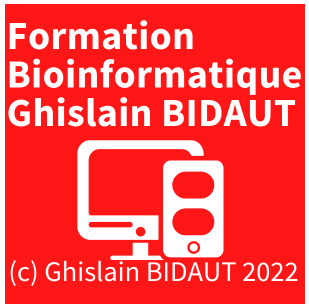


Introduction à Linux et à l'environnement Conda

Ghislain Bidaut, Plateforme Cibi, CRCM, Aix-Marseille Université

24/03/2022

Introduction à Linux et à l'environnement Conda

- **Partie 1: Linux Basique**
  - Qu'est ce que **Linux** ?
  - La ligne de commande
  - La manipulation de fichiers
- **Partie 2: Utilisation de Linux - plus avancée**
- **Partie 3: La manipulation de processus (jobs)**
- **Environnement Conda**
  - L'environnement de gestion de logiciels **Conda**
- **Utilisation d'un cluster de calcul sous *Slurm***
  - Utilisation du **cluster de l'IFB**

Première Partie: Linux Basique

Linux

- C'est un système d'exploitation **ouvert** et **libre**, issu du monde de la recherche.
- Cela signifie que **tout le monde** peut l'utiliser, le modifier et l'étudier.
- Vous l'utiliser probablement déjà car il est à la base d'Android et fait tourner un grand nombre de serveurs Web (67% source: W3Tech).
- **Linux** est stricto-sensu le **noyau**. Le système complet est **GNU-Linux**, souvent **distribué** avec de nombreux programmes (éditeurs, terminaux, etc..)

Il est accessible sous forme de **Distribution:** dont les deux familles **principales** sont:

- *Ubuntu*, *Mint*, *Debian*,
- *RedHat*, *CentOS*, etc…

(Voir le site http://distrowatch.com)

Organisation

- **Multi-utilisateur:** chaque utilisateur possède un **espace propre** accessible par un **mot de passe**
- Plusieurs utilisateurs peuvent travailler en même temps sur une même machine, par la console, ou par des connections distantes
- Chaque fichier possède des **permissions** associées aux **utilisateurs**. Il est la **propriété** d'un utilisateur et il est possible, pour cet utilisateur, de choisir des permissions spécifiques

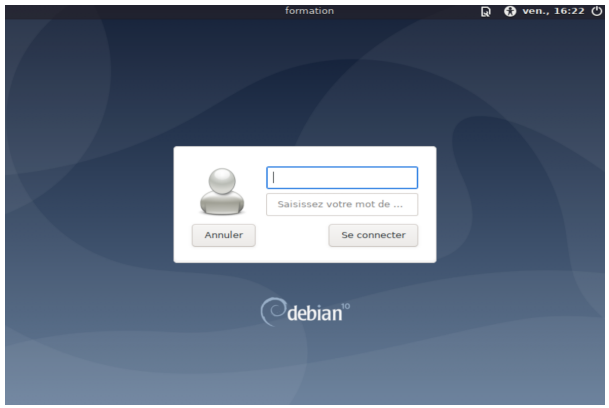
Pourquoi ne pas utiliser Windows pour faire de la bioinformatique ?

- Linux est à code source ouvert. Il est issu de **l'industrie** et de la **recherche scientifique** et a été concu pour le **traitement** de gros corpus de données
- Linux est **multi-tâches** et **multi-utilisateurs**
- Windows est sous **licence payante**
- Linux est **scalable**, (fonctionne aussi bien qur votre PC que sur de gros serveurs multiprocesseurs) (2048 max!) avec une grosse mémoire (256To max!)
- Linux est plus **fiable** que Windows, et mieux **sécurisé (Pas de virus)**
- **Linux est très personnalisable** et il est possible d'installer uniquement les composants dont l'utilisateur a besoin
- Les programmes utilisés en bioinformatique, dont font partie **Python** et **R** ont été développés sous Linux
- Les programmes **vendus** pour Windows sont chers et sont des **boites noires** (pas de code source dispo)

Plusieurs types de connection

Linux étant un système **multi-utilisateur**, il est capable de gérer plusieurs connections simultanément. Il est donc **nécessaire** de se **connecter au système** pour l'utiliser, ce qui lui permet de nous identifier.

- Soit sous forme de connection *Locale* sur la console graphique (ou texte)

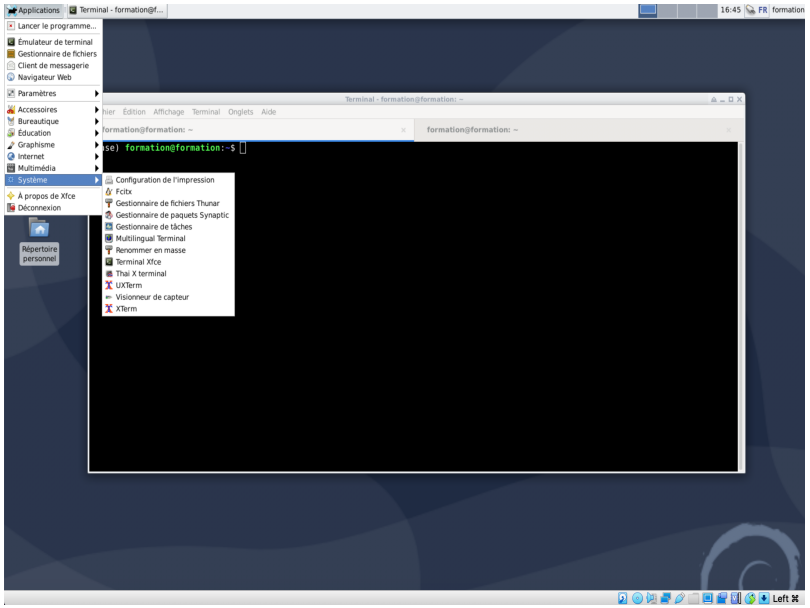


- Soit sous forme de connection *Distante* (Secure Shell - *ssh*). Utlilisation de *Putty* ou *MobaXterm* sous *Windows*, et du programme *ssh* sous *MacOS/Linux*.

Les environnements graphiques

Sous Linux, l'environnement graphique est très fortement customisable est est organisé de la façon suivante:

- Le gestionnaire de fenêtres: Une couche logicielle de base pour l'affichage: **XWindow** (Sans s)
- Le Bureau: plusieurs versions existent, plus ou moins complexes et gourmandes en ressources: **KDE**, **Gnome**, **XFCE**, etc…
- Dans le cadre de la formation, nous travaillerons sous **XFCE**.



La ligne de commande: appelée également le Shell

Pourquoi une ligne de commande ?

