Enseignement d'Informatique : Unix

Titre :

« Introduction aux systèmes d'exploitation
Cas d'étude : système UNIX »

Objectif :

Etudier la structure du système d'exploitation Unix et manipuler son langage de commande SHELL

Remerciement :

Je remercie Malik Mallem, avec lequel j'ai eu le plaisir d'assurer l'enseignement d'Unix à l'IUP, de m'avoir permis de reprendre son support de cours et de pouvoir le modifier.

Auteur : Samir Otmane

Contenu :

Bibliographie :

Maurice J. Bach "Conception du système UNIX" Masson paris 1989
Jean-Marie Rifflet "La programmation sous UNIX" Ediscience International 1998

1 Introduction aux systèmes d'exploitation

Sans ses logiciels, un ordinateur n'est qu'un morceau de métal inutile.

Les logiciels se répartissent en 2 grandes catégories :

Programme système :

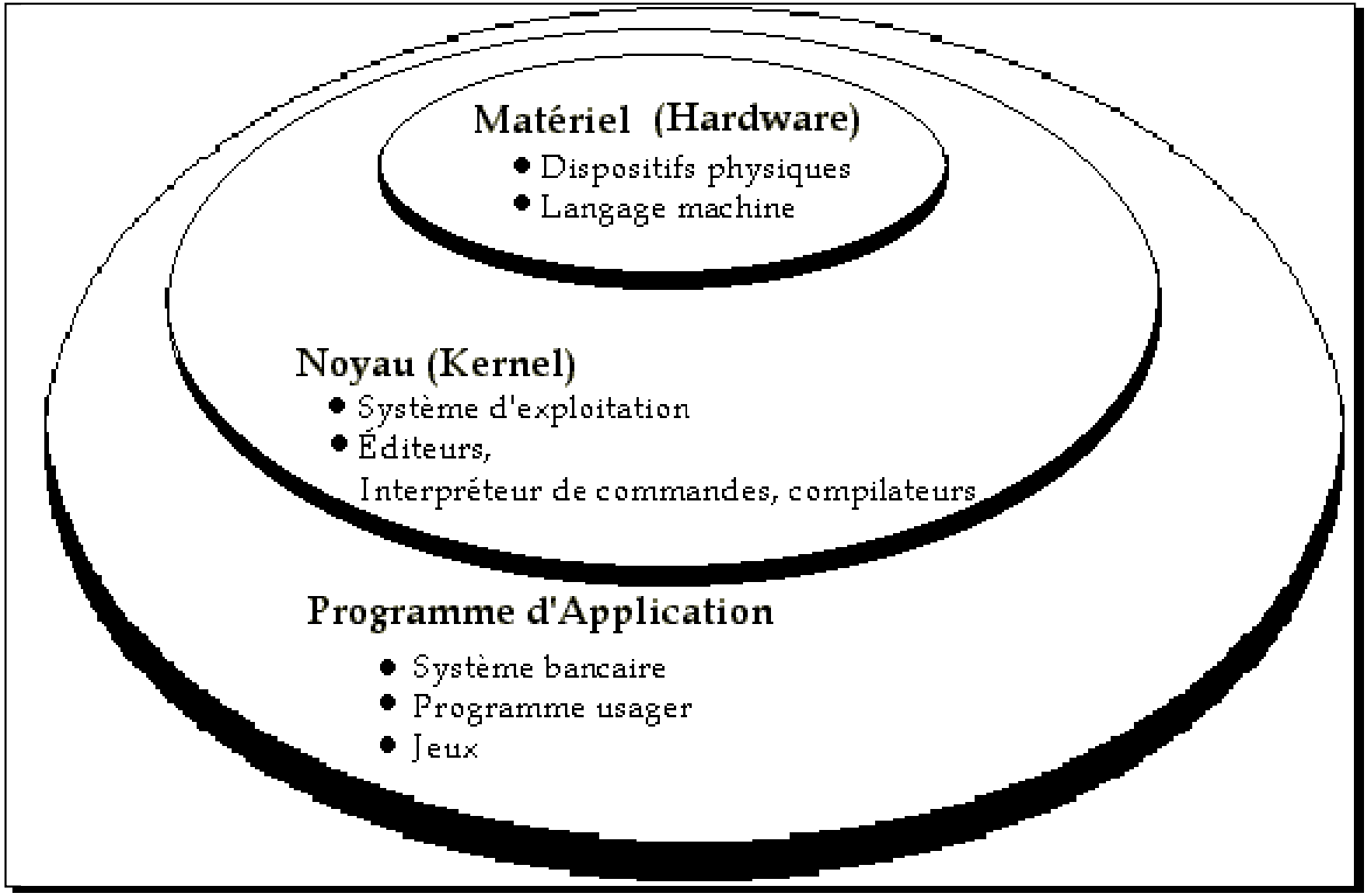
Ensemble de programmes qui permettent de gérer les principales fonctions d'un ordinateur.

