

# Commandes Unix: pour les débutants

**Denis Puthier TAGC/Inserm**, U1090, [denis.puthier@univ-amu.fr](mailto:denis.puthier@univ-amu.fr)

**Julien Seiler**, IGBMC, [seilerj@igbmc.fr](mailto:seilerj@igbmc.fr)

**Gildas Le Corguillé**, UPMC/CNRS, [lecorguille@sb-roscoff.fr](mailto:lecorguille@sb-roscoff.fr)

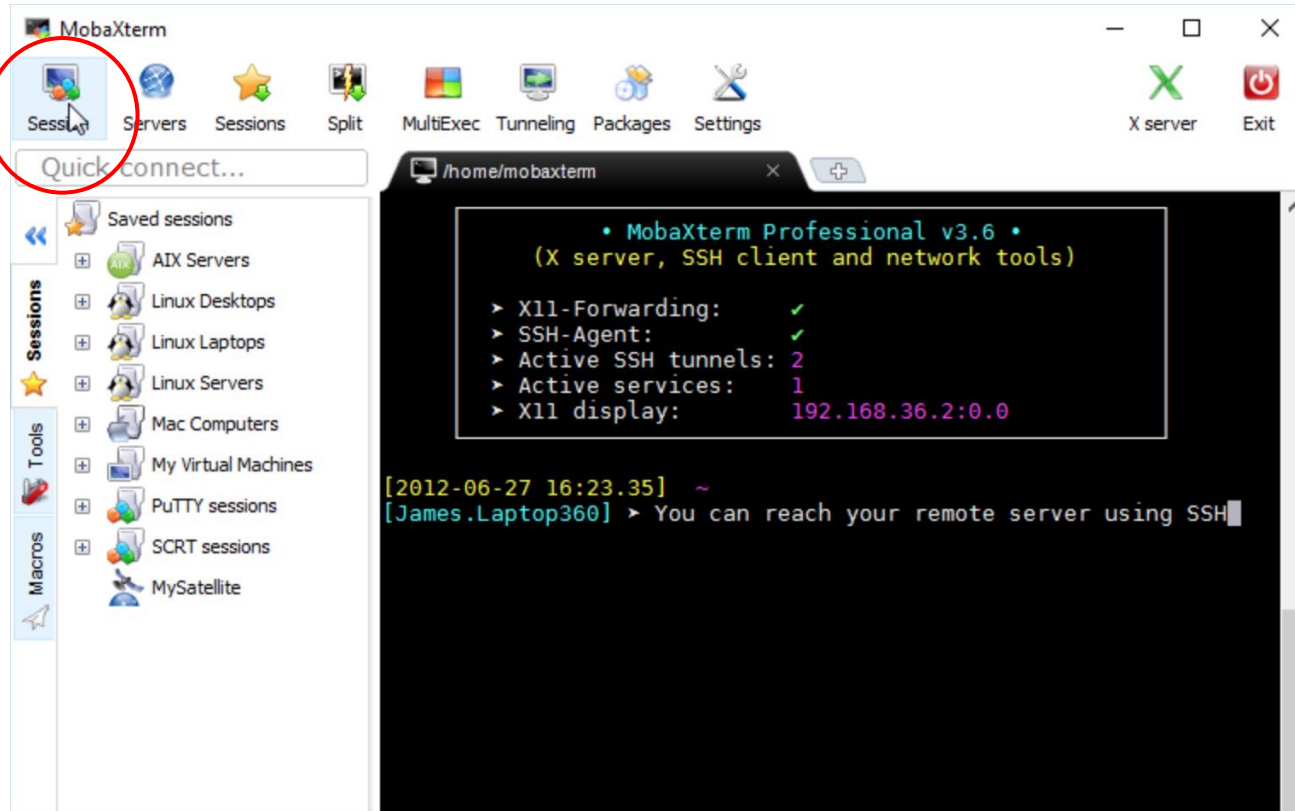
**Et tout le staff !!**

École de bioinformatique AVIESAN-IFB 2018

# 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 un **terminal**
      - **Linux ou MacOSX**
      - **Windows 10** (Bash on Ubuntu on Windows)
    - Via une application réseau comme **MobaXterm**
      - **Windows**
      - Permet l'accès à un terminal

