

- *Partie 2: Programmation Shell sous Linux*

*Prof. Dr. Ing. Abdellah Amine*

# Table des matières

1. **PRÉSENTATION**
2. **CONNEXION ET DÉCONNEXION**
3. **UNE PREMIÈRE COMMANDE : « ECHO »**
4. **LE SYSTÈME DE FICHIERS**
5. **L'ÉDITEUR**
6. **REDIRECTIONS**
7. **LES DROITS D'ACCÈS**
8. **LES FILTRES ET UTILITAIRES**
9. **PROGRAMMATION SHELL**

# 1. Présentation

## • 1.1 Définition

**SO: Programme ou ensemble de programmes et d'API servant d'interface entre le matériel (hardware) et les applications (software).**

**Unix** est un système d'exploitation multi-tâches et multi-utilisateurs.

- **Portable** : Écrit majoritairement en C, seules quelques parties sont en assembleur.
- **Multi-tâches** : Le système peut exécuter plusieurs tâches en même-temps, de manière **préemptive**, sur un ou plusieurs processeurs.
- **Multi-utilisateurs** : Plusieurs utilisateurs peuvent se connecter et travailler en même temps sur une machine, soit directement sur celle-ci (Linux, BSD, Sco) soit depuis un terminal distant.
- **Stable** : protection mémoire, les plantages du système par lui-même sont très rares.
- **Deux standards principaux** : System V et BSD, qui tout en restant compatibles diffèrent au niveau de certains appels systèmes, de la hiérarchie du système de fichier, de la séquence de démarrage...

Les composants de base d'un Unix sont le **noyau (kernel)** et les **outils (shell et commandes)**.

# 1. Présentation

Le système d'exploitation a pour principales tâches les points suivants :

1. Gestion de la mémoire
2. Accès aux périphériques
3. Accès disque / Système de fichiers
4. Gestion des programmes (processus)
5. Sécurité / Accès aux données
6. Collecte d'informations système : Statistiques

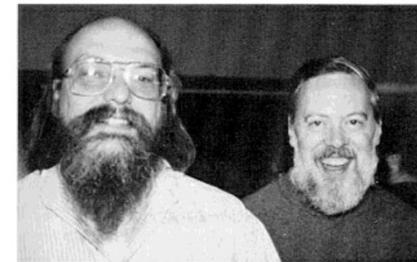
# 1. Présentation

## 1.2 Historique

### 1.2.1 Les origines

- UNIX est né au sein des laboratoires BELL (Filiaire d'AT&T)
- Développé à partir de 1969 par Ken Thompson et Dennis Ritchie
- Des 1973, UNIX est réécrit à 90% en langage C
- En 1975, les sources d'Unix sont diffusées dans les universités
- Développement de 2 branches :
  - **BSD** développé à l'Université de Berkeley (Californie)
  - **System V** vendu par AT&T à Sun Microsystems, IBM, DEC et HP

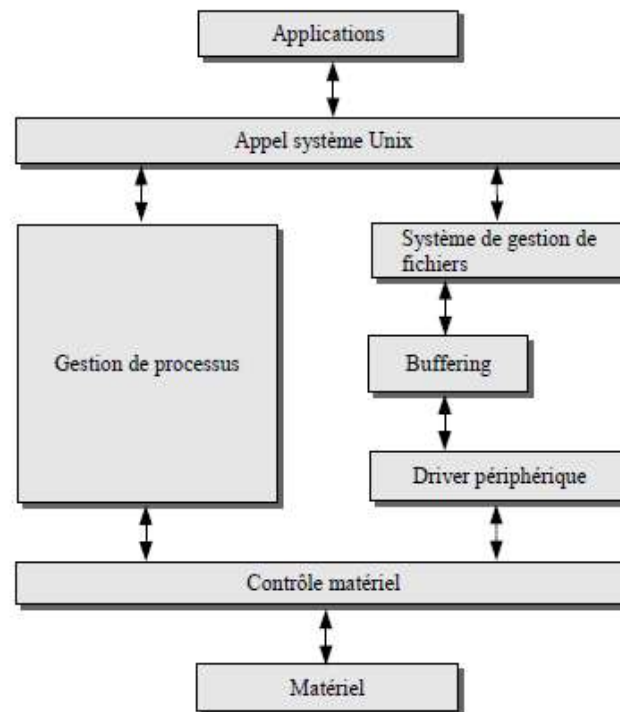
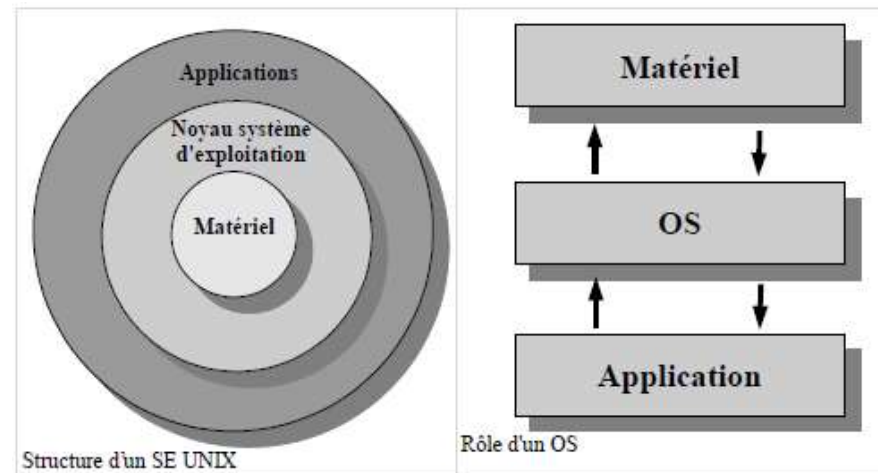
UNIX est une marque déposée depuis 1994



Ken Thompson (L) and Dennis Ritchie (R)

# 1. Présentation

## 1.3 Architecture



## 2 Connexion et déconnexion

- *2.1 Première Connexion*

Pour pouvoir travailler sous Unix il faut ouvrir une session

root ou utilisateur privilégié

Utilisateur normal

Pour se connecter:

Login : <tapez ici votre nom d'utilisateur>

Password : <tapez ici votre mot de passe>

## 2 Connexion et déconnexion

Pour se familiariser avec la saisie de commandes, nous pouvons tester quelques programmes d'information :

- **date** : affiche la date et l'heure
- **who** : liste des utilisateurs connectés (who am i : qui suis-je)
- **cal** : affiche le calendrier (cal 12 1975)
- **man** : mode d'emploi (man cal), Elle permet de visionner les contenus d'une documentation formatée pour être exploitable **par man**, Par exemple, pour voir le manuel de la commande ftp, il faut taper:

**\$>man ftp**

Pour interrompre une commande on peut utiliser la combinaison **Ctrl+C**. Pour lier les commandes, on sépare les commandes par le caractère « ; »

who ; date

Pour se déconnecter, il suffit de taper

exit

On peut aussi utiliser la combinaison de touches **Ctrl+D**.



## 2 Connexion et déconnexion

### *2.2 Changer son mot de passe*

On utilise la commande **passwd** pour modifier son mot de passe.

```
$ passwd
```

Old password:

New password:

Reenter password:

## 2 Connexion et déconnexion

### *2.3 Utilisateur, groupes*

On distingue sous Unix les utilisateurs et les groupes, notion que nous verrons et détail lors de la gestion des droits. Un groupe définit un ensemble d'utilisateurs, un utilisateur fait obligatoirement partie d'au moins un groupe, ou de plusieurs. Le groupe par défaut d'un utilisateur est « **users** ».

### *2.4 Prendre la place d'un autre*

Le système permet dans certains cas à un utilisateur connecté de changer de nom en cours de travail avec la commande `su`. Le mot de passe du nouvel utilisateur sera demandé.

`su [-] utilisateur [-c commande]`

Si `-` est précisé, l'environnement du nouvel utilisateur est chargé, et si `-c` est précisé les commandes qui suivent sont exécutées

### *2.5 Obtenir son nom de connexion*

La commande **logname** affiche le nom de login de l'utilisateur, en principe toujours le nom utilisé lors de la première connexion.

`$ logname`

oracle

### 3 Une première commande : « echo »

la commande **echo** est une commande centrale du shell : elle transmet tous ses paramètres sur écran (ou canal de sortie standard).

echo texte

Le texte est quelconque mais peut aussi admettre quelques caractères de formatage.

<i>Caractère</i>	<i>Effet</i>
<code>\n</code>	Saut de ligne (newline)
<code>\b</code>	Retour arrière (backslash)
<code>\t</code>	Tabulation
<code>\c</code>	Pas de retour à la ligne (carriage)
<code>\\</code>	Affiche \
<code>\\$</code>	Affiche \$
<code>\valeur</code>	Affiche le caractère spécial de code octal valeur

Exemple:

```
$ echo "Bonjour\nComment ça va ?\c"
```

## 4. Le système de fichiers

### 4.1 Définition

- Un système de fichiers / FileSystem / FS: comment sont gérés et organisés les fichiers par le système d'exploitation. Le FS d'Unix est hiérarchique
- Sous UNIX, tout élément est représenté sous forme de fichier
- 4 types de fichiers :
  - **Ordinaire (ordinary files)** : données, programme
  - **Répertoire (directory)**: contient d'autres données ou répertoires
  - **Lien symbolique** : pointe vers un autre fichier
  - **Spécial** : permet l'accès à un périphérique
- Chaque fichier est caractérisé par son nom, sa taille, ses droits d'accès, son propriétaire, ses dates de création, de modification...
- Structure arborescente de fichiers

## 4. Le système de fichiers

### 4.2 Les divers types de fichiers

#### 4.2.1 fichiers ordinaires (ordinary files)

Ce sont soit des fichiers contenant du texte, soit des exécutables (ou binaires), soit des fichiers de données. Par défaut, rien ne permet de différencier les uns des autres, sauf à utiliser quelques options de certaines commandes (ls -F par exemple) ou la commande **file**.

\$ file nom\_fic

nom\_fic : 32 Bits ELF Executable Binary (stripped)

## 4. Le système de fichiers

### 4.2.2 catalogues (les répertoires ou directory)

Les répertoires permettent d'organiser le disque dur en créant une hiérarchie. Un répertoire peut contenir des fichiers normaux, des fichiers spéciaux et d'autres répertoires, de manière récursive.

### 4.2.3 fichiers spéciaux

situés dans /dev, ce sont les points d'accès préparés par le système aux périphériques. Le montage va réaliser une correspondance de ces fichiers spéciaux vers leur répertoire "point de montage". Par exemple, le fichier /dev/hda permet l'accès et le chargement du 1er disque IDE

## 4. Le système de fichiers

### 4.3 Nomenclature des fichiers

On ne peut pas donner n'importe quel nom à un fichier, il faut pour cela suivre quelques règles simples. Ces règles sont valables pour tous les types de fichiers.

Sur les anciens systèmes un nom de fichier ne peut pas dépasser 14 caractères. Sur les systèmes récents, on peut aller jusqu'à 255 caractères. Il est possible d'utiliser des extensions de fichiers mais cela ne modifie en rien le comportement du système (un exécutable n'a pas besoin d'une extension particulière).

