L'environnement UNIX/Linux: un système d'exploitation pour la bioinformatique

D. Puthier[1]

[1] http://pedagogix-tagc.univ-mrs.fr/courses/jgb53d-bd-prog/
Inserm U1090
Technologies Avancées pour le Génome et la Clinique

PLAN

Pourquoi UNIX/Linux ?

Architecture d'un ordinateur

- Composants d'un ordinateur
  - Un micro-processeur (CPU, Central Process Unit) : pour le traitement des données
  - De la mémoire RAM (Random Access Memory, mémoire à accès aléatoire) : stockage temporaire de l'information (données, programmes).
  - Une mémoire morte (ROM ou Read-Only Memory) : mémoire non volatile dans laquelle est stocké le BIOS (Basic Input Output System, permet le contrôle et l'intitialisation/contrôle des composants au démarrage).
  - Des périphériques (disque dur pour stocker les programmes et les données, carte graphique,...).
  - Des bus qui relient les éléments.

Qu'est ce qu'un système d'exploitation ?

- Assure les communications entre les ressources matérielles et les applications diverses (explorateur, traitement de texte,...).
  - Programme -> requête vers OS (Operating system / système d'exploitation) -> pilote -> matériel.
  - Limite la redondance (sinon chaque application devrait assurer l'interface avec le matériel)
- Exemple de systèmes d'exploitation : Windows, Mac OS (Basé sur un système UNIX), Linux (Ubuntu, Debian, Opensuse...), Android (basé sur Linux), Chrome OS (basé sur Linux)...

Historique

- Quelques repères chronologiques :
  - 1969 : Ken Thomson et Dennis Ritchie (Bell Labs AT&T) développent UNIX.
  - 1973 : 1ère version d'UNIX en langage C.
  - 1978 : Unix V7 (officielle).
  - 1991 : Freax (Linus Torvalds)
  - 1994 : Linux V1.0 (Intégre le noyau développé par Linus Torvalds et les outils GNU dévéloppés par Richard matthew Stallman).
  - 1996 : début du projet KDE d'interface graphique
  - 1997 : début du projet GNOME comme projet concurrent de KDE

- Constituants des systèmes Unix/Linux :
  - le noyau (”kernel“) :
    - Gestion des processus (programmes).
    - Gestion de la mémoire
    - Gestion des entrées-sorties
    - Communication avec le matériel
  - La coquille (”shell“) : permet de donner des instructions au système d’exploitation.
  - Le système de fichiers (FS, ”File System“) : permet de stocker et d’organiser les fichiers.

Pourquoi utiliser Linux en biologie ?

- Nécessité de manipuler des fichiers nombreux et volumineux.
  - Comment ouvrir de tels fichiers sous Windows ?
  - Comment réaliser des actions simples sur ces fichiers (compter, filter, trier, ...)
  - Comment itérer sur ces fichiers avec Windows ?
- Nécéssité de réaliser des calculs nécessitant des serveurs puissants accessibles à distance.
  - Système multi-utilisateurs,
    - Utilisateur standard (droits réduits)
    - Super-utilisateur (root) : tous les droits sur la machine.
    - Groupes (partage).
  - Nombreux outils réseaux
    - Nombreux outils de communication en réseau : FTP (File Transfert Protocol), SSH (Secure Shell), NFS (Network File System NFS), ...

Pourquoi utiliser Linux en biologie ?

- De nombreux logiciels développés spécifiquement pour les systèmes UNIX/Linux.
- Nécessité de réaliser des “pipelines de traitement”.
  - Possibilité de piloter la machine à travers des commandes unitaires pouvant être chaînées.
    - Création de scripts réalisant des tâches complexes.
- UNIX/Linux est le meilleur équipement pour la Bioinformatique (et bien d’autres choses)...
  - Believe me...

Si vous souhaitez installer Linux

- De nombreuses “Distributions” disponibles (Ubuntu, Debian, RedHat, Opensuse, Mint...).
  - Ubuntu : solution facile à prendre en main (de même que Mint).
  - http ://www.ubuntu.com/.
  - Solution 1 : graver le CD, l’insérer, partitionner le disque* (-> dual boot) *
  - Solution 2 : graver le CD. Utiliser VirtualBox (”virtualisation“, possibilité de lancer le système ubuntu depuis windows).

* Préférable de faire une partition ”/“ et une partition ”/home“.

Notions de base pour l'utilisation du shell

Quelques bonnes habitudes

Remarque sur les noms des répertoires et fichiers :

