

Commandes Unix: pour les débutants

Denis Puthier
Claire Toffano-Nioche
Julien Seiler
Gildas Le Corguillé

TAGC/ Inserm U1090, denis.puthier@univ-amu.fr
I2BC, claire.toffano-nioche@u-psud.fr
IGBMC, seilerj@igbmc.fr
Sorbonne Université/CNRS, lecorguille@sb-roscoff.fr

Et tout le staff !!

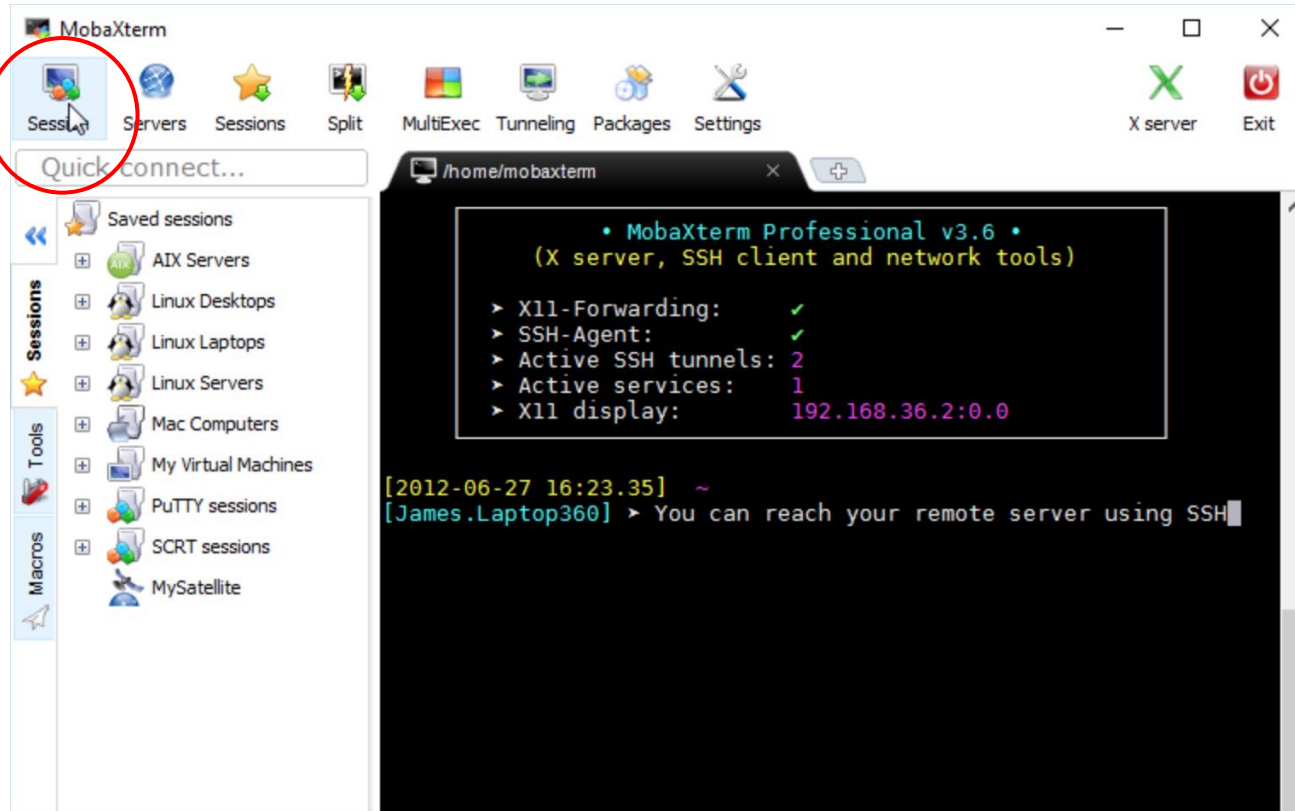
ACCÈS RAPIDE: <https://bit.ly/3ERODCS>

École de bioinformatique AVIESAN-IFB 2021

La connexion ssh vers le cluster

- **ssh (secure shell)**
 - **Protocole sécurisé**
 - les informations passant sur le réseau sont protégées (chiffrées)
 - **Plusieurs outils pour la connexion**
 - Via une application réseau comme **MobaXterm**
 - **Windows**
 - Permet l'accès à un terminal
 - Via un **terminal**
 - **Linux ou MacOSX**
 - **Windows 10** (Bash on Ubuntu on Windows)
 - Via une application web
 - JupyterHub

Se connecter depuis Windows avec MobaXterm (1)



Se connecter depuis Windows avec MobaXterm (2)

1. Session

ssh

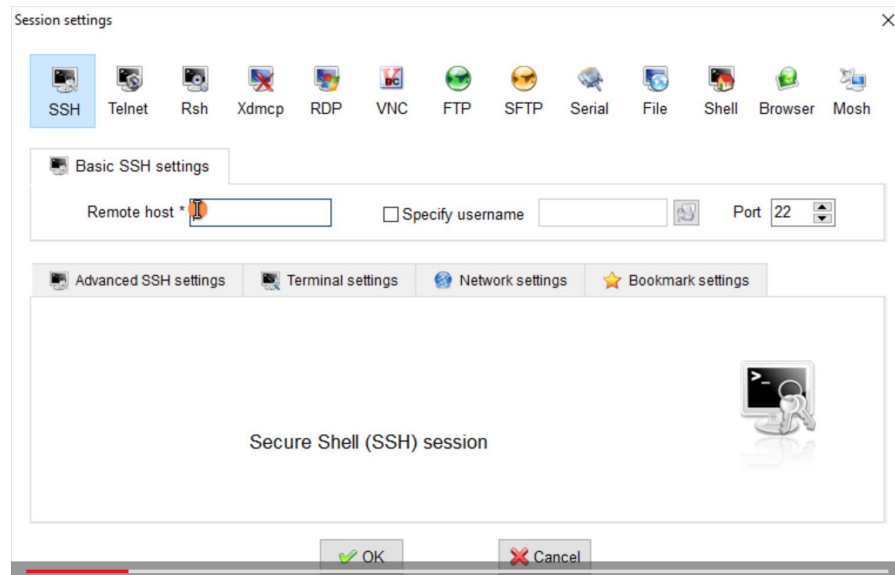
2. Remote host

core.cluster.france-bioinformatique.fr

3. Username

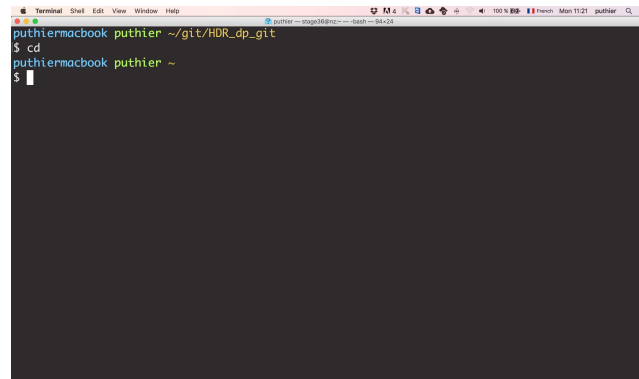
Indiquez votre login

4. Pressez OK



Se connecter depuis Mac OSX ou Linux

- MacOSX
 - **Finder > Applications > Utilities > Terminal**
 - Tapez la commande ci-dessous
- Linux
 - La localisation du terminal varie selon les bureaux (Faites-vous aider si vous ne trouvez pas).
 - Utilisez la commande ci-dessous



```
Terminal Shell Edit View Window Help
puthier-->stage30@pc-->ssh-->52c2
puthiermacbook puthier ~/git/HDR_dp_git
$ cd
puthiermacbook puthier ~
$
```

