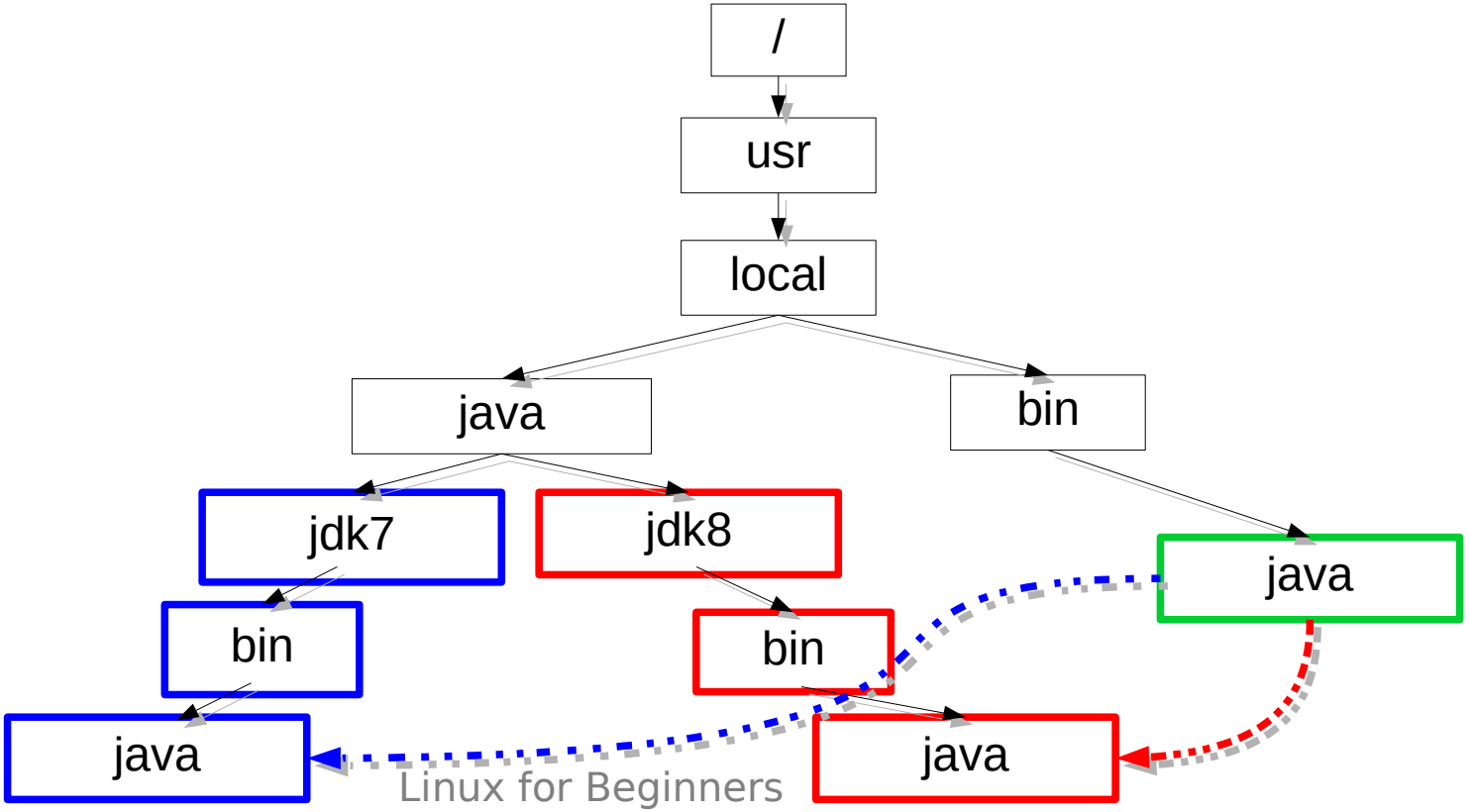


Using shortcuts : symbolic links

It is easy to create series of symbolic links using patterns.