# Se connecter depuis Windows avec MobaXterm (1)



# Se connecter depuis Windows avec MobaXterm (2)

## 1. Session

ssh

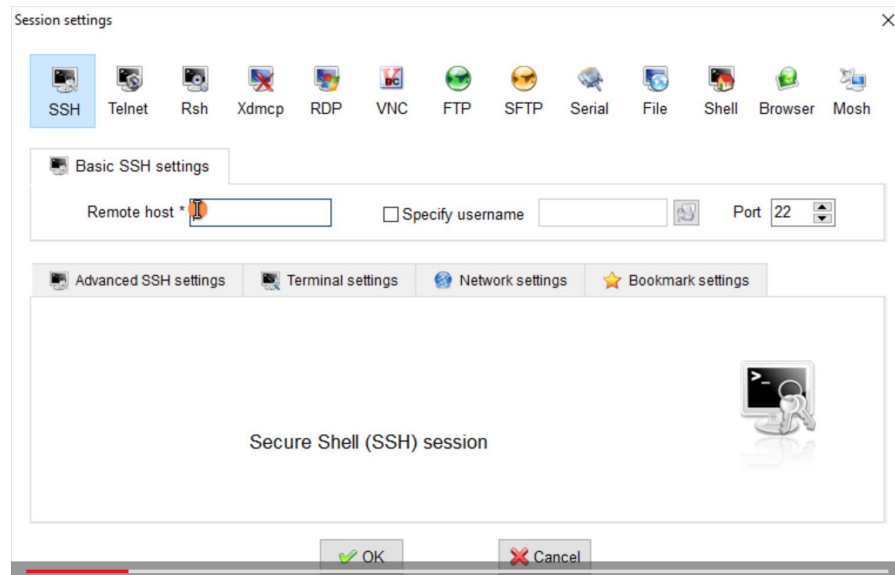
## 2. Remote host

core.cluster.france-bioinformatique.fr

## 3. Username

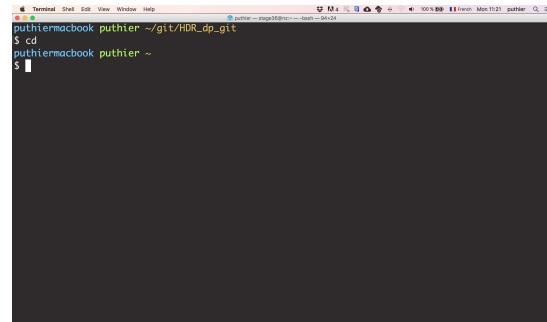
Indiquez votre login

## 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



```
$ # Connection en ssh au serveur ABiMS
$ # 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'option **-Y** permet d'ouvrir des fenêtres graphiques à distance (e.g. éditeurs)

# L'écran d'accueil (un peu sommaire)

- Prompt (invite de commande)
  - Nom d'utilisateur@hostname répertoire courant

```
[dpuhier@clust-slurm-client ~]$
```

# 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.

```
$ module load conda # Ajouter conda au PATH
```

```
$ # Charger l'environnement logiciel
```

```
$ source activate eba2018_unix
```

# Le terminal...

## Demo

Tapez 'ls' dans le terminal  
(lister les fichiers)

# lister les fichiers

\$ ls

```
[stage36@nz ~]$ ls  
mybackup  STDIN.e1262930  STDIN.o1262930  tmp  
[stage36@nz ~]$
```



# 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 (sh, 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 :)

# 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 moins 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:

**/truc** : pour chercher le terme 'truc'.

**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` (**long**) donne beaucoup d'informations sur les fichiers.
  - `-a` (**all**) montre tous les fichiers y compris ceux qui sont cachés\*.
  - `-t` (**time**) trie par date de modification.
  - `-h` (**human-readable**) affiche les tailles des fichiers en unités lisibles
  - `-r` (**reverse**) inverse l'ordre du tri.
- On peut combiner les arguments
  - `ls -l -a`
  - `ls -la`
  - `ll # alias ls -l`

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

# La commande ls et ses arguments