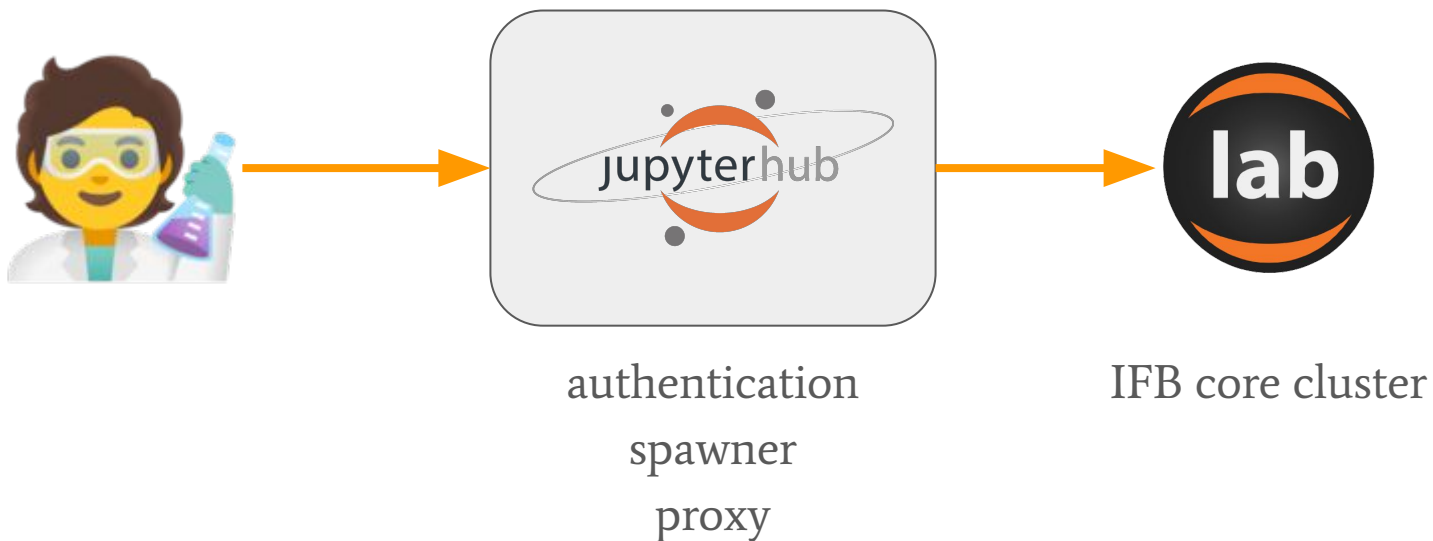
```
$ # Connection en ssh au serveur IFB
$ # Remplacez [login] par votre compte utilisateur
$ # Tapez votre mot de passe
$ ssh -Y [login]@core.cluster.france-bioinformatique.fr
```

Remarques:

- Le caractère # indique un commentaire, qui sera ignoré par le shell.
- Le \$ en rose représente l'invite de commande, qui varie selon les configurations.
- La commande à taper dans votre terminal commence juste après l'invite de commande: **ssh ...**
- L'argument -Y permet d'ouvrir des fenêtres graphiques à distance (e.g. éditeurs)

What is JupyterHub?

JupyterHub is a web application that let you spawn JupyterLab servers on a cluster or cloud infrastructure.



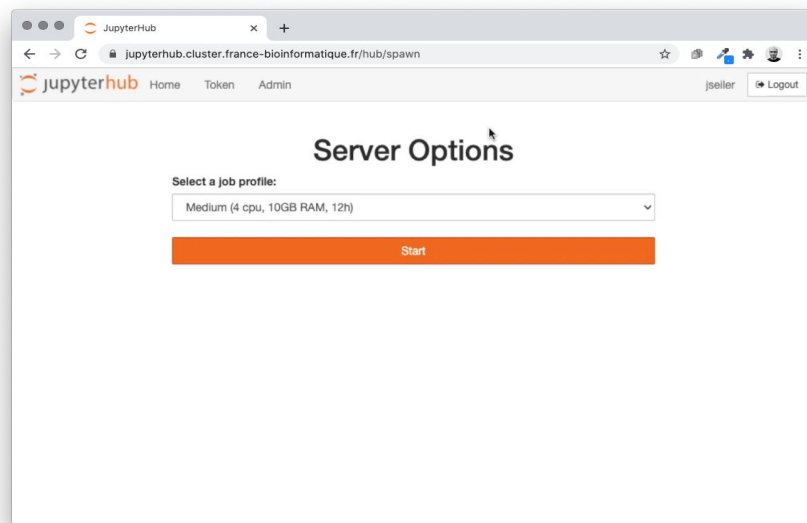
JupyterHub @ IFB

<https://jupyterhub.cluster.france-bioinformatique.fr>

Use your **IFB cluster account**
to log in

Spawn JupyterLab server
in **SLURM jobs**

Work on the **same storage**
as the cluster (ssh)



Se connecter depuis JupyterLab

File Edit View Run Kernel Diagram Tabs Settings Help

Filter files by name

Name Last Modified

- bin 3 years ago
- boot a year ago
- cvmfs 8 days ago
- dev 2 months ago
- etc 4 months ago
- home 4 years ago
- lib 3 years ago
- lib64 3 years ago
- media 4 years ago
- misc 5 months ago
- mnt 4 years ago
- net 5 months ago
- opt 3 years ago
- proc 5 months ago
- run 8 days ago
- sbin 3 years ago
- shared 8 months ago
- srv 4 years ago
- sys 5 months ago
- tmp a minute ago
- usr 3 years ago
- var 3 years ago

Launcher

Notebook

- Python 3.7
- Bash
- Python 3.9
- R 4.0.3
- RStudio

Console

- Python 3.7
- Bash
- Python 3.9
- R 4.0.3

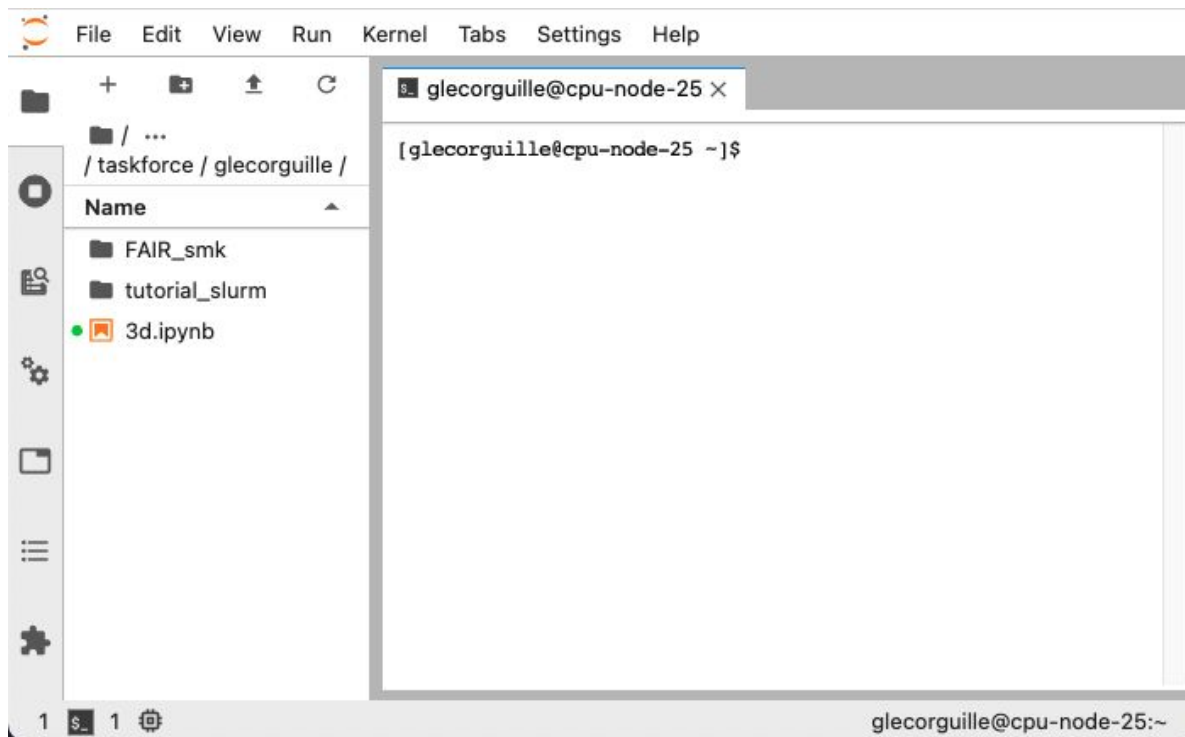
Other

- Terminal
- Diagram
- Text File
- Markdown File
- Python File
- R File
- Show Contextual Help