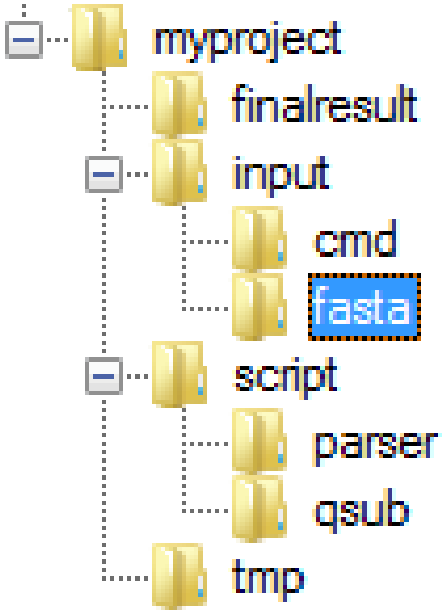
```
[stage11@nz ~]$ ln -s allsequences/ecto_seq*.fas process_ecto/
[stage11@nz ~]$ ln -s allsequences/mouse_seq*.fas process_mouse/
```

Symbolic links can also be used to transparently manage software package updates : a command can “point” to a specific version of a tool. When the tool is updated, a new version of the command can be installed alongside the previous one, and the link to the command is adjusted to point to the latest version.



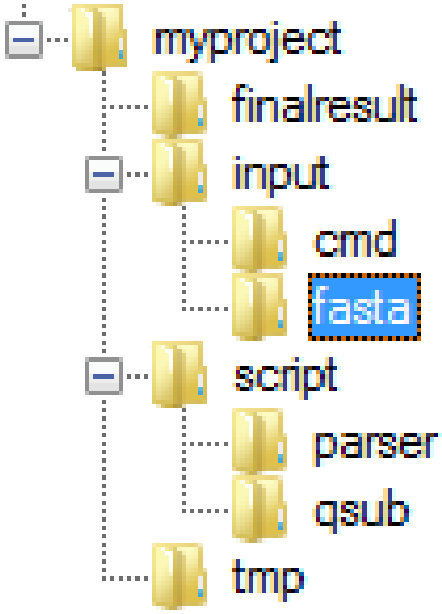
Exercises

- Create a symbolic link in your *home directory* pointing to the **`script`** directory.
- Create a symbolic link in your *home* directory pointing to the **`test-TP.txt`** file located in the in the **`finalresult`** directory.
- Display the contents of the **test-TP.txt** file located in your *home directory*
- Delete the **finalresult/test-TP.txt** file
- Conclusion ?



Exercises

- Create a symbolic link in your *home directory* pointing to the **`script`** directory.
- Create a symbolic link in your *home* directory pointing to the **`test-TP.txt`** file located in the in the **`finalresult`** directory.
- Display the contents of the **test-TP.txt** file located in your *home directory*
- Delete the **finalresult/test-TP.txt** file
- Conclusion ?




```
[stage11@nz ~]$ ln -s myproject/script ~
[stage11@nz ~]$ ln -s myproject/finalresult/test-TP.txt ~
[stage11@nz ~]$ cat test-TP.txt
(...)
[stage11@nz ~]$ rm -f myproject/finalresult/test-TP.txt
[stage11@nz ~]$ cat test-TP.txt
test-TP.txt : No such file or directory
```

Copying Files to/from a Remote Machine

The **scp** command is used to copy files to/from another Linux (UNIX) machine. To use it, it's necessary to have an account (user name and password) on the remote machine. **scp** is used like **cp** sbut one of its arguments (source or destination) includes information about the remote machine as follows:

username@hostname:

- If needed, the password on the remote machine will be requested
- Transfers are encrypted: **scp** uses the SSH protocol.

Ex1. : Copying a local file to a remote machine : information about the remote machine is found in the **destination argument** of the command.

```
[stage08@nz ~]$ scp Linux-Initiation.tar.gz stage08@sbr2:cours/
```

Ex2. : Copying a directory structure from a remote machine : the information about the remote machine is in the **source argument** of the command.

```
[stage08@nz ~]$ scp -r stage08@sbr2:cours/  .
```

Copying files from a Web server through a URL

The **wget** command is used to fetch a local copy of a (set of) file(s) located on a Web server, and whose URL is known.