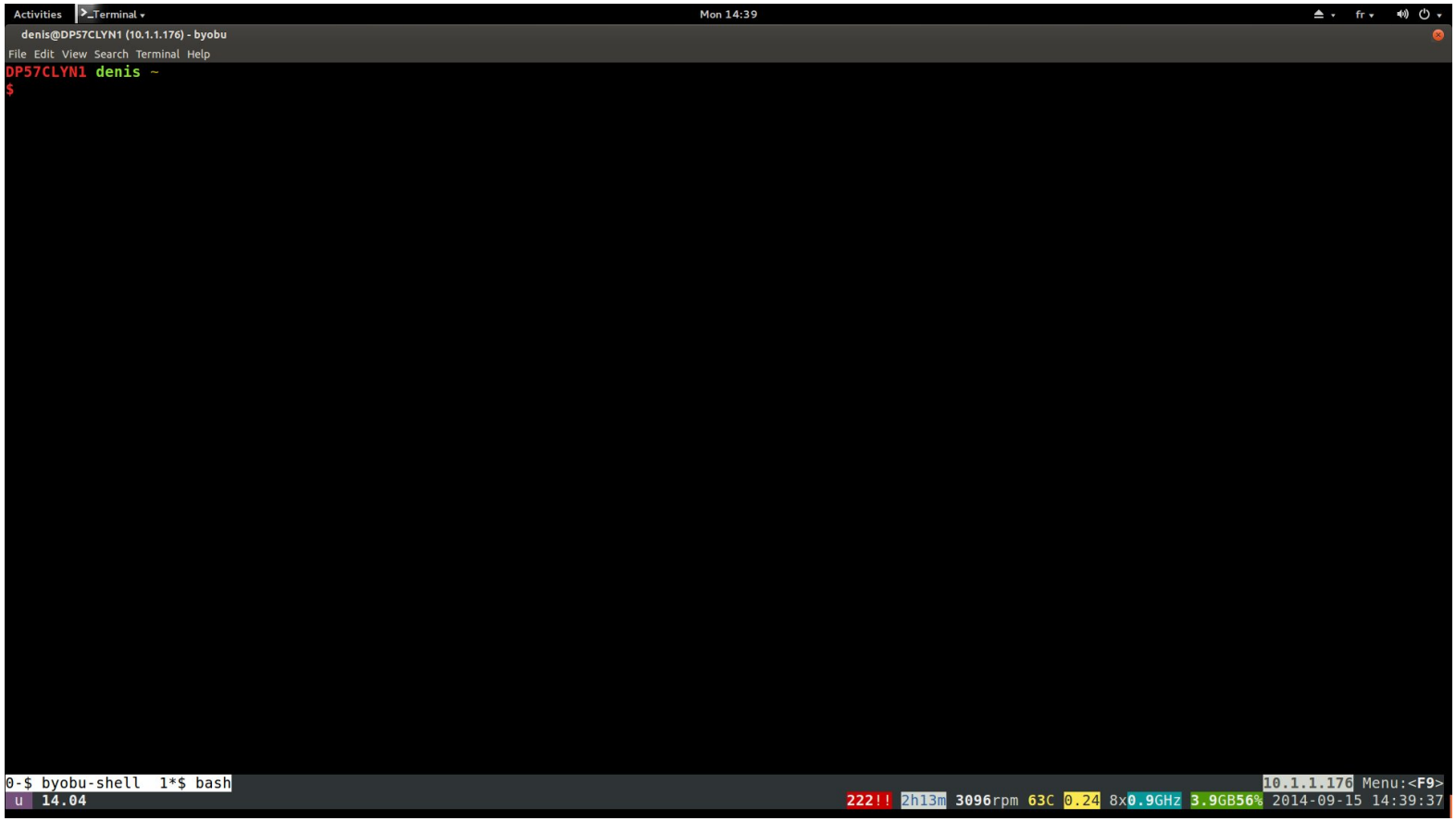
- Uniquement des caractères alphanumériques et les caractères '-' et '_'
  - Pas d'espaces dans les noms. Eviter les caractères accentués et les caractères spéciaux.
- Mettre une extension aux noms de fichiers permettant de deviner leurs types
  - .txt pour un fichier de texte
  - .sh pour un script shell
  - .py pour un script Python
  - ...
- Attention : distinction entre les lettres majuscules et les lettres minuscules (sensible à la casse).

Quelques bonnes habitudes

Remarque sur les noms des répertoires et fichiers :

- Uniquement des caractères alphanumériques et les caractères '-' et '_'
  - Pas d'espaces dans les noms. Eviter les caractères accentués et les caractères spéciaux.
- Mettre une extension aux noms de fichiers permettant de deviner leurs types
  - .txt pour un fichier de texte
  - .sh pour un script shell
  - .py pour un script Python
  - ...
- Attention : distinction entre les lettres majuscules et les lettres minuscules (sensible à la casse).

Le terminal : Arghhh !!!



Aide sur les commandes

Toutes les commandes unix, sont bien documentées. L'accès à cette documentation se fait à l'aide de la commande **man** (**man**ual).

```
[puthier@mamachine] man tail

TAIL(1)                          Manuel de l utilisateur Linux                                    TAIL(1)

NOM
       tail - Afficher la dernière partie d'un fichier.

SYNOPSIS
       tail  [-c  [+]N[bkm]] [-n [+]N] [-fqv] [--bytes=[+]N[bkm]] [--lines=[+]N] [--follow] [--quiet] [--silent] [--ve
       [--version] [fichier...]
       tail [{-,+}Nbcfklmqv] [fichier...]

DESCRIPTION
       Cette page de manuel documente la version GNU de tail ([NDT] tail = queue).
                          ...
```

On trouvera dans l'aide une section "SYNOPSIS" –> vue d'ensemble des options du programme. Dans cette section, le principe pour les arguments est le suivant :

- Tout ce qui se trouve entre crochets est facultatif. Ce qui n'est pas entre crochet est nécessaire.
- Tout ce qui se trouve entre accolade correspond à un choix (souvent exclusif).

La rubrique "OPTIONS" explique l'influence de chacune des options sur le déroulement du programme.

- Pour effectuer une recherche dans l'aide on tapera la chaine de caractère recherchée précédée du caractère */*
  (ex : */OPTIONS* pour rechercher le terme "OPTIONS".
- Pour aller à la prochaine occurrence on utilisera la touche "n" (**n**ext) pour se rendre à l'occurrence précédente on utilisera "<shift> + n".
- On utilisera "<ctrl> + <" et "<ctrl> + <shift>+<" pour se rendre à la fin et au début du fichier d'aide respectivement.
- On utilisera "q" pour quitter l'aide (**q**uit).

Recherche de termes à travers les fichiers d'aide.