```
$ # Demo
$ cd /shared/bank/eba2018/fasta    # On se déplace dans le dossier
$ ls      # On liste les fichiers
$ ls -l   # Information détaillée sur les fichiers (taille, date modif,...)
$ # Vu détaillée des fichiers et taille en Ko,Mo,Go,To...
$ ls -lh
$ # Vu détaillée des fichiers, taille en Ko,Mo,Go,To..., trie par date
$ ls -tlh
$ # Vu détaillée des fichiers, taille en Ko,Mo,Go,To..., trie par date
$ # du plus ancien au plus récent
$ ls -rtlh
```

\* Le comportement par défaut est de trier par ordre alphabétique en tenant compte de la casse (i.e majuscule minuscule).

\*\* **ATTENTION** aux **espaces**, nécessaires entre la commande et ses arguments. La commande `ls-l` n'existe pas !

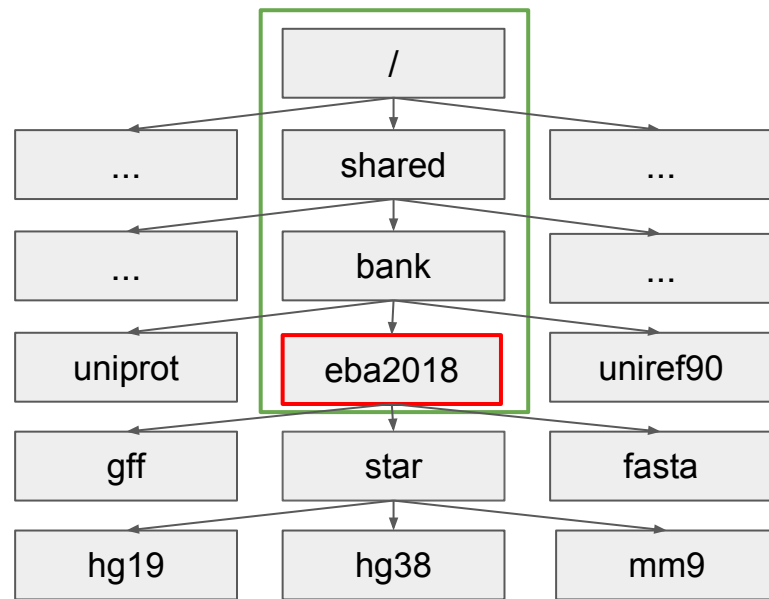


# Arborescence de fichiers



# L'arborescence du système de fichier

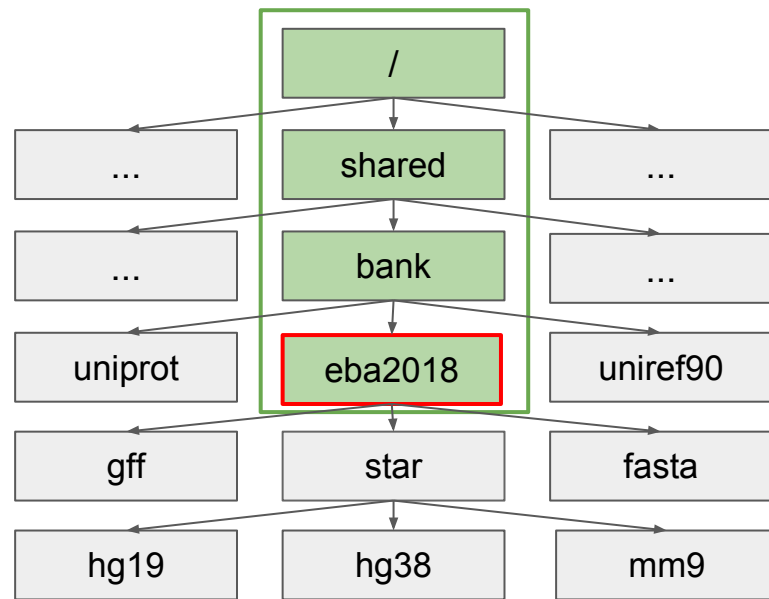
- Le système de fichier peut être vu comme un **arbre** dont les **feuilles** sont des dossiers et fichiers. On peut se **déplacer** dans cet arbre.
- Cet arbre contient une racine, le dossier **/**
- Le dossier **/** contient notamment
  - un dossier **shared** \*
    - qui lui même contient un dossier **bank**
      - qui lui même contient un dossier **eba2018**
      - ...
- Chemin du dossier
