Introduction à Unix/Linux

« Bioinformatique pour les NGS »

Guy Perrière

Pôle Rhône-Alpes de Bioinformatique

10 septembre 2018

Système d'exploitation

Il existe deux types de programmes :

- Les *applications* ou programmes utilisateurs = services directement utiles (*e.g.* traitement de texte, calcul scientifique).
- Les *programmes système*, utilisé pour interagir avec le matériel :
  - Lancer un programme.
  - Écrire sur un disque dur.
  - Afficher un texte à l'écran.
  - ...

Système de fichiers

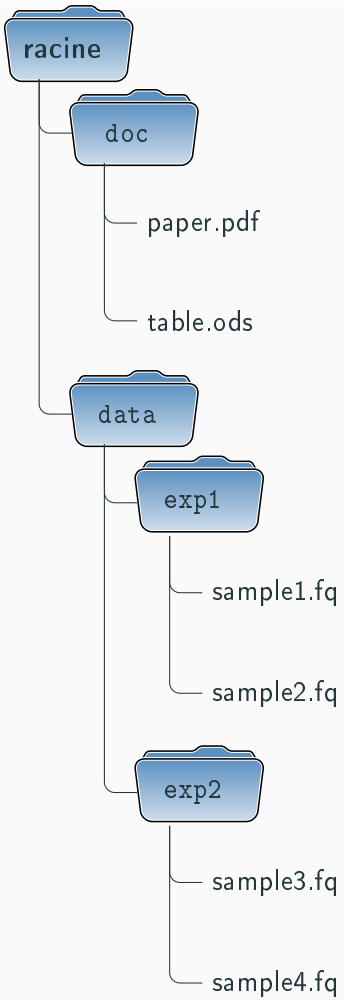
En première approximation

Façon d'organiser le stockage des données

Ingrédients typiques :

- Le *fichier* (= document ou programme).
- Le *répertoire* (= dossier pouvant contenir de fichiers et des répertoires.

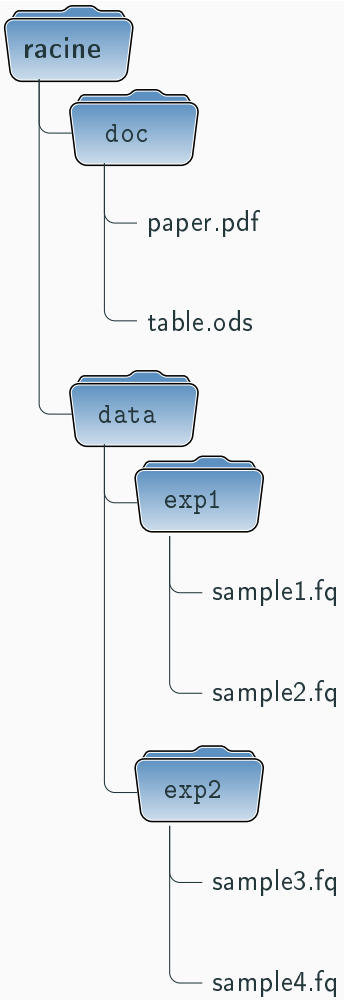
Arborescence et chemin



- Répertoire et fichiers forment un arbre (à l'envers).
- Le répertoire contenant l'ensemble des autres répertoires et fichiers est appelé la racine.
- Chaque fichier ou répertoire peut être identifié par la suite de répertoires menant de la racine à lui :