On pourra utiliser la commande **man** avec l'option "-k". On recherchera alors l'occurrence d'une chaîne de caractères donnée dans tous les paragraphes "description" des fichiers d'aide. On obtiendrait le même résultat avec la commande **apropos**

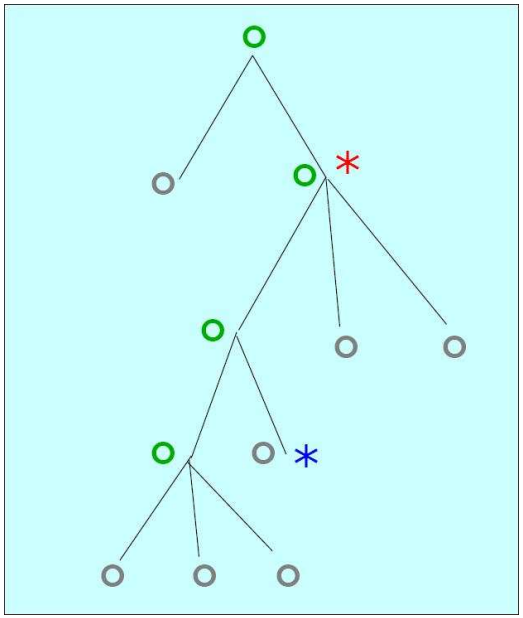
```
[puthier@mamachine] man -k jpeg
```

```
jpeg2ktopam (1)      - convert JPEG-2000 code stream to PAM/PNM
jpeg2yuv (1)         - Convert jpeg images to the yuv format.
jpegicc (1)          - little cms ICC profile applier for JPEG.
jpegtopnm (1)        - convert JPEG/JFIF file to PPM or PGM image
jpegtran (1)         - lossless transformation of JPEG files
lav2wav (1)          - Extract the audio out of MJPEG container files to stdout
lav2yuv (1)          - Convert a MJPEG file to raw yuv
```

Fichiers et répertoires

Organisation des fichiers sous Linux

- les objets (fichiers, répertoires) sont organisés en arborescence.
  - chaque objet est désigné par un chemin d'accès.
  - 2 éléments à connaitre : votre position, la position du fichier/répertoire d'intérêt.



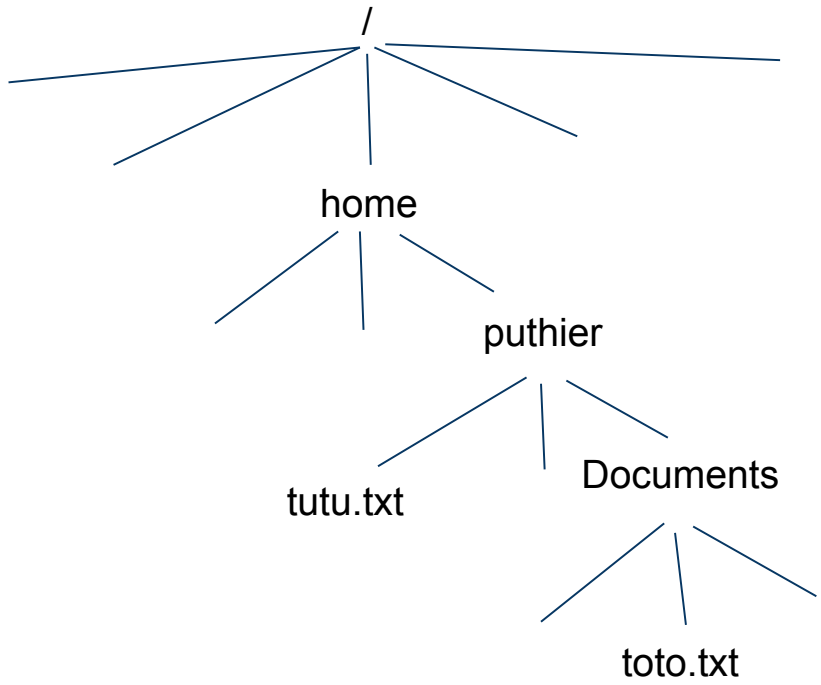
- Le repertoire “racine” est désigné par “/”.
  - –> $\sim$ C :\ sous windows
  - contient un certain nombre de sous répertoires (/bin, /boot, ..., /var).

/bin Programmes système (**bin**aries).
/boot Noyau, Bootmanager.
/dev Fichiers des périphériques (**dev**ices).
/etc Fichiers de configuration.
/home Répertoires des utilisateurs.
/lib Librairies partagées.
/mnt Répertoire de montage pour cdrom, floppy... (**m**ou**nt**).
/opt Installations supplémentaires.
/proc Informations sur le système et les processus en cours (**proc**ess).
/root Répertoire personnel de **root**.
/sbin Programmes système pour le root.
/tmp Données **tem**poraires.
/usr Programmes des utilisateurs.
/var Fichiers divers et certains fichiers de logs (**var**iable)

- Le répertoire “home”.
  - Contient les dossiers de travail et de configuration de chacun des utilisateurs /home/puthier, /home/dupont, /home/duchsmock, ...
  - Il peut être symbolisé par $\sim$:
    Si je suis connecté en tant que “puthier” :
    $\sim$ == /home/puthier
    Si je suis connecté en tant que “martin” :
    $\sim$ == /home/martin

Chemins relatifs et absolus

- Chemin absolu : se réfère à la racine “/”.
- Chemin relatif : se réfère au répertoire courant.
- Ex : On se trouve dans le répertoire “Document”. On désigne le fichier “toto.txt”
  - chemin relatif au répertoire “Document” : toto.txt
  - chemin absolu du fichier toto.txt : /home/puthier/Documents/toto.txt


/
home
puthier
tutu.txt
Documents
toto.txt

- En écriture relative “..” signifie “répertoire supérieur”
- Ex : On se trouve dans le répertoire “Document”. On désigne le fichier “tutu.txt”
  - chemin relatif au fichier tutu.txt :
    ../tutu.txt
  - chemin absolu du fichier tutu.txt :
    /home/puthier/tutu.txt

- En ecriture relative “./” signifie “le répertoire courant”
- Ex : On se trouve dans le répertoire “Document”.
  ./toto.txt <=> toto.txt