Linux est un dérivé d'*Unix* , un système concu dans les années 70, sur lesquels un moyen naturel de communiquer était par ligne de commande:

- C'est un moyen plus puissant que l'interface graphique
- Peu gourmand en ressources
- Défaut: il faut une phase de formation
- Le logiciel qui permet de donner des ordres au noyau est le ***shell.*** *Un exemple de shell est* Bash* (*Bourne Again Shell*).
- C'est un véritable **Langage de programmation** possédant une structure de contrôle de très grande puissance!

Que permettent les commandes ?

Une **commande** permet de:

- **Récupérer des informations système** (Exemple: nom du serveur, date, heure utilisation du disque ou de la mémoire)
- **Faire des manipulation de fichiers basiques** (création ou suppression de répertoires, copies ou déplacement de fichiers)
- **Exécuter des traitements complexes** (Exemple: alignement de séquences en analyse bioinformatique)
- **Modifier le comportement des processus** (arrêt, interruption, reprise, modification de priorité)

Comment interagir avec Linux ?

Nous allons travailler avec la machine Virtuelle **Linux Debian** fournie pour cette formation.

Nom d'utilisateur: *formation*, mot de passe: *formation*

Nous utilisons le logiciel *Terminal* qui permet d'accéder au **Shell**.

Nous utilisons le **Bourne Again Shell** (*bash*). Il s'utilise par la saisie d'une ligne de commande, suivi par un appui sur **Entree**.

L'utilisateur (*formation*) est présenté avec l'invite de commande (*prompt*) sur la machine (*debiancibi*)

```
utilisateur@machine:~$
```

Pour se connecter sur une machine Linux distante

```
ssh utilisateur@machine_distante #-ajouter X pour rediriger l'écran graphique
```

Exemple: Connexion sur le **cluster de calcul de l'IFB**: `IFB-core`

```
ssh utilisateur@core.cluster.france-bioinformatique.fr
```

Message apparaissant à la **première connexion seulement!**. Répondre `yes`.

```
The authenticity of host 'machine_distante (::1)' can't be established.
ECDSA key fingerprint is SHA256:xk24jY2GxbgKTXXavyWtQqjBZwxQUYrt0FpLbrATxW8.
Are you sure you want to continue connecting (yes/no)?
```

Votre première commande sous Linux

Votre première commande sous Linux: `ls`

```
ls
```

Pour fermer une connexion distante

```
exit
```

Recherche d'aide sous Linux

Recherche d'aide sur une commande spécifique: les pages de *manuel*:

```
man ls
```

Vous pouvez utiliser les flèches pour vous déplacer dans la page. Utilisez la touche **q** pour quitter.

Aide basique

```
whatis ls
```

Recherche dans les pages de manuel

```
apropos "list directory content"
```

Et bien sur, votre **moteur de recherche favori est votre ami**!

Navigation dans l'arborescence avec `cd`

Chemin personnel à l'arrivée dans le système:

```
/home/formation (ou ~)
```

Afficher le chemin courant

```
pwd
```

Retour au répertoire personnel

```
cd
```

Naviguer dans un autre répertoire

```
cd /bin
```

Retour au répertoire précédent

```
cd -
```

Remonter d'un cran dans l'arborescence

```
cd .. # . est le répertoire courant
```

Autres commandes liées à la navigation

Mémorisation du répertoire courant suivi d'un cd rep

```
pushd rep
```

Rappel et cd dans le répertoire mémorisé

```
popd
```

Lister les fichiers avec `ls`

Lister les fichiers du répertoire courant

```
ls
```

Lister les fichiers avec plus d'information (taille, permissions)

```
ls -l
```

lister les fichiers cachés (dont le nom commence par '.')

```
ls -a
```

lister les fichiers avec affichage en couleur

```
ls -laC
```

lister les fichiers avec tri par date

```
ls -lat
```

lister les fichiers avec tri par date inversé

```
ls -lart
```

lister les fichiers avec tri par taille (plus gros en premier)

```
ls -laS
```

lister les fichiers avec affichage de la taille plus lisible par un humain

```
ls -lh
```

Création et suppression de fichiers

Création d'un fichier vide (ou mise à jour de la date d'un fichier existant)

```
touch fic
```

Suppression d'un fichier

```
rm fic
```

Création d'un répertoire

```
mkdir rep
```

Suppression d'un répertoire

```
rm –r rep
```

Copier et renommer un fichier ou un réperoire

Copier un ou plusieurs fichier dans un répertoire

```
cp fic1 rep
cp fic1 fic2 rep
```

Copier un ou plusieurs fichier dans le répertoire courant: `.`

```
cp fic1 .
```

Copier un répertoire. Si `rep2` existe, `rep1` est copié **dans** `rep2`

```
cp -r rep1 rep2
```

Renommer un fichier

```
mv fic1 fic2
```

Déplacer un fichier

```
mv fic1 rep
```

Renommer un réperoire. Si `rep2` existe, `rep1`est déplacé **dans** `rep2`.

```
mv rep1 rep2
```

Copier et renommer un fichier ou un répertoire de manière distante: `scp`

Copier un fichier d'une machine distante vers le répertoire courant:

```
scp  <username>@core.cluster.france-bioinformatique.fr:/shared/home/<username>/fic1  .
```

Copier un répertoire d'une machine distante vers le répertoire courant:

```
scp -r <username>@core.cluster.france-bioinformatique.fr:/shared/home/<username>/rep  .
```

Copier un répertoire vers une machine distante:

```
scp -r rep <username>@core.cluster.france-bioinformatique.fr:/shared/home/<username>/
```

Customiser son terminal: alias

Commande alias: Création d'une "commande" *customisée*:

```
alias ll='ls -l'
```

Liste des alias courants:

```
alias
```

Quelques alias utiles à ajouter par sécurité:

```
cp='cp -ip' (interactive + preserve)
rm='rm -i' (interactive)
```

Nommage des fichiers

Il est préférable de choisir des noms explicites pour vos fichiers, tout en restant raisonnables sur la longueur du nom.

- **Nom absolu :** chemin de la racine `/` au fichier en suivant la liste des répertoires, les noms des répertoires sont séparés par des / (Exemple: `/usr/bin`)
- **Nom relatif :** chemin du répertoire courant au fichier en suivant la liste des répertoires, les noms des répertoires sont séparés par des / (Exemple: `bin`)
- Pas de limitation trop stricte sur la longueur (plus de 200).
- Composé de lettres (minuscules et majuscules sont différenciées), de chiffres et de caractères
- **les autres caractères**, accentués par exemple, sont **acceptés mais non recommandés**
- `.` désigne le répertoire courant, `..` désigne le répertoire parent, par exemple:

```
ls -l "../bin"
```

Raccourcis utiles

Il est possible d'éviter de taper entièrement une commande: **utiliser la complétion par `TAB`**

Recherche dans les commandes: **Flèches haut** ou **Flèche bas**.