  - **/shared/bank/eba2018**



# Faire référence à un dossier ou fichier ?

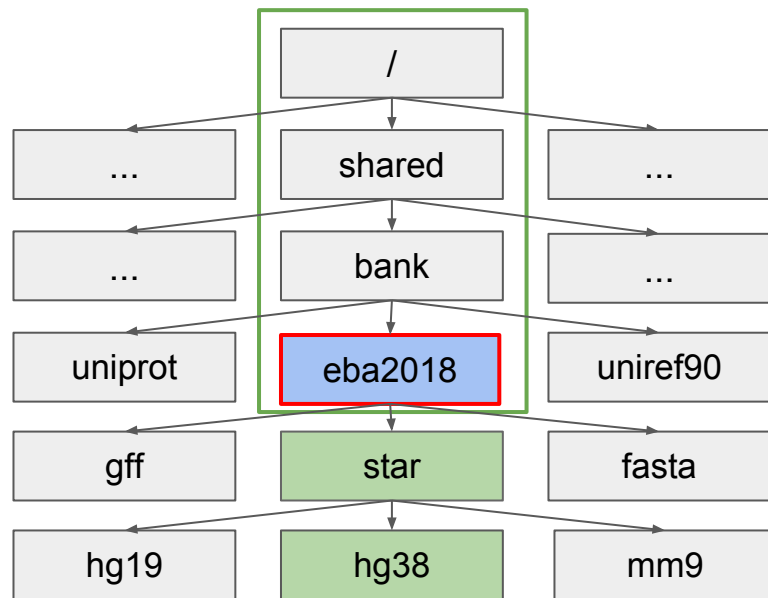
- 1) En spécifiant un chemin depuis la racine.
  - On parle de **chemin absolu**

e.g; `/shared/bank/eba2018`



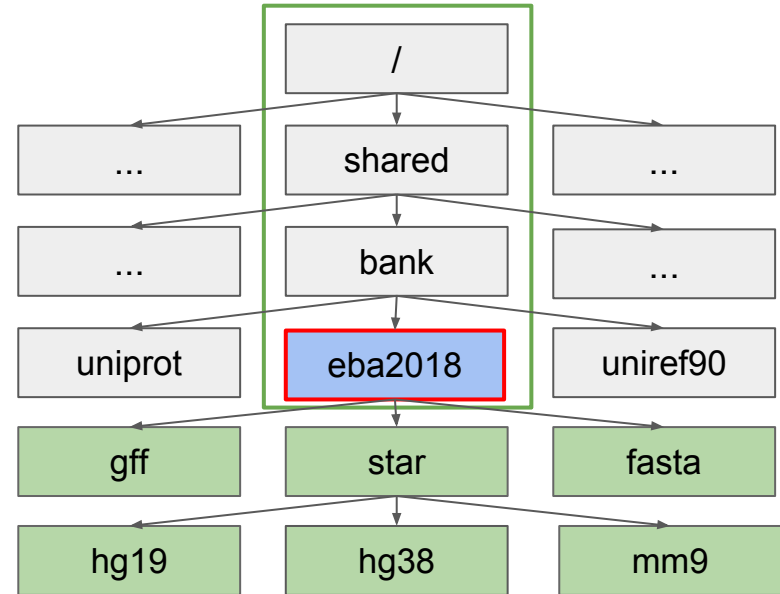
# Faire référence à un dossier ou fichier ?

- 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 eba2018 on peut aller dans star puis hg38
    - star/hg38
  - Notation équivalente, avec "." pour indiquer le dossier courant.
    - ./star/hg38



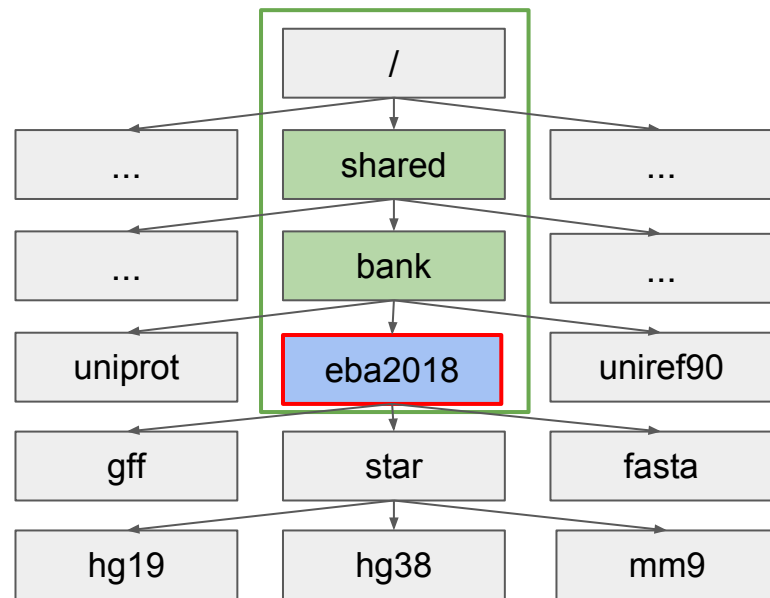
# Afficher l'arbre des sous-dossiers du dossier courant

- La commande **tree** affiche l'arbre de tous les sous-dossiers du dossier courant.



# Faire référence à un dossier ou fichier ?

- 3) Depuis le répertoire courant on peut avoir besoin de **remonter dans l'arborescence**.
  - L'opérateur '..' permet de remonter d'un niveau
  - e.g pour aller dans le dossier **shared** depuis le dossier **eba2018**
    - ../../

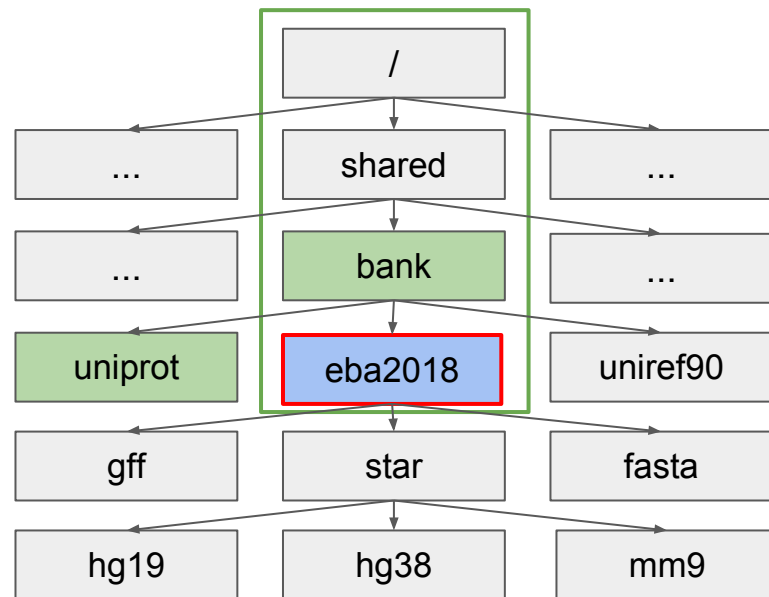


# Faire référence à un dossier ou fichier ?

- **Chemin relatif**

- E.g pour aller dans le dossier uniprot depuis le dossier eba2018

- `../uniprot`



# 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
```