Ex1. : Fetching an ENA entry in FASTA format from its accession number.

```
[stage08@nz ~]$ wget "http://www.ebi.ac.uk/ena/data/view/BN000065&display=fasta
"
--2017-04-14 12:42:09--  http://www.ebi.ac.uk/ena/data/view/BN000065&display=fasta
Resolving www.ebi.ac.uk... 193.62.193.80
Connecting to www.ebi.ac.uk|193.62.193.80|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: “BN000065&display=fasta”

    [ <=>                                                ] 320,575     --.-K/s   in 0.1s

2017-04-14 12:42:09 (2.07 MB/s) - “BN000065&display=fasta” saved [320575]
```

Ex2. : **Recursively** (**-r** option) fetching contents from an HTML page **(handle with care !)**

```
[stage08@nz ~]$ wget -r "http://abims.sb-roscoff.fr/training"
```

The remote directory structure is recreated in the directory where the **wget** command is run.

Exercises

Supposing your are in your project directory, copy, **using a single command**, the **`test-TP.txt`** file located in the **`/tmp`** directory of the **`sbr2`** machine in the **`myproject/tmp`** directory.

Exercises

Supposing your are in your project directory, copy, **using a single command**, the **`test-TP.txt`** file located in the **`/tmp`** directory of the **`sbr2`** machine in the **`myproject/tmp`** directory.

Solution

```
$ scp sbr2:/tmp/test-TP.txt myproject/tmp
```

Linux for Beginners | Outline

Users & Groups : the Concept

In a Linux (UNIX) system, each resource (file, directory, running program…) is owned by a user having a valid account on the machine. This user is the **owner** of the resource.
Every user belongs to at least one **group**. There is no limit to the number of groups a user can belong to.
At any moment, each user has only one **active** group. This is the group that will be take into account by the system when the user accesses a resource.
Users and groups have numerical identifiers, called **`uid`** (user id) and **`gid`** (group id).

The **`id`** command displays the information about the identity of the user running the command.

```
[stage08@nz ~]$ id
uid=7069(stage08) gid=1013(stage) groups=1013(stage)
```

The **`ls -l`** command displays information about the ownership (which user/group) of files and directories.

```
[stage08@nz ~]$ ls -l acteur.csv
-rw-r--r-- 1 stage08 stage 84 May 13 21:19 acteur.csv
```

Users & Groups : the Concept

The operations a user can perform on a resource are defined by the rights she has both as user and as member of groups to which she belongs.

➡ Every user is limited to what she has access to on the system: file read/write access, program execution, allocated disk or memory space.

➡ A single user has no access limitations : the system administrator or **`root`**

File system Related Access Rights

The **ls -l** command displays information on file/directory ownership and access rights. Access rights are grouped in triplets made of the **r,w,x** or – characters.

```
[stage08@nz ~]$ ls -l acteur.csv
-rw-r--r-- 1 stage08 stage 84 May 13 21:19 acteur.csv
```

```
[stage08@nz ~]$ ls -l acteur.csv
-rw-r--r-- 1 stage08 stage 84 May 13 21:19 acteur.csv
```

Owner access rights (**u : user**)

```
[stage08@nz ~]$ ls -l acteur.csv
-rw-r--r-- 1 stage08 stage 84 May 13 21:19 acteur.csv
```

Group access rights (**g : group**)

```
[stage08@nz ~]$ ls -l acteur.csv
-rw-r--r-- 1 stage08 stage 84 May 13 21:19 acteur.csv
```

Access rights for users not members of the file's group (**o : others**)

File Access Right Management

Position in the triplet	Character	Matching right	Character	Matching right
1	r	Read access allowed	-	No read access
2	w	Write access allowed	-	No write access
3	x	Execution allowed	-	Execution forbidden

Example `-rw-r--r--`

Owner `rw-` Read and write access allowed, execution forbidden

Group `r--` Read access, no writing and execution

Others `r--` Same as for the group

File system Related Access Rights

The **chmod** command is used to modify access rights to files and directories. Its first argument defines the access right modifications to apply. Its second argument defines the file(s) or directory(ies) on which to apply the modifications.