Simple 0 0

Se connecter depuis JupyterLab

- Prompt (invite de commande)
 - `login@hostname current working directory`



Comment converser avec le terminal ?

```
[stage36@nz ~]$ Bonjour mon nom est denis. Et toi ?  
-bash: Bonjour: command not found  
[stage36@nz ~]$
```

- **Réponse** : lui parler en langage **BASH (Bourne Again Shell)** *
 - Le langage BASH est un des nombreux **dialectes** Shell (bsh, ksh, csh, zsh,...).
 - Tous ces langages Shell sont extrêmement similaires.
 - Ce langage repose notamment sur un ensemble de **commandes**.
 - Ces commandes **modulaires** permettent de réaliser des **tâches**.
 - Ces **tâches** permettent de **piloter un ordinateur à distance**.

* Référence (calembour) au premier langage Shell écrit par Stephen Bourne (bsh) :)

Charger l'environnement logiciel



- Les administrateurs du cluster ont effectué les installations via le gestionnaire de logiciels **Conda**.
 - Permet l'installation facile des programmes et de leurs dépendances.
 - Le dépôt Bioconda met à disposition de nombreux outils bioinformatiques.
- La commande `module` va charger l'environnement Conda dans votre session

```
$ module avail -l      # Afficher la liste des environnements disponibles
```

```
$ # Charger fastqc dans votre environnement
```

```
$ module load fastqc/0.11.8 # La version est facultative
```

```
$ # module unload fastqc/0.11.8 # On peut aussi décharger
```

Prototype(s) d'une commande (1)

- Une **commande** réalise une **tâche** (trier, sélectionner, ouvrir, aligner des reads,...).
- Elle dispose d'un certain nombre d'**arguments** qui peuvent être facultatifs et qui peuvent **modifier** son **mode de fonctionnement**.
 - Les noms des arguments ne sont **pas standardisés**
- Ces arguments peuvent ou non prendre des **valeurs**.
- De manière générale une **instruction** dans le terminal **commence toujours par le nom d'une commande**
- Dans le premier exemple ci-dessous on dira '**moins v**'.

```
$ # Exemple d'argument sans valeur associée
```

```
$ # v pouvant signifier verbose, version (ou autre suivant la commande).
```

```
$ fastqc -v # quelle est la version du logiciel fastqc sur ce serveur ?
```

```
$ # Exemple d'argument avec valeur associée
```

```
$ man -k pdf # rechercher toutes les commandes qui traitent du format pdf
```

Prototype(s) d'une commande (2)

- De manière générale, les arguments peuvent être utilisés sous leurs **formes courtes** ou sous leurs formes **longues** (plus **explicites/lisibles** mais plus longues à taper...).
- Les formes longues sont généralement précédées de deux tirets (dans l'exemple ci dessous on dira '**moins-moins help**)

```
# Demander de l'aide (help) sur fastqc avec l'argument -h
```

```
$ fastqc -h
```

```
# La commande précédente est équivalente mais un peu plus lisible
```

```
$ fastqc --help
```

Trouver de l'aide !



Appeler la police, appeler son collègue, chercher sur internet ou utiliser la commande **man** (**manuel**)

Demo

\$ man ls # obtenir de l'aide sur la commande ls

\$ man man # obtenir de l'aide sur la commande man ...

Raccourcis dans l'aide:

/color : pour chercher le terme 'color'.

n : (**n**ext) pour chercher la prochaine occurrence de 'truc'.

p: (**p**revious) pour chercher l'occurrence précédente de 'truc'.

q : **pour quitter l'aide.**



Zoom sur la commande `ls`

La commande ls et ses arguments

- La commande `ls` peut prendre un certain nombre d'arguments.
- Parmi les arguments principaux:
 - `-l` (**l**ong/**l**ot) donne beaucoup d'informations sur les fichiers.
 - `-a` (**a**ll) montre tous les fichiers y compris ceux qui sont cachés*.
 - `-t` (**t**ime) trie par date de modification.
 - `-h` (**h**uman-readable) affiche les tailles des fichiers en unités lisibles
 - `-r` (**r**everse) inverse l'ordre du tri.
- On peut combiner les arguments :
 - `ls -l --all`
- On peut fusionner les arguments (au format court) :
 - `ls -la`

* Sous Linux les noms des fichiers cachés commencent par un point (e.g `'bashrc'`).

La commande ls et ses arguments

```
$ # Demo
$ cd /shared/bank/homo_sapiens/hg38/fasta # On se déplace dans le dossier
$ ls # On liste les fichiers
$ ls -l # Information détaillée sur les fichiers (taille, date modif,...)
$ # Vue détaillée des fichiers et tailles en Ko,Mo,Go,To...
$ ls -lh
$ # Vue détaillée des fichiers, tailles en Ko,Mo,Go,To..., tri par date
$ ls -tlh
$ # Vue détaillée des fichiers, tailles en Ko,Mo,Go,To..., tri par date
$ # du plus ancien au plus récent
$ ls -rtlh
```

NB:

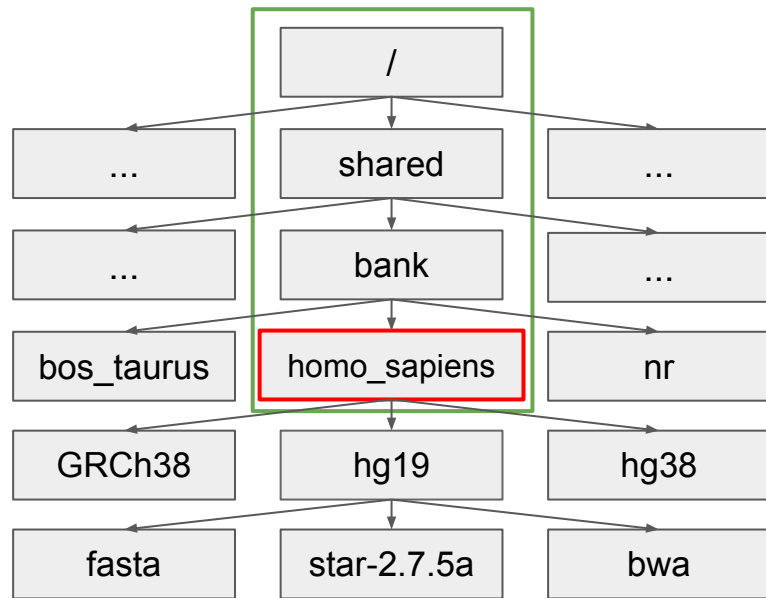
- **ATTENTION** aux **espaces**, nécessaires entre la commande et ses arguments. La commande `ls-l` n'existe pas !
- Le comportement par défaut de `ls` est de trier par ordre alphabétique en tenant compte de la casse (i.e majuscule minuscule).