Programme d'applications :

Ensemble de programmes qui effectuent les principales tâches de l'usager.

Le programme système le plus fondamental est le ***système d'exploitation***

Le **système d'exploitation** se situe dans la couche « **programme système** ». Il y a 3 grandes couches qui caractérisent les ordinateurs. La figure ci-contre illustre ces différentes couches.



1.1 Rôle et utilité d'un système d'exploitation

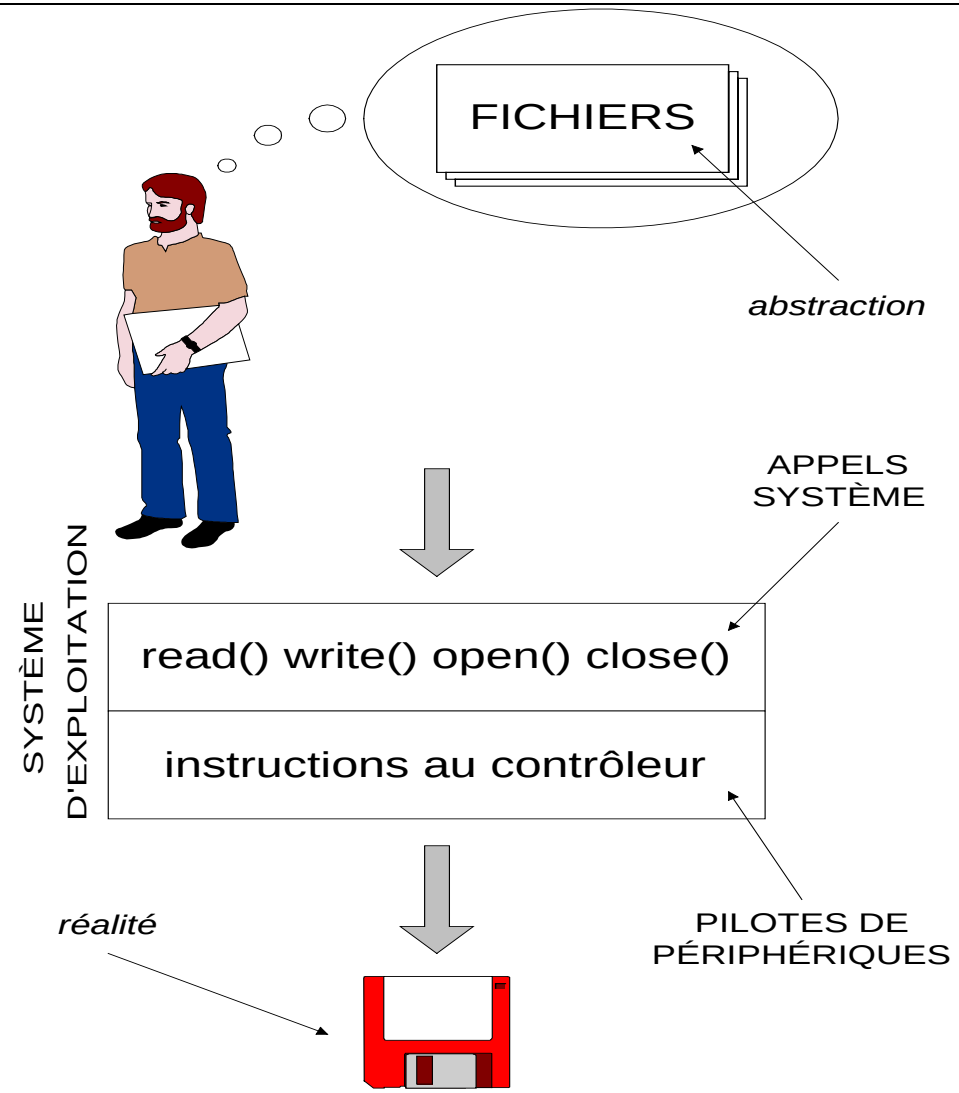
Les systèmes d'exploitation peuvent être vus comme :