  `racine` → `data` → `exp1` → `sample2.fq`

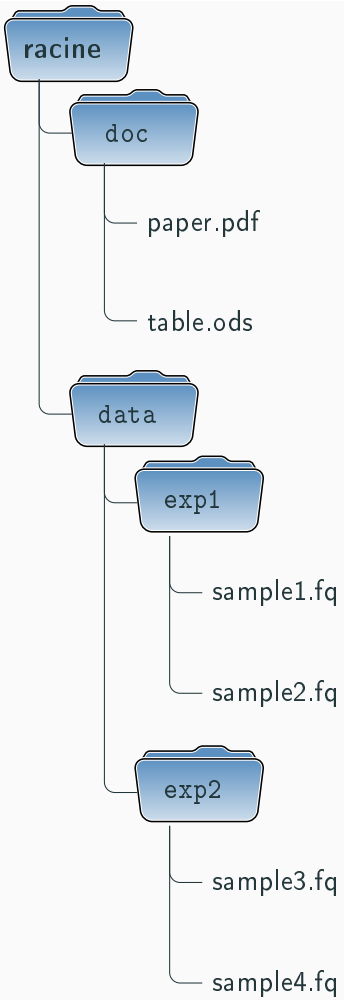
Arborescence et chemin



- Répertoire et fichiers forment un arbre (à l'envers).
- Le répertoire contenant l'ensemble des autres répertoires et fichiers est appelé la racine.
- Chaque fichier ou répertoire peut être identifié par la suite de répertoires menant de la racine à lui :

  `racine/data/exp1/sample2.fq`

Arborescence et chemin



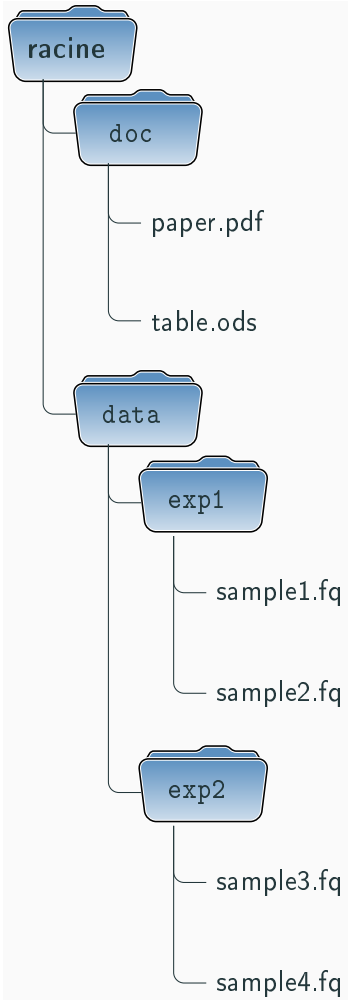
- Répertoire et fichiers forment un arbre (à l'envers).
- Le répertoire contenant l'ensemble des autres répertoires et fichiers est appelé la racine.
- Chaque fichier ou répertoire peut être identifié par la suite de répertoires menant de la racine à lui :

  `/data/exp1/sample2.fq`

Chemins absolus et relatifs (1/2)

- Un chemin à partir de la racine est dit *absolu*.
- Il commence nécessairement par / (*slash* ou barre oblique).
- Il est possible de désigner un objet à partir d'une autre position (dite position courante) en utilisant deux symboles :
  - . désigne le répertoire courant.
  - .. désigne le répertoire contenant le répertoire courant.

Chemins absolus et relatifs (2/2)



- Posons que le répertoire courant est : `/data/exp1`
- Dans ce cas, le chemin relatif vers : `/data/exp2/sample3.fq` est :

  est :

  `../exp2/sample3.fq`

Le terminal

- Programme permettant d'interagir avec le système.
- Il s'agit d'un interpréteur, *i.e.* un programme répétant la boucle :
  1. Lecture d'une commande.
  2. Exécution de la commande.
  3. Affichage des sorties de la commande.
- On tape la commande après l'invite.
- Envoi de la commande par ⏎

Premières commandes

`pwd` (*print working directory*)

Retourne la position courante du terminal dans l'arborescence

`ls` (*list segments*)

Affiche la liste des fichiers et répertoires présents dans le répertoire courant

Arguments

Le comportement de certaines commandes peut être modifié en ajoutant des paramètres (ou arguments) à la commande :

```
ls ./foo
```

Affiche le contenu du répertoire `./foo`

```
ls -l
```

Affiche la liste détaillée des fichiers présents dans le répertoire courant

```
ls -l ./foo
```

Affiche la liste détaillée du contenu du répertoire `./foo`

Manuel

Il est possible d'accéder à l'ensemble des paramètres d'une commande en utilisant une commande particulière :

`man` (*manual*)

```
man CMD
```

Exemple :

```
man ls
```

Et comme le dit le vieil adage informaticien, RTFM (*Read The F... Manual*) !