Arborescence de fichiers

L'arborescence du système de fichier

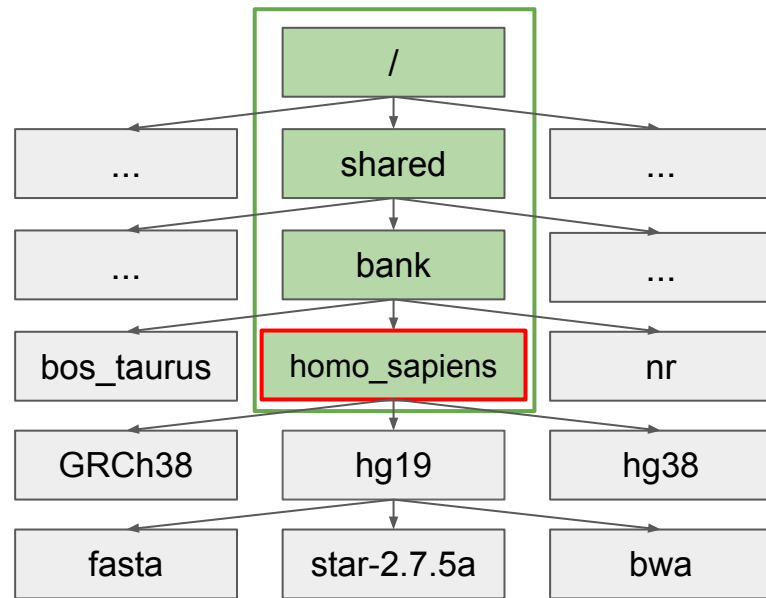
- Le **système de fichier** peut être vu comme un **arbre** dont les **feuilles** sont des **dossiers** et **fichiers**. On peut se **déplacer** dans cet arbre.
- Cet arbre contient **une racine**, le dossier **/**
- Le dossier **/** contient notamment
 - un dossier **shared** *
 - qui lui même contient un dossier **bank**
 - qui lui même contient un dossier **homo_sapiens**
 - ...
- Chemin du dossier
 - **/shared/bank/homo_sapiens**



Faire référence à un dossier ou fichier ?

- 1) En spécifiant **un chemin depuis la racine**.
 - On parle de **chemin absolu**

```
$ cd /shared/bank/homo_sapiens
```



Faire référence à un dossier ou fichier ?

- 2) En spécifiant un chemin depuis le répertoire courant.
 - Le répertoire courant est celui dans lequel l'utilisateur se trouve à un instant t .
 - Le chemin sera relatif au répertoire courant.

Depuis `homo_sapiens` on peut aller dans `hg19` puis `star`

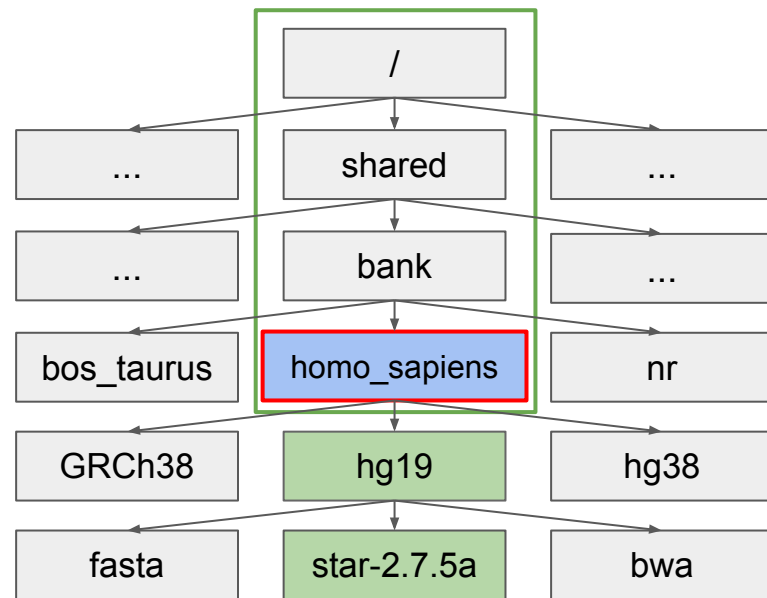
```
$ cd hg19/star-2.7.5a
```

```
# ou cd ../hg19/star-2.7.5a
```

```
# Avec "." pour signifie "le répertoire  
# courant".
```

```
$ cd - # le répertoire précédent
```

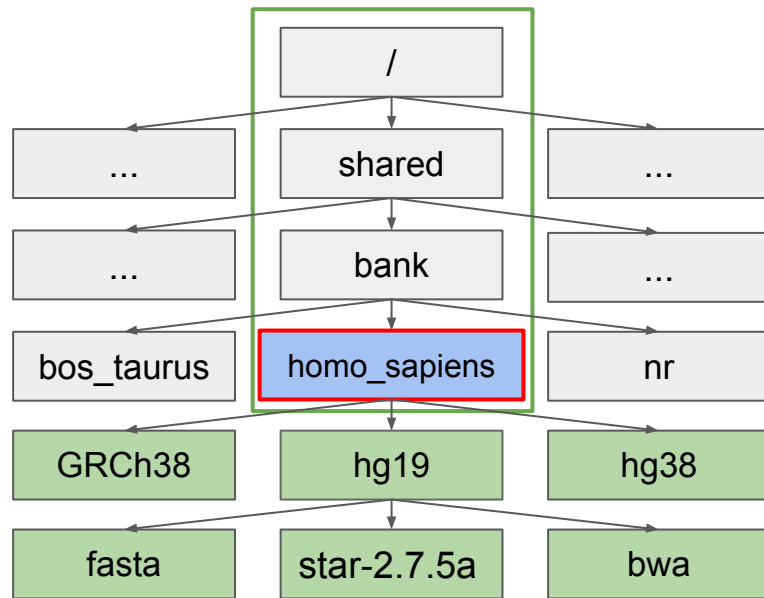
```
$ cd ../hg19/star-2.7.5a
```



Afficher l'arbre des sous-dossiers du dossier courant

```
$ cd /shared/bank/homo_sapiens  
$ tree -d # List directories only
```

```
.  
|-- GRCh38  
| |-- bwa  
| |-- fasta  
| |-- gff3  
| |-- gnomad  
| | |-- 2.1  
| | | |-- exomes  
| | | `-- genomes  
| | `-- 3.0  
| |   `-- genomes  
| |-- gtf  
| |-- star -> star-2.6.1a  
| |-- star-2.6.1a  
| `-- star-2.7.5a  
|-- hg19  
...
```



Autocompletion

- Si vous voulez **briller en société ou en famille** en donnant l'impression de **taper vite**, utilisez l'**auto-complétion**
 - De manière plus générale c'est **essentiel pour taper un chemin sans se tromper**.

```
pedagogix puthier ~  
$ cd
```

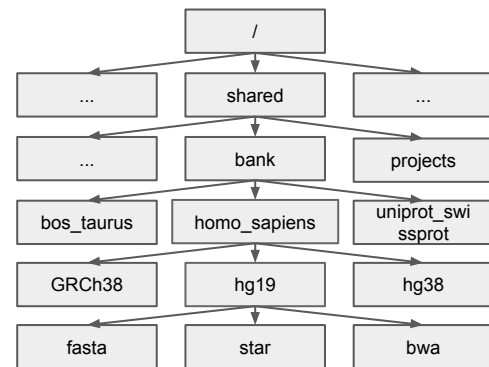
- E.g. Aller dans le répertoire
 - /usr/local/bin

Vous n'avez pas fini d'entendre

<TAB><TAB>

L'arborescence: Demo

On utilise ci-dessous la commande **pwd** (print working directory) et la commande **cd** (change directory). *

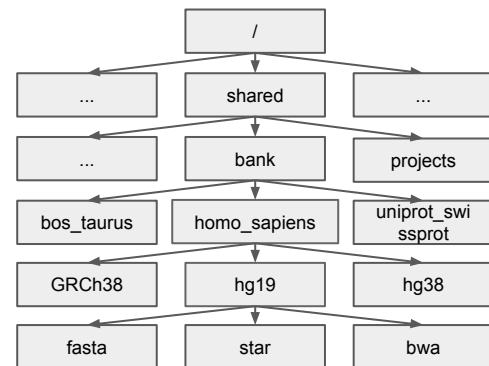


```
$ # On se déplace dans le dossier star
$ cd /shared/bank/homo_sapiens/hg38/star-2.7.5a/
$ pwd                                     # On imprime le chemin vers le répertoire courant
$ cd ..                                 # On remonte d'un répertoire (hg38)
$ pwd                                  # /shared/bank
$ cd ../../..                           # On se déplace dans le dossier bank
$ pwd                                  # /shared/bank
$ cd ../projects                         # On se retrouve 1 cran plus haut puis projects
$ ls                                    # On voit le contenu du dossier "projects"
$ ls .                                  # On voit le contenu du répertoire courant "."
$ cd ../b<TAB>/u<TAB><TAB><TAB>p<TAB><TAB>_<TAB> # Aller dans uniprot_swissprot
$ pwd                                  # On se retrouve dans /shared/bank/uniprot_swissprot
```