## 4. Le système de fichiers

**Unix fait la distinction entre les minuscules et majuscules. Toto, TOTO, ToTo et toto sont des noms de fichiers différents.**

La plupart des caractères (chiffres, lettres, majuscules, minuscules, certains signes, caractères accentués) sont acceptés, y compris l'espace (très déconseillé). Cependant quelques caractères sont à éviter :

& ; ( ) ~ <espace> \ | ` ? - (en début de nom)



## 4. Le système de fichiers

### *Quelques noms valides :*

Fichier1

Paie.txt

123traitement.sh

Paie\_juin\_2002.xls

8

### *Quelques noms pouvant poser problème :*

Fichier\*

Paie(decembre)

Ben&Nuts

Paie juin 2002.xls

-f

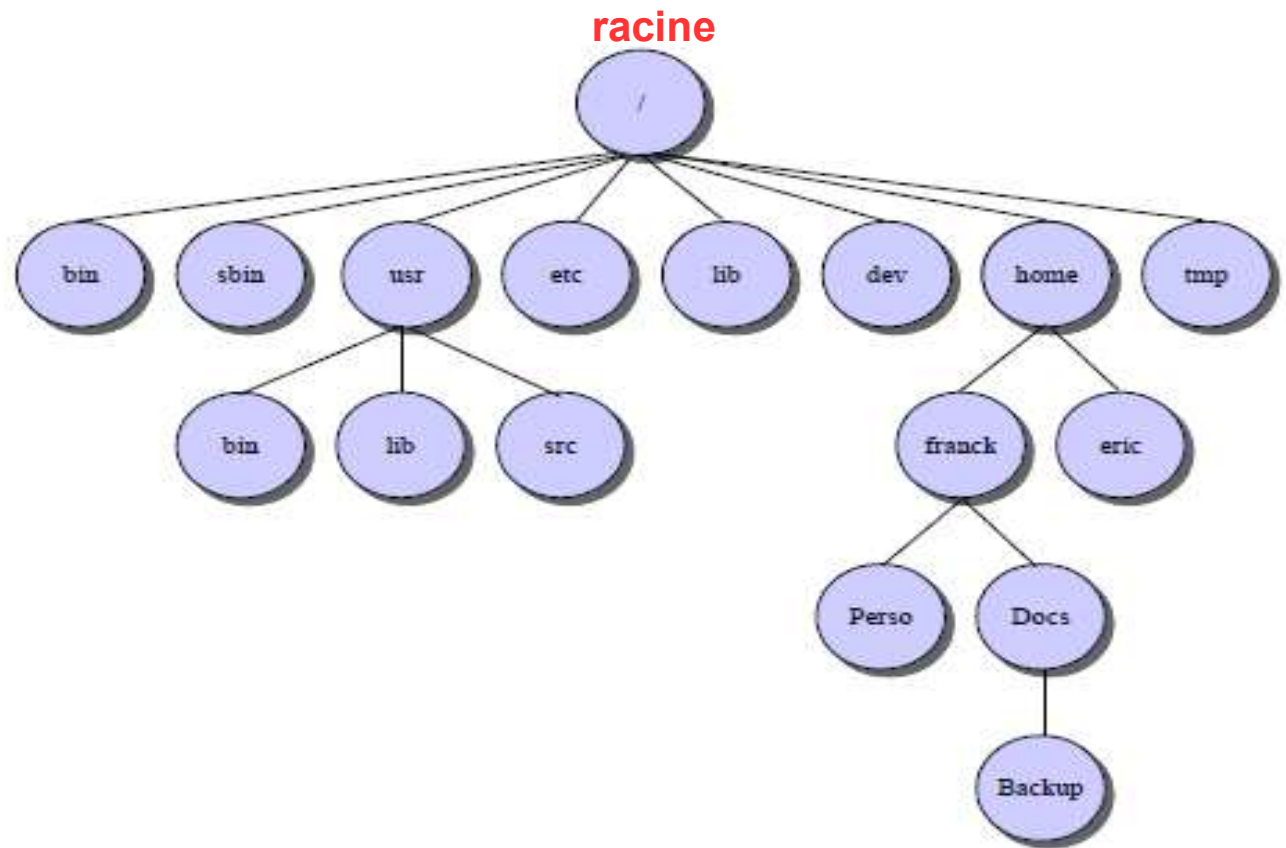
...

## 4. Le système de fichiers

### 4.4 Chemins

#### 4.4.1 Structure et nom de chemin

Le **nom de chemin** ou **path name** d'un fichier est la concaténation, depuis la racine, de tous les répertoires qu'il est nécessaire de traverser pour y accéder, chacun étant séparé par le caractère « / ». C'est un **chemin absolu**.  
Le « / » situé tout en haut s'appelle la **racine** ou **root directory**



## 4 Le système de fichiers

### 4.4.2 *Chemin relatif*

Un nom de chemin peut aussi être relatif à sa position courante dans le répertoire.

Le système (ou le shell) mémorise la position actuelle d'un utilisateur dans le système de fichier, le répertoire actif. On peut donc accéder à un autre répertoire de l'arborescence depuis l'emplacement actuel sans taper le chemin complet.

Pour se déplacer dans les répertoires, on utilise la commande **cd**. Le « **..** » permet d'accéder au répertoire de niveau supérieur. Le « **.** » définit le répertoire actif (répertoire courant). La commande **ls** permet de lister le contenu du répertoire. La commande **pwd** (print working directory) affiche le chemin complet du répertoire actif.

## 4 Le système de fichiers

### 4.4.2 *Chemin relatif*

Un tel chemin (exprimé à partir de la racine) est appelé chemin absolu. Voici un exemple de chemin absolu sous MS-DOS **c:\cours\chapitre4.txt** et sous Unix **/home/user1/rapport.txt**. Par contre, un chemin qui ne commence pas par la racine est un chemin relatif

```
$ cd /  
$ ls  
bin  
sbin  
usr  
etc  
lib  
dev  
home  
tmp
```

```
$ cd /usr/lib  
$ pwd  
/usr/lib  
$ cd ../bin  
$ pwd  
/usr/bin  
$ cd ../../etc  
$ pwd  
/etc
```

## 4 Le système de fichiers

- **Exemple**

- Chemin **absolu** :

**/home/queudet/programmes/progC/p1.c**

**/home/queudet/textes/lettre.txt**

- Chemin **relatif** au répertoire d'accueil :

**~/progC/p1.c**

**~/textes/lettres.txt**

- Chemin **relatif** au répertoire courant :

**progC/p1.c**

**../textes/lettres.txt**



Permet de remonter d'un niveau dans l'arbre

## 4 Le système de fichiers

### 4.4.3 Répertoire personnel

Lors de la création d'un utilisateur, l'administrateur lui alloue un répertoire utilisateur. Après une connexion, l'utilisateur arrive directement dans ce répertoire, qui est son **répertoire personnel**.

C'est dans ce répertoire que l'utilisateur pourra créer ses propres fichiers et répertoires. La commande **cd** sans argument permet de retourner directement dans son répertoire utilisateur.

Login : toto

Password :

\$ pwd

/home/toto

## 4 Le système de fichiers

### 4.4.4 ls et quelques commandes

- La commande **ls** permet de lister le contenu d'un répertoire (catalogue) en lignes ou colonnes. Elle supporte plusieurs options.

<i>Option</i>	<i>Signification</i>
-l	Sortie de chaque information des fichiers
-F	Ajoute un « / » au nom d'un répertoire, un « * » au nom d'un exécutable, un «   » pour un tube nommé et « @ » pour un lien symbolique.
-a	Affiche toutes les entrées, y compris « . », « .. » et les fichiers cachés (qui commencent par un .)
-d	affiche le nom (et les attributs) des répertoires et pas leur contenu.
-i	Affiche les numéros d'inode.
-R	Mode récursif. Rentre dans les répertoires et affiche leur contenu.
-r	Inverse l'ordre du tri (à l'envers)
-t	Tri par date de modification
-c	Affiche la date de création (si -l) ou tri par date de création (si -t)
-C	Les noms sont affichés sur plusieurs colonnes
-u	Affiche la date d'accès (-l) ou tri par date d'accès (-t)
-1	Liste sur une seule colonne.

## 4 Le système de fichiers

Sortie de la commande ls -l :

-rw-r--r--	1	oracle	dba	466	Feb 8 2001	input.log
1	2	3	4	5	6	7

1. Le premier caractère représente le type de fichier, les autres, par blocs de trois, les droits pour l'utilisateur, le groupe et tous.
2. Compteur de liens
3. Propriétaire du fichier.
4. Groupe auquel appartient le fichier
5. Taille du fichier en octets
6. Date de dernière modification (parfois avec l'heure)
7. Nom du fichier



## 4 Le système de fichiers

Deux autres commandes utiles :

- **cat** : concaténation de fichiers, le résultat étant affiché par défaut sur la sortie standard (écran).
- **touch** : permet de créer un fichier s'il n'existe pas, et s'il existe de modifier sa date d'accès et sa date de modification, touch toto crée le fichier toto s'il n'existe pas.

## 4 Le système de fichiers

### 4.5 Gestion des fichiers et répertoires

#### 4.5.1 Création de répertoires

- La commande **mkdir** (make directory) permet de créer un ou plusieurs répertoires, ou une arborescence complète.

```
mkdir rep1 [rep2] ... [repn]
```

```
$ mkdir documents
```

```
$ mkdir documents/texte documents/calcul documents/images
```

- La commande **mkdir** accepte un paramètre « -p » permettant de créer une arborescence.

```
$ mkdir -p documents/texte
```

va créer à la fois documents et texte. C'est valable pour tous les répertoires de niveau supérieur :

```
$ mkdir -p documents/texte/perso
```

va créer les répertoires documents, texte et perso s'ils n'existent pas. S'il existent ils ne sont pas modifiés

## 4 Le système de fichiers

### 4.5.2 Suppression de répertoires

La commande **rmdir** (remove directory) supprime un ou plusieurs répertoires. Elle ne supprime pas une arborescence. Si des fichiers sont encore présents dans le répertoire, la commande retourne une erreur. Le répertoire ne doit donc contenir ni fichiers ni répertoires.

```
rmdir rep1 [rep2] ... [repn]
```

```
$ cd documents
```

```
$ rmdir texte/perso
```

## 4 Le système de fichiers

### 4.5.3 Copie de fichiers

La commande **cp** (copy) copie un ou plusieurs fichiers vers un autre fichier ou vers un répertoire.

`cp fic1 fic2`

`cp fic1 [fic2 ... ficn] rep1`

Dans le premier cas, fic1 est recopié en fic2. Si fic2 existe, il est écrasé sans avertissement (sauf droit particulier). Dans le second cas, fic1, fic2 et ainsi de suite sont recopiés dans le répertoire rep1. Les chemins peuvent être absolus ou relatifs.

## 4 Le système de fichiers

### 4.5.3 Copie de fichiers

La commande peut prendre les options suivantes :

- **-i** : demande une confirmation pour chaque fichier avant d'écraser un fichier existant.
- **-p** : préservation des permissions, dates d'accès de modification
- **-r** : Récursif. Si la source est un répertoire copie de ce répertoire et de toute son arborescence.
- Les liens symboliques ne sont pas recopiés tels quels, mais seulement les fichiers pointés (avec le nom du lien cependant).
- **-R** : comme -r mais la recopie est identique à l'original (les liens symboliques sont copiés tels quels)

## 4 Le système de fichiers

### 4.5.4 Déplacer et renommer un fichier

.La commande **mv** (move) permet de déplacer et/ou de renommer un fichier. Elle a la même syntaxe que la commande **cp**. On peut à la fois déplacer et changer de nom.

```
$ cd
```

```
$ mv cv.txt cv_toto.txt
```

```
$ mv image.jpg documents/images/photo_toto_cv.jpg
```

## 4 Le système de fichiers

### 4.5.5 Supprimer un fichier ou une arborescence

La commande **rm** (remove) supprime un ou plusieurs fichiers, et éventuellement une arborescence complète, suivant les options. La suppression est définitive (à moins d'avoir un utilitaire système propre au filesystem).

```
rm [Options] fic1 [fic2...]
```

Options :

- i** : la commande demandera une confirmation pour chacun des fichiers à supprimer. Suivant la version d'Unix, le message change et la réponse aussi : y, Y, O, o, N, n, parfois toutes.
- r** : le paramètre suivant attendu est un répertoire. Dans ce cas, la suppression est récursive : tous les niveaux inférieurs sont supprimés, les répertoires comme les fichiers.
- f** : force la suppression. Si vous n'êtes pas le propriétaire du fichier à supprimer, rm demande une confirmation, mais pas avec l'option -f. Aucun message n'apparaîtra si la suppression n'a pu avoir lieu.

```
$ cd
```

```
$ rm -rf documents
```

## 4 Le système de fichiers

### 4.5.6 Les liens : plusieurs noms pour un fichier

Un lien permet de donner plusieurs noms à un même fichier, ou de faire pointer un fichier sur un autre. Plutôt que de faire plusieurs copies d'un même fichier pour plusieurs utilisateurs, on peut par exemple permettre à ceux-ci d'accéder à une copie unique, mais depuis des endroits et des noms différents. On utilise la commande **ln**.

```
ln [options] fic1 fic2
```

```
ln [options] fic1 rep1
```

```
ln [options] rep1 fic2
```

- Il existe deux types de liens : les liens en dur « **hard links** » et les liens symboliques « **symbolic links** ».



## 4 Le système de fichiers

### 4.5.6 Les liens : plusieurs noms pour un fichier

Il existe un nœud d'index ou un inode

Inode est comme une BD d'un fichier, il contient plusieurs informations (numéro d'inode, la taille, le propriétaire, le type d'autorisation, le numéro du lien ...) sur le fichier, mais il ne contient pas deux choses importantes: le contenu et le nom du fichier.