Exercice

- Ouvrir un terminal.
- Déterminer sa position courante.
- Trouver le fichier le plus récent du répertoire.
- Trouver le fichier de plus grande taille du répertoire.

Se déplacer dans l'arborescence

`cd` (*change directory*)

```
cd CHEMIN
```

Modifie le chemin courant du terminal

- Déplacement absolu : `cd /data/exp2`
- Déplacement relatif : `cd ../../doc`
- Que fait `cd`

Options pour `ls` (1/2)

`ls [OPTIONS] [CHEMIN]`

Affiche le nom des fichiers ou le contenu des répertoires

Arguments :

- `-l` affiche les détails du fichier
- `-a` affiche les fichiers débutant par `.` (fichiers cachés)
- `-h` affiche les tailles de fichiers
- `-R` lister récursivement les sous-répertoires
- `-t` trier du plus récent au plus ancien
- `-r` inverser le tri

Options pour `ls` (2/2)

- Les option de la forme `-x` sont appelés *interrupteurs*, ils permettent d'activer un comportement.
- Dans le cas de `ls` (et de nombreuses autres commandes), les interrupteurs peuvent être combinés :

  `ls -l -a -t -h -r` ≡ `ls -lathr`

Exercice

Tester chaque interrupteur de la commande `ls` et quelques combinaisons

Format d'un fichier

- On appelle extension d'un fichier le morceau de son nom à droite du *dernier* point.
- L'extension indique *conventionnellement* le format utilisé pour encoder le contenu :
  - Les fichiers n'ont pas nécessairement d'extension et rien ne dit que l'extension est bien cohérente avec le contenu réel du fichier.
- Pour obtenir l'information sur le type de contenu, il est possible d'utiliser la commande `file`.

`file`

```
file CHEMIN
```

Détermine le type de contenu et l'encodage d'un fichier

Inspecter le contenu d'un fichier (1/2)

cat

`cat CHEMIN`

Affiche le contenu d'un fichier à l'écran

Attention à ne pas utiliser `cat` sur de gros fichiers !

Inspecter le contenu d'un fichier (2/2)

`less CHEMIN`

Affiche de manière interactive le contenu d'un fichier à l'écran

Commandes (quand `less` est lancé) :

- `q` affiche les détails du fichier
- ↑ se déplacer d'une ligne vers le haut
- ↓ se déplacer d'une ligne vers le bas
- `ESPACE` se déplacer d'une page vers le bas
- `g` aller au début du fichier
- `G` aller à la fin du fichier
- `/` menu interactif de recherche
- `n` occurrence suivante du motif recherché
- `N` occurrence précédente du motif recherché

Afficher le début d'un fichier

head

```
head [OPTIONS] CHEMIN
```

Affiche les premières lignes d'un fichier

Arguments :

`-n ENTIER` nombre de lignes

Les arguments d'une commande (ici `-n`) peuvent avoir une valeur associée (dans ce cas un entier).

Afficher la fin d'un fichier

tail

```
tail [OPTIONS] CHEMIN
```

Affiche les dernières lignes d'un fichier

Arguments :

`-n ENTIER` nombre de lignes

Comparer deux fichiers

diff

```
diff -u FICHIER1 FICHIER2
```

Affiche un alignement de deux fichiers pour repérer les différences

Créer

touch

```
touch CHEMIN
```

Crée un fichier vide

mkdir

```
mkdir [OPTIONS] CHEMIN
```

Crée un répertoire vide

Argument :

`-p` crée les répertoires parents si nécessaire

Détruire

rmdir

```
rmdir CHEMIN
```

Supprime un répertoire *vide*

rm

```
rm [OPTIONS] CHEMIN
```

Supprime un fichier

Argument :

`-r` efface récursivement les fichiers d'un répertoire

Attention, sous Unix la suppression est définitive !