* Utilisez la **complétion** pour les noms des fichiers (touche <TAB>) et éventuellement les noms de commandes

L'arborescence quelques astuces

- Si vous êtes l'utilisateur *cnorris*. Le dossier qui stocke vos **documents** est par défaut **/shared/home/cnorris ***
 - i.e 'dossier utilisateur' ou dossier **home**.
 - Il est symbolisé par **~** (tilde).
 - AltGr + 2 (PC), Alt + n + espace (OSX)



```
$ whoami          # votre login
$ cd /            # On est à la racine
$ pwd             # /
$ ls ~            # On liste le contenu du home
$ mkdir ~/tmp     # On crée un répertoire 'tmp' dans le home (make directory)
$ cd ~/tmp        # On se déplace dans le dossier tmp nouvellement créé
$ cd              # Equivalent de cd ~
$ cd /tmp         # n'est pas la même chose que ~/tmp, il est vidé automatiquement
```

Où stocker vos fichiers/dossiers ?

- **Varie** selon les **plateformes**.

- **Se renseigner !**

- A l'IFB

- Votre “**maison**”: ~ (e.g. `/shared/home/cnorris`)

- Ce dossier est **très limité en stockage**
 - Pour les fichiers et dossiers **très peu volumineux**
 - **Pas pour faire des analyses**
 - Utiliser la commande **cd** (sans argument) pour vous rendre dans ce dossier
 - Le chemin vers le dossier home est **symbolisé par ~**
 - Quota de **5Go**

- Votre dossier **projet** (e.g. `/shared/projects/<project>`)

- **C'est le dossier dans lequel vous devez téléverser* vos données**
 - **C'est LE dossier pour lancer vos analyses ...**

* Vous pouvez aussi dire 'uploader' :)

Créer des répertoires

- On utilisera la commande **mkdir** (make directory).

```
$ cd /shared/projects/<project>/ # remplacer <project>
$ mkdir chip-seq                # On crée le dossier
$ ls -l                         # Vérifier la création du dossier
$ cd chip-seq                   # Equivalent de cd ./chip-seq *
$ mkdir bam fastq               # On crée deux dossiers d'un coup
$ ls -l                         # On a bien deux dossiers
$ cd fastq                      # On se déplace dans fastq
$ cd ../../                     # On remonte de deux niveaux
$ mkdir -p rna-seq/output/bam   # On crée un chemin vers un dossier 'bam'
$ ls -R                         # On liste Récursivement les dossiers
$ tree
```

* L'utilisation de ./ est souvent facultative.



Exercices

1. Déplacez-vous dans votre dossier projet
2. Créez un répertoire *annotations* dans votre dossier projet
3. Déplacez-vous dans le dossier *annotations*
4. Dans ce dossier créez un dossier *hg38* et son sous-dossier *gff* (*hg38/gff*)
5. Déplacez-vous dans le répertoire *hg38/gff*
6. Déplacez-vous dans *annotations*.
7. Créez un sous-dossier *mm10* et son sous-dossier *gff* (*mm10/gff*)
8. Déplacez-vous dans *mm10/gff*
9. Depuis ce dossier listez le contenu du dossier *hg38*

Solution de l'exercice

```
$ cd /shared/projects/<project>/
$ mkdir annotations
$ cd annotations
$ pwd
$ # On crée le dossier gff dans hg38
$ mkdir -p hg38/gff
$ cd hg38/gff
$ cd ../../..
$ # On crée le dossier gff dans mm10
$ mkdir -p mm10/gff
$ cd mm10/gff
$ ls ../../../../hg38

# remplacer <project>
# On crée le dossier
# On se déplace
# Où sommes nous ?

# On se déplace
# On se déplace dans annotations

# On voit un dossier gff
```



A propos du serveur:

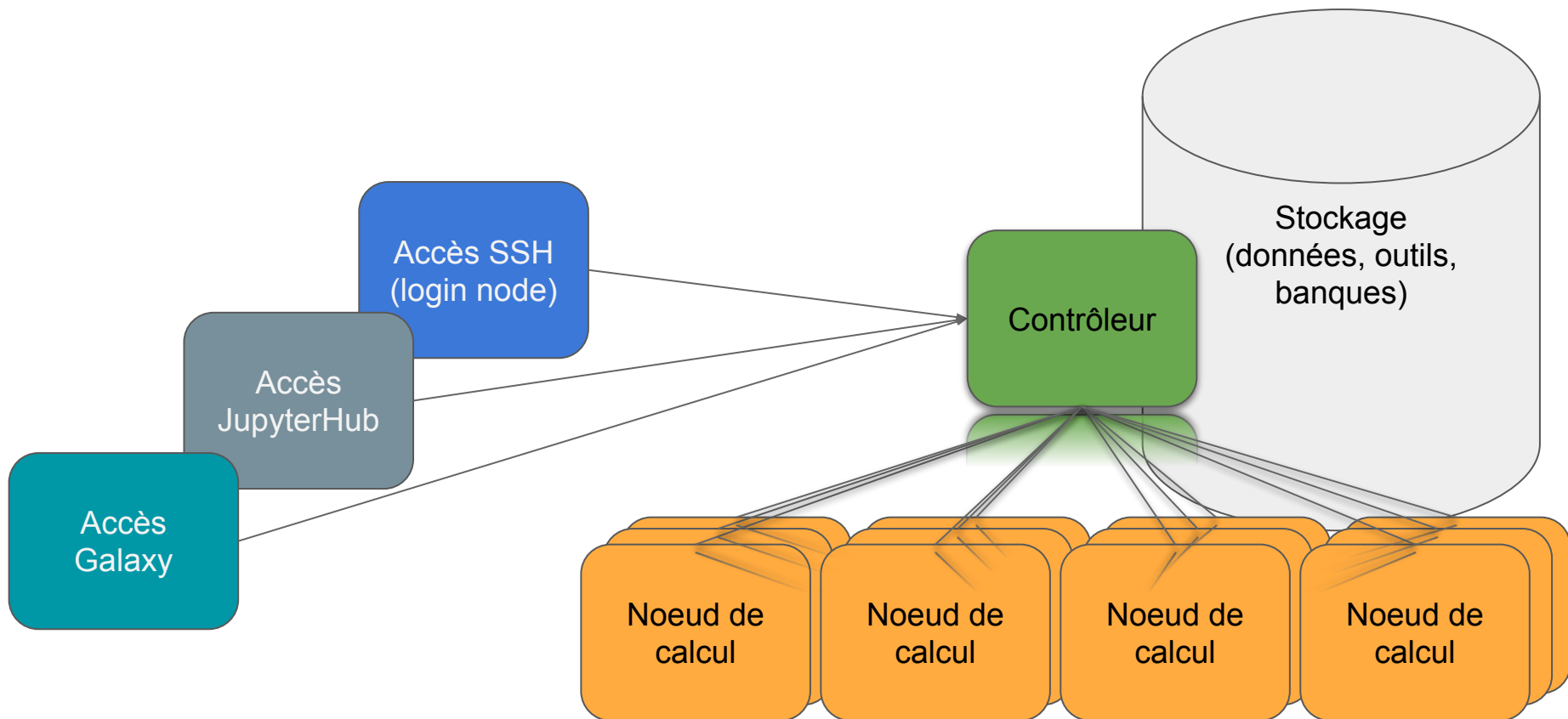
Un controller (*controler*) et des
ouvriers (*workers*)

Le cluster de calcul

- Regroupement de machines
 - Machines: "noeuds/node"
- Gestion transparente pour les utilisateurs
- Connexion depuis le monde entier
 - Nécessité de se connecter à distance
- Accès partagé
 - De nombreux utilisateurs
 - Nécessité d'adopter des règles.
 - Gestion des ressources de calcul
 - Gestion des ressources de stockage



L'architecture d'un cluster de calcul



Reserver des ressources

- Ici **Jupyter** a demandé pour nous la **réserve** des **ressources** (ouvriers) lors de la **connexion**.
 - Quand vous quittez Jupyter les ressources sont libérées
- Un programme disponible sur le controller (**SLURM**) permet de **réserver** des ressources (des processeurs sur des noeuds).
 - Vous verrez dans une autre session comment faire des réservation en ligne de commande avec SLURM.