## 4 Le système de fichiers

### 4.5.6.1 Hard Link

- Un **hard link** permet d'ajouter une référence sur un inode.
- Sous Unix chaque fichier est en fait référencé au sein de deux tables : une table d'**inode** (information node, noeud d'information, une par filesystem) qui contient outre un numéro de fichier, des informations comme des droits, le type et des pointeurs sur données, et une table catalogue (une par répertoire) qui est une table de correspondance entre les noms de fichiers et les numéros d'inodes. Le hard link rajoute donc une association dans cette seconde table entre un nom et un inode. Les droits du fichier ne sont pas modifiés.
- Un lien physique est simplement un nom supplémentaire pour un fichier existant sous Linux ou d'autres systèmes d'exploitation de type Unix. ... Des liens physiques peuvent également être créés vers d'autres liens physiques. Cependant, ils ne peuvent pas être créés pour des répertoires, et ils ne peuvent pas traverser les limites du système de fichiers ou s'étendre sur des partitions.

## 4 Le système de fichiers

### 4.5.6.1 Hard Link

- Un hard link ne permet pas d'affecter plusieurs nom à un même répertoire, et ne permet pas d'effectuer des liens depuis ou vers un autre filesystem.
- Pas de différence entre un fichier original et un lien physique (une copie et non un raccourcis)
- **If the original file is deleted, the hard links will still contain the data that were in the original file**

**\$ cd**

**\$ touch fic1**

**\$ ln fic1 fic2**

**\$ ls**

**fic1 fic2**

**\$ ls -l**

**-rw-r--r-- 2 oracle system 0 Jul 25 11:59 fic1**

**-rw-r--r-- 2 oracle system 0 Jul 25 11:59 fic2**

**\$ ls -i**

**484 fic1 484 fic2**

## 4 Le système de fichiers

### 4.5.6.2 Symbolic Link (pointeur sur les fichiers)

Un lien symbolique ne rajoute pas une entrée dans la table catalogue mais est en fait une sorte d'alias, un fichier spécial contenant une donnée pointant vers un autre chemin (on peut le concevoir comme une sorte de fichier texte spécial contenant un lien vers un autre fichier ou répertoire).

De par cette nature, un lien symbolique ne possède pas les limitations du hard link. Il est donc possible d'effectuer des liens entre plusieurs FileSystems, et vers des répertoires. Le cas échéant le lien se comportera à l'identique du fichier ou du répertoire pointés (un `cd nom_lien` est possible dans le cas d'un répertoire).

La suppression de tous les liens symboliques n'entraîne que la suppression de ces liens, pas du fichier pointé. La suppression du fichier pointé n'entraîne pas la suppression des liens symboliques associés. Dans le cas le lien pointe dans le vide(**if we delete the original file, the softlinks will become useless**)

## 4 Le système de fichiers

### Exemple

```
$ rm fic2
```

```
$ ln -s fic1 fic2
```

```
$ ls -l
```

```
-rw-r--r-- 1 oracle system 0 Jul 25 11:59 fic1
```

```
lrwxrwxrwx 1 oracle system 4 Jul 25 12:03 fic2 -> fic1
```

```
$ls -i
```

```
484 fic1 635 fic2
```

```
$ ls -F
```

```
fic1 fic2@
```

## 4 Le système de fichiers

Trois zones distinctes sur le disque dur:

### 1. Super Block

Type de FS

Taille

Etat

### 2. Inode Table(table d'index)

Information sur le fichier

mode (chmod)

Information autres tables

taille, date de modification

méta données :

NB: (Les métadonnées sont des données qui en décrivent d'autres. Dans la plupart de ses usages informatiques, le préfixe méta signifie « définition ou description de référence ».)

### 3. Data Blocks

# Le système de fichiers

- Table Inode: référence à l'endroit où les données réelles sont stockées (Blocs de données).
- Chaque fichier a une référence dans la table inode qui pointe vers les données dans les blocs de données.
- Si on supprime quelques données dans le disque dur, les données ne seront pas supprimées dans les blocs de données, elles sont seulement dissociées dans la table d'inode.

## 4 Le système de fichiers

### 4.5.7 Critères de recherche sur noms de fichier

Lors de l'utilisation de commandes en rapport avec le système de fichier, il peut devenir intéressant de filtrer la sortie de noms de fichiers à l'aide de certains critères, par exemple avec la commande `ls`. Au lieu d'afficher toute la liste des fichiers, on peut filtrer l'affichage à l'aide de divers critères et caractères spéciaux.



## 4 Le système de fichiers

### 4.5.7 Critères de recherche sur noms de fichier

<i>Caractère spécial</i>	<i>Rôle</i>
*	Remplace une chaîne de longueur variable, même vide
?	Remplace un caractère unique quelconque
[]	Une série ou une plage de caractères
[!...]	Inversion de la recherche

Ainsi,

**ls a\*** : tous les fichiers commençant par a

**ls a??** : tous les fichiers de trois caractères commençant par a

**ls a??\*** : tous les fichiers d'au moins trois caractères et commençant par a

**ls [aA]\*** : tous les fichiers commençant par a ou A

**ls [a-m]?\*txt** : tous les fichiers commençant par les lettres de a à m, possédant au moins un second caractère avant la terminaison txt.

## 4 Le système de fichiers

### 4.5.8 Verrouillage de caractères

Certains caractères spéciaux doivent être verrouillés, par exemple en cas de caractères peu courants dans un nom de fichier.

Le backslash \ permet de verrouiller un caractère unique. ls paie\ \*.xls va lister tous les fichiers contenant un espace après paie.

Les guillemets "..." les guillemets permettent l'interprétation des caractères spéciaux, variables, au sein d'une chaîne.

Les apostrophes '...'verrouillent tous les caractères spéciaux dans une chaîne ou un fichier.

## 5. L'éditeur

L'éditeur Unix par défaut se nomme **vi** (visual editor). Il a l'avantage d'être disponible et d'utiliser la même syntaxe de base sur tous les Unix. Chaque Unix propose généralement une syntaxe étendue au-delà de la syntaxe de base. Pour en connaître les détails : `man vi`.

**vi [options] Fichier [Fichier2 ...]**Trois modes de fonctionnement :

- mode commande : les saisies représentent des commandes. On y accède en appuyant sur « Echap ».
  - Mouvements (déplacement) et quantificateurs
  - effacement, copier/couper/coller,
  - rechercher
- mode saisie : saisie de texte classique
  - qui permet d'ajouter/insérer des caractères
- mode ligne de commande « à la ex » : utilisation de commandes spéciales saisies et se terminant par Entrée. Accès pas la touche « : ».
  - quitter, enregistrer, Fermer
  - Remplacer
  - Exécuter une commande externe

## 5. L'éditeur

### 5.1 Commandes de saisie

En mode commande

<i>Commande</i>	<i>Action</i>
a	Ajout de texte derrière le caractère actif
A	Ajout de texte en fin de ligne
i	Insertion de texte devant le caractère actif
I	Insertion de texte en début de ligne
o	Insertion d'une nouvelle ligne sous la ligne active
O	Insertion d'une nouvelle ligne au-dessus de la ligne active

## 5. L'éditeur

### 5.2 *Quitte*

La commande **ZZ** quitte et sauve le fichier

- **:q!** quitte sans sauver
- **:q** quitte si le fichier n'a pas été modifié
- **:w** sauve le fichier
- **:wq** ou **x** sauve et quitte

## 5. L'éditeur

### 5.3 Déplacement en mode commande

<i>Commande</i>	<i>Action</i>
h	Vers la gauche
l	Vers la droite
k	Vers le haut
j	Vers le bas
0 (zéro)	Début de ligne (:0 première ligne)
\$	Fin de ligne (:\$ dernière ligne)
w	Mot suivant
b	Mot précédent
fc	Saut sur le caractère 'c'
Ctrl + F	Remonte d'un écran
Ctrl + B	Descend d'un écran
G	Dernière ligne du fichier
NG	Saute à la ligne 'n' (:n identique)

## 5. L'éditeur

### 5.4 Correction

<i>Commande</i>	<i>Action</i>
x	Efface le caractère sous le curseur
X	Efface le caractère devant le curseur
re	Remplace le caractère sous le curseur par le caractère 'c'
dw	Efface le mot depuis le curseur jusqu'à la fin du mot
d\$ (ou D)	Efface tous les caractères jusqu'à la fin de la ligne
dO	Efface tous les caractères jusqu'au début de la ligne.
dfc	Efface tous les caractères de la ligne jusqu'au caractère 'c'
dG	Efface tous les caractères jusqu'à la dernière ligne, ainsi que la ligne active
D1G	Efface tous les caractères jusqu'à la première ligne, ainsi que la ligne active
dd	Efface la ligne active

Ces commandes peuvent être répétées. 5Dd supprime 5 lignes.

**On peut annuler la dernière modification avec la commande « u ».**

## 5. L'éditeur

### *5.5 Recherche dans le texte*

Contrairement à un éditeur de texte classique, **vi** peut rechercher autre chose que des mots simples et fonctionne à l'aide de caractères spéciaux et de critères. La commande de recherche est le caractère « / ». La recherche démarre du caractère courant à la fin du fichier. Le caractère « ? » effectue la recherche en sens inverse. On indique ensuite le critère, puis Entrée.

/echo

recherche la chaîne 'echo' dans la suite du fichier. Quand la chaîne est trouvée, le curseur s'arrête sur le premier caractère de cette chaîne.

La commande « **n** » permet de continuer la recherche dans le sens indiqué au début. La commande « **N** » effectue la recherche en sens inverse.



## 5. L'éditeur

### 5.5.1 Quelques critères :

- `/[FfBb]oule` : Foule, foule, Boule, boule
- `/[A-Z]e` : Tout ce qui commence par une majuscule avec un e en deuxième position.
- `/[A-Za-Z0-9]` : tout ce qui commence par une majuscule, minuscule ou un chiffre
- `/[^a-z]` : plage négative : tout ce qui ne commence pas par une minuscule
- `/vé.o` : le point remplace un caractère, vélo, véto, véro, ...
- `/Au*o` : l'étoile est un caractère de répétition, de 0 à n caractères, Auo, Auto, Automoto, ...
- `/.*` : l'étoile devant le point, une chaîne quelconque de taille variable
- `/[A-Z][A-Z]*` : répétition du motif entre [] de 0 à n fois, recherche d'un mot comportant
- au moins une majuscule (en début de mot)
- `/^Auto` : le ^ indique que la chaîne recherchée devra être en début de ligne
- `/Auto$` : le \$ indique que la chaîne recherchée devra être en fin de ligne

## 5. L'éditeur

### 5.5.2 Quelques commandes de remplacement

Pour remplacer du texte, il faut se placer au début de la chaîne à modifier, puis taper l'une des commandes suivantes.

<i>Commande</i>	<i>Action</i>
cw	Remplacement du mot courant
c\$	Remplacement jusqu'à la fin de la ligne
cO	Remplacement jusqu'au début de la ligne
cfx	Remplacement jusqu'au prochain caractère 'x' dans la ligne courante
c/Auto (Entrée)	Remplacement jusqu'à la prochaine occurrence de la chaîne 'Auto'

Après cette saisie, le caractère \$ apparaît en fin de zone à modifier. Il suffit alors de taper son texte et d'appuyer sur Echap.

## 5. L'éditeur

### 5.6 Copier-Coller

On utilise la commande « **y** » (Yank) pour copier du texte, elle-même devant être combinée avec d'autres indications. Pour couper (déplacer), c'est la commande « **d** ». Pour coller le texte à l'endroit choisi, c'est la commande « **p** » (derrière le caractère) ou « **P** » (devant le caractère). Si c'est une ligne complète qui a été copiée, elle sera placée en-dessous de la ligne active.

Pour copier une ligne : **yy**

Pour copier cinq lignes : **5yy**

Pour placer les lignes copiées à un endroit donné : **p**

L'éditeur vi dispose de 26 tampons pour y stocker les données que l'on peut nommer comme on le souhaite. On utilise pour ça le « " ».

Pour copier cinq mots dans la mémoire m1 : **"m1y5w**

Pour coller le contenu de la mémoire m1 à un endroit donnée : **"m1p**

## 5. L'éditeur

### 5.7 Substitution

La substitution permet de remplacer automatiquement plusieurs occurrences par une autre chaîne.

**:*[1ere ligne, dernière ligne]*s/Modèle/Remplacement/[gc]**

Les numéros de lignes sont optionnels. Dans ce cas la substitution ne se fait que sur la ligne courante. En remplacement des numéros de lignes, « . » détermine la ligne courante, « 1 » la première ligne, « \$ » la dernière ligne.

Le modèle est l'un des modèles présenté plus tôt. Remplacement est une chaîne quelconque qui remplacera le modèle.

Par défaut seule la première occurrence est remplacée. La lettre « g » indique qu'il faut remplacer toutes les occurrences. Avec « c », vi demande une confirmation pour chacune des occurrences.

