

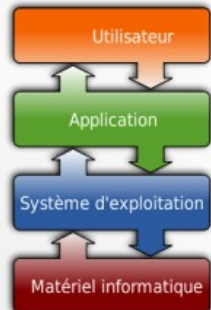
Systeme Exploitation

Systeme d'exploitation

Un *système d'exploitation* (**Operating System** ou **OS**) est un logiciel destiné à faciliter et simplifier l'utilisation d'un ordinateur

But

Un *système d'exploitation* permet d'exploiter les capacités du matériel en mettant à disposition de l'utilisateur un ensemble de services



Source : Wikipedia

Les principales tâches réalisées par un OS

Gestion de la mémoire

- transfert des programmes et des données nécessaires à la création des processus depuis un support secondaire (disque dur) vers un support central (RAM)
- allocation de mémoire
- transfert entre mémoire principale et mémoire secondaire

Linux : Avantages

- Ecrit dans un langage de haut niveau \Rightarrow portabilité
- Interface utilisateur simple
- Appels systèmes réutilisables pour écrire de nouvelles commandes
- Système de gestion de fichiers hiérarchiques
- Multi-Utilisateurs / Multi-Tâches

- + Gratuit
- + Moins sensible aux virus, spywares
- + Beaucoup de logiciels/applications gratuits
- + Centralisation des MAJ système & applications
- + Besoin de moins de ressources matérielles
- + ...

Gestion des processus

- concerne l'exécution des programmes
 - création
 - gestion de l'exécution simultanée
 - synchronisation
 - communication

Processus

Programme en cours d'exécution

Affichage de la valeur d'une variable :
`echo $NOM`

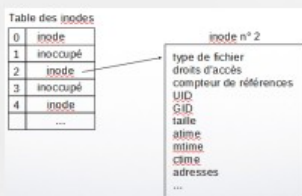
a noté que pour créer une variable il faut faire
`nomvar=valeur`

```
> a="bonjour tout le monde"
> echo $a
bonjour tout le monde
```

`pwd` => affiche le répertoire où l'on se trouve actuellement ex :

```
36002835@fst-unix3:~$ pwd
/home/etudiants/36002835
36002835@fst-unix3:~$
```

- Un **inode** est un descripteur de fichier, il contient les attributs du fichier et une table d'adressage des données sur le disque.
- Une **table d'inodes** par disque. Elle est stockée sur le disque lui-même dans un espace réservé lors de la création du système de fichier.
- La taille de la table d'inodes est statique et détermine le nombre maximum d'inodes, i.e. de fichiers.



Lien physique :

- sur un fichier déjà existant
- le fichier créé a le même numéro d'inode que le fichier lié
- le compteur de références de l'inode est incrémenté
- uniquement sur un même système de fichiers
- pas de lien physique sur un répertoire
- Un lien physique est une autre désignation du fichier lié, les données ne sont pas dupliquées

Lien symbolique :

- le fichier créé possède son propre inode et pointe sur l'inode du fichier lié
- le lien symbolique contient le chemin relatif vers le fichier lié
- pas de vérification de la cohérence
- possible entre deux systèmes de fichiers différents
- Un lien symbolique est un alias.

petite précision : le chemin **relatif** (absence de slash initial)
le chemin **absolu** (slash en préfixe du chemin)

- Création d'un lien physique fich2 sur fich1
In fich1 fich2
- Création, dans le répertoire rep, des liens physiques sur les fichiers fich1 fich2 ...
In fichier1 fichier2 ... rep
- l'option -s permet de créer des liens symboliques

comme par exemple :....

Pour manipuler l'arborescence plusieurs commande sont utilisé (à connaître) :

-la commande `pwd` qui affiche le nom du répertoire ou on se trouve actuellement (répertoire courant) voir l'exemple plus haut

-la commande `cd` qui signifie (change directory) nous permet de changer de répertoire de travail. En gros permet la navigation. Si l'on écrit :

cd sans argument on retourne sur notre répertoire d'accueil :

```
36002835@fst-unix3:~/rep1$ cd
36002835@fst-unix3:~$ ls
c  examples.desktop  Raptor  rep1  texte1  texte2  texte3  toto
```

cd avec argument cad avec un nom de répertoire on se dirige vers lui :

```
36002835@fst-unix3:~$ cd rep1
36002835@fst-unix3:~/rep1$ ls
ok  toi  totobis
```

on pourra noter qu'il y a d'autre façon de naviguer comme par exemple :

cd .. qui nous permet de remonter d'un niveau de répertoire

```
36002835@fst-unix3:~/rep1$ cd ..
36002835@fst-unix3:~$ ls
c  examples.desktop  Raptor  rep1  texte1  texte2  texte3  toto
```

-la commande *ls* qui permet de lister le contenu d'un répertoire.

ls sans argument nous liste le contenu du répertoire où l'on se trouve actuellement :

```
36002835@fst-unix3:~$ ls
c  examples.desktop  Raptor  rep1  texte1  texte2  texte3  toto
```

ls avec argument nous liste le contenu de l'argument :

```
c  examples.desktop  Raptor  rep1  texte1  texte2  texte3  toto
36002835@fst-unix3:~$ ls rep1
ok  toi  totobis
```

il y a également d'autres options comme le :

ls -a qui liste tout les fichiers y compris ceux qui commencent par un point et les fichiers cachés.

```
36002835@fst-unix3:~$ ls -a
.          .bash_logout  .cache          .local  rep1  texte3
..         .bashrc       examples.desktop .profile  rep1  texte1  toto
.bash_history  c             .gnupg          Raptor  texte2
```

ls -d affiche le nom des répertoires sans leur contenu.

ls -l format long avec détails

```
36002835@fst-unix3:~$ ls -l
1444284 c              137807 rep1      137940 texte2
1442204 examples.desktop 138247 rep2      1443845 texte3
1451948 Raptor        1444275 texte1    1451905 toto
```

et plein d'autres consultables avec la commande *man ls*

```

dgay@fst-unix3:~$ ls -ld *
drwxrwxr-x 2 dgay dgay 4096 août 28 23:25 0rep0
-rw-rw-r-- 1 dgay dgay 61 nov. 15 2017 1711151159.txt
-rw-rw-r-- 1 dgay dgay 75 nov. 15 2017 1711151750.txt
-rw-rw-r-- 1 dgay dgay 99 nov. 15 2017 1711151806.txt
-rw-rw-r-- 1 dgay dgay 0 sept. 28 2017 1file.txt
-rw-rw-r-- 1 dgay dgay 0 oct. 4 2017 3file.txt
-rw-rw-r-- 1 dgay dgay 85 oct. 25 2017 afile.txt
-rw-rw-r-- 1 dgay dgay 99 nov. 15 2017 AideMemoire.txt
-rwxrwxr-x 1 dgay dgay 30 nov. 15 2017 allparam.sh
-rwxrwxr-x 1 dgay dgay 115 nov. 8 2017 amoupm.sh
-rwxrwxr-x 1 dgay dgay 61 nov. 22 2017 argrank.sh
-rwxrwxr-x 1 dgay dgay 390 nov. 15 2017 arg.sh
-rwxrwxr-x 1 dgay dgay 8608 nov. 15 2017 bonjour
-rw-rw-r-- 1 dgay dgay 72 nov. 15 2017 bonjour.c
-rw-rw-r-- 1 dgay dgay 1504 nov. 15 2017 bonjour.o