Info: le serveur IFB-core-cluster

- Hébergé à l'IDRIS (<http://www.idris.fr/>)
- Capacité actuelle 1904 coeurs physiques
 - soit 3808 coeurs “hyperthreadés”
- Plus d'info
 - <https://ifb-elixirfr.gitlab.io/cluster/doc/cluster-desc/>



Manipuler des fichiers

Télécharger et décompresser un fichier

- Pour le téléchargement, on pourra utiliser par exemple la commande **wget**.
- Pour la décompression on utilisera la commande **gunzip** si le fichier a été compressé avec l'algorithme **gzip** (extension **.gz**)

```
$ # On se déplace dans annotations
$ cd /shared/projects/<project>/annotations/hg38 # remplacer <project>
$ mkdir bed; cd bed
$ # On télécharge le fichier
$ wget --no-check-certificate https://zenodo.org/record/5716151/files/hg38_exons.bed.gz
$ ls -lh                                # Le fichier compressé (finit par ".gz"). Il pèse 8,4Mo.
$ file hg38_exons.bed.gz                # Vérification du type de fichier
$ gunzip hg38_exons.bed.gz              # On le décompresse
$ ls -lh                                # Le fichier a perdu l'extension gz. Il pèse 81Mo.
$ file hg38_exons.bed                   # Vérification du type de fichier : il a changé.
```

Le fichier hg38_exons.bed

Contient les coordonnées (début/fin) des exons humains au format [BED](#).

Le format bed (Bed6) (<http://genome.ucsc.edu/FAO/FAOformat.html#format1>) *

Format tabulé (les colonnes sont séparées par des tabulations)

Chromosome Start End Name Score Strand (Others...)

chr1	100719763	100719924	VCAM1 ENST00000370115 protein_coding	.	+
chr1	10072027	10072214	UBE4B ENST00000253251 protein_coding	.	+
chr1	10072027	10072214	UBE4B ENST00000343090 protein_coding	.	+
chr1	10072027	10072214	UBE4B ENST00000377153 protein_coding	.	+
chr1	100720475	100720565	VCAM1 ENST00000370119 protein_coding	.	+
chr1	100720475	100720751	VCAM1 ENST00000294728 protein_coding	.	+
chr1	100720475	100720751	VCAM1 ENST00000347652 protein_coding	.	+
chr1	100720475	100720751	VCAM1 ENST00000370115 protein_coding	.	+

* Positions Start et End sont toujours données par rapport au sens 5'/3' du brin +. Les coordonnées sont 'zero-based, half-open'.

Visualiser le contenu d'un fichier

- On utilisera `less` ou `more` (*) pour parcourir le fichier ligne à ligne (logiciels de type '**pager**').
- On utilisera `head` ou `tail` pour voir les **n premières** ou **n dernières** lignes d'un fichier.
- La commande `cat` permet de renvoyer tout le contenu d'un fichier sur la sortie standard (l'écran).
<ctrl> + c (cancel) pour arrêter.
- Les **raccourcis clavier dans less** sont les mêmes que pour la commande `man`.

Raccourcis dans less :

↑ : se déplacer vers le haut.

↓ : se déplacer vers le bas.

> : Aller à la première ligne.

< : Aller à la dernière ligne.

/**chr** : pour chercher le terme 'chr' (puis touche 'enter').

n : (**n**ext) pour chercher la prochaine occurrence de 'truc'.

p: (**p**revious) pour chercher l'occurrence précédente de 'truc'.

q : pour **q**uitter.

(*) *Less does more or less the same as more, but rather more than less, I like less more than more* (Jacques van Helden)

(*) *Un avantage de less est qu'on peut remonter en arrière; avec more ... c'est mort* (Marc Deloger)

Exercices

1. Utilisez la commande **head** pour regarder les 10 premières lignes du fichier `hg38_exons.bed`
2. Utilisez la commande **tail** pour regarder les 10 dernières lignes du fichier `hg38_exons.bed`
3. Promenez-vous dans le fichier `hg38_exons.bed` en utilisant la commande **less**. Quittez `less`.
4. Renvoyer le contenu du fichier à l'écran avec **cat**.

Truc: si la lecture vous fatigue, utilisez `<ctrl> + c` (cancel) pour arrêter le défilement.

Solutions

1. Utilisez la commande **head** pour regarder les 10 premières lignes du fichier `hg38_exons.bed`
2. Utilisez la commande **tail** pour regarder les 10 dernières lignes du fichier `hg38_exons.bed`
3. Promenez-vous dans le fichier `hg38_exons.bed` en utilisant la commande **less**. Quittez `less`.
4. Renvoyer le contenu du fichier à l'écran avec **cat**.

Truc: si la lecture vous fatigue, utilisez `<ctrl> + c` (cancel) pour arrêter le défilement.

Solution

```
$ head -n 10 hg38_exons.bed
$ tail -n 10 hg38_exons.bed
$ less hg38_exons.bed # q to quit
$ cat hg38_exons.bed
```


Compter les lignes d'un fichier

Utiliser la commande `wc` (**word count**) avec l'argument `-l` (**line**).

```
$ wc -l hg38_exons.bed # 1261870 exons
```

Extraire des colonnes

- Pour extraire des colonnes on utilisera la commande `cut` avec l'argument `-f` (field)
- Les colonnes du fichiers doivent nécessairement être séparées par une **tabulation** (sinon utiliser l'argument `-d` pour 'delimiter')

Truc: si la lecture vous fatigue, utilisez `<ctrl> + c` (cancel) pour arrêter le défilement.

```
$ cut -f 1 hg38_exons.bed          # extraire la colonne 1
$ cut -f 1,2 hg38_exons.bed        # extraire les colonnes 1 et 2
$ cut -f 2,1 hg38_exons.bed        # cut ignore malheureusement
                                   # l'ordre indiqué
$ cut -f 3-5 hg38_exons.bed        # extraire la colonne 3 jusqu'à 5
$ cut -f 3- hg38_exons.bed         # extraire depuis la colonne 3
                                   # jusqu'à la fin de la ligne
```

Trier un fichier

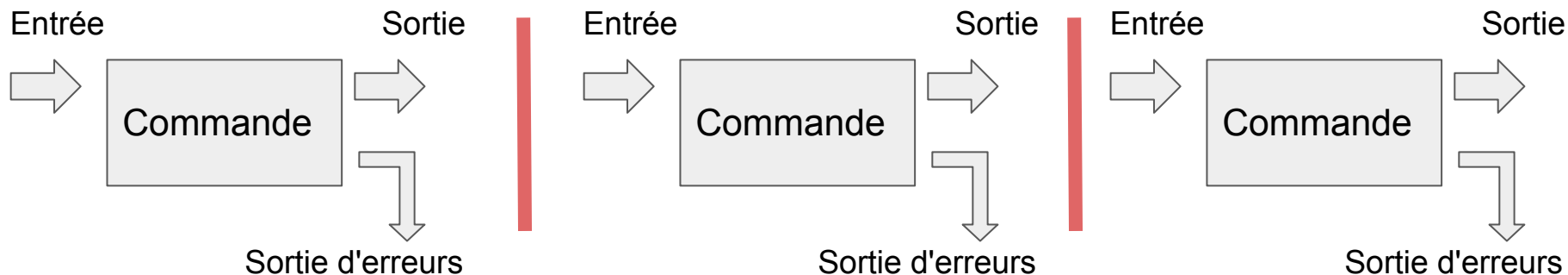
- Il faut utiliser la commande `sort` (tri alphabétique par défaut).
 - `-k` (**key**): e.g
 - `-k1,1` utiliser les caractères de la **k**olonne 1 à 1 pour le tri.
 - `-k2,2nr` utiliser les caractères de la colonne 2 à 2 pour faire un tri **num**érique (entiers) en inversant l'ordre (**reverse**).
 - `-k2,2g` (**--general-numeric-sort**) pour effectuer, sur la colonne 2, un tri sur des valeurs décimales.

```
$ # Tri alphabétique en considérant tous les caractères de chaque ligne
$ sort hg38_exons.bed
$ # Tri par chromosome (colonne 1)
$ sort -k1,1 hg38_exons.bed
$ # Trier le fichier sur la colonne 1 (chromosomes, tri alphabétique)
$ # puis par colonne 2 (starts, tri numérique):
$ sort -k1,1 -k2,2nr hg38_exons.bed # Trier les lignes par coordonnées
```