**:1,\$s/[Uu]nix/UNIX/g**

Cet exemple remplace, dans tout le fichier, Unix ou unix par UNIX.

# 5. L'éditeur

## 5.9 Commande set

La commande **set** permet de configurer l'éditeur.

La commande **set** sans aucun paramètre liste à la fois les variables d'environnement et les variables liées au shell. Cette commande est spécifique du Shell Unix utilisée ([bash](#), [ksh](#), [sh](#), etc.)

La commande [unset](#) permet de détruire une variable d'environnement.

- **set all** : affiche l'ensemble des options possibles
- **set number (ou nu) / nonumber (ou nonu)** : affiche / supprime les numéros de lignes.
- **set autoindent / noautoindent** : l'indentation est conservée lors d'un retour à la ligne.
- **set showmatch / noshowmatch** : lors de la saisie d'une accolade ou d'une parenthèse de fermeture, celle d'ouverture est affichée un très court instant, puis l'éditeur revient au caractère courant.
- **set showmode / noshowmode** : vi affichera une ligne d'état (INPUT MODE).
- **set tabstop=x** : définit le nombre de caractères pour une tabulation.

## 6. Redirections

- Les redirections sont l'une des plus importantes possibilités offerte par le shell.
- Par redirection, on entend la possibilité de rediriger l'affichage de l'écran vers un fichier, une imprimante ou tout autre périphérique, les messages d'erreurs vers un autre fichier, remplacer la saisie clavier par le contenu d'un fichier.
- Unix utilise des canaux d'entrées/sorties pour lire et écrire ses données. Par défaut le canal d'entrée est le clavier, et le canal de sortie, l'écran. Un troisième canal, le canal d'erreur, est aussi redirigé vers l'écran.
- Il est donc possible de rediriger ces canaux vers des fichiers, ou du flux texte de manière transparente pour les commandes Unix.

## 6. Redirections

### 6.1 En sortie

On se sert du caractère « > » pour rediriger la sortie standard (celle qui va normalement sur écran).

On indique ensuite le nom du fichier où seront placés les résultats de sortie.

```
$ ls -l > resultat.txt
```

```
$ cat resultat.txt
```

```
total 1
```

```
-rw-r--r-- 1      Administ ssh_user      0 Jul 4 12:04 TOTO  
-rw-r--r-- 1      Administ ssh_user      0 Jul 25 15:13 resultat.txt  
-rw-r--r-- 1      Administ ssh_user     171 Jul 25 15:13 test.txt
```

Si le fichier n'existe pas, il sera créé. S'il existe, son contenu sera écrasé, même si la commande tapée est incorrecte. **Le shell commence d'abord par créer le fichier puis exécute ensuite la commande.**

## 6. Redirections

Pour rajouter des données à la suite du fichier, donc sans l'écraser, on utilise la double redirection

« >> ». Le résultat est ajouté à la fin du fichier.

```
$ ls -l > resultat.txt
```

```
$ date >> resultat.txt
```

```
$ cat resultat.txt
```

```
total 1
```

```
-rw-r--r-- 1 Administ ssh_user 0 Jul 4 12:04 TOTO
```

```
-rw-r--r-- 1 Administ ssh_user 0 Jul 25 15:13 resultat.txt
```

```
-rw-r--r-- 1 Administ ssh_user 171 Jul 25 15:13 test.txt
```

```
Thu Jul 25 15:20:12 2002
```



## 6. Redirections

### 6.2 *En entrée*

Les commandes qui attendent des données ou des paramètres depuis le clavier peuvent aussi en recevoir depuis un fichier, à l'aide du caractère « < ». Un exemple avec la commande « **wc** » (word count) qui permet de compter le nombre de lignes, de mots et de caractères d'un fichier.

```
$ wc < resultat.txt
```

```
4      29    203
```

## 6. Redirections

### 6.3 *Les canaux standards*

- On peut considérer un canal comme un fichier, qui possède son propre descripteur par défaut, et dans lequel on peut lire ou écrire.
  - ✓ Le canal d'entrée standard se nomme « **stdin** » et porte le descripteur **0**.
  - ✓ Le canal de sortie standard se nomme « **stdout** » et porte le descripteur **1**.
  - ✓ Le canal d'erreur standard se nomme « **stderr** » et porte le descripteur **2**.

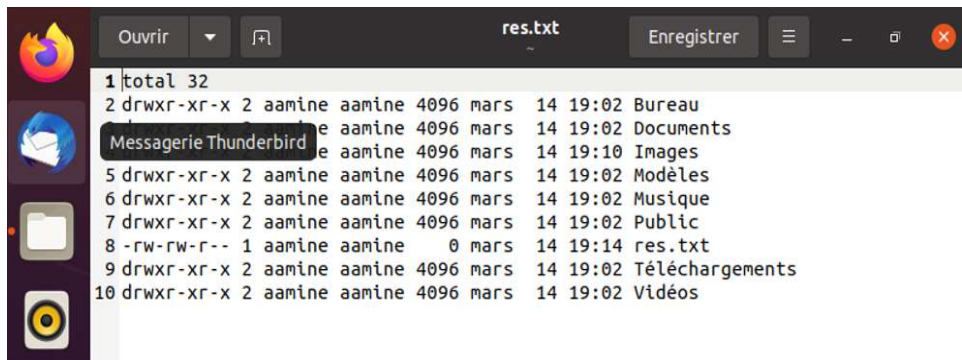
# Démonstration

```
aamine@aamine-VirtualBox:~$ ls -l
total 32
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Bureau
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Documents
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Images
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Modèles
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Musique
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Public
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Téléchargements
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Vidéos
aamine@aamine-VirtualBox:~$
```

Au lieu d'avoir le résultat dans le terminal on doit l'avoir dans le fichier

```
drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Vidéos
aamine@aamine-VirtualBox:~$ ls -l > res.txt
aamine@aamine-VirtualBox:~$
```

Le résultat sera affiché dans le fichier res.txt



The screenshot shows a file manager window titled 'res.txt' with the following content:

```
1 total 32
2 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Bureau
3 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Documents
4 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:10 Images
5 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Modèles
6 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Musique
7 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Public
8 -rw-rw-r-- 1 aamine aamine 0 mars 14 19:14 res.txt
9 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Téléchargements
10 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Vidéos
```

# Démonstration

Pour l'affichage des répertoires et des fichiers cachés

```
aamine@aamine-VirtualBox:~$ ls -l > res.txt
aamine@aamine-VirtualBox:~$ ls -la > res.txt
aamine@aamine-VirtualBox:~$
```

```
1 total 76
2 drwxr-xr-x 15 aamine aamine 4096 mars 14 19:14 .
3 drwxr-xr-x 3 root root 4096 mars 14 18:59 ..
4 -rw----- 1 aamine aamine 22 mars 14 19:04 .bash_history
5 -rw-r--r-- 1 aamine aamine 220 mars 14 18:59 .bash_logout
6 -rw-r--r-- 1 aamine aamine 3771 mars 14 18:59 .bashrc
7 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Bureau
8 drwxrwxr-x 10 aamine aamine 4096 mars 14 19:03 .cache
9 drwx----- 11 aamine aamine 4096 mars 14 19:15 .config
10 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Documents
11 drwx----- 3 aamine aamine 4096 mars 14 19:02 .gnupg
12 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:10 Images
13 drwxr-xr-x 3 aamine aamine 4096 mars 14 19:02 .local
14 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Modèles
15 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Musique
16 -rw-r--r-- 1 aamine aamine 807 mars 14 18:59 .profile
17 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Public
18 -rw-rw-r-- 1 aamine aamine 0 mars 14 19:21 res.txt
19 drwx----- 2 aamine aamine 4096 mars 14 19:02 .ssh
20 -rw-r--r-- 1 aamine aamine 0 mars 14 19:03 .sudo_as_admin_successful
21 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Téléchargements
22 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Vidéos
```

```
aamine@aamine-VirtualBox:~$ cat doc1.txt >> res.txt
aamine@aamine-VirtualBox:~$
```

```
1 total 76
2 drwxr-xr-x 15 aamine aamine 4096 mars 14 19:14 .
3 drwxr-xr-x 3 root root 4096 mars 14 18:59 ..
4 -rw----- 1 aamine aamine 22 mars 14 19:04 .bash_history
5 -rw-r--r-- 1 aamine aamine 220 mars 14 18:59 .bash_logout
6 -rw-r--r-- 1 aamine aamine 3771 mars 14 18:59 .bashrc
7 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Bureau
8 drwxrwxr-x 10 aamine aamine 4096 mars 14 19:03 .cache
9 drwx----- 11 aamine aamine 4096 mars 14 19:15 .config
10 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Documents
11 drwx----- 3 aamine aamine 4096 mars 14 19:02 .gnupg
12 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:10 Images
13 drwxr-xr-x 3 aamine aamine 4096 mars 14 19:02 .local
14 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Modèles
15 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Musique
16 -rw-r--r-- 1 aamine aamine 807 mars 14 18:59 .profile
17 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Public
18 -rw-rw-r-- 1 aamine aamine 0 mars 14 19:21 res.txt
19 drwx----- 2 aamine aamine 4096 mars 14 19:02 .ssh
20 -rw-r--r-- 1 aamine aamine 0 mars 14 19:03 .sudo_as_admin_successful
21 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Téléchargements
22 drwxr-xr-x 2 aamine aamine 4096 mars 14 19:02 Vidéos
23 bonjour estbm
```

# Sortie Erreur

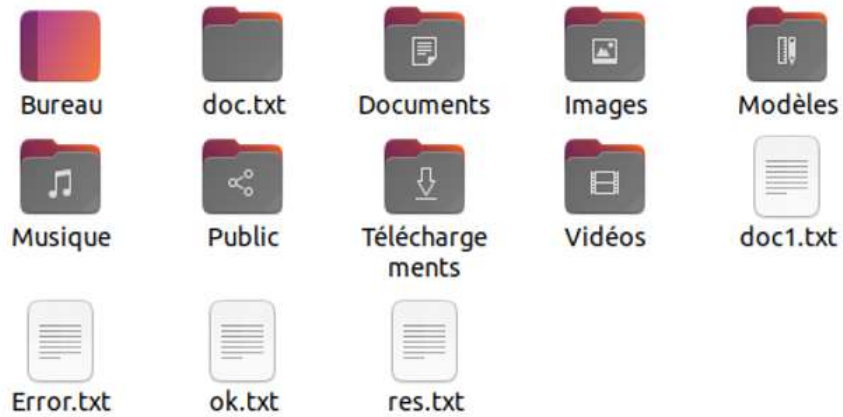
```
Activités Terminal 14 mars 20:31
aamine@aamine-VirtualBox: ~
a Rhythmbox ne-VirtualBox:~$ cat doc1.txt
bonjour es cbm
aamine@aamine-VirtualBox:~$ cat dossier.txt
cat: dossier.txt: Aucun fichier ou dossier de ce type
aamine@aamine-VirtualBox:~$ cat dossier.txt > res.txt
cat: dossier.txt: Aucun fichier ou dossier de ce type
aamine@aamine-VirtualBox:~$
```

```
Ouvrir res.txt Enregistrer
res.txt doc1.txt
1
Aide
cat: dossier.txt: Aucun fichier ou dossier de ce type
aamine@aamine-VirtualBox:~$ cat dossier.txt 2> res.txt
aamine@aamine-VirtualBox:~$
```

```
Activités Éditeur de texte 14 mars 20:35
Ouvrir res.txt Enregistrer
res.txt doc1.txt
1 cat: dossier.txt: Aucun fichier ou dossier de ce type
```

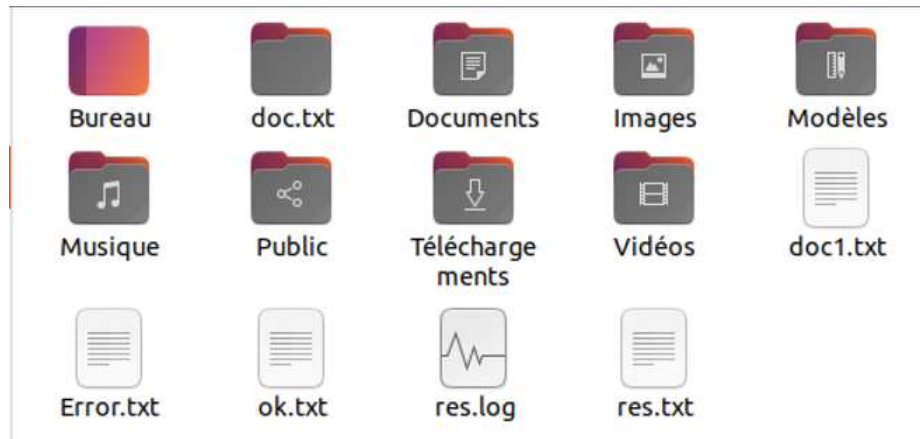
# Sortie Erreur

```
aamine@aamine-VirtualBox:~$ cat dossier.txt > ok.txt 2>Error.txt  
aamine@aamine-VirtualBox:~$
```



# Liaison de la sortie standard avec la liaison d'erreur

- `cat dossier.txt > res.txt 2>&1`



# Redirection vers l'entrée

**cat < doc.txt**

**cat doc.txt**

Même résultat sauf dans le premier c'est que doc.txt est chargé par le terminal qui va ouvrir le contenu du fichier, par contre dans le deuxième c'est la commande cat qui est responsable.