```

- , d, l, type de fichiers
 - rwxrwxr-x droits d'accès au fichier
 - nombre de liens
 - propriétaire
 - groupe
 - taille
 - date de modification
 - nom

-la commande `mkdir` (make directory) nous permet de crer un nouveau répertoire. Par exemple :

`mkdir rep2` va créer le répertoire `rep2` dans notre répertoire actuel :

```

c exemples.desktop Raptor rep1 texte1 texte2 texte3 toto
36002835@fst-unix3:~$ mkdir rep2
36002835@fst-unix3:~$ ls
c exemples.desktop Raptor rep1 rep2 texte1 texte2 texte3 toto

```

- la commande `rmdir` (remove directory), elle permet de supprimer un repertoire dont le nom est donnée en argument. Par exemple si je veut supprimer notre `rep2` il faudra taper : `rmdir rep2` et alors le répertoire sera supprimer :

```

c exemples.desktop Raptor rep1 rep2 texte1 texte2 texte3 toto
36002835@fst-unix3:~$ rmdir rep2
36002835@fst-unix3:~$ ls
c exemples.desktop Raptor rep1 texte1 texte2 texte3 toto

```

il y a également la commande `rmdir -p` qui supprime tous les sous répertoire vide

et d'autre disponible dans le `man rmdir`

ATTENTION : impossible de supprimer un répertoire qui n'est pas vide sinon message d'erreur

```

36002835@fst-unix3:~$ rmdir rep1
rmdir: impossible de supprimer 'rep1': Le dossier n'est pas vide

```

Droit acces aux fichiers :

L'accès aux fichiers du système est contrôlé par des droits :

- lecture (read)
- écriture (write)
- exécution (execution)

Selon trois catégories d'utilisateurs :

- propriétaire/utilisateur (user)
- membres du groupe du fichier (group)
- autres utilisateurs du système (others)

La commande `ls -l` permet d'afficher les droits d'accès au fichiers :

```
36002835@fst-unix3:~$ ls -l
total 28
-rw-rw-r-- 1 36002835 36002835 0 sept.  6 14:54 c
-rw-r--r-- 1 36002835 36002835 8980 avril 20 2016 examples.desktop
-rw-rw-r-- 1 36002835 36002835 0 sept. 19 17:13 Raptor
drwx----- 3 36002835 36002835 4096 sept. 19 17:28 rep1
-rwx----- 1 36002835 36002835 9 sept.  7 14:28 texte1
-rw-rw-r-- 1 36002835 36002835 0 sept. 19 16:28 texte2
-rw-rw-r-- 1 36002835 36002835 9 sept. 19 15:19 texte3
-rw----- 2 36002835 36002835 21 sept. 19 15:14 toto
```

- Description d'un fichier par ligne.
- Le premier caractère d'une ligne indique le type de fichier,
- les 9 suivants (3×3) indiquent les droits d'accès aux fichiers (read, write, execution) pour les trois catégories d'utilisateurs (user, group, others).

$\begin{array}{ccc} r & w & x \\ \hline & \text{user} & \end{array}$ $\begin{array}{ccc} r & - & x \\ \hline & \text{group} & \end{array}$ $\begin{array}{ccc} r & - & x \\ \hline & \text{others} & \end{array}$

«le premier caractère d'une ligne indique le type de fichier» exemple : le «d» de la 4 eme ligne signifie que le type de fichier est un répertoire , ici rep1. Le «d» signifie directory.

Comment lire les droit d'accès en valeurs octales ? :

- Lecture, read : symbole **r**, valeur octale **4**
- Ecriture, write : symbole **w**, valeur octale **2**
- Exécution, execution : symbole **x**, valeur octale **1**

$\begin{array}{ccc} 4 & 2 & 1 \\ \hline r & w & x \\ \hline & 7 & \end{array}$ $\begin{array}{ccc} 4 & 2 & 1 \\ \hline r & - & x \\ \hline & 5 & \end{array}$ $\begin{array}{ccc} 4 & 2 & 1 \\ \hline r & - & - \\ \hline & 4 & \end{array}$

Valeur octale des droits d'accès à un tel fichier : 754

read ? Write ? :

- Lecture : droit de lister le contenu d'un répertoire
- Ecriture : droit de créer/supprimer une entrée dans le répertoire
- Exécution : droit de se positionner dans ou de traverser le répertoire

Pour changer les droit d'accès d'un fichier il faut utiliser la commande `chmod` qui signifie «change file mode bits».

Pour l'utiliser il faut taper par exemple `chmod 777 nom_du_fichier`

la commande permet de changer les droit d'accès à un fichier en donnant le mode soit de manière octale :

```
36002835@fst-unix3:~$ chmod 777 Raptor
36002835@fst-unix3:~$ ls -l
total 28
-rw-rw-r-- 1 36002835 36002835 0 sept. 6 14:54 c
-rw-r--r-- 1 36002835 36002835 8980 avril 20 2016 exemples.desktop
-rwxrwxrwx 1 36002835 36002835 0 sept. 19 17:13 Raptor
drwx----- 3 36002835 36002835 4096 sept. 19 17:28 repl
-rwx----- 1 36002835 36002835 9 sept. 7 14:28 texte1
-rw-rw-r-- 1 36002835 36002835 0 sept. 19 16:28 texte2
-rw-rw-r-- 1 36002835 36002835 9 sept. 19 15:19 texte3
-rw----- 2 36002835 36002835 21 sept. 19 15:14 toto
```

ici on utilise `chmod 777 Raptor` qui donnera tout les droit d'accès au fichier Raptor.