Copier

cp

```
cp [OPTIONS] SOURCE DESTINATION
```

Copie des fichiers/répertoires

Arguments :

- `-r` déplace aussi les répertoires
- `-i` demande confirmation avant d'écraser

scp

```
cp [OPTIONS] SOURCE DESTINATION
```

Copie des fichiers/répertoires au travers du réseau (d'une machine à une autre)

Déplacer/renommer

Sous Unix, il n'y a pas de différences entre déplacer et renommer un fichier.

mv

```
mv [OPTIONS] SOURCE DESTINATION
```

Renommer un fichier ou un répertoire

Arguments :

`-i` demande confirmation avant d'écraser

Si la destination existe déjà et est un répertoire, la source y est déplacée.

Exercice

- A l'aide de `cp`, faites une copie du fichier `.bash_history`
- A l'aide de `cd`, `ls` et `file`, trouvez un maximum de formats présents sur votre machine.
- Ouvrez le fichier `.bash_history` avec `less` et essayez chacune des commandes présentées.
- Essayez les commandes `head` et `tail` sur le fichier `.bash_history`
- A l'aide de `diff`, comparez votre copie de `.bash_history` et sa version actuelle.

Ergonomie

- Rappel des commandes précédentes ($\Uparrow$ et $\Downarrow$) :
  - Notion d'historique des commandes.
- Complétion des commandes et de leurs arguments ($\longrightarrow\!\!|$).
- Raccourcis clavier :
  - `ctrl-A` aller en début de ligne.
  - `ctrl-E` aller en fin de ligne.
  - `ctrl-K` couper la commande à droite du curseur.
  - `ctrl-U` effacer toute la ligne.
  - `ctrl-R` recherche dans l'historique.

Joker

Le symbole `*` est interprété par le terminal pour désigner un ensemble de chemins :

- `ls doc.pdf` seulement `doc.pdf`
- `ls *` tous les fichiers du répertoire courant
- `ls *.pdf` tous les fichiers dont le nom se termine par `.pdf`
- `ls d*.pdf` idem + le nom commence par `d`

Cas typique d'usage, déplacer un ensemble de fichiers :

```
mv *.pdf ./vers/ailleurs
```

Notion de processus

Définition

Représentation d'un programme en cours d'exécution

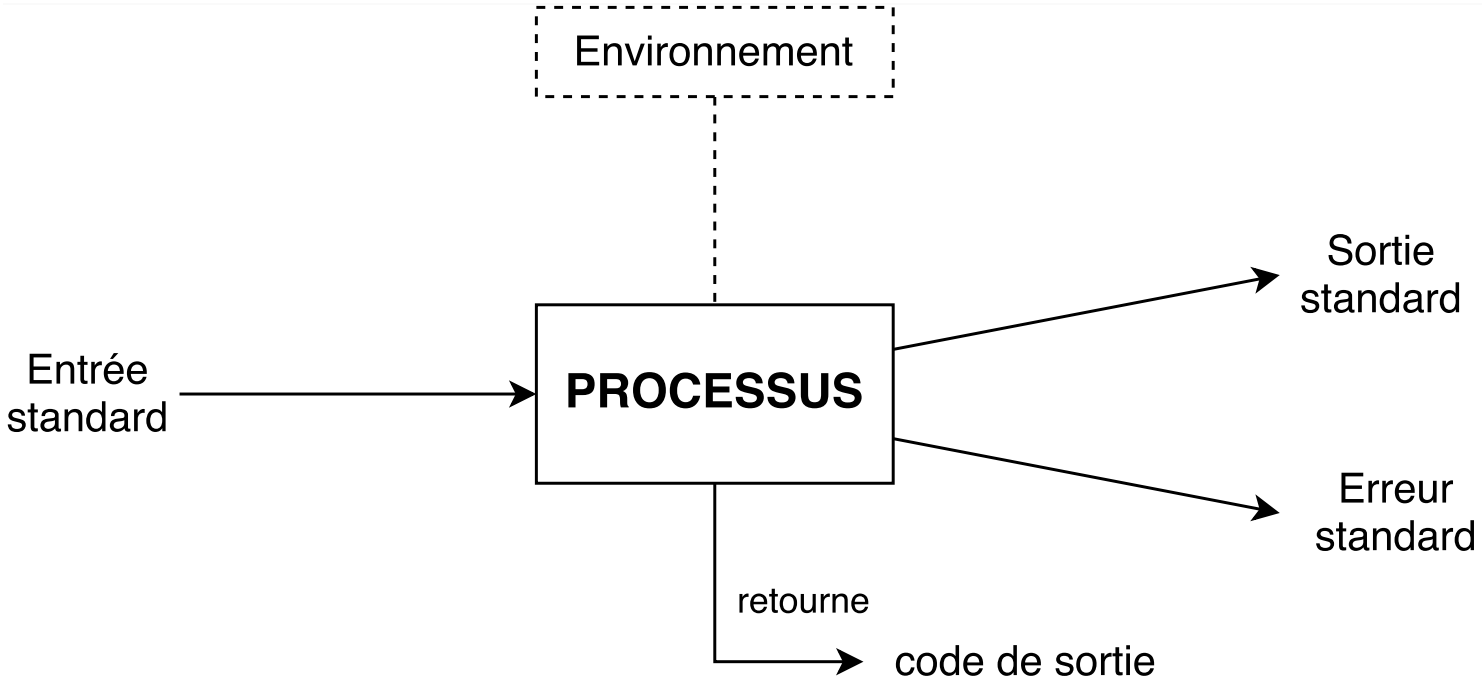
Un *processus* est constitué :