Redirections

Enchaînement de commandes



- Entrée standard (stdin): **un fichier** ou du texte (**un flux** de texte).
- Sortie standard (stdout, “output 1”): renvoyée à l’écran par défaut **et transférée via le tube**.
- Erreur standard (stderr, “output 2”): renvoyée à l’écran par défaut **et non transférée via le tube**.
- Opérateurs de redirection
 - `|` : le caractère “pipe” passe le flux de texte stdin à une autre commande
 - `> file.txt` : stocke le flux stdout en créant (ou écrasant) le fichier file.txt
 - `>> file.txt` : stocke le flux stdout en ajoutant des lignes dans le fichier file.txt
 - `2> log.txt` : stocke le flux stderr dans un fichier nommé log.txt
 - `1> file.txt 2> log.txt` : stocke stdout dans un fichier et stderr dans un autre

Exercices

- **Hommage à Marseille**
 - Utilisez les commandes `head` pour visualiser les 51 premières lignes du fichier `hg38_exons.bed` et renvoyer le résultat dans `less`
- **Hommage au finistère**
 - Utilisez les commandes `head` et `tail` pour récupérer la 29ème ligne du fichier `hg38_exons.bed`.

Solution

```
# On utilise head puis un tube qui renvoie vers less.  
# Tapez q pour quitter.  
$ head -n 51 hg38_exons.bed | less  
  
# On récupère les 29 premières lignes avec head,  
# puis on extrait de ce flux de texte la dernière ligne avec tail  
$ head -n 29 hg38_exons.bed | tail -n 1
```

Demo: enchaînements de commandes

```
$ # Obtenir la liste non redondante de chromosomes présents dans le fichier
$ cut -f1 hg38_exons.bed | sort | uniq
$ # Nombre de chromosomes différents
$ cut -f1 hg38_exons.bed | sort | uniq | wc -l
$ # Obtenir la liste des chromosomes présents dans le fichier et
$ # le nombre d'occurrence de chacun d'entre eux
$ cut -f1 hg38_exons.bed | sort | uniq -c      # '-c=count'
$ # La liste des chromosomes présents dans le fichier et leur nombre trié
$ # par ordre décroissant (-r: reverse, -n: numeric, -k: 'kolonne')
$ cut -f1 hg38_exons.bed | sort | uniq -c | sort -nr -k 1,1
```

Note: La commande **uniq** permet d'éliminer les doublons dans un flux de texte trié.

Exercice I

- En quoi la commande `less` diffère-t-elle de la commande `more` ? (15 pts)
- En utilisant la commande `grep` indiquez : (5 pts)
 - Combien y-a-t-il **d'exons sur le chromosome 22** ?
 - Combien de lignes correspondant aux exons présents sur le chromosome chr22 contiennent le terme `lincRNA` ?

Exercice I

En utilisant la command grep

- Q1: *less does more or less the same as more, but rather more than less, I like more less than more*
- Combien y-a-t-il **d'exons sur le chromosome 22** ?
- Combien de lignes correspondant aux exons présents sur le chromosome chr22 contiennent le terme lincRNA ?

Solution

```
$ grep "chr22" hg38_exons.bed | wc -l # n = 27854
$ grep "chr22" hg38_exons.bed | grep "lincRNA" | wc -l # 846
```



Merci pour votre attention.

**Remerciements à toute l'équipe
pédagogique et technique pour le
support**

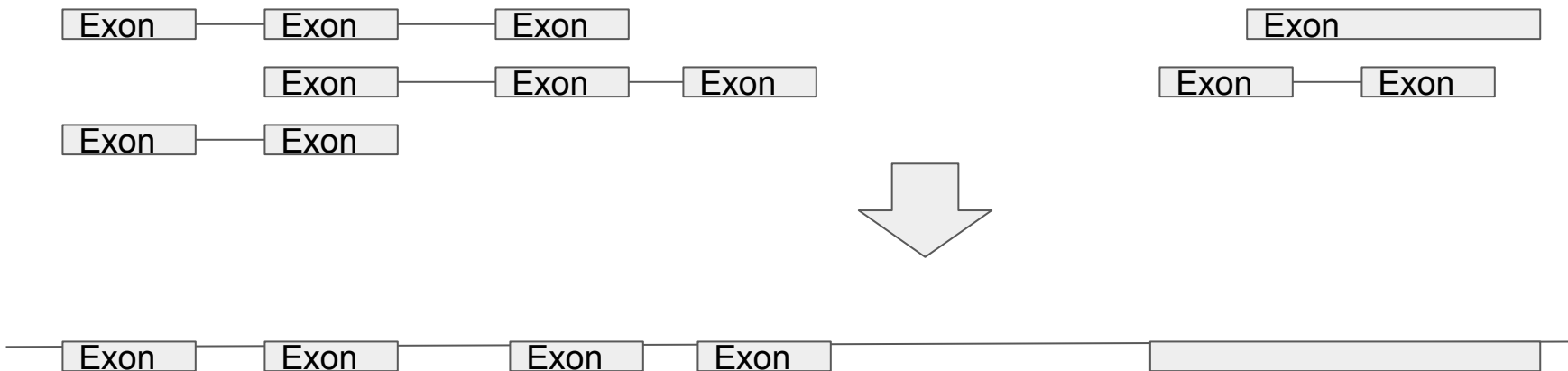


Bonus

Les diapos suivantes vous permettent, si vous le désirez, d'approfondir votre pratique de la ligne de commande Unix

Exercice II (optionnel)

- What is the genome fraction covered by exons ?
 - We must perform the operation below



- We will need bedtools

Bedtools

- A software to perform arithmetic operations on genomic coordinates.
 - <http://bedtools.readthedocs.org/en/latest/content/overview.html>
- Some example usages:
 - Extend/slop regions.
 - Compare regions (intersect).
 - Merge regions.
 - Format conversion.
 - ...
- The **bedtools** command is associated with a set of **sub-commands**.

Exercise with bedtools

- Use bedtools with -h argument.
 - What do you see ?
- Ask for some help about the **merge command** (**bedtools merge -h**)
 - Looks at the arguments.
 - Read the **note** at the end of the command. Why is it important ?

Exercice with bedtools

- Use bedtools with -h argument.
 - What do you see ?
- Ask for some help about the **merge command** (**bedtools merge -h**)
 - Looks at the arguments.
 - Read the **note** at the end of the command. Why is it important ?
-

Solution

```
$ module load bedtools/2.29.2
```

```
$ bedtools -h # l'ensemble des sous commandes
```

```
$ bedtools merge -h # utiliser l'argument -i
```

```
# la note indique que les régions génomiques doivent être triées au préalable.
```




Exercise

- Use bedtools sort and bedtools merge to merge overlapping regions/exons.