Ou on peut aussi changer les droit en utilisant la manière symbolique (ex `ugo` `+-rwx`) :

le `u` pour user
le `g` pour group
le `o` pour other
et le `a` pour all

le `+` pour ajouter
le `-` pour enlever
le `=` pour affecter

exemple :

si on garde le fait que le fichier Raptor a tout les droit d'accès donc `rwx rwx rwx` on peut faire :

```
36002835@fst-unix3:~$ chmod o-rwx Raptor
36002835@fst-unix3:~$ ls -l
total 28
-rw-rw-r-- 1 36002835 36002835 0 sept. 6 14:54 c
-rw-r--r-- 1 36002835 36002835 8980 avril 20 2016 exemples.desktop
-rwxrwx--- 1 36002835 36002835 0 sept. 19 17:13 Raptor
drwx----- 3 36002835 36002835 4096 sept. 19 17:28 repl
-rwx----- 1 36002835 36002835 9 sept. 7 14:28 texte1
-rw-rw-r-- 1 36002835 36002835 0 sept. 19 16:28 texte2
-rw-rw-r-- 1 36002835 36002835 9 sept. 19 15:19 texte3
-rw----- 2 36002835 36002835 21 sept. 19 15:14 toto
```

ici on a enlever les droit de «other» donc on a utiliser la commande `chmod o-rwx Raptor`
soit on enleve `rwx` pour la catégorie «other».

Les remettre de manière symbolique tjrs ? :

```
36002835@fst-unix3:~$ chmod o+rwx Raptor
36002835@fst-unix3:~$ ls -l
total 28
-rw-rw-r-- 1 36002835 36002835 0 sept. 6 14:54 c
-rw-r--r-- 1 36002835 36002835 8980 avril 20 2016 exemples.desktop
-rwxrwxrwx 1 36002835 36002835 0 sept. 19 17:13 Raptor
drwx----- 3 36002835 36002835 4096 sept. 19 17:28 repl
-rwx----- 1 36002835 36002835 9 sept. 7 14:28 texte1
-rw-rw-r-- 1 36002835 36002835 0 sept. 19 16:28 texte2
-rw-rw-r-- 1 36002835 36002835 9 sept. 19 15:19 texte3
-rw----- 2 36002835 36002835 21 sept. 19 15:14 toto
```

ici on a ajouter les droit de «other» donc on a utiliser la commande `chmod o+rwx Raptor`
soit on ajoute `rwx` pour la catégorie «other».

Pour fini l'affectation ?

```
36002835@fst-unix3:~$ chmod ugo=rwx Raptor
36002835@fst-unix3:~$ ls -l
total 28
-rw-rw-r-- 1 36002835 36002835 0 sept. 6 14:54 c
-rw-r--r-- 1 36002835 36002835 8980 avril 20 2016 examples.desktop
-rwxrwxrwx 1 36002835 36002835 0 sept. 19 17:13 Raptor
drwx----- 3 36002835 36002835 4096 sept. 19 17:28 repl
-rwx----- 1 36002835 36002835 9 sept. 7 14:28 texte1
-rw-rw-r-- 1 36002835 36002835 0 sept. 19 16:28 texte2
-rw-rw-r-- 1 36002835 36002835 9 sept. 19 15:19 texte3
-rw----- 2 36002835 36002835 21 sept. 19 15:14 toto
```

ici on a affecter tout les droit du fichier Raptor (ugo) donc on a utiliser la commande `chmod ugo=rwx Raptor` soit on affecte `rwx` pour la catégorie «user», «group» et «other».

Ne pas oublier que on peut utiliser aussi le `chmod a=... Raptor` par exemple ici on utilise `chmod a=r Raptor` qui va mettre le droit de lecture sur l'ensemble des cathégorie.

```
36002835@fst-unix3:~$ chmod a=r Raptor
36002835@fst-unix3:~$ ls -l
total 28
-rw-rw-r-- 1 36002835 36002835 0 sept. 6 14:54 c
-rw-r--r-- 1 36002835 36002835 8980 avril 20 2016 examples.desktop
-r--r--r-- 1 36002835 36002835 0 sept. 19 17:13 Raptor
drwx----- 3 36002835 36002835 4096 sept. 19 17:28 repl
-rwx----- 1 36002835 36002835 9 sept. 7 14:28 texte1
-rw-rw-r-- 1 36002835 36002835 0 sept. 19 16:28 texte2
-rw-rw-r-- 1 36002835 36002835 9 sept. 19 15:19 texte3
-rw----- 2 36002835 36002835 21 sept. 19 15:14 toto
```

Droit d'accès par défaut :

- Par défaut les fichiers sont créés avec les droits `r w- r - - r - -`
- La commande **umask** (user file creation mode **mask**) permet de changer les droits attribués par défaut.
- Elle prend en argument un masque constitué des 3 valeurs octales représentant les droits à retirer aux fichiers créés. Autrement dit le complément à 7 des droits qu'on veut attribuer aux fichiers.

`umask 027` ⇒ `rwx r-x - - -` ou `750`

ici par exemple on fait `umask 027` soit pour la catégorie «user» donc la première cathégorie on enleve rien d'où le «0» ensuite pour la deuxième cathégorie donc la cathégorie «group» le «2» cela signifie que l'on enleve «w» car «w» a pour équivalent octale 2 puis pour la dernière cathégorie «other» le «7» signifie que l'on enleve tout.

- Le **masque** est valable pour les fichiers créés après la commande, sans effet rétroactif.
- Sans argument la commande **umask** affiche le masque courant.
- Les droits à la création d'un fichier dépendent aussi des utilitaires utilisés pour les créer : ils peuvent être restreints ou augmentés par rapport au droits par défaut.
- Par exemple l'éditeur vi supprime le droit exécutable, mais un compilateur l'ajoute.

d'autre commande de chngement :

chown (**ch**ange **own**er). Pour changer le propriétaire (nom ou uid) d'un fichier :

chown utilisateur fichier

chgrp (**ch**ange **grp**). Pour changer le groupe (nom ou gid) d'un fichier

chgrp groupe fichier

Options :

- R** : opère récursivement sur un répertoire
- h** : agit sur les liens symboliques

Seul le propriétaire et root ont l'autorisation.

cp : copy (copie de fichier(s))

cp fichier1 fichier2

- Effectue une copie de fichier1 dans fichier2.
- Si fichier2 existe, il est écrasé

cp source1 [source2 ...] répertoire

- Copie chaque source dans le répertoire donné

Options courantes :

- n** n'écrase pas un fichier existant
- i** mode interactif, demande confirmation
- r** opère de manière récursive si la source est un répertoire
- p** conserve les dates du fichier source

ici la commande cp permet la copie de fichier exemple :

```
36002835@fst-unix3:~$ cp texte1 texte2
36002835@fst-unix3:~$ ls
c  examples.desktop  Raptor  repl  texte1  texte2  texte3  toto
36002835@fst-unix3:~$ cat texte1
teste n1
36002835@fst-unix3:~$ cat texte2
teste n1
```


ici on a copier le contenu de *texte1* sur *texte2*.