1. Machine étendue
2. Gestionnaire de ressources.

Une des fonctions du système d'exploitation est de présenter à l'utilisateur l'équivalent d'une **machine virtuelle** ou **machine étendue**.

1.1.1 Système d'exploitation en tant que machine étendue

Le système d'exploitation (SE) est un ensemble de programmes qui fournit une abstraction à l'utilisateur. Le SE masque le matériel aux regards de l'utilisateur et offre une vue simple et agréable des ressources. Par exemple lors de la lecture/écriture de fichiers sur disquette, le SE peut se réduire par un ensemble d'appels système (read(), write(), etc) vers les pilotes de périphériques (instructions au contrôleur).



Vu sous cet aspect, le rôle du système d'exploitation est de présenter à l'utilisateur l'équivalent d'une machine virtuelle plus facile à programmer que le matériel.

1.1.2 Le système d'exploitation vu comme gestionnaire de ressources

Le système d'exploitation est un ensemble de programmes qui sert à gérer les différentes fonctions d'un système complexe.

L'autre rôle primordial d'un système d'exploitation est de connaître à tout moment l'utilisateur d'une ressource, de gérer les accès à cette ressource, d'en accorder l'usage et d'éviter les conflits d'accès entre les différents programmes ou entre les utilisateurs.

Le système d'exploitation agit aussi comme un gestionnaire de ressources qui doit s'occuper des quatre aspects suivants :

équipements nombreux
⇩
multiprogrammation (plusieurs tâches à la fois)
⇩
temps partagé (plusieurs utilisateurs à la fois)
⇩
situation complexe à gérer!

1)la gestion des processus
2)la gestion de la mémoire
3)des entrées-sorties
4)le système de fichiers

1.2 Historique des systèmes d'exploitation

- **Babbage et le 1er ordinateur**
  - Le premier véritable ordinateur a été construit par le mathématicien anglais **Charles Babbage** (1792-1871).

Il n'y avait évidemment aucun système d'exploitation dans cette machine.

- **1ère génération (1945-55) :**
  - Les tubes à vide et les cartes perforées.
  - Programmation en langage machine

- **2ème génération (1955-1965) :**
  - Les transistors et le traitement par lots
  - Programmation FORTRAN et Assembleur
  - Quelques systèmes d'exploitation de la deuxième génération : FMS (Fortran Monitor System) et IBSYS (sur IBM 7094).

Premier système d'exploitation :

IBSYS et FMS.

• 3ème génération (1965-1980) :

- les circuits intégrés et la multiprogrammation
- Système d'exploitation **OS/360** faisait appel à la **multiprogrammation**.
- 1er système à temps partagé: **MULTICS (MULT**iplexed **I**nformation and **C**omputing **S**ervice)

- Développement sur un mini-ordinateur "PDP-7" de **UNICS** (**UN**iplexed **I**nformation and **C**omputing **S**ervice) qui devient par la suite **UNIX. Unics** était à l'époque un système d'exploitation **mono-utilisateur**. C'est aussi l'époque où est apparue le **langage C**.

Systèmes d'exploitation :

- **OS/360**
- **MULTICS**
- **UNIX**

• 4ème génération (1980-): Ordinateurs Personnels

C'est l'époque des circuits LSI *(Large Scale Integration)* et VLSI *(Very Large Scale Integration)*. Les puces contiennent maintenant des centaines de milliers de transistors.