Ex. 1 : Make a file “private” a.k.a remove (using the minus sign -) all rights (letters r, w and x) to the group (letter g) and to others (letter o).

```
[stage08@nz ~]$ chmod  go-rwx acteur.csv
-rw------- 1 stage08 stage 84 May 13 21:19 acteur.csv
```

Ex. 2 : Add (+ sign) write access (letter w) for the group (letter g) to a file.

```
[stage08@nz ~]$ chmod  g+w acteur.csv
-rw-rw-r-- 1 stage08 stage 84 May 13 21:19 acteur.csv
```

Ex. 4 : Prevent file modification by removing (- sign) write (letter w) access to everyone (including the owner)

```
[stage08@nz ~]$ chmod  -w acteur.csv
-r--r----- 1 stage08 stage 84 May 13 21:19 acteur.csv
```

Ex. 4 : Add (+ sign) execution rights (letter x) to a file.

```
[stage08@nz ~]$ chmod  +x monprogramme
-rwxr-xr-x 1 stage08 stage 84 May 13 21:19 monprogramme
```

File system Related Access Rights - Directories

In the output of **`ls -l`**, directories are flagged with a **`d`** letter before the string defining the access rights.

```
[stage08@nz ~]$ ls  -l
drwxr-xr-x 4 stage08 stage    4096 May 01 11:41 Linux-Initiation
```

Position in the triplet	Char.	Matching right	Char.	Matching right
1	**`r`**	**Reading *the list of* files is allowed.**	**`-`**	**Reading *the list of files* is forbidden**
2	**`w`**	**Creating, renaming and removing files is allowed.**	**`-`**	**File creation, renaming or removal are forbidden**
3	**`x`**	**Going (with cd) in the directory is allowed.**	**`-`**	**Cd'ing in the directory is forbidden**

The **`chmod`** command has an **`-R`** (recursive) option recursively applying the access rights to all files and subdirectories of its destination argument.

File system Related Access Rights – Group Changeing

The **chgrp**, command allows to define a new group for a file or directory. **The user running the command must be member of the group.**

```
[stage08@nz ~]$ chgrp  autregroupe acteur.csv
[stage08@nz ~]$ ls -l  acteur.csv
-r--r----- 1 stage08 autregroupe 84 May 13 21:19 acteur.csv
```

There is a command to change file ownership **(chown)** but its use is restricted to the system administrator...

Exercises

Authorize all members of the **`stage`** group to write in the **`Linux-Initiation`** directory and its subdirectories. Check with your neighbor that (s)he can deposit and remove files there. But forbid any modification to the **`acteur.csv`** file.

Exercises

Authorize all members of the **`stage`** group to write in the **`Linux-Initiation`** directory and its subdirectories. Check with your neighbor that (s)he can deposit and remove files there. But forbid any modification to the **`acteur.csv`** file.

Solution

```
$ chmod -R g+rwx Linux-Initiation
$ chmod g-rwx Linux-Initiation/acteur.csv
```

Linux for Beginners | Outline

Some Definitions

A process is a currently running program. Each time a user issues a command (runs a program), the operating system loads it into memory and starts its execution.

In order to run smoothly, a process needs **memory** and **processor time** (CPU). It is the duty of the operating system to proceed to the optimal allocation of these resources among all the processes running “simultaneously”. The **load** of a machine reflects the activity of all the active processes at any given moment.

As for files, a process has an **owner** (user) and a **group** ; and associated **rights or permissions.**

The user has the ability -to some extent- to control process’ execution: she can **stop** them “by force”, **interrupt them** to **resume** them later on, or modify their **priority**.

Running / Stopping / Interrupting processes

Each time a command is issued in the current session, a process is created and executed. Only when the excution ends is it possible to issue new commands.

```
[stage08@nz ~]$ gedit  acteur.csv
[stage08@nz ~]$
```

New commands can be run only after exiting gedit

A process running in the current session can be (brutally) stopped by typing the **Ctrl-C** key combination.

```
[stage08@nz ~]$ gedit  acteur.csv
^C
[stage08@nz ~]$
```

Ctrl-C "kills" the process. The user regains control.

A process killed by **Ctrl-C** frees all the resources (memory, open files) in its possession.