Opérations sur les dossiers

- La commande “cd” (**c**hange **d**irectory)
- Ex 1 : On se trouve dans Document. On souhaite se rendre dans “/home/puthier”
  - En absolue :

    ```
    cd /home/puthier
    ```

    ou

    ```
    cd ~
    ```
  - En relatif :

    ```
    cd ..
    ```

- Ex 2 : On se trouve dans “Document”. On souhaite se rendre à la racine
  - En absolue :

    ```
    cd /
    ```

  - En relatif :

    ```
    cd ../../..
    ```

- **NB** : Pensez toujours à utiliser la “complétion” (touche tabulation)

- **pwd**, **p**rint **w**orking **d**irectory (afficher le répertoire courant)
  Ex : on se trouve dans “Document”

```
[puthier@mamachine] pwd
 /home/puthier/Documents
```

- **ls**, **ls**t (lister les fichiers et dossiers d’un répertoire)
  Ex : on se trouve dans “Document”

```
[puthier@mamachine] ls
 toto.txt
[puthier@mamachine] ls  /home/puthier
 tutu.txt
[puthier@mamachine] ls  ~
 tutu.txt
```

- Les options de **ls**
  - ls -l : (**l**ong) affiche les droits, les tailles et les dates de création/modification des fichiers et répertoires
  - ls -a : (**a**ll) affiche tous les fichiers et répertoires, même les fichiers/dossiers cachés (leurs noms commencent par ".")
  - ls -R : (**r**ecursive) affiche tous les fichiers et le contenu des dossiers.
  - ls –sort=time : trie les fichiers par date de création
  - ls –sort=size : trie les fichiers par tailles
  - ls -1 : présente les nom des fichiers/dossiers en une seule colonne
- On peut combiner les options des commmandes :

```
[puthier@mamachine] ls -la /home/puthier/
```

- **mkdir**, **m**a**k**e **d**irectory (créer un répertoire)
  Ex : on se trouve dans “Document”

```
[puthier@mamachine] mkdir analyses
[puthier@mamachine] ls
 analyses tutu.txt
```

- **rmdir**, **rem**ove **d**irectory (déléter un répertoire)
  Ex : on se trouve dans “Document”

```
[puthier@mamachine] rmdir analyses
[puthier@mamachine] ls
 tutu.txt
```

- **rmdir** équivaut à “rm -Rf analyses”. NE PAS UTILISER CETTE COMMANDE !!!!

NB : Les caractères “**joker**”. Permettent de désigner un ensemble de fichiers

- ”?“ : Désigne un caractére quelconque (présent).
- ”*“ : Désigne un ensemble de caractères quelconques (présents ou absents).

```
[puthier@mamachine] ls
 file12.txt  file1.txt  file2.txt  file.txt f.sh

[puthier@mamachine] ls file*
 file12.txt  file1.txt  file2.txt  file.txt

[puthier@mamachine] ls file?.txt
 file1.txt  file2.txt
```

- **du** (**d**isk usage) afficher la taille totale des fichiers contenus dans un répertoire :

```
[puthier@mamachine] du -sh /home/puthier/
 824M    /home/puthier/
```

- Options :
  du -h : (**h**uman readable) (affiche la taille en ko, Mo, Go)
  du -s : (**s**um) (affiche le somme de tous les fichiers et sous répertoires)

Opérations sur les fichiers

- **cat** : afficher le contenu d'un fichier :

```
[puthier@mamachine] cat /home/puthier/tutu.txt
```

- **less/more** : afficher le contenu d'un fichier page par page :

```
[puthier@mamachine] less /home/puthier/tutu.txt
```

On utilisera "<ctrl> + <" et "<ctrl> + <shift>+<" pour se rendre à la fin et au début du fichier d'aide respectivement. On utilisera "q" pour quitter l'aide (**q**uit).

```
[puthier@mamachine] more /home/puthier/tutu.txt
```

Opérations sur les fichiers

- **NB** : Sous Unix et windows le retour à la ligne est codé différemment dans les fichiers :

```
[puthier@mamachine] od -c exemple.unix.txt
0000000   A  \n   B  \n   C  \n
0000006

[puthier@mamachine] od -c exemple.win.txt
0000000   A  \r  \n   B  \r  \n   C  \r  \n
0000012
```

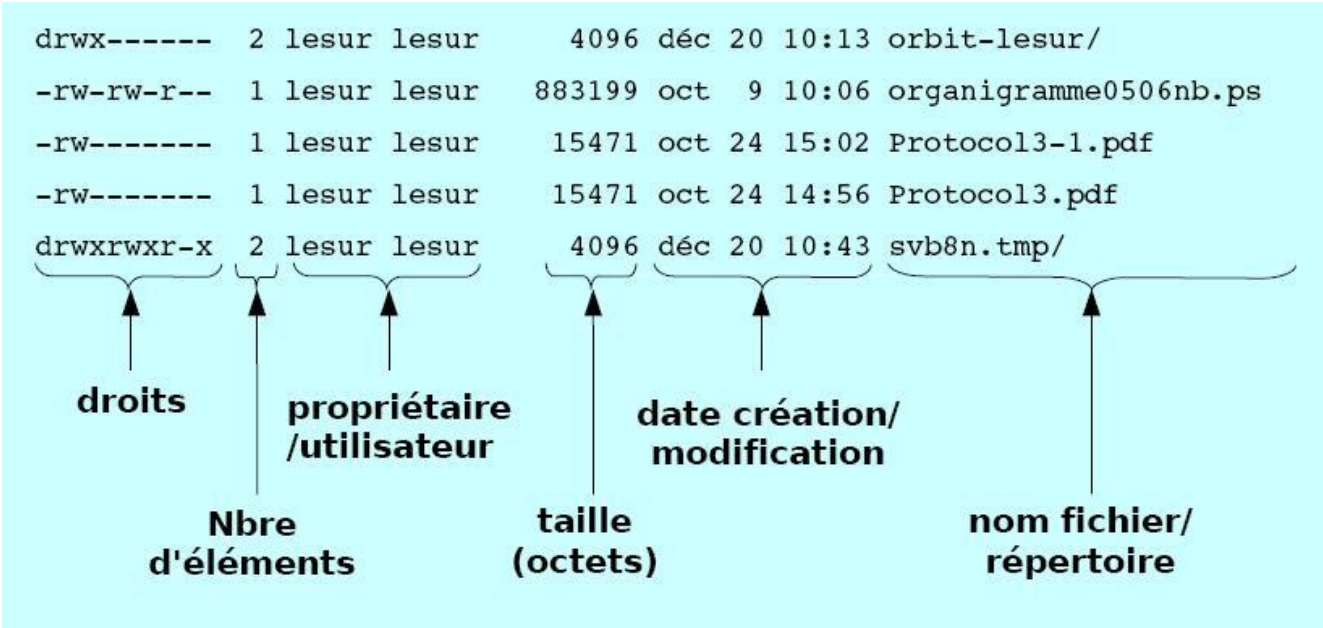
- Connaître les droits des fichiers, leurs tailles ... :

```
[puthier@mamachine] ls -l /home/puthier/
```