Recherche de commandes: `CTRL-R` puis taper dans le champ de recherche.

```
(reverse-i-search)`ls': ls -la .bash*
```

Navigation dans une commande: `CTRL-<Fleche droite>` ou `CTRL-<Fleche gauche>`

```
(reverse-i-search)`ls': ls -la .bash*
```

Annuler l'exécution d'une commande: `CTRL-C`

```
du -sh^C
```

Lister l'historique:

```
history 30 # 30 dernières commandes
```

Manipulation de fichier

Examen du contenu d'un fichier texte

Télécharger l'exemple de MyCore -> data_Linux -> Fichier VCF

```
cat SRR10426968.vcf
```

A l'envers

```
tac SRR10426968.vcf
```

Filtrage des lignes d'un fichier texte

```
grep "<DEL>" SRR10426968.vcf
```

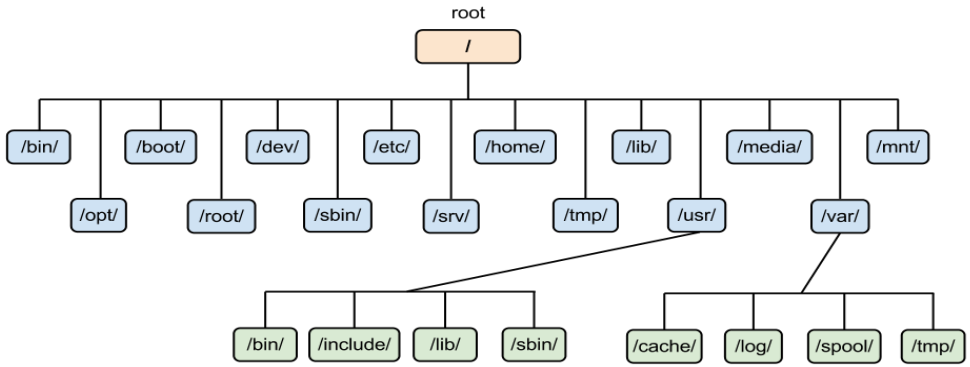
Défilement contrôlé: `more` (mieux: `less` ;-))

```
less SRR10426968.vcf
```

Types de fichiers

Type de fichier	Utilisation
ordinaire	Fichier contenant des documents ordinaires (texte, images, sons, vidéo, …)
répertoire	répertoire (dossier) contenant des répertoires ou des fichiers
lien (voir commande `ln`)	lien logique vers un fichier existant
bloc	fichier associé à un périphérique de stockage (clé USB, DVD, disque dur, …)
caractère	fichier associé à un périphérique d'entrée-sortie (clavier, écran, …)
socket	fichier associé à un point de connexion réseau (échange de données)
tube (|)	fichier associé à l'exécution automatique d'une commande

Organisation du système de fichiers sous Linux



(*)

- Chemin `absolu`: `/usr/bin`
- Chemin relatif: `bin` ou `./bin` si on est déjà dans `/usr`

(*) Extrait de https://www.rs-online.com/designspark/an-intro-to-linux-file-system-management

Quelques exemples de navigation dans l'arbrescence à l'aide de la commande `cd`

```
cd /bin
cd /usr/bin
cd bin
cd /
cd ..
cd ../.. # Remonter deux niveaux
cd .
cd
```

Seconde Partie: Utilisation de Linux - plus avancée

Commandes d'environnement

Qui suis-je ?

```
whoami
```

Où suis-je ?

```
hostname
```

Information sur votre compte

```
id
```

Information sur votre Système d'exploitation

```
uname (-a)
```

Changer votre mot de passe (**Ne le faites pas!**)

```
passwd
```

Qui travaille en ce moment sur le système ?

```
w ou who
```

Customisation avancée

Modification du prompt: il est défini par la variable `$PS1`

```
echo $PS1

\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\
[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
```

Fichiers de configuration

```
.bash_profile
.bashrc
```

Ajouts de raccourcis (‘alias’) soit dans **.bash_aliases**, soit dans **.bash_profiles**

```
alias ll='ls -la' # Pas d'espaces autour du signe '='
```

Voir les variables d’environnement

```
env
```

Métacaractères/Jokers

Caractère	Utilisation
`*`	toute chaîne de caractères, potentiellement vide
`?`	exactement un caractère
`{v1,v2,v3}`	exactement un caractère, soit **v1**, soit **v2**, soit **v3**
`{v1..v2}`	exactement un caractère, entre **v1** à **v2**
`{v1..v2..v3}`	exactement un caractère, entre **v1** à **v2** par pas de **v3**

Exemples:

```
echo {a..c}{1,9} {a..c}*
a1 a9 b1 b9 c1 c9 activite.png bc-lampe.jpg comparaison.py
echo {a..g..2}{1..7..2}
a1 a3 a5 a7 c1 c3 c5 c7 e1 e3 e5 e7 g1 g3 g5 g7
echo *.{jpg,png}
activite.png bc-lampe.jpg
echo {a..c}*.???
activite.png bc-lampe.jpg
```

Affichages de caractères spéciaux

En Bash, certains caractères (`*`, `$`, `?`, `>`, `&`) jouent un rôle spécial et ne peuvent pas être affichés simplement.

```
echo $
$
echo $1 # paramètre positionnel 1

echo $$ # process ID du script
 20105
echo $? # statut de sortie de la dernière commande
 0
echo *
hg19_order.fa sample_19098-007.log sample_19098-007.sam fichier.gp trace.png
echo > # redirection
bash: syntax error near unexpected token ‘newline’
```

Pour les afficher, il faut les désactiver (les *échapper*) avec `\`

```
echo \*
```

Organisation des droits

```
ls -la
total 60184
drwxr-xr-x 19 formation formation     4096 juil. 21 09:57 .
drwxr-xr-x  3 root      root          4096 mai   28 14:54 ..
-rw-------  1 formation formation      679 juil. 18 16:18 .bash_history
-rw-r--r--  1 formation formation     4018 mai   28 16:19 .bashrc
drwxr-xr-x  2 formation formation     4096 mai   28 15:24 Bureau
-rwxr-xr-x  1 formation formation 61451533 déc.  21  2020 Miniconda3-py39_
  4.9.2-Linux-x86_64.sh
```