Running / Stopping / Interrupting processes

A process running in the current session can be **interrupted** with the key **Ctrl-Z** combination.

```
[stage08@nz ~]$ gedit  acteur.csv
^Z
[1]+  Stopped                 gedit acteur.csv
[stage08@nz ~]$
```

The user regains control after typing **Ctrl-Z**. The process has been **interrupted**.

An interrupted process is "frozen", and keeps all resources it was allocated (except for processor time). It is assigned a ***job identifier*** *(not to be confused with the process identifier, cf. following slides).*

The **jobs** command lists all interrupted processes of the current session.

```
[stage08@nz ~]$ gedit  insulin.fas
^Z
[2]+  Stopped                 gedit insulin.fas
[stage08@nz ~]$ jobs
[1]-  Stopped                 gedit acteur.csv
[2]+  Stopped                 gedit insulin.fas
```

Resuming an Interrupted Process

The **fg** *(foreground)* command resumes the execution of an interrupted process. Without argument, the most recently interrupted process wil be resumed. To resume a specific process, it is possible to use the % sign followed by the job identifier.

```
[stage08@nz ~]$ jobs
[1]-  Stopped                 gedit acteur.csv
[2]+  Stopped                 gedit insulin.fas
[stage08@nz ~]$ fg %1
gedit acteur.csv
```

The **bg** *(background)* command also resumes an interrupted process but **immediately gives back control to the user in in the current session.** Execution of the process continues in the **background**

```
[stage08@nz ~]$ jobs
[1]-  Stopped                 gedit acteur.csv
[2]+  Stopped                 gedit insulin.fas
[stage08@nz ~]$ bg %2
[2]+ gedit insulin.fas &
[stage08@nz ~]$
```

Running a process in background mode

A process can be run directly in background mode by adding an ampersand (**&**) to the command line.

```
[stage08@nz ~]$ gedit acteur.csv &
[3] 26357
[stage08@nz ~]$
```

Both a **job identifier** and a **process identifier (PID)** are displayed .

Displaying process information : ps

The **ps** (process status) command is used to display more or less detailed information about processes.

Ex. 1 : Listing the processes in the current session.

```
[stage08@nz ~]$ ps
  PID TTY          TIME CMD
16175 pts/14   00:00:00 bash
20693 pts/14   00:00:00 dbus-launch
26357 pts/14   00:00:00 gedit
27505 pts/14   00:00:00 ps
```

The main information are the PID (process identifier) and the name of the command (CMD).

Ex. 2 : Getting the **detailed** list of the processes in the current session with the **-f** *(full)* option.

```
[stage08@nz ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
stage08  16175 16174  0 17:21 pts/14   00:00:00 -bash
stage08  20693     1  0 17:37 pts/14   00:00:00 dbus-launch --autolaunch 9b7328b
stage08  26357 16175  0 17:58 pts/14   00:00:00 gedit insulin.fas
stage08  28638 16175  4 18:06 pts/14   00:00:00 ps -f
```

Additionnal information is shown about the user (UID), the PID of the parent process (PPID), the start time (STIME), the execution time (TIME) and the complete command line (CMD).

Displaying process information : ps

Ex. 3 : Listing the processes of a specific user with the **-u** *(user)* option.