- chmod, (**ch**ange **mod**e) changer les droits d'un fichier :

  `chmod [ugoa] {+,-,=} [rwx] <Fichier>`

- **u**ser, **g**roup, **o**ther, **a**ll
- **r**ead, **w**rite, e**x**ecute



- Ex : fichier week.html

```
--> -rw-r--r-- 1 puthier users  16K week.html
```

- Enlever les droits de lecture sur le fichier week.html pour tous sauf l'utilisateur et le groupe :

```
[puthier@mamachine] chmod o-r week.html
-rw-r----- 1 puthier users 16K week.html
```

- Ajouter les droits de lecture/écriture pour tous :

```
[puthier@mamachine] chmod a+rw week.html
-rw-rw-rw- 1 puthier users 16K week.html
```

- la commande **wc** (**w**ord **c**ount) affiche 3 valeurs :
  le nombre de lignes
  le nombre de mots (séparés par des blancs)
  le nombre d'octets

```
[puthier@mamachine] wc GSE7671_family.soft
 7899  60663 461473 GSE7671_family.soft
```

- connaître le nombre de lignes dans un fichier :

```
[puthier@mamachine] wc -l GSE7671_family.soft
 7889
```

- la commande **tail**. Affiche les n dernières lignes d'un fichier.

```
[puthier@mamachine] tail -4 GSE7671_family.soft
 376      -1.02     15393
 377      NULL      NULL
 378      NULL      NULL
 379      0.008       1111
```

- la commande **head**. Affiche les n premières lignes d'un fichier.

```
[puthier@mamachine] head -4 GSE7671_family.soft
 ^DATABASE
 !Database_name
 !Database_institute
 !Database_web_link
```

- Copier un fichier :

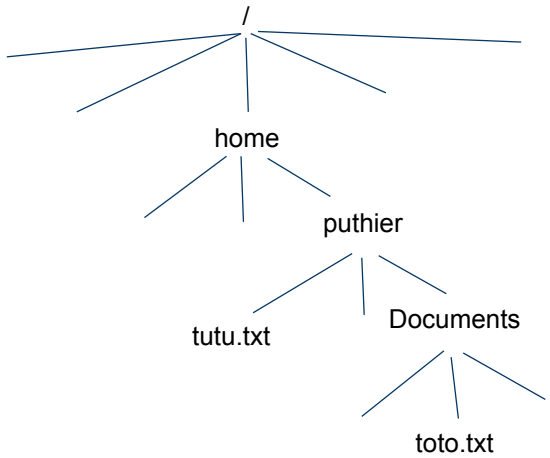
```
cp <origine> <destination>
```

- Exemple (le fichier garde le nom toto.txt) :

```
cp toto.txt ..
```

- Exemple (copie avec changement de nom) :

```
cp toto.txt ../titi.txt
```



- **mv** : (**mov**e), déplacer un fichier :

  ```
  mv <origine> <destination>
  ```

- Exemple (le fichier est déplacé et conserve son nom original) :

  ```
  mv /home/puthier/Documents/toto.txt /tmp
  ```

- Exemple (le fichier est deplacé et renommé) :

  ```
  mv /home/puthier/Documents/toto.txt /tmp/f.txt
  ```

- **mv** est utilisé pour renommer les fichiers

  ```
  mv file.old.txt file.new.txt
  ```

La fonction **cut** :

- Pour les fichiers contenant plusieurs colonnes, il est utile de pouvoir extraire certaines d'entre elles.

  ```
  cut [-fd] <fichier>
  ```

- -**f**=**f**ield, -**d**=separator.
- Exemple :

  ```
  [puthier@mamachine] cut -f3,5 fichier.txt
  ```

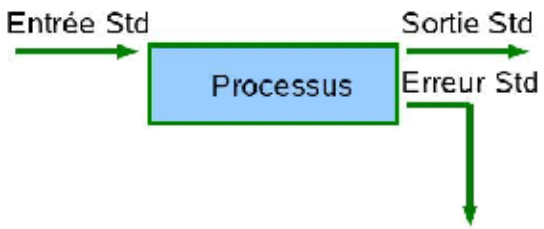
  Extrait la 3ième et la 5ième colonnes du fichier "fichier.txt". Par défaut, le séparateur de colonne est une tabulation.

Redirection

Entrées-Sorties (E\S) d'un processus.

Dans un processus les flux E\S sont au nombre de trois.