Ou encore :

```
36002835@fst-unix3:~$ cp texte1 rep1
36002835@fst-unix3:~$ ls rep1
ok  texte1  toi  totobis
```

ici on a copier *texte1* dans le repertoire *rep1*.

mv : move (déplacement de fichier(s))

mv source1 [source2 ...] destination

- Déplace une entrée dans l'arborescence de fichiers.
- Si destination est un répertoire alors chaque source est supprimée de son répertoire d'origine et insérée dans destination (déplacement)
- Si destination est un fichier alors source1 est renommé en destination.

Options courantes :

- n n'écrit pas un fichier existant
- i mode interactif, demande confirmation

Ici la commande *mv* permet de déplacer un fichier mais aussi de renommer !

Exemple :

```
36002835@fst-unix3:~$ mv texte1 texte2
36002835@fst-unix3:~$ ls
c  examples.desktop  Raptor  rep1  texte2  texte3  toto
36002835@fst-unix3:~$ cat texte2
teste n1
```

ici on a renommer *texte1* en *texte 2* car la source 1 «*texte1*» a pour destination «*texte2*» qui est un fichier comme la source 1.

autre exemple :

```
36002835@fst-unix3:~$ mv texte3 rep1
36002835@fst-unix3:~$ ls
c  examples.desktop  Raptor  rep1  texte1  texte2  toto
36002835@fst-unix3:~$ ls rep1
ok  texte1  texte3  toi  totobis
```

ici on a déplacer *texte3* qui été dans notre répertoire d'accueil dans le «*rep1*» car si la source 1 a pour destination un répertoire elle est déplacer et non pas renommé

note :

```
36002835@fst-unix3:~$ cd rep1
36002835@fst-unix3:~/rep1$ mv texte3 ..
36002835@fst-unix3:~/rep1$ ls
ok  texte1  toi  totobis
36002835@fst-unix3:~/rep1$ ls ..
c  examples.desktop  Raptor  rep1  texte1  texte2  texte3  toto
```

on peut utiliser le «..» pour déplacer dans le répertoire d'un niveau au dessus. Ici on redéplace texte3 qui été dans rep1 dans notre repertoire d'accueil.

Par contre il ne faut jamais oublier de se placer dans le répertoire ou se trouve le fichier que l'on veut déplacer.

rm : remove (supprimer des fichiers)

rm fichier1 [fichiers2...]

- Supprime les fichiers donnés.
- Plus précisément : supprime les références données. Si des liens physiques existent sur un fichier, c'est uniquement la référence désignée par le nom donné qui est supprimée.
- Les blocs de données (le fichier) ne sont physiquement supprimés que lors de la suppression de la dernière référence.

La commande rm nous permet de supprimer nos fichier

exemple :

```
36002835@fst-unix3:~$ ls
blade  c          Raptor  texte1  texte3  toto
burn   examples.desktop rep1    texte2  texte4  xbi
36002835@fst-unix3:~$ rm xbi
36002835@fst-unix3:~$ ls
blade  c          Raptor  texte1  texte3  toto
burn   examples.desktop rep1    texte2  texte4
```

ici on un fichier du nom de «xbi» on réalise donc la commande `rm xbi` et on remarque que le fichier a été supprimer de notre répertoire d'accueil.

Si on veut supprimer plusieurs fichier en même temps il suffit d'enchaîner le nom des fichiers cotés à cotés comme par exemple :

```
36002835@fst-unix3:~$ ls
blade  c          Raptor  texte1  texte3  toto
burn   examples.desktop rep1    texte2  texte4
36002835@fst-unix3:~$ rm blade burn
36002835@fst-unix3:~$ ls
c  examples.desktop Raptor  rep1  texte1  texte2  texte3  texte4  toto
36002835@fst-unix3:~$
```

ici on voit que grâce à la commande `rm blade burn` on a supprimé les deux fichiers d'un coup.

NOTE : voir au début pour la suppression de répertoire avec la commande `rmdir`

touch

touch fichier

- ▢ Permet de modifier les dates de dernier accès et de dernière modification d'un fichier sans modifier son contenu.
- ▢ Si le fichier n'existe pas, cela permet de créer un fichier vide avec les droits par défaut.

Options courantes :

- a change la date de dernier accès
- m change la date de dernière modification
- t fixe les dates selon un format donné (cf **man touch**)

La commande `touch` nous permet de créer un fichier
il suffit de taper la commande `touch nom_de_votre_fichier`

par exemple si on veut crer un fichier du nom de `bonjour` on va faire :

```
36002835@fst-unix3:~$ ls
c  examples.desktop  Raptor  repl  textel  texte2  text
36002835@fst-unix3:~$ touch bonjour
36002835@fst-unix3:~$ ls
bonjour  examples.desktop  repl  texte2  texte4
c        Raptor      textel  texte3  toto
36002835@fst-unix3:~$
```

find : rechercher des fichiers (2)

Quelques options de sélection (c.f. **man**)

- ▢ **-name** fic : le nom du fichier est fic
- ▢ **-type** x : le type du fichier est x = **d** répertoire, **f** fichier ordinaire, **l** lien symbolique ...
- ▢ **-size** n : taille (en blocs de 512Ko par défaut)
- ▢ **-user** x : le propriétaire est x
- ▢ **-ctime** n : la date de dernière modification remonte à n (jours par défaut)
- ▢ **-newer** fic : plus récent que le fichier fic

la commande `find` permet de rechercher des fichiers ou repertoire etc. si dessus nous avons tous les exemple de commande possible comme par exemple :

la commande `find -type d` va nous avicher tout les fichiers de type repertoires

```
36002835@fst-unix3:~$ find -type d
.
./rep1
./rep1/ok
./.cache
./.local
./.local/share
./.local/share/nano
./.local/share/keyrings
./.gnupg
./.gnupg/private-keys-v1.d
```




on peut par exemple aussi chercher par exemple le fichier car on ne sait pas où il est (imagine qu'on l'a oublié le chien) :

```
36002835@fst-unix3:~$ ls rep1
bye ok textel toi totobis
36002835@fst-unix3:~$ find -name bye
./rep1/bye
36002835@fst-unix3:~$
```

la commande nous affiche bien que le fichier `bye` est dans le répertoire `rep1`.

find : rechercher des fichiers (3)

On peut combiner logiquement les critères de sélection à l'aide des opérateurs

-  **-not**
-  **-and**
-  **-or**

Exemple : rechercher à partir de la racine les liens symboliques créés par l'utilisateur `etud12` :

```
find / -type l -and -user etud12
```

D.Gay

SE – Bash

82

Introduction

Utilisateurs

Système de Fichiers




Processus

Shell

S

find : rechercher des fichiers (4)

Actions :

-  **-print**
affiche les chemins d'accès des fichiers trouvés
-  **-exec *commande* {} \;**
exécute la commande sur chacun des fichiers trouvés.
Attention à la syntaxe!
-  **-ok *commande* {} \;**
idem `-exec` mais avec demande de confirmation

Nous avons vu ci-dessus les différentes possibilités en rapport avec la commande `find` comme l'enchaînement avec `le` and etc.