## 7. LES DROITS D'ACCÈS

### • 7.1 Utilisateurs et groupes

Le fichier **/etc/passwd** contient toutes les informations relatives aux utilisateurs (logins, mots de passe, ...).

A sa création par l'administrateur, un utilisateur se voit affecté un **UID** (« **User Identifier**») unique. Les utilisateurs sont définis dans le fichier **/etc/passwd**. De même chaque utilisateur est rattaché à au moins un groupe (groupe principal), chaque groupe possédant un identifiant unique, le **GID** (« **Group Identifier**»). Les groupes sont définis dans **/etc/group**.

La commande **id** permet d'obtenir ces informations. En interne, le système travaille uniquement avec les UID et GID, et pas avec les noms par eux-mêmes.

```
aamine02> id
```

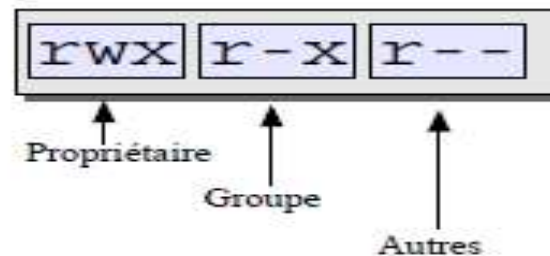
```
uid=75(oracle) gid=102(dba)
```

## 7. LES DROITS D'ACCÈS

### • 7.1 Utilisateurs et groupes

A chaque fichier (inode) sont associés un UID et un GID définissant son propriétaire et son groupe d'appartenance. On peut affecter des droits pour le propriétaire, pour le groupe d'appartenance et pour le reste du monde. On distingue de ce fait trois cas de figure:

- 1) UID de l'utilisateur identique à l'UID défini pour le fichier : cet utilisateur est propriétaire du fichier.
- 2) Les UID sont différents : le système vérifie si le GID de l'utilisateur est identique au GID du fichier : si oui l'utilisateur appartient au groupe associé au fichier.
- 3) Dans les autres cas (aucune correspondance) : il s'agit du reste du monde (others), **ni les propriétaires, ni appartenant au groupe.**



## 7. LES DROITS D'ACCÈS

- 7.2 Les droits d'accès
- Sortie de la commande ls -l :

d	rwxr-xr-x	1	oracle	dba	466	Feb 8 2001	oracle
-	rwxrwxrwx	1	oracle	dba	14536	Feb 8 2001	output.log
-	rwxr-x---	1	oracle	dba	1456	Feb 8 2001	launch.sh

Sur la première ligne du tableau, le répertoire oracle appartient à l'utilisateur oracle et au groupe dba, et possède les droits rwxr-xr-x.

Les trois types de droits:

r: Lecture (read)

w: Ecriture (write)

x: (execute)

## 7. LES DROITS D'ACCÈS

### • 7.2.1 Les droits d'accès en octal

- Les droits d'un fichier sont représentés par une chaîne de 10 caractères :

`- rxw rw- r--`

<i>Droits</i>	<i>Valeur</i>
<code>---</code>	<code>0</code>
<code>--x</code>	<code>1</code>
<code>-w-</code>	<code>2</code>
<code>-wx</code>	<code>3</code>
<code>r--</code>	<code>4</code>
<code>r-x</code>	<code>5</code>
<code>rw-</code>	<code>6</code>
<code>rwX</code>	<code>7</code>

Exemples :

```
-rw----- :  
drwxr--r-- :  
drwxr-x--- :
```

## 7. LES DROITS D'ACCÈS

- *7.3. Modification des droits*

Lors de sa création, un fichier ou un répertoire dispose de droits par défaut. On utilise la commande **chmod** (change mode) pour modifier les droits sur un fichier ou un répertoire. Il existe de méthodes pour modifier ces droits : par la forme symbolique et par la base 8. Seul le propriétaire d'un fichier peut en modifier les droits (sauf l'administrateur système).

**Le chmod sur un lien symbolique est possible comme sur tout autre fichier, mais cela ne modifie pas les droits du lien par lui-même mais les droits du fichier pointé.**

## 7. LES DROITS D'ACCÈS

- **7.3. 1 Par symboles**

La syntaxe est la suivante :

`chmod modifications Fic1 [Fic2...]`

S'il faut modifier les droits de l'utilisateur, on utilisera le caractère « **u** ». pour les droits du groupe, le caractère « **g** », pour le reste du monde le caractère « **o** », pour tous le caractère « **a** ».

Pour ajouter des droits, on utilise le caractère « **+** », pour en retirer le caractère « **-** », et pour ne pas tenir compte des paramètres précédents le caractère « **=** ».

Enfin, le droit d'accès par lui-même : « **r** », « **w** » ou « **x** ».

On peut séparer les modifications par un espace, et cumuler plusieurs droits dans une même commande.

## 7. LES DROITS D'ACCÈS

- **5.3. 1 Par symboles**

```
$ ls -l
```

```
total 0
```

```
-rw-r--r-- 1 oracle system 0 Aug 12 11:05 fic1
```

```
-rw-r--r-- 1 oracle system 0 Aug 12 11:05 fic2
```

```
-rw-r--r-- 1 oracle system 0 Aug 12 11:05 fic3
```

```
$ chmod g+w fic1
```

```
$ ls -l fic1
```

```
-rw-rw-r-- 1 oracle system 0 Aug 12 11:05 fic1
```

```
$ chmod u=rwx,g=x,o=rw fic2
```

```
$ ls -l fic2
```

```
-rwx--xrw- 1 oracle system 0 Aug 12 11:05 fic2
```

```
$ chmod o-r fic3
```

```
$ ls -l fic3
```

```
-rw-r----- 1 oracle system 0 Aug 12 11:05 fic3
```

## 7. LES DROITS D'ACCÈS

### • 7.3.2 Par base 8

La syntaxe est identique à celle des symboles. A chaque droit correspond une valeur **octale** c'est à dire de **zéro (0)** à **sept (7)**, positionnelle et cumulable.

<i>Propriétaire</i>			<i>Groupe</i>			<i>Reste du monde</i>		
<b>r</b>	<b>w</b>	<b>x</b>	<b>r</b>	<b>w</b>	<b>x</b>	<b>r</b>	<b>w</b>	<b>x</b>
400	200	100	40	20	10	4	2	1

Pour obtenir le droit final il suffit d'additionner les valeurs. Par exemple si on veut `rw-rw-rw-` alors on fera  $400+200+100+40+20+4+2=766$ , et pour `rw-r--r-`  $400+200+40+4=644$ .

```
$ chmod 766 fic1
```

```
$ ls -l fic1
```

```
-rwxr-xr-x 1 oracle system 0 Aug 12 11:05 fic1
```

```
$ chmod 644 fic2
```

```
$ ls -l fic2
```

```
-rw-r--r-- 1 oracle system 0 Aug 12 11:05 fic2
```



## 7. LES DROITS D'ACCÈS

- *7.4 Masque des droits*

Lors de la création d'un fichier ou d'un répertoire et qu'on regarde ensuite leurs droits, on obtient généralement `rw-r--r--` (644) pour un fichier et `rw-rw-rw-` (755) pour un répertoire. Ces valeurs sont contrôlées par un masque, lui-même modifiable par la commande **umask**. la commande prend comme paramètre une valeur octale qui sera soustraite aux droits d'accès maximum. Par défaut, tous les fichiers sont créés avec les droits 666 (`rw-rw-rw-`) et les répertoires avec les droits 777 (`rw-rw-rw-`), puis le masque est appliqué.

Sur la plupart des Unix, le masque par défaut est 022, soit `---w--w-`. Pour obtenir cette valeur, on tape `umask` sans paramètre.

## 7. LES DROITS D'ACCÈS

- *7.4 Masque des droits*

Pour un fichier :

Maximum rw-rw-rw- (666)

Retirer ----w--w- (022)

Reste rw-r--r-- (644)

Pour un répertoire :

Maximum rwxrwxrwx (777)

Retirer ----w--w- (022)

Reste rwxr-xr-x (755)

**ATTENTION : le calcul des droits définitifs (effectifs) n'est pas une simple soustraction de valeurs octales ! Le masque retire des droits mais n'en ajoute pas.**

Pour un fichier :

Maximum rw-rw-rw- (666)

Retirer ---r-xrw- (056)

Reste rw--w---- (620) **et PAS 610 !**

## 7. LES DROITS D'ACCÈS

- **7.4 *Changement de propriétaire et de groupe***

Il est possible de changer le propriétaire et le groupe d'un fichier à l'aide des commandes **chown** (change owner) et **chgrp** (change group).

`chown utilisateur fic1 [Fic2...]`

`chgrp groupe fic1 [Fic2...]`

- **7.5 Extractions des noms et chemins**

La commande **basename** permet d'extraire le nom du fichier dans un chemin.

`$ basename /tmp/seb/liste`

`liste`

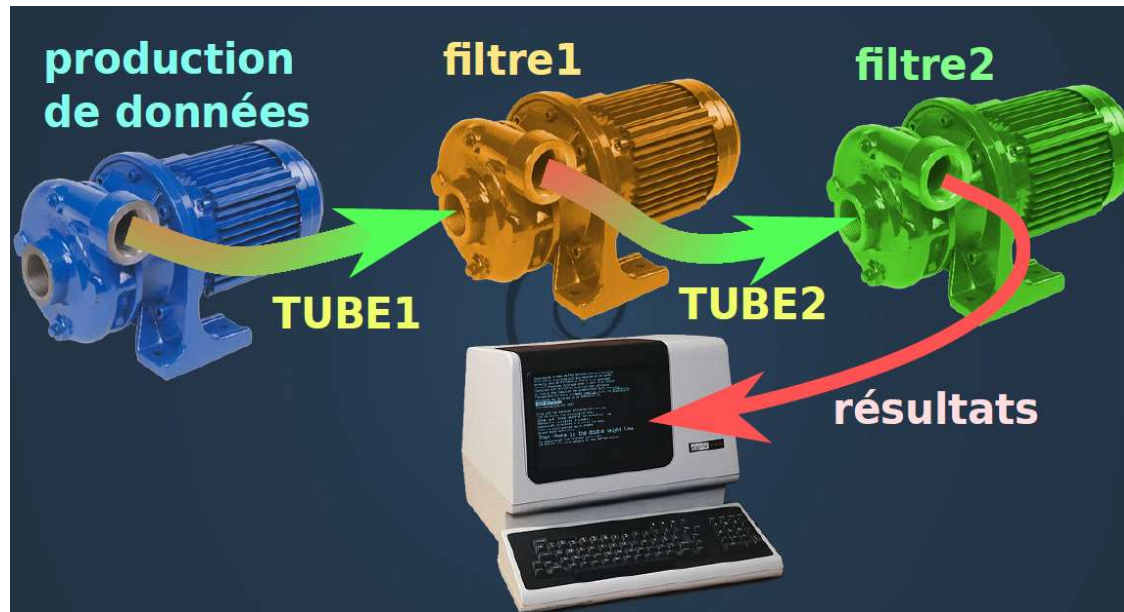
La commande **dirname** effectue l'inverse, elle extrait le chemin.

`$ dirname /tmp/seb/liste`

`/tmp/seb`

## 8- Les filtres et utilitaires

- Commandes utiles pour de petits traitements sur des données de type texte organisées en tableau
- Principe général:



## 8.1 Ordre des traitements

Dans un tube, chaque commande traite des données venant de sa gauche et fournit les résultats à sa droite

– Analogue à la composition de fonctions en maths



## 8.2 Rôles de ces filtres

- Extraire des informations dans des *textes*
  - compter les occurrences d'une information
  - comparer différentes informations
  - faire des classements, des sélections...
- Pour aller plus loin
  - il existe des outils programmables : sed, awk, perl...
- Traitement d'images avec la librairie netpbm
  - changer la taille, les couleurs...