- L'**entrée standard** est le flux d'entrée par lequel du texte ou toute autre donnée peut être entré dans un programme.
- La **sortie standard** est le flux de sortie dans lequel les données sont écrites par le programme. Les données sont habituellement écrites à l'écran.
- L'**erreur standard** est le flux de sortie permettant aux programmes d'émettre des messages d'erreur et des diagnostics.



Les opérateurs de redirection

- On utilise des opérateurs pour rediriger un fichier vers l'entrée d'un processus ou rediriger les sorties d'un processus.
  - "$<$" Suivi du nom du fichier indique sa redirection vers un processus donné.
  - "$>$" : redirection de la sortie standard d'un processus vers un fichier (celui-ci est écrasé).
  - "$>>$" : redirection de la sortie standard d'un processus vers un fichier (ajout)
  - "2 $>$" : redirection de l'erreur standard vers un fichier.

- Exemple 1 : les noms des fichiers présents dans “/tmp” sont stockés dans le fichier “tmp.txt”

```
[puthier@mamachine] ls /tmp > tmp.txt
```

- Exemple 2 : On ajoute la liste des fichiers du dossier “/home/puthier”

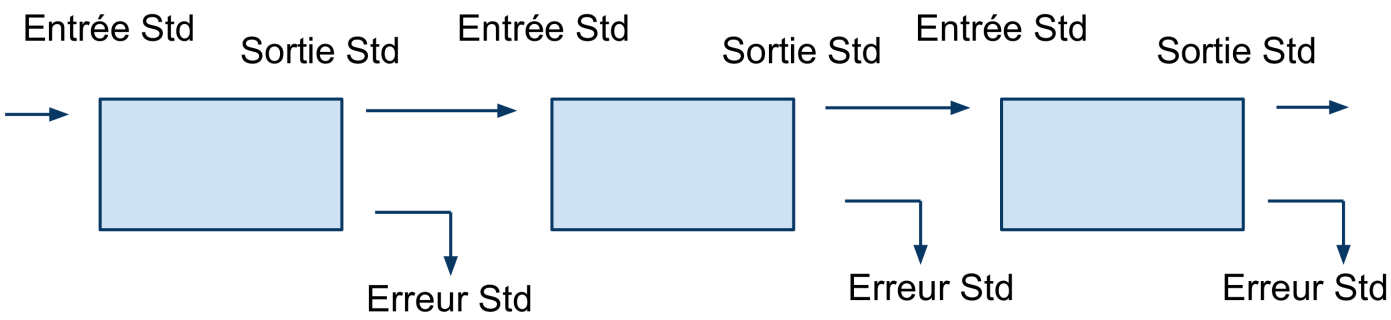
```
[puthier@mamachine] ls /home/puthier >> tmp.txt
```

- Exemple 3 : On concatène des fichiers.

```
[puthier@mamachine]  echo "123" > f1.txt
[puthier@mamachine]  echo "456" > f2.txt
[puthier@mamachine]  cat f*.txt >> result.txt
[puthier@mamachine]  cat result.txt
123
456
```

Les tubes (pipes)

- Les processus générés par les commandes peuvent s'enchaîner.
- Un tel enchainement est symbolisé “par le caractère ”|“ (tube ou ”pipe“).



Exemple d'enchaînement :

```
[puthier@mamachine] cat file_test.txt
 Martine
 Alain
 Julien
 Aline
 Aline
 Robert

[puthier@mamachine] cat file_test.txt | sort | uniq
 Alain
 Aline
 Julien
 Martine
 Robert
```

- Exemple d'enchaînement :

```
[puthier@mamachine]  cat test.txt
Martine
Alain
Julien
Aline
Aline
Robert

[puthier@mamachine] grep -E "^[AM].*e$" test.txt | head -2
Martine
Aline
```

Expressions régulières

Définition

- Elles permettent de décrire une motif au sein d'une chaîne de caractères.

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

- Exemples

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

Les filtres

Définition

- Un filtre est une commande capable de lire un flux sur son entrée standard, d'effectuer un traitement et d'écrire le résultat sur sa sortie standard. On peut les enchaîner avec des "tubes".
- Exemple de filtres :
  - **grep**, **sort**, **tr**, **sed**, **wc**, **head**, **tail**, **paste**, **awk**, **(perl)**, ...

La commande grep

La commande **grep** : (**g**eneral **r**egular **e**xpression **p**rocessor) permet de filtrer l'entrée via l'utilisation d'une expression regulière. Son entrée peut être un fichier ou un flux.

- L'option **-E** permet l'utilisation d'expressions réguliéres étendues (ERE).
- L'option **-E** permet l'utilisation d'expressions réguliéres compatibles avec Perl (PCRE).

```
[puthier@mamachine] echo -e "456\n567\n775" |grep -P "^.
456
```

La commande **grep** (suite).

- A noter l'option **-r** (**r**ecursive), permet de rechercher un terme en mode récursif.

```
[puthier@mamachine]  grep -r "toto" ./*
```

- A noter l'argument -v–invert-match (affiche les lignes ne contenant pas le motif).

```
[puthier@mamachine]  echo -e "123\n456"|grep -v "^1"
456
```

La commande sort

- La commande **sort** (tri).

```
sort [-t  separateur] [-kPOS1[,POS2]] [-nr] <fichier>
```

- **n**umérique, **r**everse. Exemple : tri sur la troisième colonne

```
[puthier@mamachine]  cat toto.txt
        b       1       C
        a       2       B
        c       5       A

[puthier@mamachine]  sort -k3 toto.txt
        c       5       A
        a       2       B
        b       1       C
```

La commande tr

Transpose ou élimine des caractères (**tr**anslate).

```
tr [options] chaîne1 chaîne2 <entree >sortie
```

- Exemple 1 : transposition.

```
[puthier@mamachine]  echo "ABCDE"|tr  'A' 'C'
 CBCDE

[puthier@mamachine]
 tr 'url' 'URL'  < week.html > result.txt

[puthier@mamachine] echo "ABCDE"|tr -d 'A'
 BCDE
```

La commande sed

La commande **sed** : "**s**tream **ed**itor". etant donné un flux, recherche l'occurence d'une expression réguliére et effectue les modifications.