Processus : Intuition

- Un processus est lié à un programme en cours d'exécution
- Attention : un programme peut donner lieu à plusieurs processus
- Linux est Multi-Tâches : à un instant t , plusieurs processus peuvent s'exécuter en "même temps"
- Le système doit être capable d'identifier les différents processus
- Chaque processus a son propre contexte d'exécution

D.Gay

SE – Bash

Introduction

Utilisateurs

Système de Fichiers

Processus

Shell

Processus : Identification

- PID** (Process **ID**entifier) : numéro unique du processus
- PPID** (Parent Process **ID**entifier) : numéro du processus père
- Tout processus a été créé par un autre (son père) sauf le premier processus lancé par le système lors du démarrage qui a pour PID 1 et pas de PPID.
- La numérotation recommence à 1 à chaque démarrage.

ps : process status

La commande **ps** permet de visualiser les processus qui "tournent" sur une machine.

Options courantes :

- e** ou **-a** affiche tous les processus
- u** uid affiche pour un utilisateur particulier
- f** affichage détaillé

cette commande nous permet de voir les processus qui tourne sur notre machine avec plusieurs options différentes. Par exemple juste la commande «nue» nous renvoie :

```
36002835@fst-unix3:~$ ps
  PID TTY          TIME CMD
  7815 pts/22    00:00:00 bash
 16068 pts/22    00:00:00 ps
36002835@fst-unix3:~$
```

avec :

Processus status

- UID : nom ou id du propriétaire du processus
- PID : numéro du processus
- PPID : numéro du processus père
- C : facteur de priorité (plus élevé = moins prioritaire dans l'attribution des ressources)
- STIME : heure de lancement
- TTY : numéro du terminal associé (?? = aucun)
- TIME : temps CPU utilisé
- COMMAND : commande associé au processus

Plein d'autres options d'affichage et d'informations sur les processus en cours : **man ps**

La commande **kill** (comme son nom ne l'indique pas) permet d'envoyer un signal à un processus :

kill -sig pid

envoie le signal de numéro ou de nom *sig* au processus *pid*.

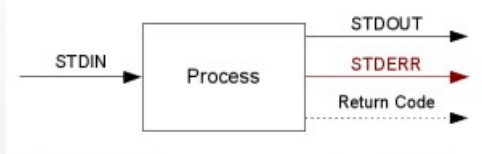
Deux signaux à retenir absolument :

SIGKILL (9) : tuer le processus, terminaison brutale et imparable

SIGTERM (15) : terminaison propre

Entrées/Sorties d'une commande

Entrées/Sorties d'une commande (processus)



- STDIN** : entrée standard, par défaut le clavier
Descripteur de fichier : **0**
- STDOUT** : sortie standard, par défaut l'écran
Descripteur de fichier : **1**
- STDERR** : sortie standard des erreurs, par défaut l'écran
Descripteur de fichier : **2**

De plus chaque commande renvoie un code de retour d'exécution (**exit status**) qui vaut 0 si tout s'est bien passé.

Pour l'enchaînement de commande il faut faire :

Séquentiel

`commande1 ; commande2`

Conditionnel

`commande1 && commande2`

`commande1 || commande2`

Séparateur	exit status commande1	exécution commande2
&&	0	OUI
&&	≠ 0	NON
	0	NON
	≠ 0	OUI

Exemple : ici on crée le fichier «kabo» et on écrit directement à l'intérieur :

```
36002835@fst-unix3:~$ touch kab0 ; cat > kab0
hello mother fucker
36002835@fst-unix3:~$ ls
bonjour  examples.desktop  paaa  paad  paag  textel  texte4
c        kab0          paab  paae  Raptor  texte2  toto
essai.sh okaa          paac  paaf  repl   texte3
36002835@fst-unix3:~$ cat kab0
hello mother fucker
```

l'enchaînement de commande nous permet au lieu de créer le fichier puis retaper la commande cat pour écrire à l'intérieur, ici on enchaîne les deux commandes et cela nous permet de gagner du temps:)

Les redirections

Il est possible et très utile de rediriger les entrées/sorties d'une commande.

Redirections des sorties standard

■ de la sortie standard (STDOUT 1) :

```
commande > fichier  
commande >> fichier
```

■ de la sortie standard des erreurs (STDERR 2) :

```
commande 2 > fichier  
commande 2 >> fichier
```

Si le fichier cible n'existe pas, il est créé.

Avec >, si le fichier existe, il est écrasé.

Avec >>, s'il existe le résultat est ajouté à la fin.

Les redirections

Redirections de l'entrée standard STDIN : 0

```
commande < fichier  
commande << mot
```

Avec <, le contenu du fichier est envoyé à la commande

Avec <<, la commande lit tout ce qui suit jusqu'au mot donné seul sur une ligne

Tout peut être facilité avec le «pipe» ou tube de communication, qui permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande :

```
commande1 | commande2
```

ce qui est équivalent à :

```
commande1 > fichier ; commande2 < fichier
```

< ps : le « | » c'est «altgr»+ »6» pour les plus handicapé XD >

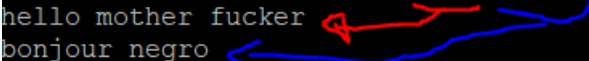
cat : concaténation de fichiers

cat [fichier1 ...]

- Concatène le contenu des fichiers donnés sur la sortie standard, c'est à dire par défaut l'écran.
- Si aucun fichier n'est donné, la commande concatène tout ce qui est tapé au clavier sur la sortie standard jusqu'à ce qu'on tape CTRL-D.
- Cette commande permet entre autres de facilement visualiser le contenu d'un fichier en affichant son contenu à l'écran.

concatenation c'est à dire la fusion entre guillemet de deux fichier par exemple si l'on fait `cat fichier1 fichier2` il va nous afficher tout le contenu du fichier1 suivi du contenu du fichier2 :

```
36002835@fst-unix3:~$ cat kabo
hello mother fucker
36002835@fst-unix3:~$ cat textel
bonjour negro
36002835@fst-unix3:~$ cat kabo textel
hello mother fucker
bonjour negro
```



head & tail

head [-n] fichier

Affiche les n (par défaut 10) premières ligne du fichier

tail [-n nombre] fichier

Affiche les n (par défaut 10) dernières ligne du fichier

tail +n fichier

Affiche le fichier à partir de la ligne numéro n

la commande head et la commande tail nous permettent d'afficher les lignes de nos fichiers soit en commençant par le début (head) ou la fin (tail) et de plus la commande tail peut nous permettre de commencer la lecture à un certain nombre de lignes.