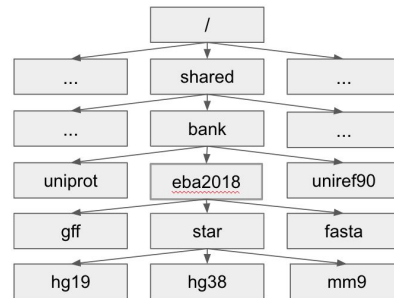
- E.g. Aller dans le répertoire
  - /usr/local/bin

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). \*



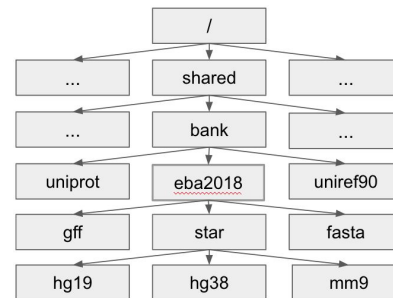
```
$ cd /shared/bank/eba2018/star/hg38 # On se déplace dans le dossier hg38
$ pwd                               # On imprime le chemin vers le répertoire courant
$ cd ..                             # On remonte d'un répertoire (star)
$ cd ../../..                       # On se déplace dans le dossier bank
$ pwd                               # /shared/bank
$ cd ../projects                    # On se retrouve 1 crans 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>/          # Aller dans uniprot_swissprot u<TAB>p<TAB>_
$ pwd                               # On se retrouve dans /shared/bank/uniprot_swissprot
```

\* Utilisez la **complétion** pour les noms les noms de fichier (**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/eba2018_cnorris`)
    - **C'est le dossier dans lequel vous devez téléverser\* vos données**
    - **C'est LE dossier pour lancer vos analyses ...**



\* Si JvH est dans la salle d'à côté, vous pouvez aussi dire 'uploader' :)

# Quels dossiers utiliser ?



- Votre dossier **scratch** (e.g [/shared/scratch/eba2018\\_cnorris](#))
  - **Le dossier pour lancer vos analyses qui vont générer des données intermédiaires volumineux et ... relativement inutiles**
- Pros and Cons
  - **Bâton:** les fichiers plus vieux que X jours seront effacés automatiquement
  - **Carotte:** les disques de l'espace scratch sont beaucoup plus performants (SSD)

# Créer des répertoires

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

```
$ cd /shared/projects/eba2018_cnorris
$ 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
```

\* 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/gff**
5. Déplacez vous dans le répertoire **hg38/gff**
6. Déplacez vous dans **annotations**.
7. Créez un sous dossier **mm10/gff**
8. Déplacez vous dans **mm10/gff**
9. Depuis ce dossier listez le contenu du dossier **hg38**