- Exemple

```
[puthier@mamachine] echo -e "ABCD\nDEF"
ABCDEF
DEFAA

[puthier@mamachine]  echo -e 'ABCA\nAEA' | sed 's/A/G/'
GBCA
GEA
[puthier@mamachine]  echo -e 'ABCA\nAEA' | sed 's/A/G/g'
GBCG
GEG
```

La commande awk

- Un utilitaire extrêmement pratique qui bénéficie d'un langage de programmation propre.
- Un programme awk se présente souvent sous la forme d'une ligne de commande (des programmes plus évolués peuvent être écrits).
- En général, il se présentera sous la forme d'une structure BEGIN (action à réaliser avant de lire le flux), d'une structure centrale (une action à réaliser sur chaque ligne) et d'une structure END (action à réaliser après la lecture du flux) :

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

La commande awk

- Variables spéciales de awk :
  - $1,$2, $3... Colonne 1, 2, 3 ...
  - $0 la ligne courante
  - FS : **F**ield **S**eparator
  - OFS : **O**utput **F**ield **S**eparator
  - NR : **N**umber of **R**ows. Le numéro de la ligne courante
  - NF : **N**umber of **F**ields. Le nombre de colonnes (champs).

La commande awk

- Quelques exemples de *oneliners* avec awk :
  - Imprimer le nombre de colonnes pour chaque ligne.

```
awk 'BEGIN{FS="\t"}{print NF}' file.txt
```

  - ajouter un numéro de ligne et imprimer les colonnes 3 et 13

```
awk 'BEGIN{FS=OFS="\t"}{print NR, $3,$13}' file.txt
```

  - Calculer la somme de la colonne 2

```
awk 'BEGIN{FS="\t";s=0}{s=s+$2}END{print s}' file.txt
```

  - Et bien d'autres (*cf* awk onliners sur internet)

Contrôle des processus

Définition

- Lorsque vous lancez une commande ou un programme, vous démarrez un processus.
- A ces processus sont associés un PID (**Process ID** : nombre unique permettant de les identifier.
- Les commandes **top** et **ps** permettent de lister ces processus.
- La commande kill permet de “tuer” ces processus.

La commande top

- L'option **-u** permet de ne présenter que les processus d'un utilisateur donné.

```
[puthier@mamachine] top -u puthier
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4406 puthier   15   0  656m 165m  29m S   11  8.2   3:30.46 java
 4359 puthier   15   0 33364  16m  12m R    0  0.8   1:38.10 konsole
11559 puthier   15   0  2380 1064  764 R    0  0.1   0:00.06 top
 3813 puthier   18   0  3224 1460 1184 S    0  0.1   0:00.17 kde
 3875 puthier   15   0  2500  388  268 S    0  0.0   0:00.00 gpg-agent
...
```

- Dans **top** la touche "**k**" permet de choisir un processus à détruire (indiquer son PID). l'arrêt d'un processus peut aussi être effectué via la commande **kill**.

Premier plan et arrière plan

- un terminal n'accepte qu'un processus au premier plan.
- On peut arrêter temporairement un processus avec <ctrl> + z.
- On peut mettre fin au processus avec <ctrl> + c. "
- *bg* -> processus en arrière plan (background).
- *fg* -> processus au premier plan (foreground).

```
[puthier@mamachine] kate # je n'ai plus la main -> <ctrl> + z puis "bg".
```

- Lorsqu'on lance un processus on peut le mettre directement en arrière plan avec le caractère &.

```
[puthier@mamachine] kate& # Le processus se lance en arrière plan.
```

La commande nohup

- Une commande est associée à un terminal. Si on ferme le terminal, les processus qui en dépendent sont tués.
- Pour éviter cela, il faut utiliser nohup.

```
[puthier@mamachine] nohup macommande&
```

Réseau

La communication réseau est trés bien intégrée dans unix (ftp, ssh, http,...).

```
[puthier@mamachine] ftp tagc.univ-mrs.fr # ou lftp, ncftp
[puthier@mamachine] ssh 10.1.1.53
[puthier@mamachine] ssh -X 10.1.1.53 # avec affichage graphique.
[puthier@mamachine] curl http://tagc.univ-mrs.fr # recupération des sources d'un page web.
[puthier@mamachine] wget http://tagc.univ-mrs.fr/welcome/IMG/logo_inserm.gif #récupération d'un fichier sur le web.
[puthier@mamachine] curlftpfs ftp://tagc.univ-mrs.fr
...
```

Bioinformatique et reproductibilité.

Tout reproduire, même les erreurs...