Exemple :

soit un fichier chien :

```
36002835@fst-unix3:~$ cat chien
bonjour je suis la ligne 1
et moi la ligne 2
3
4
5
etc connard...
report
bruler les hérés... chut
ne rend pas les choses plus difficile XD

36002835@fst-unix3:~$
```

la commande head :

```
36002835@fst-unix3:~$ head -1 chien
bonjour je suis la ligne 1
36002835@fst-unix3:~$ head -3 chien
bonjour je suis la ligne 1
et moi la ligne 2
3
```

ici par exemple la commande `head -1 chien` nous permet d'afficher la première ligne du fichier chien et la commande `head -3 chien` nous permet d'afficher les 3 premières lignes du fichier chien.

Par défaut la commande head sans l'indicateur du nb de ligne est mis à 10 par défaut.

La commande tail :

```
36002835@fst-unix3:~$ tail -1 chien

36002835@fst-unix3:~$ tail -3 chien
bruler les hérés... chut
ne rend pas les choses plus difficile XD

36002835@fst-unix3:~$ tail +2 chien
et moi la ligne 2
3
4
5
etc connard...
report
bruler les hérés... chut
ne rend pas les choses plus difficile XD

36002835@fst-unix3:~$
```

*la commande `tail -1 chien` affiche la dernière ligne du fichier .. et ATTENTION une ligne vide comme dans notre exemple reste une ligne à afficher
ensuite la commande `tail -3 chien` affiche les 3 dernières lignes du fichier.*

La commande `tail +2 chien` ici nous a permis d'afficher les lignes de notre fichier à partir de la ligne 2.

wc : word count

wc [-lwc] [fichier]

- ▣ Compte le nombre de lignes (-l) , mots (-w), caractères (-c) d'un fichier.
- ▣ Par défaut compte les trois et affiche les résultats séparés par des tabulations sur une ligne dans l'ordre : nombre de lignes, mots et caractères.

la commande wc nous permet de compter le nombre de ligne mots et caractère.

Si on utilise la commande à nue .. donc wc on aura les 3 par défaut :

```
36002835@fst-unix3:~$ wc chien
10 29 141 chien
36002835@fst-unix3:~$
```

ici on a juste utiliser la commande wc pour le fichier chien donc on tape wc chien et on a 10 lignes , 29 mots et 141 caractères.

Si l'on veut cibler par exemple on veut que le nombre de caractères on devra taper :

wc -c chien [-c pour caractère -l pour ligne et -w pour word (mot)]

exemple :

```
36002835@fst-unix3:~$ wc -c chien
141 chien
36002835@fst-unix3:~$
```

ici on a que le nombre de caractères soit 141

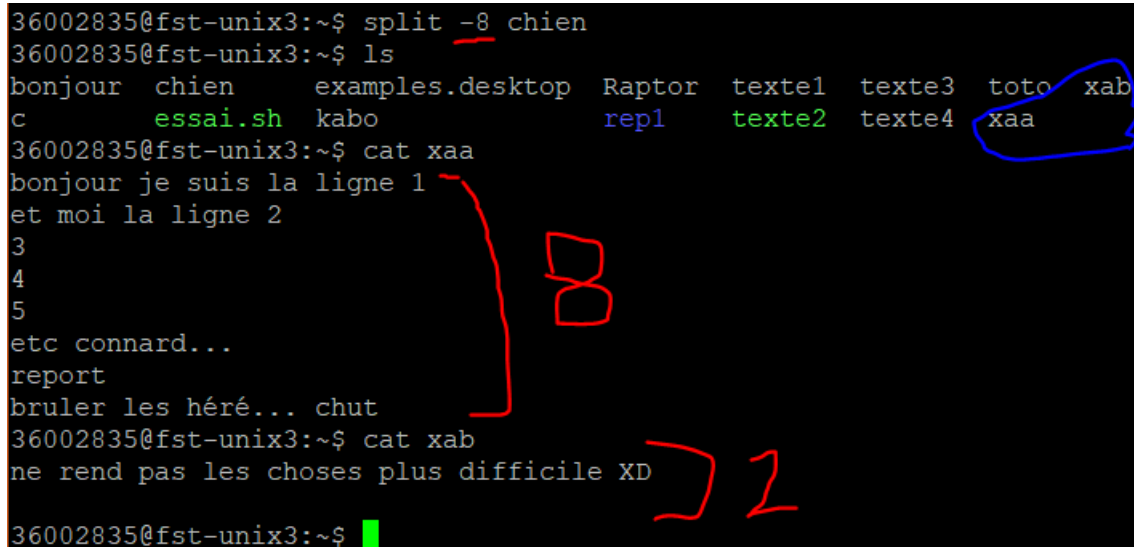
split : scinder un fichier

split [-n] [fichier [nom]]

- ▣ Scinde le fichier donné (par défaut l'entrée standard) en plusieurs fichiers de n lignes (par défaut 1000).
- ▣ Les fichiers créés sont nommés par défaut xaa, xab, xac ... ou bien en suffixant le nom donné par aa, ab, ac ...

ici la commande split nous permet de scinder le fichiers

```
36002835@fst-unix3:~$ split -8 chien
36002835@fst-unix3:~$ ls
bonjour  chien      exemples.desktop  Raptor  texte1  texte3  toto  xab
c        essai.sh  kabo              repl    texte2  texte4  xaa
36002835@fst-unix3:~$ cat xaa
bonjour je suis la ligne 1
et moi la ligne 2
3
4
5
etc connard...
report
bruler les hérés... chut
36002835@fst-unix3:~$ cat xab
ne rend pas les choses plus difficile XD
36002835@fst-unix3:~$
```



prenons cet exemple

ici le fichier chien contient 10 lignes

on utilise la commande `split -8 chien` qui va scinder le fichier en 8 lignes

par défaut comme vu sur le screen d'avant la commande `split` va créer des fichiers `xaa`, `xab`, `xac` et chaque fichier va contenir le nombre de lignes que l'on a indiqué ..