# Exercice

```
$ cd /shared/projects/eba2018_cnorris
$ 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
```

# Dossier projet  
# 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 maître et des esclaves

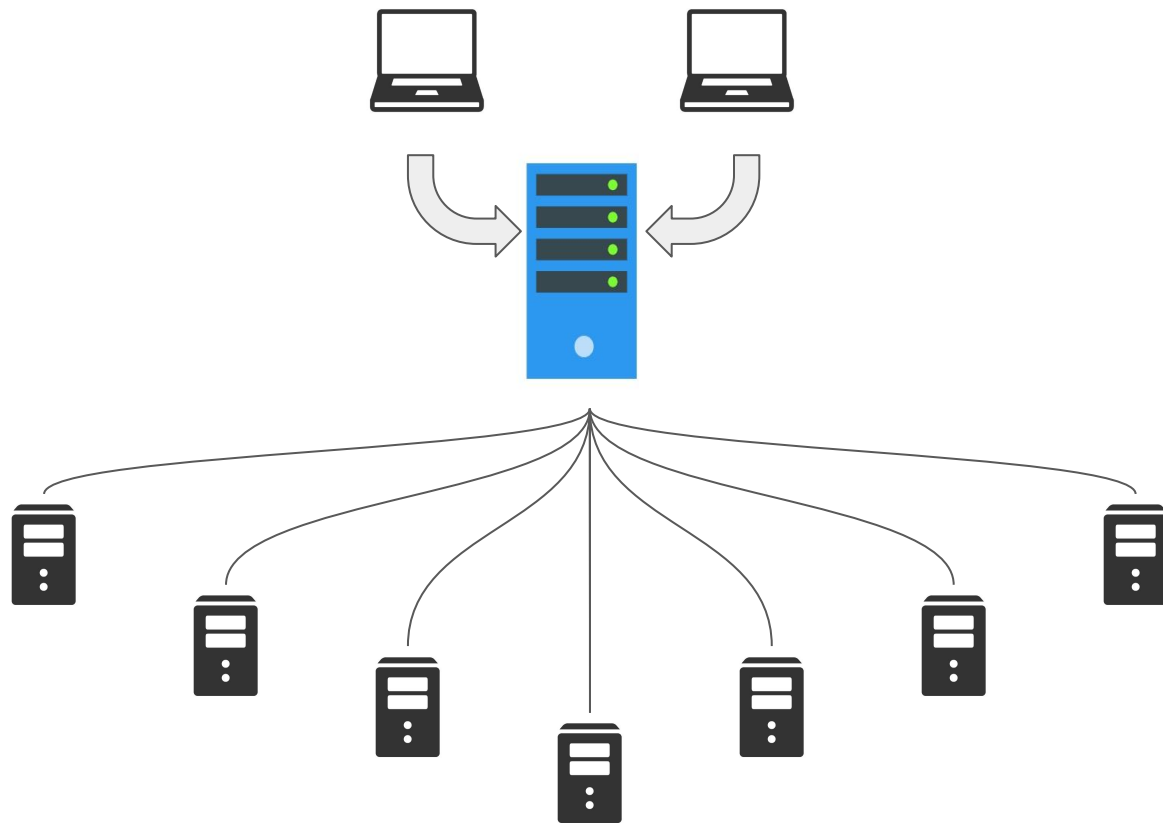
# Le cluster de calcul

- 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





# Maître et esclaves



Master



Esclave



Users

# Ne pas surcharger le maître de tâches

- Le maître a beaucoup de travail. Il s'occupe des esclaves
  - **Ne pas le déranger**
    - Si le maître est trop sollicité, il ne peut plus s'occuper des esclaves
    - Seules les commandes **peu gourmandes** en ressources (e.g **ls**, **cd**) sont acceptées sur le **maître**, dont le nom est, ici, **clust-slurm-client**.
    - Les commandes plus gourmandes doivent être lancées sur l'un des noeuds esclaves.

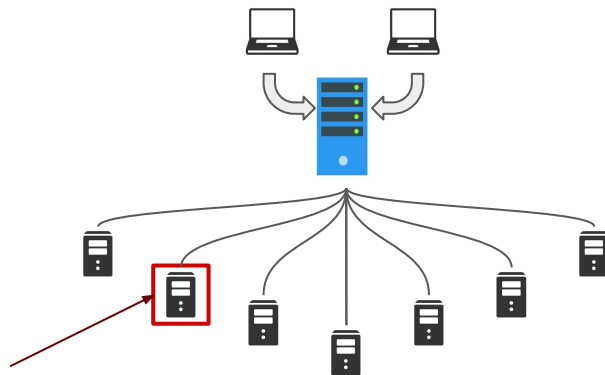
# Demander à un noeud de travailler

- Un programme disponible sur le maître (**SLURM**) permet de **réserver** des ressources (des processeurs sur des noeuds).
  - Comme à l'hôtel
    - On **réserve des chambres** dans des hôtels (des processeurs sur un noeud)
      - Mode **interactif** (on est redirigé sur le noeud cible).
        - **Les ressources ne sont libérées qu'après avoir quitté (exit)**
      - Mode **non interactif** (on reste sur le maître).
        - Quand la tâche est réalisée les ressources sont libérées

# SLURM: mode interactif

- On réserve un processeur sur un noeud par défaut.
- Utiliser sinteractive.