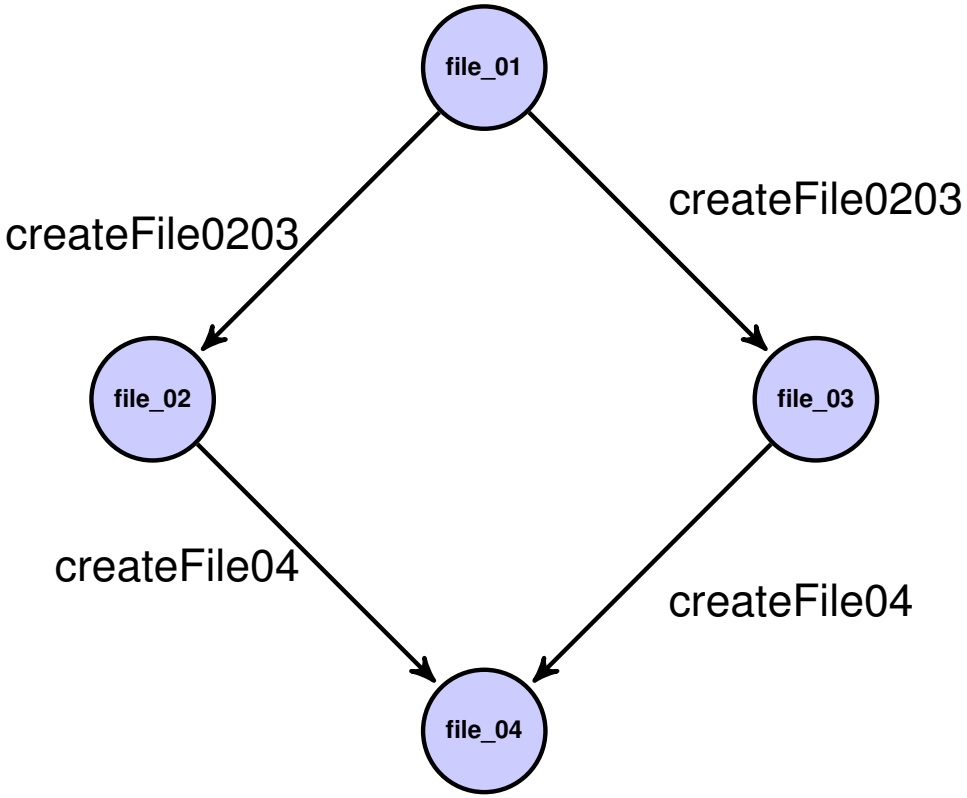
- Etant donné la complexité de certains traitement bioinformatiques, le problème de la reproductibilité se pose.
- Coder l'ensemble de la procédure permet de pouvoir identifier des erreurs *a posteriori*.
- Quelques solutions :
  - Make (makefiles)
  - Snakemake (Snakefiles)
  - R/Sweave

Make/makefile : création de 'pipelines/workflows'

- Permet une analyse reproductible (reproducible research).
- Permet de définir des fichiers cibles (targets) à partir de fichiers sources (prerequisites).
- A chaque création de fichier cible est associée une recette (recipe) codée en shell.
- Make assure une cohérence dans le déroulé du 'pipeline' en analysant les dates de modification des fichiers sources et cibles.

```
target: prerequesites
        recipe
```

Exemple de pipeline



Exemple de makefile :

```
all: file_04.txt

file_02.txt file_03.txt: file_01.txt
    @echo "Creating file_02 and file_03"
    @cat file_01.txt > file_02.txt
    @echo "123" >> file_02.txt
    @cat file_01.txt > file_03.txt
    @echo "456" >> file_03.txt


file_04.txt: file_03.txt file_02.txt
    @echo "Creating file_04"
    @cat file_02.txt file_03.txt > file_04.txt
```

Execution du makefile

- Se placer dans le dossier contenant le fichier makefile puis taper :

```
[puthier@mamachine] make
Creating file_02 and file_03
Creating file_04

# Changing the date of  file_02.txt
[puthier@mamachine] touch file_02.txt

# make regenerate file_04
[puthier@mamachine] make
Creating file_04
```

Quelques éléments pour la programmation (Bash Scripting).

Les variables

En plus d'un ensemble de fonctions très diverses le script shell permet de déclarer des variables et dispose de structures de contrôle et d'itération.

- Création d'une variable.
- Exemple

```
[puthier@mamachine] a=2
[puthier@mamachine] echo $a
 2
```

Les boucles

- Exemple d'utilisation de variables : les boucles.

```
[puthier@mamachine] ls *.txt
f1.txt  f2.txt  f3.txt
[puthier@mamachine] for i in *.txt;do mv $i $i.tmp;done
[puthier@mamachine] ls *.tmp
f1.txt.tmp  f2.txt.tmp  f3.txt.tmp
[puthier@mamachine] ls *.bmp |wc -l
12
[puthier@mamachine] for i in *.bmp; do convert $i $i.jpg;done
[puthier@mamachine] ls *.jpg |wc -l
12
```

Backquoting

- Permet de stocker le résultat d'une commande dans une variable.
  Ex : renommer des fichiers ou parcourir les lignes d'un fichier

```
[puthier@mamachine] for i in `seq 1 4`; do touch tp.file.$i.txt;done
tp.file.1.txt  tp.file.2.txt  tp.file.3.txt  tp.file.4.txt
[puthier@mamachine] rm -f `ls --color=none| grep -v 4`
file.4.txt
[puthier@mamachine] for i in `cat file.4.txt`; do curl $i;done
```

Bash script

- Automatisation de tâches.
- Nécessite de sauvegarder les commandes dans un fichier (e.g. ; myScript.sh).

- Example (the commands enclosed in "myScript.sh") :

```
#!/bin/bash

#This is a comment
echo "Hello" $1
```

- Running the script

```
[puthier@mamachine] chmod u+x myScript.sh
[puthier@mamachine] ./myScript "world"
Hello world
```

Pour en savoir plus...

- Le point de vue de R. Stallman (conf. à L'ENS)

  `http://framablog.org/index.php/post/2007/04/11/Stallman-en-grande-forme-conference-ENST-03-avril-2007`
- Unix RefCard

  `http://www.ai.univ-paris8.fr/~djedi/poo/unix-refcard.pdf`
- Unix Refcard (the One Page Linux Manual)

  `http://homepage.powerup.com.au/~squadron/linux_manual.pdf`
- Linux : Initiation et utilisation (J.P. Armspach, P. Colin, F. Ostré-Waerzeggers)
- Introduction aux scripts-shell (A. Robbins, N.H.F Beebe)
- Les TD Linux (D. Puthier)
- ...

Merci ! !