- D'un ensemble d'*instructions* interprétables par le processeur.
- D'un *espace mémoire* attribué (pour stocker son état d'exécution).
- De *ressources* allouées (fichiers ouverts, ports réseaux).

Il faut bien distinguer programme de processus !

Interaction avec le système

Les processus ont de nombreux moyens de recevoir des informations du système ou lui transmettre :



et il y en a d'autres...

Entrée standard

- Source de données par défaut pour le processus.
- Se présente comme un fichier depuis lequel on peut lire des caractères.
- La fin de l'entrée est signalée par un caractère spécial, appelé `EOF` (*End Of File*).
- Il est possible de produire `EOF` sur le terminal avec la combinaison `ctrl-D`.
- Comme tout fichier, l'entrée standard peut représenter des sources variées (fichier, connexion réseau, ...)
- Par défaut, elle est connectée à l'entrée clavier.

Sortie standard

- Destination par défaut des résultats du processus.
- Se présente aussi comme un fichier dans lequel on peut écrire.
- De même, la sortie standard peut représenter des destinations variées.
- Par défaut, la sortie standard est affichée par le terminal.

Sortie d'erreur

- Destination par défaut pour rapporter des problèmes à l'exécution.
- Se présente aussi comme un fichier dans lequel on peut écrire.
- Par défaut la sortie d'erreur est affichée dans le terminal.
- Les deux sorties sont donc mélangées dans le terminal :
  - Possibilité (souvent utile) de les envoyer à des destinations distinctes.

Les redirections

Définition

Mécanisme et syntaxe par lesquels il est possible d'associer de nouvelles sources/destinations aux flux d'un processus

Avantage :

- Le programme n'a pas besoin de savoir précisément la nature des sources et destinations :
  - Il est donc plus facilement réutilisable dans des contextes variés.

Rediriger la sortie standard

Syntaxe :

`CMD > CHEMIN` (écrase la destination)

ou :

`CMD >> CHEMIN` (ajoute en fin de fichier)

Exemple :

```
ls -l > liste.txt
cat liste.txt
```

Rediriger l'entrée standard

C'est une façon de simuler une entrée au clavier en utilisant le contenu d'un fichier.

Syntaxe :

```
CMD < CHEMIN
```

Exemples :

```
echo "1 + 1" > script.R
```

```
R -vanilla < script.R
```

Rediriger la sortie d'erreur

Syntaxe :

`CMD 2> CHEMIN` (écrase la destination)

ou :

`CMD 2>> CHEMIN` (ajoute en fin de fichier)

Pour fusionner les deux sorties :

`CMD &> CHEMIN` (écrase la destination)

ou :

`CMD &>> CHEMIN` (ajoute en fin de fichier)

Notion d'environnement

Définition

Collection de variables transmises au processus au démarrage pour influer son comportement, de la forme :

```
HOME=/home/perriere
LANG=fr
SHELL=/bin/bash
```

- Le processus peut les modifier, en ajouter ou en enlever.
- Si un processus en crée un autre, il lui transmet son environnement.