Organisation en 3 groupes de droits

r = lecture

w = écriture

x = exécution

o = propriétaire

g = groupe

w = autres (other)

Gestion par la commande `chmod`. Exemple:

```
chmod u+x Miniconda3-py39_4.9.2-Linux-x86_64.sh # Executable pour l'utilisateur
```

Commande chmod

```
chmod [u g o a] [+ - =] [r w x] nom_du_fichier
```

- **u** = user
- **g** = group
- **o** = other
- **a** = all

Commande `chmod`: correspondances de représentation des droits

Droit	Alphanumérique	Octal	Binaire
aucun droit	—	0	000
exécution seulement	–x	1	001
écriture seulement	-w-	2	010
écriture et exécution	-wx	3	011
lecture seulement	r–	4	100
lecture et exécution	r-x	5	101
lecture et écriture	rw-	6	110
tous les droits (lecture, écriture et exécution)	rwx	7	111

chmod: exemples

Rendre exécutable pour l'utilisateur

```
chmod u+x Miniconda3-py39_4.9.2-Linux-x86_64.sh
```

Rendre exécutable pour tous

```
chmod a+x Miniconda3-py39_4.9.2-Linux-x86_64.sh
```

Donner tous les droits à l'utilisateur, et lecture seulement pour le groupe et les autres

```
chmod 744 Miniconda3-py39_4.9.2-Linux-x86_64.sh
```

Informations détaillées de `ls -la`

```
ls -la
total 60184
drwxr-xr-x 19 formation formation     4096 juil. 21 09:57 .
drwxr-xr-x  3 root      root          4096 mai   28 14:54 ..
-rw-------  1 formation formation      679 juil. 18 16:18 .bash_history
-rw-r--r--  1 formation formation     4018 mai   28 16:19 .bashrc
drwxr-xr-x  2 formation formation     4096 mai   28 15:24 Bureau
-rwxr-xr-x  1 formation formation 61451533 déc.  21  2020 Miniconda3-py39_
1     2    3  4         5              6    7             8
```

Position	Caractère	Utilisation
1	d	fichier ordinaire (**-**), lien logique ou symbolique (**l**), répertoire (**d**), bloc (**b**), caractère (**c**), tube (**p**), socket (**s**)
2	rw-r–r–	modalité d'accès au fichier (propriétaire, groupe et les autres). Modalités : lecture (**r**), écriture (**w**), exécution (**x**) ou pas de modalité (**-**)
3	19	nombre de liens (ici, lien vers un fichier existant)
4	formation	identifiant du propriétaire du fichier

Informations détaillées (Suite)

```
ls -la
total 60184
drwxr-xr-x 19 formation formation     4096 juil. 21 09:57 .
drwxr-xr-x  3 root      root          4096 mai   28 14:54 ..
-rw-------  1 formation formation      679 juil. 18 16:18 .bash_history
-rw-r--r--  1 formation formation     4018 mai   28 16:19 .bashrc
drwxr-xr-x  2 formation formation     4096 mai   28 15:24 Bureau
-rwxr-xr-x  1 formation formation 61451533 déc.  21  2020 Miniconda3-py39_
1     2    3  4         5              6    7              8
```

Position	Caractère	Utilisation
5	formation	identifiant du groupe du fichier
6	4096	nombre de nodes occupées par le fichier
7	juil. 21 09:57	date de modification
8	.	nom du fichier

Commande `find`

Cette commande permet de trouver un fichier dans le système de fichier et éventuellement d'y appliquer une action. Sa syntaxe est la suivante:

```
find <répertoire> action
```

`find` fonctionne de manière récursive par défaut.

Exemples:

```
 # trouve tout les fichiers du répertoire courant
find . -print
# recherche le fichier .bashrc
find . -name .bashrc -print
# fichiers portant l'extension .sh
find . -name "*.sh" -print
# trouve les répertoires Images dans les répertoires utilisateurs
# et renvoie les messages d'erreurs dans /dev/null
find /home -type d -name "Images" 2> /dev/null
# recherche les fichiers portant l'extension tmp et les
# efface après confirmation;
find . -type f -name "*.tmp"  -exec rm -i  {} \;
```

Commande `find`

Voir la page http://www.linux-france.org/article/memo/node126.html

`-name`: Recherche par nom de fichier.

`-type`: Recherche par type de fichier.

`-user:` Recherche par propriétaire.

`-group:` Recherche par appartenance à un groupe.

`-size:` Recherche par taille de fichier.

`-atime:` Recherche par date de dernier accès.

`-mtime` Recherche par date de dernière modification.

`-ctime`: Recherche par date de création.

`-perm`: Recherche par autorisations d'accès.

`links` : Recherche par nombre de liens au fichier

Compression/Décompression

Il existe plusieurs commandes de **compression** et **d'aggrégation** de fichiers. Ces deux opérations sont **distinctes** sous **Linux**.

Utilitaire de compression: `gzip`

```
gzip fich
gunzip fich.gz
```

Plus puissant: `bzip2`

```
bzip2 fich
bunzip2 fich.bz2
```

Compression/Décompression (suite)

Pour plusieurs fichiers ou un répertoire, il faut d'abord utiliser l'utilitaire d'agrégation `tar` (Tape Archiver), puis compresser.

```
tar cf rep.tar rep # Add -v for verbosity
gzip rep.tar
gunzip rep.tar
tar xf rap.tar     # Add -v for verbosity
```

Identique mais en combinant les opérations

```
tar czf rep.tar.gz rep # Add -v for verbosity
tar xzf rep.tar.gz     # Add -v for verbosity
```

Il existe également un utilitaire `zip`.

```
zip fic.zip fic # Compression d'un fichier
unzip fic.zip
zip -r rep.zip rep     # Compression d'un répertoire
```

Manipulation de fichiers texte

Affichage du début d'un fichier

```
head fic.txt
head -10 fic.txt # affichage des 10 premières lignes
```

Affichage de la fin d'un fichier

```
tail fic.txt
tail -f fic.txt # avec rafraichissement
```

Extraire des colonnes d'un fichier délimité

```
cut  -f 2 fic.tab
cut  -f 1,3 fic.tab
cut  -f 1,3 fic.tab
cut -f 1-2,5-6 fic.tab
cut -d , -f 2 fic.csv # comma delimiter
```

Boucles for

Il est possible d'appliquer des traitement de manière répétée par l'instruction `for`:

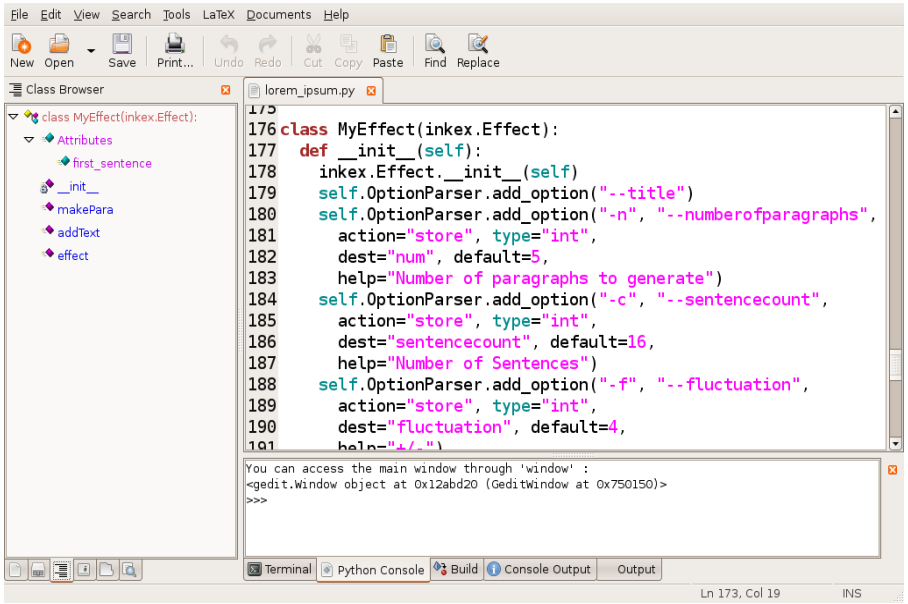
Un exemple simple:

```
for f in *txt
do
  echo "$f"
done
```

Voir: https://www.cyberciti.biz/faq/bash-for-loop/

Editeurs interactifs sous Linux

- Gedit



- Nano
- Pour administrateurs: **Vi, Emacs**

Editeur série `sed`

`sed` est un *editeur série*. Il permets d'effectuer des manipulations de texte de manière automatisée et est particulièrement utile pour manipuler de grosses masses de données.

Exemple:

On remplace *Linus* par *Linux* dans le fichier `test.txt`

```
sed 's/Linus/Linux/' text.txt
```

On peut l'utiliser pour remplacer des noms de fichiers: **Exemple**: Remplacement des **-** par des **_** dans les fichiers `.txt` du répertoire courant (**Attention, c'est irréversible**).

```
for f in *.txt; do fn=`echo $f|sed 's/-/_/g'`; mv -i "$f" "$fn"; done
```

Un site dédié à `sed`: https://buzut.net/apprendre-commande-sed-linux/

Téléchargement en ligne de commande

Commande `wget`

```
wget http://ftp.sra.ebi.ac.uk/vol1/fastq/SRR900/008/SRR9005678/SRR9005678.fastq.gz
--2021-08-04 17:08:22--  http://ftp.sra.ebi.ac.uk/vol1/fastq/SRR900
                                                 /008/SRR9005678/SRR9005678.fastq.gz
Résolution de ftp.sra.ebi.ac.uk (ftp.sra.ebi.ac.uk)... 193.62.193.138
Connexion vers ftp.sra.ebi.ac.uk (ftp.sra.ebi.ac.uk)|193.62.193.138|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 1383706547 (1,3G) [application/octet-stream]
Sauvegarde en : «SRR9005678.fastq.gz»

 5% [===>
                                                 ] 79 563 695  2,79MB/s  eta 7m 53s
```

Installation de programmes sous Debian

L'installation de programmes sous Linux est généralement réservé aux utilisateurs ayant les **permissions** appropriées.

Pour installez un programme dans le système, il faut être **superutilisateur** (`root`) ou bien faire partie du groupe des **sudoer**, c'est à dire avoir la permission d'utliser la commande **sudo**.

Quelques commandes utiles pour changer d'utilisateur:

```
su - # pour passer sur le compte root
su - <utilisateur> # pour passer sur le compte 'utilisateur'
```

L'installation se fait par un **gestionnaire de paquets**, qui gère les dépendances et les versions en fonction de la distribution utilisée.

Dans le cas de **Debian** et affiliées (**Ubuntu**, **Mint**), nous utilisons **APT**.

Utilisation sommaire d'APT

Installation d'un programme, par exemple `gedit`:

```
sudo apt-get install gedit
[sudo] Mot de passe de formation :
...
Souhaitez vous continuer ? [O/n]
```

Suppression d'un paquet

```
sudo apt-get remove gedit # Supression d'un paquet
sudo apt-get purge gedit # Supression d'un paquet et de ses dépendances
```

Création et exécution de scripts

But: Automatiser des tâches.

Principe: Créer un fichier texte (le *script*) contenant les instructions à exécuter.

- **Important**: Le fichier texte doit comporter le "*sha-bang*" à la première ligne:

```
#!/bin/bash
```

- Il doit être rendu **exécutable** par la commande `chmod`:

```
chmod +x monscript.sh
```

Partie 3: Gestion des Processus

Gestion des processus

- Un processus est un programme géré par le système. C'est autour de ce concept que sont organisés les
  - **Partage du temps CPU**,
  - **Partage de de la mémoire**
- Un processus peut avoir différents états : en **exécution**, en **sommeil**, en **sommeil sans possibilité d'interruption**, **stoppé**, **zombie** ;
- Il existe différentes **priorités d'exécution** : utilisation différenciée du temps CPU
- Processus **bloquant** (la commande associée au processus en cours bloque l'exécution de toute autre commande ultérieure):

```
sleep 100 ; date    # la commande date sera exécutée uniquement
                    # après les 100 secondes d'attente de la commande sleep
```

- Le système des gestion des processus est arborescent : principe parenté, duplication ; répertoire `/proc`.

Gestion des processus (suite)

Il est important de gérer ce que l'on exécute sur la machine de manière à ne **pas la saturer** en **mémoire** ou **CPU**.