donc ici il a scindé le fichier `chien` de 8 en 8 donc il a créé que 2 fichiers .. `xaa` et `xab` car `chien` ne contenait que 10 lignes

fichiers `xaa` et `xab` ont donc été créés (en bleu)

et donc le fichier `xaa` contient les 8 premières lignes du fichier `chien` (en rouge)

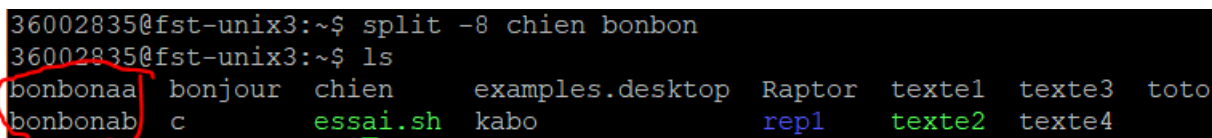
et vu qu'il ne restait que 2 lignes il a créé le fichier `xab` et a mis les 2 lignes restantes (en rouge également)

a noté que l'on peut changer le nom des fichiers créés : au lieu de créer des fichiers `xaa`, `xab` etc.. on peut faire la commande :

`split -8 chien bonbon`

et ici les fichiers ne se nomment plus `xaa` et `xab` mais `bonbonaa` et `bonbonab` :

```
36002835@fst-unix3:~$ split -8 chien bonbon
36002835@fst-unix3:~$ ls
bonbonaa  bonjour  chien      exemples.desktop  Raptor  texte1  texte3  toto
bonbonab  c        essai.sh  kabo              repl    texte2  texte4
```



paste : fusion de fichiers

`paste [-d liste] [fichier ...]`

- Permet de fusionner les lignes de différents fichiers en regroupant les lignes correspondantes de chaque fichier, séparées par des tabulations (ou par les délimiteurs spécifiés avec l'option `-d`), et terminées par un saut de ligne.
- Si aucun fichier n'est indiqué, ou si le nom `-` est mentionné, la lecture se fait sur l'entrée standard.

la commande paste qui permet la fusion des fichiers mais au niveau des lignes.
Exemple :

soit deux fichiers :

```
36002835@fst-unix3:~$ cat chien
bonjour je suis la ligne 1
et moi la ligne 2
3
4
5
etc connard...
report
bruler les hérés... chut
ne rend pas les choses plus difficile XD
36002835@fst-unix3:~$
```

```
36002835@fst-unix3:~$ cat kabo
hello mother fucker
36002835@fst-unix3:~$
```

le fichier chien

et le fichier kabo

si on utilise la commande paste elle va fusionner les lignes des fichiers et donc nous afficher les lignes de chaque fichier côte à côte séparées de tabulation.

Exemple :

```
36002835@fst-unix3:~$ paste chien kabo
bonjour je suis la ligne 1      hello mother fucker
et moi la ligne 2
3
4
5
etc connard...
report
bruler les hérés... chut
ne rend pas les choses plus difficile XD
36002835@fst-unix3:~$
```

comme le montre le screen la ligne 1 de chien qui est « bonjour je suis la ligne 1 » est à côté de la ligne 1 du fichier kabo qui est « hello mother fucker ».

Comme le fichier n'a pas d'autres lignes que la première il n'y a rien à côté des lignes du fichier chien mais si il y a par exemple 10 lignes pour le fichier chien et 10 pour le fichier kabo alors les 10 lignes du premier seront à côté des 10 lignes du deuxième.

sort : tri de fichiers

`sort [option] [fichier ...]`

Permet de trier les lignes de la concaténation des fichiers donnés selon les options données. Quelques options (c.f. man pour les autres) :

- n tri numérique (par défaut selon le code ASCII)
- d tri alphanumérique uniquement
- r ordre décroissant (par défaut croissant)
- f ignorer la casse
- b ignorer les blancs en début de champ
- u supprimer les doublons
- kn indique sur quel champ trier
- tc précise le caractère de séparation de champs

cette commande va nous permettre de trier selon nos souhaits l'intérieur du fichier donné.

Par exemple si l'on veut trier notre fichier chien par ordre décroissant on va utiliser `sort -r chien`

```
36002835@fst-unix3:~$ sort -r chien
report
ne rend pas les choses plus difficile XD
et moi la ligne 2
etc connard...
bruler les hérés... chut
bonjour je suis la ligne 1
5
4
3
36002835@fst-unix3:~$
```

comme on le voit sur le screen les lignes ont été triées par ordre décroissant alors que de base le fichier se présente ainsi :

```
36002835@fst-unix3:~$ cat chien
bonjour je suis la ligne 1
et moi la ligne 2
3
4
5
etc connard...
report
bruler les hérés... chut
ne rend pas les choses plus difficile XD
36002835@fst-unix3:~$
```

il y a beaucoup d'options disponibles à voir sur le screen du cours plus haut:)

tr : transcodage de caractères

tr chaîne1 chaîne2

- Transforme chaque caractère de l'entrée standard qui apparaît dans la chaîne1 par celui de la chaîne2 lui correspondant.
- Si la chaîne2 est plus courte que la chaîne1 elle est complétée en dupliquant le dernier caractère.
- Options : c.f. man

Exemple 1 : tr "abc" "ABC"

transforme les a en A, les b en B et les c en C

Exemple 2 : tr "a-z" "A-Z"

transforme les minuscules en majuscules

*la commande tr qui permet la transformation de caractère
quelque exemple disponible au dessus*

```
36002835@fst-unix3:~$ tr "bonjour" "BONJOUR"
bonjour
BONJOUR
hello
hello
salut
salUt
36002835@fst-unix3:~$
```

par exemple ici on dit que bonjour doit être transformé en BONJOUR mais ATTENTION ce n'est pas le mot qui est transformé mais les caractères soit ici on dit que quand on entre un b minuscule il doit être transformé en B majuscule

comme montré au dessus quand on tape «bonjour» il nous renvoie «BONJOUR»

si on tape «hello» il renvoie «hellO» car il a vu un o minuscule et comme dit auparavant il transforme le o en O majuscule

pareil pour le «salut» il n'a transformé que le u car il y avait un u dans bonjour et il a compris qu'il fallait changer le u en majuscule.

A noter que pour sortir de cette phase d'écriture il faut effectuer un CTRL-d comme lorsque l'on a fini d'écrire dans un fichier.

Par exemple si on fait :

`tr "a-z" "A-Z"`

la il va transformer tout ce que l'on tape en minuscule en majuscule

cut : extraction de champs

La commande **cut** permet de supprimer une partie de chaque ligne d'un fichier ou de l'entrée standard.

`cut -c liste_positions`

Affiche uniquement les caractères des positions indiquées.

`cut -f liste_champs`

Affiche uniquement les champs (séparation par défaut: tabulation) spécifiés. L'option **-dx** permet de déclarer x comme caractère de séparation de champ.