Variable d'environnement dans le terminal

- Comme tout processus le terminal a aussi un environnement :
  - Affichage au moyen de la commande `env`.
  - Affichage du contenu d'une variable au moyen de la commande `echo`, par exemple :
    `echo $HOME`
- Il est possible d'utiliser les variables dans une commande :

  `ls ${HOME}`

  ou plus simplement :

  `ls $HOME`
- Il est possible de modifier (ou créer) une variable avec la commande `export` :

  `export X=[...]`

Exercice

- Expliquez à quoi sert la commande `echo` ?
- Affichez l'environnement de votre terminal et essayez d'interpréter certaines variables.
- Modifiez une variable et vérifiez que la valeur a bien été changée.

La variable PATH

- Les commandes du terminal sont bien souvent des programmes, présents sous la forme de fichiers :

  `ls` $\longrightarrow$ `/bin/ls`

  `cp` $\longrightarrow$ `/bin/cp`

  `firefox` $\longrightarrow$ `/usr/bin/firefox`
- Lorsqu'on invoque une commande, l'interpréteur doit retrouver le bon exécutable :
  - Recherche dans une liste de répertoires, codée par la variable d'environnement `PATH`.

Syntaxe pour PATH

- Les répertoires sont explorés dans l'ordre.
- Le premier fichier exécutable avec le nom recherché est sélectionné.
- Pour savoir quel est le choix effectué par l'interpréteur il est possible d'utiliser la commande `which`.
- Modification par `export` :

```
export PATH=REPERTOIRE(:REPERTOIRE)
```

Exercice

- Déterminez l'emplacement de l'exécutable de la commande `ls`.
- Copier la valeur de `PATH` dans une variable `PATH_BCK`.
- Entrez la commande :

  `export PATH=`

  Quel en est l'effet ?
- Trouvez un moyen de lancer `ls`.

  ...puis de rétablir un fonctionnement normal.
- Comparez la sortie de `ls` dans les deux cas.

Aparté : les alias

- Les commandes que l'on veut exécuter souvent doivent être courtes (histoire de gagner du temps).
- Si ce n'est pas le cas à cause de nombreuses options, il est possible de définir un *alias* :

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

```
alias gt="git log -graph -abbrev-commit -decorate
-date=relative -all"
```

- Dans toute commande commençant par l'alias, celui-ci est remplacé par sa valeur :

`ls -la` $\longrightarrow$ `ls -color=auto -la`

- Il est possible d'afficher la liste de tous les alias au moyen de la commande `alias`.

Code de sortie

- Quand un programme arrive en fin d'exécution, il doit fournir un nombre entier appelé *code de sortie*.
- Par convention, la valeur 0 signale l'absence d'erreur.
- Une valeur différente de 0 signale une erreur dont l'interprétation dépend du programme.
- Une variable spéciale, `$?`, permet d'accéder au code de retour de la dernière commande entrée au terminal.

Séquence de commandes

Il est possible de construire des commandes en en combinant plusieurs :

`CMD1; CMD2` exécute `CMD1` puis `CMD2`

`CMD1 && CMD2` exécute `CMD1` puis `CMD2` si `CMD1` a terminé sans erreur

`CMD1 || CMD2` exécute `CMD1` puis `CMD2` si `CMD1` a échoué

Usage des connecteurs de séquence

- S'assurer que la commande précédente a bien fonctionné avant de lancer la suivante, ainsi :

  ```
  cd foo && rm *
  ```

  est beaucoup plus sûr que :

  ```
  cd foo; rm *
  ```

- Actions conditionnelles :

  ```
  wget "http://..." || (echo "Download failed" ; exit 2)
  ```

Les tubes (*a.k.a. pipes*)

Principe

Connecter la sortie standard d'un processus à l'entrée standard d'un autre

Avantages :

- Pas de fichier intermédiaire à nommer puis effacer.
- Plus efficace (pas d'écriture sur disque, parallélisme).

Syntaxe :

```
CMD1 | CMD2 | ...| CMDn
```

Exemple :

```
ls -l | wc -l
```