Le noeud qui m'a été assigné par sinteractive, le vôtre est différent



```
$ hostname                # On est connecté sur la machine-maîtresse, slurm-client
$ sinteractive             # On réserve un processeur sur un noeud.
$ hostname                # On est sur une nouvelle machine
$ exit                    # On quitte le noeud qui est alors libéré.
$ hostname                # On est de retour sur slurm-client
$ sinteractive             # On réserve un processeur sur un noeud.
$ hostname                # Un nouveau noeud nous a été attribué
```



# 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/eba2018_cnorris/annotations/hg
$ mkdir bed; cd bed
$ # On télécharge le fichier
$ wget http://pedagogix-tagc.univ-mrs.fr/courses/data/roscoff/hg38\_exons.bed.gz
$ ls -lh                                # Le fichier compressé (finit par ".gz"). Il pèse 8,4Mo.
$ gunzip hg38_exons.bed.gz             # On le décompresse
$ ls -lh                                # Le fichier a perdu l'extension gz. Il pèse 81Mo.
```

# 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.

**/truc** : pour chercher le terme 'truc'.

**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* (an hommage to JvH)



# 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.

# 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.

## # Solution

```
$ head -n 10 hg38_exons.bed
$ tail -n 10 hg38_exons.bed
$ less hg38_exons.bed
$ 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 la colonne 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

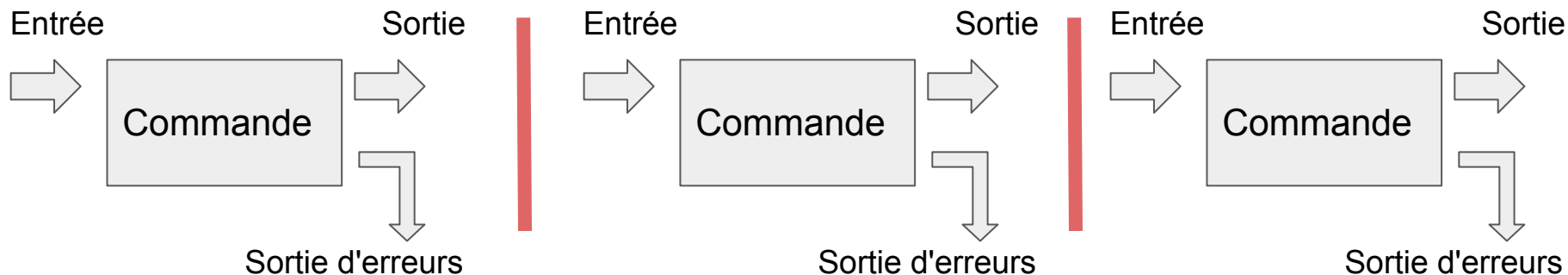
- Il faut utiliser la commande `sort` (tri alphabétique par défaut).
  - `-k` (**key**): e.g
    - `-k1,1` utiliser les caractères de la **k**olonne 1 à 1 pour le tri.
    - `-k2,2nr` utiliser les caractères de la colonne 2 à 2 pour faire un tri **numérique (entiers)** en inversant l'ordre (**reverse**).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

- Une bouteille de pastaga à gagner
  - Utilisez les commandes **head** pour visualiser les 51 premières lignes du fichier `hg38_exons.bed` et renvoyer le résultat dans **less**

- **Hommage au finistère**

- Un bouteille de chouchen à gagner !
  - 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 pour '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 -n -r k 1,1
```

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

# Exercices (notés)

- 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` ?

# Exercices (notés)

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
```

# Pour aller plus loin

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

## 2018 Courses - Session 2

Module	Date(s)	Speakers	Demande / Capacity	Registration / Course Material
Linux - Initiation	10 Dec	L. GUÉGUEN / M. HOEBEKE	0/18	Open until 20 Nov.
Cluster Usage	11 Dec	P. BORDRON / G. LE CORGUILLÉ	0/18	Open until 20 Nov.