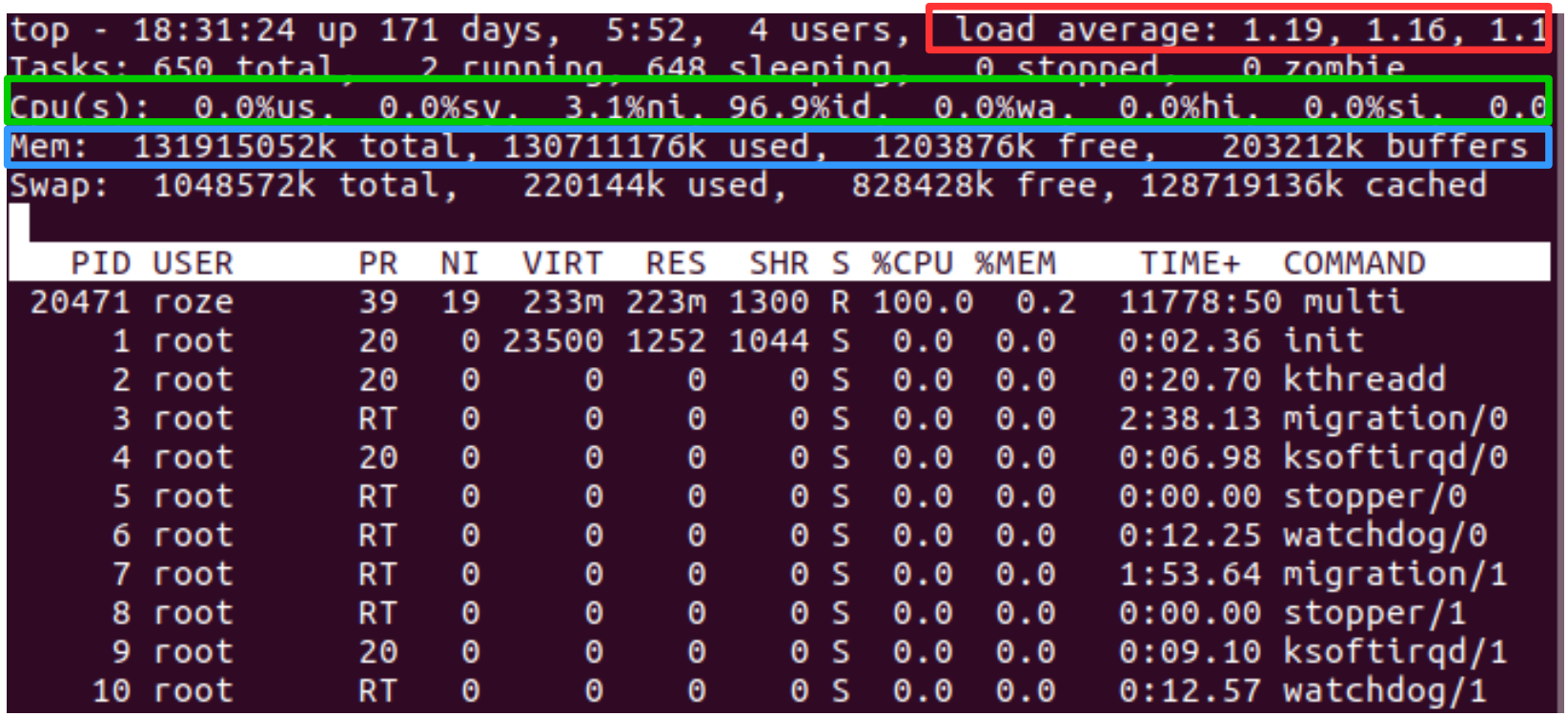
```
[stage08@nz ~]$ ps -fu stage08
UID        PID  PPID  C STIME TTY          TIME CMD
stage08  16174 16172  0 17:21 ?        00:00:00 sshd: stage08@pts/14
stage08  16175 16174  0 17:21 pts/14   00:00:00 -bash
(…)
stage08  20696     1  0 17:37 ?        00:00:00 /usr/libexec/gconfd-2
stage08  26357 16175  0 17:58 pts/14   00:00:00 gedit insulin.fas
stage08  32327 16175  0 18:20 pts/14   00:00:00 ps -fu stage08
```

Ex. 4 : Listing all the processes currently present on the machine with the **-elf** *(extended, long, full)* options.

```
[stage08@nz ~]$ ps -elf
0 S uguyet   29185 29184  0  80   0 - 28353 n_tty_ Apr19 pts/35   00:00:00 /bin/bash
0 S lgueguen 30113 10845  0  80   0 - 26364 n_tty_ May12 pts/32   00:00:00 less macros.xml
4 S root     30980  5864  0  80   0 - 28926 unix_s May12 ?        00:00:00 sshd: nhenry
[priv]
5 S nhenry   31153 30980  0  80   0 - 28961 poll_s May12 ?        00:00:01 sshd: nhenry@notty
0 S nhenry   31154 31153  0  80   0 - 14977 poll_s May12 ?        00:00:01
/usr/libexec/openssh/sft
0 S pmandon  31370  2698  0  80   0 - 27641 n_tty_ May12 pts/64   00:00:00 /bin/bash
0 S lberdjeb 31598     1  0  80   0 - 26526 wait   12:22 ?        00:00:00 /bin/sh
/opt/sge/qlogin.
```

Interactive Process Visualisation with `top`

The `top` command displays and continually refreshes the list of processes running on a machine. By default, the list is sorted according to the process load (%CPU column). The command also summarizes the overall state of the system (uptime, global load, memory availability) above the process list.