Liste des processus

```
ps -ef # Extended full display
```

Cherche un numéro de processus

```
pgrep motif
```

Utilitaire d'affichage des processus

```
top # ou htop
```

Affichage d'informations mémoires

```
pmap proc
```

Affichage de priorité

```
chrt -p proc
```

Lancement de cmd avec priorité 30

```
nice 30 cmd
```

Changement de priorité de proc à 30

```
renice 30 proc
```

Envoi d'un signal **-9** à proc

```
kill -9  proc
```

Mode bloquant

Pour exécuter trois commandes séquentiellement:

```
cmd1
cmd2 # S'exécute une fois cmd1 terminée
cmd3 # s'exécute une fois cmd2 terminée
```

ou

```
cmd1; cmd2; cmd3
```

Exemple

- Un alignement de séquences suivi d'une fusion ("merge") des séquences alignées:

```
bwa mem -a -t 8 ref.fa reads/data_1.fastq reads/data_2.fastq > GDR-18_pe.sam
bwa mem -a -t 8 ref.fa reads/data_SE.fastq > GDR-18_se.sam
samtools merge GDR-18.sam GDR-18_pe.sam GDR-18_se.sam
```

- Les deux commandes d'alignements `bva mem` sont indépendantes et peuvent donc s'exécuter indépendamment l'une de l'autre.
- La commande `samtools` doit être exécutée **uniquement** lorsque les commandes `bva mem` sont **terminées**.

Mode non bloquant

- Imaginons que `cmd1` et `cmd2` soient indépendantes et que `cmd3` soit dépendante du résultat de `cmd1` et `cmd2`.
- Il est possible de lancer une seconde commande sans bloquer le terminal en ajoutant une esperluette (`&`) à la fin de la première commande.

```
cmd1 &
cmd2
cmd3
```

Exemple de commande lancée en mode non-bloquant

- Nous lançons les deux commandes d'alignement simultanément en ajoutant un **&** à la fin de la première commande:

```
bwa mem -a -t 8 ref.fa reads/data_1.fastq reads/data_2.fastq > GDR-18_pe.sam &
bwa mem -a -t 8 ref.fa reads/data_SE.fastq > GDR-18_se.sam
```

- Le terminal nous renvoie un message quand les deux commandes sont terminées:

```
[1]+ Done bwa mem -a -t 8 ref.fa reads/data_1.fastq...
```

- Puis on peut lancer le merge par **samtools**:

```
samtools merge GDR-18.sam GDR-18_pe.sam GDR-18_se.sam
```

Contrôle des jobs

Ces commandes lancées en arrière-plan sont des **jobs**. Ils peuvent être contrôlés par les commandes suivantes:

- Affichage des jobs:

```
jobs
[1]+ Running bwa ... & # Cette commande tourne dans le canal 1
```

Il est possible de lancer plusieurs commandes en arrière plan simultanément dans différents *canaux*.

- Contrôle des jobs

```
<CTRL><C> # Terminaison d'un job (annulation)
<CTRL><Z> # Suspension d'un job
```

Exemples d'actions de contrôle de jobs

- Utilisation de **<CTRL><C>** et **<CTRL><Z>**

```
gzip human_g1k_v37.fasta ^C # job annulé
gzip human_g1k_v37.fasta ^Z # job suspendu
[1]+ Stopped gzip human_g1k_v37.fasta
jobs
[1]+ Stopped gzip human_g1k_v37.fasta
```

- Commandes de contrôle `fg`, `bg` et `kill`:

```
kill %1 # termine le job sur le canal 1
fg %1 # passe le job  du canal 1 en mode bloquant (foreground)
bg %1 # passe le job du canal 1 en mode non-bloquant (background)
```

Sorties standard/erreur

- **Entrée standard** (0) : contenu d'un fichier, clavier, flux de données
- **Sortie standard** (1): écran habituellement
- **Sortie standard des erreurs** (2): écran habituellement

Possibilité de remplacer l'entrée ou les sorties standards par un *fichier* :

- **Exemple d'utilisation:** Utilisation de données stockées dans un fichier à la place de la saisie clavier (Utile pour de gros volumes de données)
- **Exemple d'utilisation 2:** le résultat de la commande est directement stocké dans un fichier (Par exemple pour générer un fichier *log*).

Redirection de la sortie standard (1) dans un fichier

Utilisation de **>** pour stocker la sortie dans un fichier (le fichier est **écrasé**)

```
grep 'Chr' data.txt > info_Chr.txt
bwa mem -a -t 8 ref.fa reads/data_SE.fastq > GDR-18_se.sam
```

Utilisation de **>>** pour ajouter la sortie dans un fichier existant (le contenu du fichier est **conservé**)

```
extrait_chromosome.py chr8 data.fa > resultats2.fa
ls -s resultats2.fa
1302112 resultats2.fa
extrait_chromosome.py chr5 data.fa >> resultats2.fa
ls -s resultats2.fa
3009296 resultats2.fa
```

Redirection de la sortie d'erreur standard (2) dans un fichier

Utilisation de `2>` pour stocker la sortie d'erreur dans un fichier (le fichier est **écrasé**)

Exemple:

```
$ bwa mem -t 32 -P -M human_g1k_v37.fasta
sample_19098-007_R1.fastq.gz sample_19098-007_R2.fastq.gz
> sample_19098-007.sam 2> sample_19098-007.log
```

- Le résultat de la commande `bwa mem` est stocké dans le fichier `sample 19098-007.sam`
- Les erreurs et messages d'information sont stockées dans le fichier `sample 19098-007.log`

Autres redirections

- Redirection des erreurs standards (2) vers la sortie standard (1) avec `2>&1`

```
$ commande 2>&1
```

- Redirection des erreurs standards (2) vers la sortie standard (1) avec `2>&1` et redirection du tout vers un fichier `fic.log`

```
commande > fic.log 2>&1
```

- Redirection des erreurs standards (2) vers la sortie standard (1) avec `2>&1` et redirection du tout vers `/dev/null` (suppression)

```
commande  >> dev.null  2>&1
```

Chainage de commande par tube

Utilisation du symbole tube | (*pipe*).

Le tube permet de rediriger la **sortie standard** de la première commande vers l'**entrée standard** de la seconde commande

Exemple: Calcul du nombre total de read dans un fichier fastq

```
grep '+' my.fastq | wc -l
```

Execution de programmes en tâche de fond

Il existe des utilitaires pour exécuter des programmes en tâche de fond: Cela permet de faire tourner un programme après fermeture de la session par l'utilisateur.

- Directement en ligne de commande: `nohup`

```
nohup gzip human_g1k_v37.fasta &
nohup: les entrées sont ignorées et la sortie est ajoutée à « nohup.out »
```

- Plus avancé: lancement de terminaux persistants avec `screen`

```
screen # création d'un screen
screen -l # liste les session screen
screen -r <numero> # récuperer une session screen (attacher)
screen -d <numero> # détacher une session screen
```

Quizz

Quizz

Quelle est la meilleure façon d'éliminer la sortie d'une commande comme `ls`?