Exemples :

```
cut -c 5-20      conserve les caractères du 5è au 20è
cut -d: -f 5-    conserve les champs séparés par : à partir du 5è
cut -c -4,8-12,20- ?
```

ici la commande cut supprime une partie de chaque ligne de notre fichiers comme montre les exemple ci dessus :

la commande `cut -c 5-20`

va conserver les caractère du 5 eme au 20 eme exemple :

```
36002835@fst-unix3:~$ cut -c 5-10 chien
our je
oi la

connar
rt
er les
end pa
36002835@fst-unix3:~$
```

ici avec la comande cut on a conserver les caractère a partir du 5eme au 10 eme donc il nous affiche notre fichier chien mais juste avec les caractère que l'on a demander.

(bonjour je suis la ligne 1)

est devenu (our je) car il a supprimer les caractère 1 2 3 et 4 et a commencer a partir du 5 eme soit le o de bonjour

a noter que un espace compte comme un caractère !!!

exemple la il s'est aretter a «e» car le e etait le 10 eme caractère

il y a encore plein de possibilité comme vu sur le screen du cours

au niveau du shell :

on rappelle que pour crer une variable on doit faire :

`nomvariable=valeur`

exemple :

```
> a="bonjour tout le monde"
> echo $a
bonjour tout le monde
```

et pour afficher la valeur de notre variable on va faire un `echo $nomvariable`

mais ici de la nouveauté :

- Si une variable est immédiatement suivi d'autres caractères, il faut l'encadrer avec des accolades
- Si une variable n'est pas affectée (pas de valeur), le shell lui substitue la chaîne vide (aucun caractère)

```
> a=toto
> b=titi
> ab=tutu
> echo $ab
tutu
> echo ${a}b
totob

> echo a$b
atiti
> echo ${a}c
totoc
> echo $ab
tutu
> echo $ac
toto c

> echo $a c
toto c
```

ici pas besoin d'exemple tout est expliqué sur le screen du cours:)

Il est possible de donner des valeurs par défaut lors de la substitution

- `${a-zozo}` si a n'existe pas, lui substituer la valeur zozo
- `${a-$b}` si a n'existe pas, lui substituer la valeur de la variable b
- `${a=12}` si a n'existe pas, lui affecter la valeur 12
- `${a?message}` si a n'existe pas, provoquer une interruption et afficher le message

```
36002835@fst-unix3:~$ echo $a
36002835@fst-unix3:~$ echo ${a=12}
12
36002835@fst-unix3:~$ echo $a
12
36002835@fst-unix3:~$ echo ${a-zozo}
12
36002835@fst-unix3:~$
```

exemple ici a n'a pas de valeur attribué.... Donc on dit que si a n'existe pas on lui affecte la valeur 12 ... donc il a vu que a n'existait pas et lui a mis la valeur 12 ...

ensuite on lui dit si a n'existe pas lui affecter la valeur «zozo» malheureusement a existe donc il a renvoyé la valeur de a soit 12

on peut également attribuer une commande à une variable :

La sortie standard d'une commande peut être substituée :

```
> pwd
/home/enseignants/dgay
> a=`pwd`
> echo $a /home/enseignants/dgay
> echo je suis dans le repertoire `pwd`
je suis dans le repertoire /home/enseignants/dgay
```

Toute chaîne entre back-quote `cmd` est interprétée comme une commande à exécuter et est remplacée par la sortie standard de cette commande

Les jokers :

Les caractères spéciaux du shell

Les Jokers pour la génération de noms de fichiers

- ? désigne exactement un caractère quelconque
- * désigne un nombre quelconque (0 ou +) de caractères
- [liste] désigne un caractère parmi ceux donnés entre crochets

Les Jokers exemple

- t?t? filtre les noms titi, tito, tatx, ... mais pas tooto, tatata, titix, ...
- a*b filtre tous les noms commençant par a et terminant par b, y compris ab
- t[ai]t[ai] filtre uniquement tata, tati, tita, titi
- t[a-e]* filtre tous les noms commençant par t et suivi par un caractère parmi a, b, c, d, e et d'un nombre quelconque de caractères

par exemple on peut faire :

```
36002835@fst-unix3:~$ ls
bonbonaa  bonjour  chien      exemples.desktop  Raptor  texte1  texte3  toto
bonbonab  c         essai.sh   kabo              repl    texte2  texte4
36002835@fst-unix3:~$ ls t??t??
texte1  texte2  texte3  texte4
36002835@fst-unix3:~$
```

dans cet exemple on fait la commande `ls t??t??` et il va afficher tous les fichiers qui commencent par un t suivis de 2 caractères quelconques suivis d'un t et suivis de 2 caractères quelconques

donc il nous affiche texte1 texte2 etc..

comme sur le screen du cours il y a plusieurs options possibles comme `a*b` ou `t[ai]t[ai]` etc..

Caractères protégés

- # ce qui suit est considéré comme un commentaire (non-évalué)
- \$ substitution d'un nom de variable
- ; sépare 2 commandes tapées sur une même ligne
- ! permet de relancer les commandes de l'historique
- >, <, |, ... les caractères utilisés pour les redirections I/O

partis du cours a comprendre sans exemple :

Expressions régulières

Définition : Regular Expression (RE)

- Toute chaîne de caractères est une expression régulière (RE)
- Si une RE contient des caractères spéciaux, ils doivent être inhibés avec \
- Une expression régulière sert à identifier une chaîne de caractères répondant à certains critères

RE & caractères spéciaux

- [^abc] un caractère autre que ceux donnés entre crochets (complémentaire)
- ^ hors des crochets, il signifie le début de ligne
- \$ signifie la fin de ligne

Exemples :

[a-z] les chaînes ne contenant pas de lettres minuscules

^toto les lignes commençant par toto

toto\$ les lignes se terminant par toto

^toto\$ les lignes formées exactement de toto

^\$ les lignes vides

D.Gay

SE – Bash

Introduction

Utilisateurs

Système de Fichiers

Processus

Shell

RE & caractères spéciaux

- * répétition 0 ou plusieurs fois du caractère précédant

Exemples :

a* les lignes contenant 0 ou plusieurs a (donc toutes les lignes)

aa* les lignes contenant au moins un a

.* n'importe quelle chaîne de caractères

Pour les expressions régulières étendues : man grep

le reste du cours concerne les scripts DONC

il restes a regarder comprendre et tester si besoin les derniers éléments du cours ... soit de la diapo 130 a 149 car pas eu le temps de faire en fiche .. pasque mi lé faible oui XDDD donc bon courage bonne gay XD