Global system load over the last 1, 5 and 15 minutes

Snapshot of CPU usage

Snapshot of available and allocated memory

Process termination : `kill`

The **`kill`** command sends a signal to the process whose process identifier (PID) is given as argument. By using specific options (**`-HUP`**, **`-TERM`**, **`-KILL`**) the process is more or less “gently” notified.

Ex. 1 : Brutally stopping a process using **`kill`** with the **`-KILL`** option

```
[stage08@nz ~]$ gedit acteur.csv &
[1] 27908
[stage08@nz ~]$ kill -KILL 27908
[stage08@nz ~]$
```

The **`kill`** command can only be used on one’s own processes.

Processes and Inheritance

Every process is spawned from a parent process. For instance, open a session yields a new process which will be the parent process of all of the commands typed in on the command line.

The process with PID 1 is the process that, when the machine was started, gave rise to the system's process tree.

Halting a process results in halting all of its child processes (to keep in mind before using `kill`).

When a process ends, its parent process is notified. Until the parent process handles its child's termination, the latter stays in a *zombie* state. Zombie processes do not take up any system resources, except if they proliferate in an uncontrollable way. Eliminating a zombie is done by "killing" its parent process. It is then attached to the process with PID 1 who takes care of eliminating zombies.

Exercises

Open a connection. Run the **`gedit`** command in background mode. Then close the connection. What happens ?

Open a connection. Run the **`gedit`** command in background mode. Then close the connection. What happens ?

Solution

The **`gedit`** process is killed when the connection is closed : it was the child process of the connection process.

Detaching Processes from a Session

It is frequently necessary to execute programs whose running time will exceed the duration of a session. Thus, automatic killing of these processes must be avoided when the session is closed. The **nohup** (*no hang-up)* command allows to detach a process from a session.

```
[stage08@nz ~]$ nohup mon_long_programme_de_bioinformatique.pl
nohup: ignoring input and appending output to `nohup.out'
```

Any information displayed by the program will be added to a file named **nohup.out** created in the directory where the command was executed.

For programs already running (in background mode) and whose PID is known, the **disown** command allows to detach it from the session.

```
[stage08@nz ~]$ mon_long_programme_de_bioinformatique.pl &
[1] 29087
[stage08@nz ~]$ disown 29087
```

Bonus : Customizing Your Environment

The Session's Environment - SHELL

Remember : each command is executed un a **current directory** with the credentials of the **current user.**

When starting a session (local or remote), the process handling what the user types is called the **command interpreter** or **`shell`.** Its purpose is to wait for the user to hit the Enter key at the end of a command line a to try to make sense of it a.k.a run the commands whose name(s) were typed in.

The **`shell`** is highly configurable.

Environment Variables - PATH

When the user types a command name, the shell looks for a matching (binary) file in a well defined list of directories. This list is stored in an