- Genotoul : <http://bioinfo.genotoul.fr/index.php/training-2/training/>

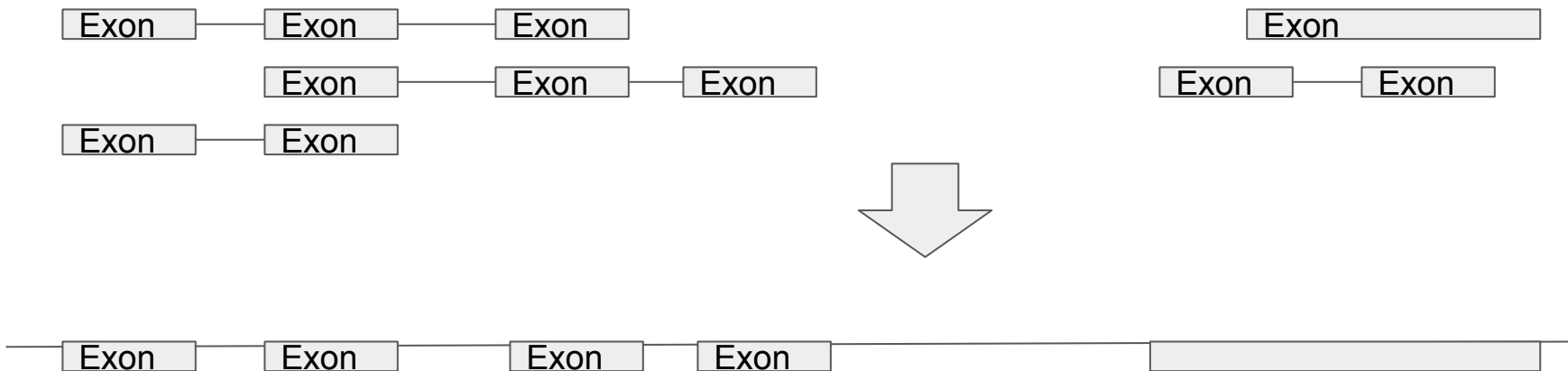


**Merci pour votre attention.**

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

# Exercise

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



- Let see how to do that...



# Using 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

```
u@m: bedtools -h          # l'ensemble des sous commandes
```

```
u@m: 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.

```
u@m: bedtools sort -i hg19_exons.bed | bedtools merge
```

100

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

```
u@m: bedtools sort -i hg38_exons.bed | bedtools merge >
hg38_exons_merged.bed
u@m: 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} {action} END{action}' fichier
```

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

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

```
{for each line}
```

```
END{after rading 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

# Exemple

```
# print columns 2 and 1
```

```
# \t is the tabulation character
```

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

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

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

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

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

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

```
u@m: awk 'BEGIN{FS=OFS="\t"}{print $3-$2}' 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
u@m: 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' (selecteurs ou critères).

```
awk 'BEGIN{} pattern {} 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
```

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

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

# Exemples avec des patterns

# La première ligne

u@m: awk 'NR == 1 {print}' hg38\_exons\_merged.bed

# La ligne 2 à 10

u@m: awk '{OFS="\t"} NR >= 2 && NR <= 10 {print NR, \$0}' hg38\_exons\_merged.bed

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

u@m: 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 caractères x.
x+	1 à n fois le caractères 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 !)

u@m: awk ' \$1 ~/^chr[129]\$/ {print}' hg38\_exons\_merged.bed



# SLURM/SGE: mode non interactif

Interactif	Non interactif / Mode batch
<p>[A Roscoff] Job limité à :</p> <ul style="list-style-type: none"><li>- 2 jours</li><li>- 2 processeurs (sauf pendant eba)</li><li>- Faible priorité</li></ul>	<p>Presque pas de limites :</p> <ul style="list-style-type: none"><li>- Pas de limite de temps</li><li>- Possibilité de réserver jusqu'à 32 / 48 voir 128 "cpu"</li><li>- Haute priorité des instructions traitées</li></ul>
"Manuel"	"Automatisable" : Possibilité de soumettre via un script (e.g. 20000 jobs)
<p>Vous pouvez surveiller vos calculs via les sorties standard voir intervenir s'ils plantent ou s'ils demandent une intervention (confirmation ou ajustement)</p>	<p>Vous perdez la main donc vous devez "rediriger" les sorties de vos programmes dans des fichiers</p>
<p>"Qui tue le père tue le fils" : quand le terminal est fermé il met fin à tous les processus lancés</p>	<p>Les calculs/jobs sont lancés, vous perdez la main dessus mais l'ordonnanceur le surveille pour vous</p>



Note: sbatch (SLURM) == qsub (SGE)

# A propos de CONDA<sup>®</sup>

- 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
```