1. `ls -q`
2. `ls > /dev/null`
3. `ls > /null`
4. `ls -out`

Si les permissions `-rwxr-xr--` sont accordées au groupe `gp1` pour un fichier celui-ci a les droits

1. de lecture, écriture et exécution
2. de lecture et d'exécution
3. de lecture uniquement
4. d'exécution uniquement

Quizz

La commande `uname`

1. n'existe pas
2. permet de créer un alias
3. indique l'ID d'un fichier
4. informe sur l'OS et sa version

Quelle commande liste les processus de tous les utilisateurs ?

1. `ps -e`
2. `ps`
3. `ps -a`
4. `ps -f`

Quizz

`tar` est un logiciel dont la fonction principale est

1. la compression
2. le cryptage
3. le ftp
4. l'archivage

Comment obtenir le nombre de mots d'un fichier texte `fich1`?

1. `cat -w fich1`
2. `cat -c fich1`
3. `wc -w fich1`
4. `ls -w fich1`

Quizz

Quelle option de `grep` renvoie les lignes qui ne correspondent pas à l'expression ?

1. `-i`
2. `-u`
3. `-r`
4. `-v`

La commande `mv rep1 rep2`

1. copie `rep1` vers `rep2`
2. déplace `rep1` dans `rep2`
3. déplace `rep1` et `rep2` dans le dossier courant
4. renomme `rep1` en `rep2`

Quizz

Comment se connecter au système par le réseau en mode texte ?

1. `sh user@machine`
2. `ksh user@machine`
3. `ssh user@machine`
4. `bash user@machine`

Que valent respectivement r w x en octal ?

1. 4 2 1
2. 1 2 4
3. 0 1 2
4. 2 1 0

Quizz

Que fait `cal 8 256` ?

1. définit un pipe de buffer 256 octets
2. calcul le modulo de 256 / 8
3. affiche 256 en octal
4. affiche le mois d'Août de l'an 256

Que signifie `2>&1` après une ligne de commande ?

1. rediriger l'erreur standard vers la sortie standard
2. rediriger l'erreur standard vers l'entrée standard
3. éliminer les messages de la sortie d'erreur standard
4. concaténer les messages de la sortie erreur standard

Quizz

A quoi correspond le caractère | dans le système d'exploitation Unix ?

1. un opérateur de factorielle
2. un opérateur de concaténation
3. un pipeline
4. un OU logique

Si un fichier a les permissions `-rw-r--r--` , son propriétaire a les droits

1. de lecture et d'exécution
2. de lecture uniquement
3. d'exécution uniquement
4. de lecture et d'écriture

L’environnement Conda

Environnement Conda

Conda est une **plateforme de gestionnaire de paquets et d'environnements logiciels** permettant d'installer à sa guise des dizaines de paquets Python, R et d'utilitaires Linux, incluant la **plupart des programmes** utilisés en bioinformatique.

C'est un système très utile pour **l'utilisateur débutant**, aussi bien que pour **l'utilisateur expérimenté** car il permet:

- L'installation de programmes sans **droits administrateurs** (!)
- De gérer des dépendances de paquets avancées lors des installations.

Voir le site: https://docs.conda.io/en/latest/

Comment fonctionne Conda ?

Conda s'apppuie sur un installateur. Dans ce cours, nous utilisons **miniconda**. une version en ligne de commande simple. Il existe également **Anaconda**, une version plus complète incluant *Anaconda Navigator*, un gestionnaire conda graphique.

En plus du programme d'installation existent les **canaux** (*channels*), contenant les programmes gérés par Conda. Le canal contenant les programmes de bioinformatique s'appelle **bioconda**.

Comment installer **miniconda + bioconda** ?

- Etape 1: récupérer le programme `miniconda`:
- Etape 2: configurer les **canaux**
- Etape 3: chercher des **programmes** et les **installer**
- Etape 4: gérer ses **environnements**

Installation

- Aller sur le site https://docs.conda.io/en/latest/miniconda.html
- Récupérer la bonne version (Linux Python 3.9)
- Exécuter l'installateur

```
chmod +x Miniconda3-py39_4.9.2-Linux-x86_64.sh
./Miniconda3-py39_4.9.2-Linux-x86_64.sh
```

Une fois installé, l'invite bash comporte l'entête "base":

```
(base) utilisateur@machine:~ $
```

Les canaux (Channels)

Les programmes sont accessible à travers un index contenu dans les canaux (*channels*).

Les programmes bioinformatiques ne sont pas accessibles par défaut. Ces programmes sont dans le channel *bioconda*, qu'il faut configurer.

La canal *conda-forge* contient des programmes supplémentaires apportés par la communauté de développeurs.

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

Gérer les environnements

Plutôt que de tout installer ce dont nous avons besoin au même niveau, il est possible de compartimentaliser les programmes dont nous avons besoin dans différents **environnement** Conda pour:

- **Regrouper logiquement** les programmes dédiés à une tâche précise, par exemple, créer un environnement dédié à l'analyse RNA-Seq.
- Avoir à disposition **plusieurs versions** d'un programme.
- Gérer des environnements **plus légers**, donc plus rapides à mettre à jour et à exporter vers d'autres systèmes.
- Eviter de se retrouver avec des programmes **incompatibles** dans le même environnement.

Commandes de gestion d'un environnement Conda

Créer un environnement (Exemple: pour le **RNA-Seq**)

```
conda create –n rnaseq
```

Activer un environnement

```
conda activate rnaseq
```

Recherche un programme par nom

```
conda search samtools
conda search *st*
```

Installer un programme

```
conda install samtools
```

Sortir de l'environnement courant (retour dans *base*)

```
conda deactivate
```

Mise à jour de Conda

Nécessite un peu de patience !

```
conda update -n base conda
conda update --all
```