environment variable

This specific variable is called **PATH**, and its value can be displayed with **echo** :

```
[stage08@nz ~]$ echo $PATH
/opt/sge/bin/lx24-amd64:/opt/python/bin:/usr/local/java/bin:/usr/lib64/qt-3.3/bin:/usr/
kerberos/sbin:/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/
sbin:/sbin:/opt/dell/srvadmin/bin:/usr/local/public/bin:/usr/local/genome2/bin:/usr/
local/genome/bin:/usr/local/adm/bin:/usr/local/admin/script:/usr/local/adm/script:/usr/
local/genome/script:/usr/local/genome2/h::/usr/local/genome2/seqclean:/usr/local/
genome2/seqclean/bin:/usr/local/genome2/tgicl_linux:/usr/local/genome2/tgicl_linux/
bin:/usr/local/genome/emboss/bin:/usr/local/genome/phylip/bin:/usr/local/genome/
mgadist:/usr/local/genome/MUMmer:/usr/local/genome/TMHMM/bin:/usr/local/genome/hmmer/
bin:/usr/local/genome/fasta/bin:/usr/local/genome/mcl64/mcl-05-321/bin:/usr/local/
genome/WoLFPSORT_package_v0.2/bin:/usr/local/genome2/abyss/bin:/usr/local/cristallo/
bin:/home/fr2424/stage/stage08/bin:/opt/openmpi/bin:/opt/6.x/matlab/r2013b/bin:/opt/
6.x/matlab/r2013b/toolbox/abims/ffca:/opt/6.x/matlab/r2013b/toolbox/abims/mexcdf:/usr/
local/cristallo/shelx97:/home/fr2424/stage/stage08/bin
```

Command Lookup Customization

To find out in which of the directories included in the **PATH** a command is actually found, the **which** command can be used :

```
[stage08@nz ~]$ which grep
/bin/grep
[stage08@nz ~]$ which java
/usr/local/java/bin/java
```

To run a command not located in one of the **PATH** directories, its (absolute or relative) path must be given :

```
[stage08@nz ~]$ ls -l myproject/script
-rwxr-xr-x 1 stage08 stage 804 May 4 20:58 myproject/script/supercalcul.sh
[stage08@nz ~]$ supercalcul.sh
-bash: supercalcul.sh: command not found
[stage08@nz ~]$ myproject/script/supercalcul.sh
```

New directories can be added to the **PATH** to allow executing new commands without having to specify their location.

```
[stage08@nz ~]$ export PATH=~/myproject/script:${PATH}
[stage08@nz ~]$ supercalcul.sh
```

export Propagates the new value of **PATH** to all subsequent processes in the same session

PATH= Means that a new value will be assigned to **PATH**

~/myproject/script The new directory that will be added (at the start) of **PATH**

:${PATH} The previous value of **PATH** is appended to the new directory.

Using Aliases

It is often handy to avoid repeatedly typing commands with the same options & arguments to define an ***alias*** for them. This can be done with the **`alias`** command.

Ex. 1 : Redefining **`grep`** to display matches in color.

```
[stage08@nz ~]$ alias grep='grep --color'
[stage08@nz ~]$ grep Homo insulin.fas
>gi|163659904|ref|NM_000618.3| Homo sapiens insulin-like growth factor 1 (somatomedin
C) (IGF1), transcript variant 4, mRNA
>gi|163659900|ref|NM_001111284.1| Homo sapiens insulin-like growth factor 1
(somatomedin C) (IGF1), transcript variant 2, mRNA
```

Ex. 2 : Creating an alias to count “human” sequences in a multi-FASTA file.

```
[stage08@nz ~]$ alias hsc='grep -c -i human'
[stage08@nz ~]$ hsc insulin_vs_nt.blast
373
```

Making the Customizations Persistent

Environment variable changes as well as alias definitions are only valid for the duration of the session they were made.

There is however a **`~/.bashrc`** (text) file whose contents is read whenever a new session is opened. The contents of this file can be any shell command, including **`alias`** definitions and **`PATH`** modifications.

After modifying this file in the current session, the **`source`** command has to be issued for the new version of the file contents to be taken into account in this session :

```
[stage08@nz ~]$ source ~/.bashrc
```

Displaying Environment Parameters

The **env** command displays all the known environment variables with their values.

```
[stage08@nz ~]$ env
(…)
LANG=fr_FR.UTF-8
GDM_LANG=fr_FR
MANAGERPID=25591
DISPLAY=:0
INVOCATION_ID=964f88f33972481e93c31234b2e4483f
COMPIZ_CONFIG_PROFILE=ubuntu
(…)
```

The **alias** command displays the list of all active aliases.

```
[stage08@nz ~]$ alias
(…)
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
(…)
```

Exercises

Add a customized version of **`grep`** to your environment, an reload your configuration in the current session.

After handing over your completed evaluation sheet, you will receive a *cheat sheet* with the essential Linux commands.