## 8.3 Fichiers gérés par les filtres

- Les **filtres** qu'on va voir travaillent sur des *textes* (lisibles, modifiables très facilement) :
  - rarement de vrais fichiers textes
  - souvent les **résultats de commandes Unix**
- ls, ps, find, cat...
  - des **fichiers de configuration** du système :
- /etc/passwd /etc/crontab

## 8.4 Mode d'emploi général

- On utilise un tube plus ou moins complexe :

**commande | filtre1 | filtre2 | filtre3...**

- La commande envoie le texte à traiter dans les filtres
- les filtres l'altèrent chacun leur tour (**attention ordre**)

- On peut aussi utiliser des redirections :

**filtre1 < fichE > fichtmp**

**filtre2 < fichtmp > fichS**



## 8.5 les principaux filtres utilisés dans le monde UNIX

### 8.5.1 Les commandes cat, more et less

**cat f1 f2 .. > f** concatène f1, f2 .. dans le nouveau fichier f

**less f1 f2 .. > f** concatène les fichiers f1 f2 .. en un seul fichier f (comme cat)

**less f3 >> f** ajoute le contenu du fichier f3 à la suite du contenu du fichier f

## 8.5 les principaux filtres utilisés dans le monde UNIX

### 8.5.2 La commande grep

Permet de rechercher une expression dans un fichier

#### Syntaxe

**grep [options] expreg [fichiers]**

Cette commande recherche dans les fichiers ou sur son entrée standard des lignes de texte qui satisfont l'expression régulière expreg indiquée. Sa sortie peut être redirigée dans un fichier.

options

- c      donne seulement le nombre de lignes trouvées obéissant au critère
- l      donne seulement le nom des fichiers où le critère a été trouvé
- v      donne les lignes où le critère n'a pas été trouvé
- i      ne pas tenir compte de la casse (ne pas différencier majuscules minuscules)
- n      pour n'afficher que les numéros des lignes trouvées
- w      pour imposer que le motif corresponde à un mot entier d'une ligne

# Démonstration

1. Les lignes qui contiennent le mot « abc »

```
aamine@aamine-VirtualBox:~$ mkdir docs
aamine@aamine-VirtualBox:~$ cd docs
aamine@aamine-VirtualBox:~/docs$ nano f1.txt
aamine@aamine-VirtualBox:~/docs$ vi f1.txt
aamine@aamine-VirtualBox:~/docs$ cat f1.txt
azerty  abc toto 2020
TOTO    ABC user xyz
aamine@aamine-VirtualBox:~/docs$ grep "abc" f1.txt
azerty  abc toto 2020
aamine@aamine-VirtualBox:~/docs$ grep -n "abc" f1.txt
1:azerty      abc toto 2020
aamine@aamine-VirtualBox:~/docs$ grep -n -i "abc" f1.txt
1:azerty      abc toto 2020
2:TOTO  ABC user xyz
aamine@aamine-VirtualBox:~/docs$ grep -ni "abc" f1.txt
1:azerty      abc toto 2020
2:TOTO  ABC user xyz
aamine@aamine-VirtualBox:~/docs$
```

# Démonstration

1. Les lignes qui contiennent « toto »
2. Les lignes qui se terminent par « 2020 »
3. Les lignes qui se terminent avec un chiffre
4. Les lignes ne contenant pas « user »
5. Le nombre de lignes contenant l'expression « toto »
6. Les noms des fichiers contenant « user »

```
aamine@aamine-VirtualBox:~/docs$ grep -i "toto" f1.txt
azerty abc toto 2020
TOTO ABC user xyz
aamine@aamine-VirtualBox:~/docs$ grep "2020" f1.txt
azerty abc toto 2020
aamine@aamine-VirtualBox:~/docs$ grep "[0-9]$" f1.txt
azerty abc toto 2020
aamine@aamine-VirtualBox:~/docs$ grep -v -n "user" f1.txt
1:azerty abc toto 2020
aamine@aamine-VirtualBox:~/docs$ grep -c -i "toto" f1.txt
2
aamine@aamine-VirtualBox:~/docs$ grep -n "^$" f1.txt
aamine@aamine-VirtualBox:~/docs$ grep "^$" f1.txt
aamine@aamine-VirtualBox:~/docs$ grep -n"^$" f1.txt
grep : option invalide -- '^'
Usage : grep [OPTION]... MOTIFS [FICHIER]...
Exécutez « grep --help » pour obtenir des renseignements complémentaires.
aamine@aamine-VirtualBox:~/docs$ grep -n "^$" f1.txt
aamine@aamine-VirtualBox:~/docs$ cp f1 f2.doc
cp: impossible d'évaluer 'f1': Aucun fichier ou dossier de ce type
aamine@aamine-VirtualBox:~/docs$ cp f1.txt f2.doc
aamine@aamine-VirtualBox:~/docs$ grep "user"
*
^C
aamine@aamine-VirtualBox:~/docs$ grep "user"*
^C
aamine@aamine-VirtualBox:~/docs$ ls -l
total 8
-rw-rw-r-- 1 aamine aamine 39 mars 27 21:32 f1.txt
-rw-rw-r-- 1 aamine aamine 39 mars 27 21:45 f2.doc
aamine@aamine-VirtualBox:~/docs$ vi f2.doc
aamine@aamine-VirtualBox:~/docs$ grep -n "^$" f1.txt
aamine@aamine-VirtualBox:~/docs$ grep "user" *
f1.txt:TOTO ABC user xyz
f2.doc:TOTO ABC user xyz
aamine@aamine-VirtualBox:~/docs$ grep -n "^$" f1.txt
aamine@aamine-VirtualBox:~/docs$
```

# La commande find

La commande find cherche un ou plusieurs fichiers

dans une arborescence de fichiers, en se basant sur différents critères

- Exemples:

1. N'affichez que les fichiers ordinaires
2. N'afficher que les dossiers
3. Les fichiers nommés « f1 » ou « f2 »
4. Les fichiers d'extensions .txt
5. Les fichiers vides
6. Les fichiers qui ont été modifié il y'a plus que deux minutes
7. Les fichiers qui ont été modifié il y'a moins de deux jours
8. Les fichiers ayant une taille supérieure à 1000K

```
aamine@aamine-VirtualBox:~/docs$ ls
f1.doc f1.txt f2.doc f2.pdf p1.pdf tmp
aamine@aamine-VirtualBox:~/docs$ find . -type f
find: './-type': Aucun fichier ou dossier de ce type
find: 'f': Aucun fichier ou dossier de ce type
aamine@aamine-VirtualBox:~/docs$ find . -type f
./f1.doc
./f2.pdf
./f1.txt
./p1.pdf
./f2.doc
aamine@aamine-VirtualBox:~/docs$ find . -type d
```

```
./tmp
aamine@aamine-VirtualBox:~/docs$ find . -name "f1.txt"
./f1.txt
aamine@aamine-VirtualBox:~/docs$ find . -iname "f1.txt"
./f1.txt
aamine@aamine-VirtualBox:~/docs$ find. -name "*.txt"
```

La commande « find. » n'a pas été trouvée, voulez-vous dire :

```
commande « findg » du deb ncl-ncarg (6.6.2-1build4)
commande « findv » du deb polylib-utils (5.22.5-4+dfsg)
commande « find » du deb findutils (4.7.0-1ubuntu1)
```

Essayez : sudo apt install <nom du deb>

```
aamine@aamine-VirtualBox:~/docs$ find . -name "*.txt"
./f1.txt
aamine@aamine-VirtualBox:~/docs$ find . -empty
./f1.doc
./f2.pdf
./p1.pdf
./tmp
```

```
aamine@aamine-VirtualBox:~/docs$ find . -mmin +2
./f1.doc
./f2.pdf
./f1.txt
./p1.pdf
./tmp
./f2.doc
aamine@aamine-VirtualBox:~/docs$ find . -mtime -48
./f1.doc
./f2.pdf
./f1.txt
./p1.pdf
./tmp
./f2.doc
aamine@aamine-VirtualBox:~/docs$ find . -Size +1000k
find: prédicat inconnu « -Size »
aamine@aamine-VirtualBox:~/docs$ find . -size +1000k
aamine@aamine-VirtualBox:~/docs$ ls -lah
total 20K
drwxrwxr-x 3 aamine aamine 4,0K mars 27 22:16 .
drwxr-xr-x 18 aamine aamine 4,0K mars 27 21:46 ..
-rw-rw-r-- 1 aamine aamine 0 mars 27 22:13 f1.doc
-rw-rw-r-- 1 aamine aamine 39 mars 27 21:32 f1.txt
-rw-rw-r-- 1 aamine aamine 39 mars 27 21:45 f2.doc
-rw-rw-r-- 1 aamine aamine 0 mars 27 22:16 f2.pdf
-rw-rw-r-- 1 aamine aamine 0 mars 27 22:13 p1.pdf
drwxrwxr-x 2 aamine aamine 4,0K mars 27 22:13 tmp
```

# cut : sélection de colonnes

La commande **cut** présente 2 formes suivant que l'on sélectionne des colonnes de caractères ou qu'on distingue des champs séparés par un caractère précis.

sélection colonne

**cut -c(sélection\_colonnes) [fichiers]**

## Exemple

1. affiche le 5ième caractère  
cut -c5 fichier
2. affiche du 5ième au 10ème caractères  
cut -c5-10 fichier
3. affiche le 5ième et le 10ème caractères  
cut -c5-10 fichier
4. affiche à partir du 5ième (jusqu'à la fin)  
cut -c5- fichier



```
aamine@aamine-VirtualBox:~$ touch note.txt
aamine@aamine-VirtualBox:~$ vi note.txt
aamine@aamine-VirtualBox:~$ cat note.txt
Prenom  nom    age    note
Ahmed   Amine   18     12
Sara    Sonya   20     14
Mohamed Achraf  22     16
```

```
aamine@aamine-VirtualBox:~$ cut -d,-f 2
cut: le délimiteur doit être un seul caractère
Saisissez « cut --help » pour plus d'informations.
aamine@aamine-VirtualBox:~$ cut -d -f2
cut: le délimiteur doit être un seul caractère
Saisissez « cut --help » pour plus d'informations.
aamine@aamine-VirtualBox:~$ cut -d, -f2
```

```
^C
aamine@aamine-VirtualBox:~$ cut -d,-f 2 note.txt
cut: le délimiteur doit être un seul caractère
Saisissez « cut --help » pour plus d'informations.
aamine@aamine-VirtualBox:~$ cut -d, -f 2 note.txt
Prenom  nom    age    note
Ahmed   Amine   18     12
Sara    Sonya   20     14
Mohamed Achraf  22     16
```

```
aamine@aamine-VirtualBox:~$ vi note.txt
aamine@aamine-VirtualBox:~$ cut -d, -f 2 note.txt
Amine
Sonya
Achraf
```

```
aamine@aamine-VirtualBox:~$ cut -d, -f 4 note.txt
12
14
16
```

```
aamine@aamine-VirtualBox:~$ cut -d, -f 2,4 note.txt
Amine,12
```

```
aamine@aamine-VirtualBox:~/docs$ pwd
/home/aamine/docs
aamine@aamine-VirtualBox:~/docs$ cd ..
aamine@aamine-VirtualBox:~$ touch test.txt
aamine@aamine-VirtualBox:~$ vi test.txt
aamine@aamine-VirtualBox:~$ cat test.txt
A Fichiers
Sara
Sara
Sara
Mohamed
Mohamed
Ali
Naina
Karim
Said
Laila
aamine@aamine-VirtualBox:~$ cut -c 1 test.txt
A
S
S
S
M
M
A
N
K
S
L
aamine@aamine-VirtualBox:~$ cut -c 1,2,3 test.txt
Ahm
Sar
Sar
Sar
Moh
Moh
Ali
Nai
Kar
Sai
Lai
aamine@aamine-VirtualBox:~$ cut -c 1,3 test.txt
Am
Sr
Sr
Sr
Mh
Mh
```

# L'utilitaire sed

Il s'agit d'un utilitaire (*sed* = "*Stream Editor*") qui sélectionne les lignes d'un fichier texte (ou d'un flot provenant d'un pipe) vérifiant une expression régulière et qui leur applique un traitement ou un remplacement.