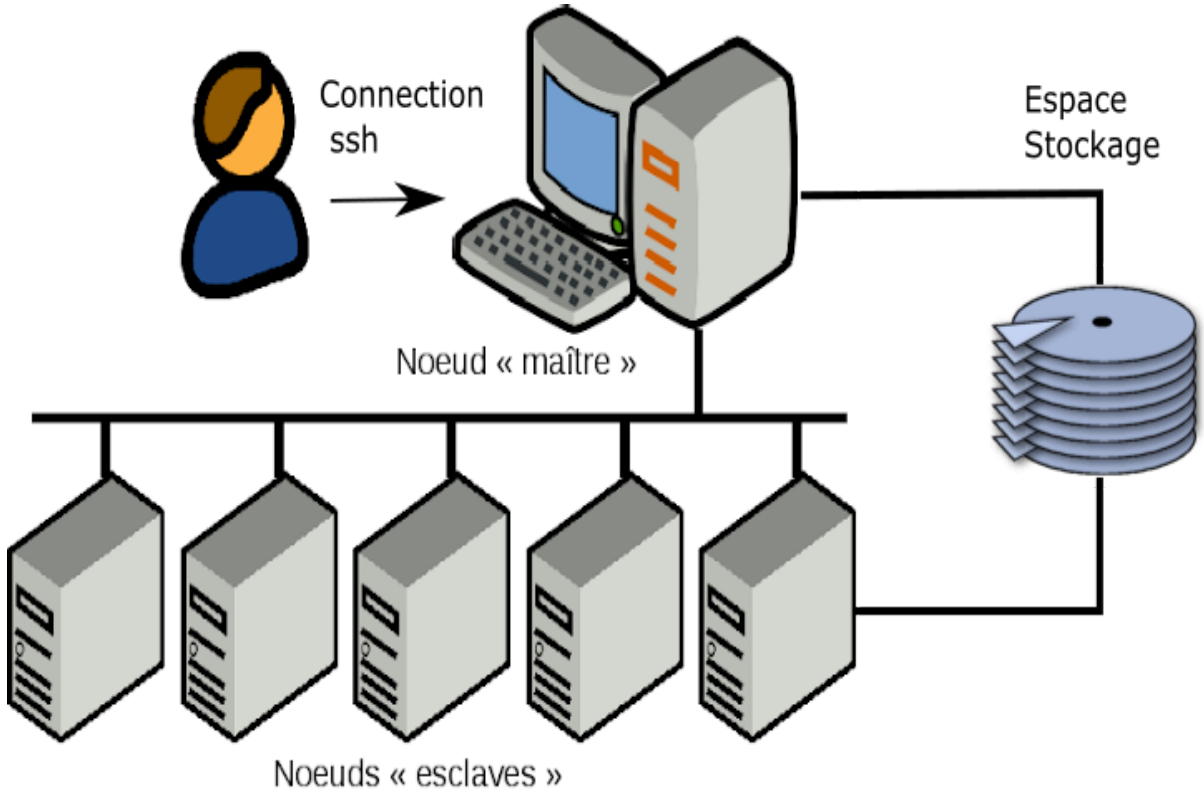
Sauver/restaurer une configuration d'environnement dans un fichier `yml``

```
conda env export -n envname > envname.yml
conda env create --file envname.yml
```

Utilisation d’un cluster de calcul sous *Slurm*

Cluster de l'IFB

Pour lancer des calculs lourds, nous utilisons le **Cluster de calcul IFB-Core** de l'**Institut Français de Bioinformatique:** https://www.france-bioinformatique.fr/cluster-ifb-core/



Organisation du cluster de l'IFB

La documentation utilisateur est disponible à https://ifb-elixirfr.gitlab.io/cluster/doc/

Le cluster de l'IFB est organisé de la manière suivante:

- Un **nœud maître** sur lequel on se connecte directement
- Des **nœuds de calculs** qui permettent d'exécuter les *jobs*.
- Un **stockage destiné au calcul** (ce n'est **pas** un **stockage permanent**).
- Accès à travers un *ordonnanceur* (**Slurm**). Cet ordonnanceur est un système de réservation de ressources (temps, mémoire et nombre de coeurs)
- Il met les jobs **en attente** en attendant leur traitement une fois que les ressources demandées sont disponibles.

Utilisation pratique: connection

Connection par `ssh`

```
ssh user@core.cluster.france-bioinformatique.fr
```

Visusalisation des jobs courants

Liste des jobs de l'utilisateur *user* tournant sur le cluster

```
squeue -u user
```

Liste des jobs de l'utilisateur *user* tournant sur le cluster

```
squeue -u user
```

Partitions

- Une partition permet de gérer les priorités des jobs:
  - Partition `fast`: Pour les **jobs de 0 à 24h** (défaut).
  - Partition `long`: Pour les **jobs de 0 à 30j** (défaut).

Pour voir la liste des partitions:

```
sinfo
```

Un **acompte** (appelé également **projet**) permet de faire de l'*accounting* au propriétaire du cluster et de voir quel projet consomme des ressources.

Par défaut, les jobs sont soumis au compte `démo` qui possède 6h de calcul. Dans le cadre de cette formation, nous utilisons l'acompte `form_2022_09`.

Gestion des jobs sous Slurm

Soumission d'un job sur un acompte:

```
sbatch -A <acompte> ./script.sh
```

Annulation d'un job

```
scancel <numéro du job>
```

Formatage d'un script pour soumission

On peut préciser les variables `SBATCH` suivantes:

```
#!/bin/bash

#SBATCH --job-name=trimming           # Job name
#SBATCH --mail-type=END,FAIL          # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --mail-user=<user_mail>       # Where to send mail
#SBATCH --ntasks=1                    # Run on a single core
#SBATCH --mem=5gb                     # Job memory request
#SBATCH --time=02:00:00               # Time limit hrs:min:sec
#SBATCH --output=trimming_%j.log      # Standard output and error log

for fastqgz in *.fastq.gz
do
        echo "process $fastqgz"
        trimmomatic ....
done
```

Soumission du job

On soumet le job sur un *Acompte* (Obligatoire sinon c'est le compte `demo` qui est utilisé). Dans le cadre de la formation, l'acompte est `form_2022_09`:

```
sbatch -A form_2022_09 myscript.sh
```

On peut surveiller le bon déroulement d'un job par

```
squeue -u <username>
```

Le résultat de la soumission génère deux sorties:

- Un ou 2 fichiers texte contenant les **sorties standards** et **sorties d'erreur standards**
- Le cas échéant, l'utilisateur reçoit un e-mail (si spécifié dans les options)

Message de l'IFB!

```
##   ************************************************************    ##
##                                                                    ##
##       Cluster Shutdown for Maintenance on April 4-5-6-7th 2022     ##
##                                                                    ##
##   The cluster and its associated services (RStudio, JupyterLab,   ##
##   UseGalaxy, community, my...) will be shutdown for maintenance:   ##
##    -  from 14:00 on monday April 4th 2022                          ##
##    -  to 12:00 (noon) on thursday April 7th 2022                   ##
##                                                                    ##
##   L'équipe de support Cluster IFB Core
```

Quelques sources d'information supplémentaires:

- Présentation du système Linux: https://jplu.developpez.com
- Tutoriel de l'ENS: https://www.tuteurs.ens.fr/unix/
- *Le Système Linux* par Matt Welsh (O'Reilly)
- Merci à Laurent Mouchard (Univ. Rouen) pour son cours Linux (L3) qui m'a inspiré certaines diapositives.

Licence