Exercise

- Use bedtools sort and bedtools merge to merge overlapping regions/exons.

```
$ bedtools sort -i hg38_exons.bed | bedtools merge
```

100

- Use the `>` redirection operator.
 - Erase file if it exists.
- `>>` can be used to add lines to an existing file.

```
$ bedtools sort -i hg38_exons.bed | bedtools merge > hg38_exons_merged.bed
$ ls
# A new file was created
```





Some arithmetic with awk



Awk

- Awk is a command available on most linux system.
- Awk has its own language.
- Awk allows to perform oneliners (and more)
- The prototype of a awk command is the following:

```
awk 'BEGIN{action}condition{action}END{action}' fichier
```

- Each set of brace is associated to a particular task:

```
BEGIN{before opening the file}
```

```
{for each line}
```

```
END{after reading all lines}
```

Awk

- Awk has **special variables**.
- Examples:

FS: Field Separator.

OFS: Output Field Separator.

NR: Number of Row.

NF: Number of **Field**.

\$0: The current line

\$1,\$2,\$3 (...): columns 1,2 ou 3 (...) of the current line

Example

```
# print columns 2 and 1 ; \t is the tabulation character
```

```
$ awk 'BEGIN{FS="\t"}{print $2,$1}' hg38_exons.bed
```

```
# print columns 2 and 1 with tabulated output
```

```
$ awk 'BEGIN{FS=OFS="\t"}{print $2,$1}' hg38_exons.bed
```

```
# print columns 2 and 1 with tabulated output and line number
```

```
$ awk 'BEGIN{FS=OFS="\t"}{print NR,$2,$1}' hg38_exons.bed
```

```
# Compute start - end for each line
```

```
$ awk 'BEGIN{FS=OFS="\t"}{print $3-$2}' hg38_exons.bed
```

```
# Apply a condition and print the 3 first column
```

```
$ awk 'BEGIN{FS=OFS="\t"} $1=="chr1"{print $1,$2,$3}' hg38_exons.bed
```



Exercice

Calculer la somme des fragments (awk)



Exercice: Calculer la somme des fragments (awk)

Calculer à chaque ligne la somme cumulée de la taille des fragments

Notez que les ";" permettent de séparer des instructions

s est une variable que l'on déclare à 0

```
$ awk 'BEGIN{FS="\t"; s=0}{s=s+$3-$2; print s}' hg38_exons_merged.bed
```

Ou encore

```
$ awk 'BEGIN{FS="\t"; s=0}{s=s+$3-$2}END{print s}' hg38_exons_merged.bed
```

A vos calculatrices (vous pouvez utiliser R).

Aller plus loin avec awk

- Le prototype d'une commande awk peut être un peu étendu en ajoutant des 'patterns' (sélecteurs ou critères).

```
awk 'BEGIN{} condition {} END{}' fichier
```

- Le critère pourra être une expression régulière (voir plus loin) ou une expression logique

```
# exemples: test si a égal b. Imprime si vrai.
```

```
awk 'a == b {} END{}' fichier
```

```
# Exemple: imprime si la colonne 1 vérifie une expression régulière.
```

```
awk '$1 ~/regExp/ {print}' fichier
```

Exemples avec des patterns

La première ligne

```
$ awk 'NR == 1 {print}' hg38_exons_merged.bed
```

La ligne 2 à 10

```
$ awk '{OFS="\t"} NR >= 2 && NR <= 10 {print NR, $0}' hg38_exons_merged.bed
```

Les lignes dont la colonne 1 contient la chaîne 'chr19'.

```
$ awk ' $1 ~ /chr19/ {print}' hg38_exons_merged.bed
```

Expressions régulières

- Permettent de décrire un motif dans une chaîne de caractères.

.	un caractère quelconque
[a-z]	une lettre minuscule (interval, ex : [u – w])
[A-Z]	une lettre majuscule (interval, ex : [U – W])
[ABc]	A ou B ou c
[^ABab]	Toute lettre différente de a et b.
^	Début de ligne.
\$	Fin de ligne
x*	0 à n fois le caractère x.
x+	1 à n fois le caractère x.
x{n,m}	Le caractère x répété n à m fois.

Exemples

<code>\.txt\$</code>	Toute chaîne finissant par “.txt”
<code>^[A – B]</code>	Une chaîne débutant par une majuscule.
<code>^\{4,6\}\.txt\$</code>	Quatre à 6 caractères suivis de “.txt”
<code>^[A – Z].*\.txt\$</code>	Une chaîne débutant par une majuscule et finissant par “.txt”
<code>^\$</code>	Une chaîne de caractères vide.
<code>^[^0 – 9]*\.sh\$</code>	Une chaîne ne contenant pas de chiffres et se terminant par “.sh”

Exercice

- En utilisant awk et un pattern, construire une expression régulière permettant de récupérer, dans le fichier hg38_exons_merged.bed, les lignes dont la colonne 1 contient chr1, chr2 et chr9 (et rien d'autre quoi que puisse contenir le fichier).



Solutions

```
# awk chr1, chr2 et chr9 (et rien d'autre !)  
$ awk ' $1 ~/^chr[129]$/ {print}' hg38_exons_merged.bed
```

Les raccourcis clavier du terminal

Ctrl + c	Annuler (c ancel)
Ctrl + u	Déléter avant le curseur
Ctrl + k	Déléter après curseur
Ctrl + w	Déléter par mot
Ctrl + r	R echercher dans les commandes précédentes
Ctrl + a	Début de ligne
Ctrl + e	Fin (e nd) de ligne
Alt + →	Déplacement mot à mot
Ctrl + l	Effacer le terminal (c lear)

Pour aller plus loin

- Supports de cours :
 - ABiMS : [Linux Initiation](#), [Linux Avancé](#), [Cluster](#)
- Agenda
 - ABiMS : <http://abims.sb-roscoff.fr/training/courses>
 - Genotoul : <http://bioinfo.genotoul.fr/index.php/training-2/training/>
 - ...

A propos de

- C'est un gestionnaire de d'outils et packages
 - Tous les langages sont supportés : Python, R, Ruby, Java, Javascript, C, ...
- Les outils viennent pré-compilés avec toutes les library systèmes et dépendances. Cela évite ainsi les problèmes d'installation
- Il permet la création d'environnement qui **isole** le ou les outils des autres. On peut ainsi disposer de plusieurs versions d'un outil et éviter les conflits de versions de dépendances.
- Les environnements sont exportables pour la reproductibilité

```
# Création d'un environnement conda (ici avec installation  
# de star, salmon, deseq2
```

```
$ conda create -n rnaseq star==2.5.3 salmon bioconductor-deseq2
```

```
# Utilisation après 'activation' de l'environnement
```

```
$ source activate rnaseq_ref
```

Configurer le ~/.bashrc

```
# We declare a new command 'll'. Equivalent to 'ls -l'
```

```
alias ll="ls -l"
```

```
# When using the 'ls' command, file and directory names will be colored.
```

```
# NB use ls --color=none or \ls to cancel this behaviour
```

```
alias ls="ls --color"
```

```
# When using the 'ls' command, file and directory names will be colored.
```

```
alias grep="grep --color"
```

```
# If the rm command is used the system will ask before... (but don't use it please)
```

```
alias rm="rm -i"
```

```
# Changing the prompt display (don't worry it this line seems obscure to you)
```

```
export PS1="\[\033[01;34m\]\h\[\033[00m\]\[\033[01;32m\] \[\033[01;32m\]\u  
\[\033[00;33m\]\w\n\[\033[01;30m\e[0m\e[1;00m\]$ "
```

Using emacs

- Emacs est un éditeur qui permet de modifier un fichier dans le terminal.
- Certains lui préféreront
 - vi
 - Nano, ...
- Some basic emacs commands
 - <ctrl> x + <ctrl> f (open a **f**ile)
 - <ctrl> x + <ctrl> s (**s**ave)
 - <ctrl> x + <ctrl> c (**c**ancel/quit)
 - <ctrl> + g (si vous êtes perdus)
 - See [emacs refcard](#)

```
$ cp ~/.bashrc ~/.bashrc_back # backup your bashrc
```

```
$ emacs ~/.bashrc # open ~/.bashrc with emacs
```

```
$ source ~/.bashrc
```