## Syntaxe

`sed [options] [commande] [-f fichier-commandes] fichier-source`

La commande sed est souvent utilisée pour les opérations d'éditions ci-dessus:

1. Extraire / supprimer les lignes d'un fichier
2. Trouver et remplacer des patterns
3. Insertion et suppression des données dans les fichiers



# Exemple

```
[eugene@localhost ~]$ cat salary.txt
Zorita Serrano      Software-Engineer   SanFrancisco   56      2012/06/01      $115,000
Zenaida Frank      Software-Engineer   NewYork        63      2010/01/04      $125,250
Yuri Berry         Chief-Marketing-Officer NewYork        40      2009/06/25      $675,000
Vivian Harrell     Financial-Controller SanFrancisco    62      2009/02/14      $452,500
Unity Butler       Marketing-Designer   SanFrancisco    47      2009/12/09      $85,675
Timothy Mooney     Office-Manager      London         37      2008/12/11      $136,200
Tiger Nixon        System-Architect    Edinburgh      61      2011/04/25      $320,800
Thor Walton        Developer           NewYork        61      2013/08/11      $98,540
Tatyana Fitzpatrick Regional-Director    London         19      2010/03/17      $385,750
Suki Burks         Developer           London         53      2009/10/22      $114,500

[eugene@localhost ~]$ sed -n '1p' salary.txt
Zorita Serrano      Software-Engineer   SanFrancisco   56      2012/06/01      $115,000
[eugene@localhost ~]$ sed -n '1p;5p' salary.txt
Zorita Serrano      Software-Engineer   SanFrancisco   56      2012/06/01      $115,000
Unity Butler       Marketing-Designer   SanFrancisco    47      2009/12/09      $85,675
[eugene@localhost ~]$ sed -n '1,5p' salary.txt
Zorita Serrano      Software-Engineer   SanFrancisco   56      2012/06/01      $115,000
Zenaida Frank      Software-Engineer   NewYork        63      2010/01/04      $125,250
Yuri Berry         Chief-Marketing-Officer NewYork        40      2009/06/25      $675,000
Vivian Harrell     Financial-Controller SanFrancisco    62      2009/02/14      $452,500
Unity Butler       Marketing-Designer   SanFrancisco    47      2009/12/09      $85,675
```

# La commande tr

tr est un utilitaire de ligne de commande dans les systèmes Linux et Unix qui traduit, supprime et serre les caractères de l'entrée standard et écrit le résultat dans la sortie standard.

La commande tr peut effectuer des opérations telles que la suppression de caractères répétés, la conversion de majuscules en minuscules et le remplacement et la suppression de caractères de base. En règle générale, il est utilisé en combinaison avec d'autres commandes via la tuyauterie.

## Syntaxe

**tr [options] ch1 ch2 <fich1 >fich2**

Remplace toutes les occurrences de TOUS les caractères de ch1 par le caractère de ch2, de même rang, dans le flot d'entrée.

## Exemple

```
# pour convertir et afficher la ligne saisie au clavier en minuscules
read ligne; echo $ligne | tr 'A-Z' 'a-z'

tr -c chaine car remplace tout caractère NON INCLUS dans la chaine chaine par le caractère car

# remplace supprime tous les caractères différents de a,b, ..z par un espace
echo $ligne | tr -c a-z ' '

tr -d chaine supprime tout caractère entré, appartenant à la chaine chaine

# supprime toutes les minuscules non accentuées
echo $ligne | tr -d a-z

tr -s chaine supprime toute répétition des caractères contenus dans chaine

# supprime les espaces multiples entre les mots
echo $ligne | tr -s ' '
```

# Grontab

## Introduction

Crontab est un outil qui permet de lancer des applications de façon régulière, pratique pour un serveur pour y lancer des scripts de sauvegardes, etc.

## L'installation

Bien que par défaut, il soit souvent installé, voici les commandes pour l'installer sur les différentes distributions

`apt-get install cron`

Pour être autorisé à utiliser la commande crontab, il faut que l'utilisateur soit présent dans le groupe cron.

Les fichiers `/etc/cron.allow` et `/etc/cron.deny` permettent de définir les droits d'utilisation sur crontab.

## Utiliser la commande crontab

### Afficher la liste des actions

Pour afficher le contenu du fichier **crontab** :

Code BASH :

```
crontab -l
```

### Supprimer toutes les actions

Pour supprimer toutes les actions du fichier **crontab** :

Code BASH :

```
crontab -r
```

### Editer les actions

Pour éditer les actions du fichier **crontab** :

Code BASH :

```
crontab -e
```

Le crontab s'ouvre avec un éditeur par défaut. Sous Fedora, c'est **vi**. Si on veut utiliser **nano** :


Code BASH :

```
export EDITOR=nano  
crontab -e
```

# Guide de survie de crontab : les syntaxes

Voici de manière schématique la syntaxe à respecter d'un **crontab**:

Code :


 Copier vers le presse-papier

```
# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user command to be executed
```

Le fichier de configuration de est constitué des différentes lignes. Chaque ligne correspond à une action.

Prenons l'exemple suivant :

Code BASH :

 Copier vers le presse-papier

```
mm hh jj MMM JJJ [user] tâche > log
```

- **mm** : minutes (00-59).
- **hh** : heures (00-23) .
- **jj** : jour du mois (01-31).
- **MMM** : mois (01-12 ou abréviation anglaise sur trois lettres : jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec).
- **JJJ** : jour de la semaine (1-7 ou abréviation anglaise sur trois lettres : mon, tue, wed, thu, fri, sat, sun).
- **user (facultatif)** : nom d'utilisateur avec lequel exécuter la tâche.
- **tâche** : commande à exécuter.
- **> log** (facultatif) : redirection de la sortie vers un fichier de log. Si un fichier de log n'est pas spécifié, un mail sera envoyé à l'utilisateur local.

Pour chaque unité, on peut utiliser les notations suivantes :

- **1-5** : les unités de temps de 1 à 5.
- **\*/6** : toutes les 6 unités de temps (toutes les 6 heures par exemple).
- **2,7** : les unités de temps 2 et 7.

# Exemple

Exécution tous les jours à 22h00 d'une commande et rediriger les infos dans sauvegarde.log :

Code BASH :

```
00 22 * * * /root/scripts/sauvegarde.sh >> sauvegarde.log
```

Exécution d'une commande toutes les 6 heures :

Code BASH :

```
00 */6 * * * /root/scripts/synchronisation-ftp.sh
```

Exécution d'une commande toute les heures :

Code BASH :

```
00 */1 * * * /usr/sbin/ntpdate fr.pool.ntp.org
```

Exécution d'une commande toutes les minutes uniquement les lundis :

Code BASH :

```
* * * * 1 /root/script/commandes-du-lundi.sh
```

Exécution d'une commande une fois par an à une heure précise (ici le 25 décembre à 00h15) :

Code BASH :

```
15 00 25 12 * echo "Le père Noël est passé !"
```

Exécuter chaque jour, de chaque mois à 2:15 la commande eix-sync

Code BASH :

```
15 02 * * * /usr/bin/eix-sync
```

## Autres

D'autres commandes similaires existent :

- `fgrep` (fast grep) : plus rapide en exécution, mais ne permettant pas l'utilisation de métacaractères dans l'expression à chercher ;
- `egrep` (enhanced grep) : élargit les possibilités de recherche en donnant accès à d'autres métacaractères pour spécifier l'expression. Cette commande revient à utiliser l'option « -E » de la commande « `grep` » ;
- `awk` : (Alfred Aho, Peter Weinberger et Brian Kernighan) utilise un script de programmation dans son propre langage (ressemblant beaucoup au langage « C ») pour traiter les données entrantes selon l'algorithme programmé par l'utilisateur.

# 9-PROGRAMMATION SHELL

## 9.1 Notre premier script

- **Création du fichier**

Commençons par créer un nouveau fichier pour notre script. Le plus simple est d'ouvrir Vim en lui donnant le nom du nouveau fichier à créer :

```
$ vim essai.sh
```

- **Indiquer le nom du shell utilisé par le script**

La première chose à faire dans un script shell est d'indiquer... quel shell est utilisé.

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

- **Exécuter le script bash**

**Donner les droits d'exécution au script**

**Il faut vérifier avec ls -l**

```
$ chmod +x essai.sh
```

- **Exécution du script**

```
$ ./essai.sh
```

- **Exécution de débogage**

- ```
$ bash -x essai.sh
```



## 9.1 Afficher et manipuler des variables

- **Déclarer une variable**

```
$ vim variables.sh
```

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

```
message='Bonjour tout le monde '
```

```
message='Bonjour c'est moi'
```

**Echo: afficher une variable**

```
$ echo Salut tout le monde
```

```
$ echo "Salut tout le monde
```

**retourner a la ligne**

```
$ echo -e "$Message\nAutre ligne"
```

## 9.1.1 Afficher et manipuler des variables

- `#!/bin/bash`
- `message='Bonjour tout le monde'`
- `echo $message`
- on veuille afficher à la fois du texte et la variable

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

```
message='Bonjour tout le monde'
```

```
echo 'Le message est : ' $message
```

## 9.1.2 read : demander une saisie

- **read nomvariable**

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

```
read nom
```

```
echo "Bonjour $nom !"
```

**Affecter simultanément une valeur à plusieurs variables**

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

```
read nom prenom
```

```
echo "Bonjour $nom $prenom !"
```

## 9.1.2 read : demander une saisie

- afficher un message de prompt: -p

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

```
read -p 'Entrez votre nom : ' nom
```

```
echo "Bonjour $nom !"
```

## 9.1.2 read : demander une saisie

- **limiter le nombre de caractères: -n**

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

```
read -p 'Entrez votre login (5 caractères max) : ' -n 5 nom
```

```
echo "Bonjour $nom !"
```

## 9.1.2 read : demander une saisie

**limiter le temps autorisé pour saisir un message: -t**

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

```
read -p 'Entrez le code de désamorçage de la bombe (vous avez 5 secondes) : ' -t 5 code
```

```
echo -e "\nBoum !"
```

## 9.1.3 Effectuer des opérations mathématiques

le bash n'est pas vraiment capable de manipuler des nombres

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

```
let "a = 5"
```

```
let "b = 2"
```

```
let "c = a + b"
```

```
echo $c
```

## 9.2 Les variables d'environnement

Les variables d'environnement sont des variables que l'on peut utiliser dans n'importe quel programme. On parle aussi parfois de **variables globales**

env



## 9.3 Les variables des paramètres

Comme toutes les commandes, vos scripts bash peuvent eux aussi accepter des paramètres. Ainsi, on pourrait appeler notre script comme ceci :

`./variables.sh param1 param2 param3`

Pour récupération des paramètres

En effet, des variables sont automatiquement créées :

- `$#` : contient le nombre de paramètres ;
- `$0` : contient le nom du script exécuté (ici `./variables.sh`) ;
- `$1` : contient le premier paramètre ;
- `$2` : contient le second paramètre ;
- ... ;
- `$9` : contient le 9e paramètre.

## 9.3.1 Les variables des paramètres

```
#!/bin/bash

echo "Vous avez lancé $0, il y a $# paramètres"
echo "Le paramètre 1 est $1"
```

```
$ ./variables.sh param1 param2 param3
Vous avez lancé ./variables.sh, il y a 3 paramètres
Le paramètre 1 est param1
```

### 9.3.2 Les variables des paramètres

Un script qui accepte une liste de fichiers en paramètre.

`./script.sh fichier1 fichier2 fichier3 fichier4 ... fichier14 fichier15`

En général, pour traiter autant de paramètres, on s'occupera d'eux un par un... On peut « décaler » les paramètres dans les variables `$1` , `$2` , etc. à l'aide de la commande `shift` .

## 9.3.2 Les variables des paramètres

```
#!/bin/bash  
  
echo "Le paramètre 1 est $1"  
shift  
echo "Le paramètre 1 est maintenant $1"
```

```
$ ./variables.sh param1 param2 param3  
Le paramètre 1 est param1  
Le paramètre 1 est maintenant param2
```

## 9.4 Les tableaux

Pour définir un tableau, on peut faire comme ceci :

```
tableau=('valeur0' 'valeur1' 'valeur2')
```

Pour accéder à une case du tableau, il faut utiliser la syntaxe suivante :

```
echo ${tableau[2]}
```

Vous pouvez aussi définir manuellement le contenu d'une case :

```
tableau[2]='valeur2'
```

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

```
tableau=('valeur0' 'valeur1' 'valeur2')
```

```
tableau[5]='valeur5'
```

```
echo ${tableau[1]}
```

Vous pouvez afficher l'ensemble du contenu du tableau d'un seul coup en utilisant

```
${tableau[*]}
```

## 9.5 Les conditions

**if : la condition la plus simple**

**if [ test ]**

**then**

**echo "C'est vrai"**

**fi**

Vous noterez — c'est très important — qu'il y a des espaces à l'intérieur des crochets. On ne doit pas écrire[test]  
mais [ test ] !

Autre facon

**if [ test ]; then**

**echo "C'est vrai"**

**fi**

## 9.5 Les conditions

Nous testons la valeur d'une chaîne de caractères

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

```
nom="Bruno"
```

```
if [ $nom = "Bruno" ]
```

```
then
```

```
    echo "Salut Bruno !"
```

```
fi
```

## 9.5 Les conditions

Test de deux variables

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

```
nom1="Bruno"
```

```
nom2="Marcel"
```

```
if [ $nom1 = $nom2 ]
```

```
then
```

```
    echo "Salut les jumeaux !"
```

```
fi
```



## 9.5 Les conditions

Le script avec else

```
if [ test ]
```

```
then
```

```
    echo "C'est vrai"
```

```
else
```

```
    echo "C'est faux"
```

```
fi
```

## 9.5 Les conditions

- **Exemple 1 avec else**

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

```
nom="Bruno"
```

```
if [ $nom = "Bruno" ]
```

```
then
```

```
    echo "Salut Bruno !"
```

```
else
```

```
    echo "J'te connais pas, ouste !"
```

```
fi
```

## 9.5 Les conditions

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

```
if [ $1 = "Bruno" ]  
then  
    echo "Salut Bruno !"  
else  
    echo "J'te connais pas, ouste !"  
fi
```

Test

```
$ ./script.sh Bruno  
Salut Bruno !
```

```
$ ./script.sh Jean  
J'te connais pas, ouste !
```

## 9.5 Les conditions

### **Sinon si**

```
if [ test ]  
then  
    echo "Le premier test a été vérifié"  
elif [ autre_test ]  
then  
    echo "Le second test a été vérifié"  
elif [ encore_autre_test ]  
then  
    echo "Le troisième test a été vérifié"  
else  
    echo "Aucun des tests précédents n'a été vérifié"  
fi
```

## 9.5 Les conditions

Notre script

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

```
if [ $1 = "Bruno" ]
```

```
then
```

```
    echo "Salut Bruno !"
```

```
elif [ $1 = "Michel" ]
```

```
then
```

```
    echo "Bien le bonjour Michel"
```

```
elif [ $1 = "Jean" ]
```

```
then
```

```
    echo "Hé Jean, ça va ?"
```

```
else
```

```
    echo "J'te connais pas, ouste !"
```

```
fi
```

## 9.6 Les tests

### Les différents types de tests

Il est possible d'effectuer trois types de tests différents en bash :

1. des tests sur des chaînes de caractères ;
2. des tests sur des nombres ;
3. des tests sur des fichiers.

## 9.6.1 Tests sur des chaînes de caractères

Vérifions par exemple si deux paramètres sont différents :

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

```
if [ $1 != $2 ]
```

```
then
```

```
    echo "Les 2 paramètres sont différents !"
```

```
else
```

```
    echo "Les 2 paramètres sont identiques !"
```

```
fi
```

```
$ ./test.sh Bruno Bernard
```

```
Les 2 paramètres sont différents !
```

```
$ ./test.sh Bruno Bruno
```

```
Les 2 paramètres sont identiques !
```

## 9.6.2 Tests sur des chaînes de caractères

| Condition                           | Signification                                                                                                                                                                                     |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\$chaine1 = \$chaine2</code>  | Vérifie si les deux chaînes sont identiques. Notez que bash est sensible à la casse : « b » est donc différent de « B ».<br>Il est aussi possible d'écrire « == » pour les habitués du langage C. |
| <code>\$chaine1 != \$chaine2</code> | Vérifie si les deux chaînes sont différentes.                                                                                                                                                     |
| <code>-z \$chaine</code>            | Vérifie si la chaîne est vide.                                                                                                                                                                    |
| <code>-n \$chaine</code>            | Vérifie si la chaîne est non vide.                                                                                                                                                                |



## 9.6.2 Tests sur des chaînes de caractères

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

```
if [ -z $1 ]
```

```
then
```

```
    echo "Pas de paramètre"
```

```
else
```

```
    echo "Paramètre présent "
```

```
fi
```

```
$ ./test.sh
```

Pas de paramètre

```
$ ./test.sh param
```

Paramètre présent

## 9.4 Tests sur des nombres

| Condition                       | Signification                                                                                                                                                        |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\$num1 - eq \$num2</code> | Vérifie si les nombres sont égaux ( <b>equal</b> ). À ne pas confondre avec le « = » qui, lui, compare deux chaînes de caractères.                                   |
| <code>\$num1 - ne \$num2</code> | Vérifie si les nombres sont différents ( <b>not equal</b> ). Encore une fois, ne confondez pas avec « != » qui est censé être utilisé sur des chaînes de caractères. |
| <code>\$num1 - lt \$num2</code> | Vérifie si <code>num1</code> est inférieur ( < ) à <code>num2</code> ( <b>lower than</b> ).                                                                          |
| <code>\$num1 - le \$num2</code> | Vérifie si <code>num1</code> est inférieur ou égal ( <= ) à <code>num2</code> ( <b>lower or equal</b> ).                                                             |
| <code>\$num1 - gt \$num2</code> | Vérifie si <code>num1</code> est supérieur ( > ) à <code>num2</code> ( <b>greater than</b> ).                                                                        |
| <code>\$num1 - ge \$num2</code> | Vérifie si <code>num1</code> est supérieur ou égal ( >= ) à <code>num2</code> ( <b>greater or equal</b> ).                                                           |

## 9.4.1 Tests sur des nombres

- Vérifions par exemple si un nombre est supérieur ou égal à un autre nombre :

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

```
if [ $1 -ge 20 ]
```

```
then
```

```
    echo "Vous avez envoyé 20 ou plus"
```

```
else
```

```
    echo "Vous avez envoyé moins de 20"
```

```
fi
```

## 9.5 Tests sur des fichiers

| Condition                              | Signification                                                                                                                    |
|----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <code>-e \$nomfichier</code>           | Vérifie si le fichier existe.                                                                                                    |
| <code>-d \$nomfichier</code>           | Vérifie si le fichier est un répertoire. N'oubliez pas que sous Linux, tout est considéré comme un fichier, même un répertoire ! |
| <code>-f \$nomfichier</code>           | Vérifie si le fichier est un... fichier. Un vrai fichier cette fois, pas un dossier.                                             |
| <code>-L \$nomfichier</code>           | Vérifie si le fichier est un lien symbolique (raccourci).                                                                        |
| <code>-r \$nomfichier</code>           | Vérifie si le fichier est lisible (r).                                                                                           |
| <code>-w \$nomfichier</code>           | Vérifie si le fichier est modifiable (w).                                                                                        |
| <code>-x \$nomfichier</code>           | Vérifie si le fichier est exécutable (x).                                                                                        |
| <code>\$fichier1 -nt \$fichier2</code> | Vérifie si <code>fichier1</code> est plus récent que <code>fichier2</code> ( <b>n</b> ewer <b>t</b> han).                        |
| <code>\$fichier1 -ot \$fichier2</code> | Vérifie si <code>fichier1</code> est plus vieux que <code>fichier2</code> ( <b>o</b> lder <b>t</b> han).                         |

## 9.5.1 Tests sur des fichiers

- faire un script qui demande à l'utilisateur d'entrer le nom d'un répertoire et qui vérifie si c'en est bien un :

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

```
read -p 'Entrez un répertoire : ' repertoire
```

```
if [ -d $repertoire ]
```

```
then
```

```
    echo "Bien, vous avez compris ce que j'ai dit !"
```

```
else
```

```
    echo "Vous n'avez rien compris..."
```

```
fi
```

## 9.5.2 Effectuer plusieurs tests à la fois

Les deux symboles à connaître sont :

**&&** : signifie « et » ;

**||** : signifie « ou ».

**#!/bin/bash**

Il faut encadrer chaque condition par des crochets. Prenons un exemple :

```
if [ $# -ge 1 ] && [ $1 = 'koala' ]
```

```
then
```

```
    echo "Bravo !"
```

```
    echo "Vous connaissez le mot de passe"
```

```
else
```

```
    echo "Vous n'avez pas le bon mot de passe"
```

```
fi
```

## 9.5.3 Inverser un test

- Il est possible d'inverser un test en utilisant la négation. En bash, celle-ci est exprimée par le point d'exclamation « ! ».

```
if [ ! -e fichier ]  
then  
    echo "Le fichier n'existe pas"  
fi
```

## 9.5.4 case : tester plusieurs conditions à la fois

- `#!/bin/bash`

```
case $1 in
    "Bruno")
        echo "Salut Bruno !"
        ;;
    "Michel")
        echo "Bien le bonjour Michel"
        ;;
    "Jean")
        echo "Hé Jean, ça va ?"
        ;;
    *)
        echo "J'te connais pas, ouste !"
        ;;
esac
```



## 9.5.5 case : tester plusieurs conditions à la fois

Nous pouvons aussi faire des « ou » dans un case. Dans ce cas, petit piège, il ne faut pas mettre deux || mais un seul ! Exemple :

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

```
case $1 in
    "Chien" | "Chat" | "Souris")
        echo "C'est un mammifère"
        ;;
    "Moineau" | "Pigeon")
        echo "C'est un oiseau"
        ;;
    *)
        echo "Je ne sais pas ce que c'est"
        ;;
esac
```

## 9.6 Les boucles

**while : boucler « tant que »**

```
while [ test ]
```

```
do
```

```
    echo 'Action en boucle'
```

```
done
```

## 9.6.1 Les boucles

- On va demander à l'utilisateur de dire « oui » et répéter cette action tant qu'il n'a pas fait ce que l'on voulait. Nous allons créer un script boucles.sh pour l'occasion

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

```
while [ -z $reponse ] || [ $reponse != 'oui' ]  
do  
    read -p 'Dites oui : ' reponse  
done
```

On fait deux tests.

1.Est-ce que \$reponse est vide ?

2.Est-ce que \$reponse est différent de oui ?

Comme il s'agit d'un OU (||), tant que l'un des deux tests est vrai, on recommence la boucle. Cette dernière pourrait se traduire par : « Tant que la réponse est vide ou que la réponse est différente de oui ». Nous sommes obligés de vérifier d'abord si la variable n'est pas vide, car si elle l'est, le second test plante

## 9.6.2 Les boucles

Il existe aussi le mot clé `until`, qui est l'inverse de `while`. Il signifie « Jusqu'à ce que ».

Remplacez juste `while` par `until` dans le code précédent pour l'essayer

## 9.6.3 for : boucler sur une liste de valeurs

**Parcourir une liste de valeurs**

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

```
for variable in 'valeur1' 'valeur2' 'valeur3'
```

```
do
```

```
    echo "La variable vaut $variable"
```

```
done
```

## 9.6.4 for : boucler sur une liste de valeurs

- Faire un script qui liste tous les fichiers trouvés dans le répertoire actuel :

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

```
liste_fichiers=`ls`
```

```
for fichier in $liste_fichiers
```

```
do
```

```
    echo "Fichier trouvé : $fichier"
```

```
done
```

## 9.6.5 for : boucler sur une liste de valeurs

On pourrait faire un code plus court sans passer par une variable

```
#!/bin/bash  
for fichier in `ls`  
do  
    echo "Fichier trouvé : $fichier"  
done
```

## 9.7 Les fonctions

### 9.7.1 Introduction

#### Syntaxe

nom\_de\_fonction()

{

commande1

[ commande2 ?]

}

# ?

nom\_de\_fonction



## 9.7 Les fonctions

### 9.7.1 Introduction

- Une fonction permet de regrouper des instructions fréquemment employées dans un ensemble identifié par un nom.
- Ce nom, utilisé ensuite dans le script comme toute autre commande Unix, exécutera l'ensemble des instructions contenues dans la fonction. Une fois le corps de la fonction créé, il n'y a aucune différence entre « appeler une fonction » et « appeler une commande Unix ».

## 9.7 Les fonctions

- En Bash, il y a deux manières pour déclarer une fonction :

# déclaration méthode 1

maFonction ()

{

bloc d'instructions

}

#appel de ma fonction

maFonction

# déclaration méthode 2

function maFonction

{ bloc d'instructions }

#appel de la fonction

maFonction

## 9.7 Les fonctions

- **Exemple**

```
#!/bin/sh
```

**# Fonction qui affiche la date puis fait un "ls"**

```
newls()
```

```
{
```

```
    date # Affichage de la date
```

```
    ls -l # Affichage du ls
```

```
}
```

**# Utilisation de la fonction**

```
newls # Appel de la fonction
```

```
var1=`newls` # Récupération de ce que la fonction affiche
```

**# Utilisation d'une commande Unix**

```
ls -l # Appel de la commande classique "ls -l"
```

```
var2=`ls -l` # Récupération de ce que la commande "ls -l" affiche
```

**# Il n'y a aucune différence syntaxique entre l'utilisation d'une fonction ou d'une commande?**