Cette formidable miniaturisation des composantes a rendu possible la mise en marché d'ordinateurs personnels *(Personnal Computer, PC)*.

Deux systèmes d'exploitation dominent:

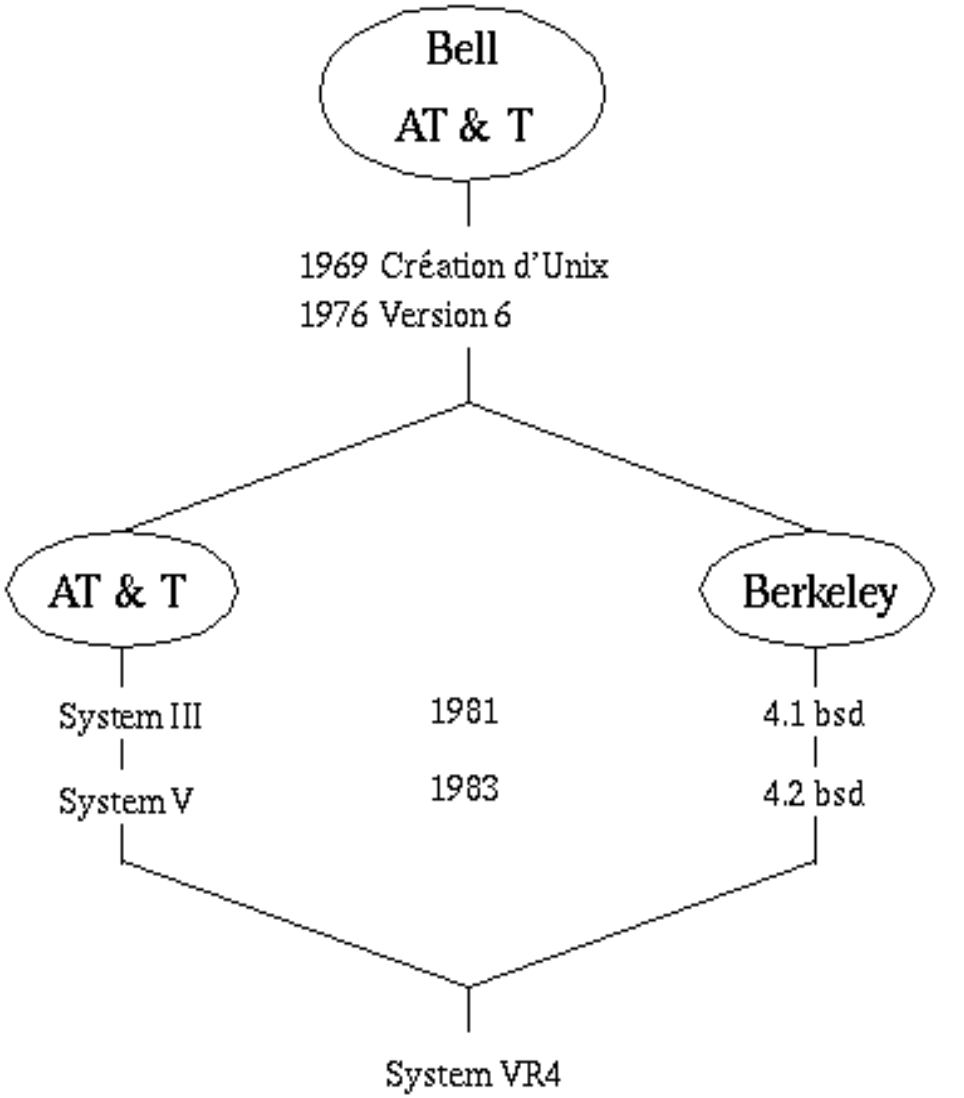
MS-DOS Mis au point pour les processeurs Intel 80x86, il est le plus utilisé dans le monde avec près de 20 millions d'installations. Il intègre petit à petit des éléments de UNIX.

UNIX Il est le système supporté par le plus grand nombre de fabricants de matériel et de logiciel.

2. Le système UNIX : aspect utilisateur

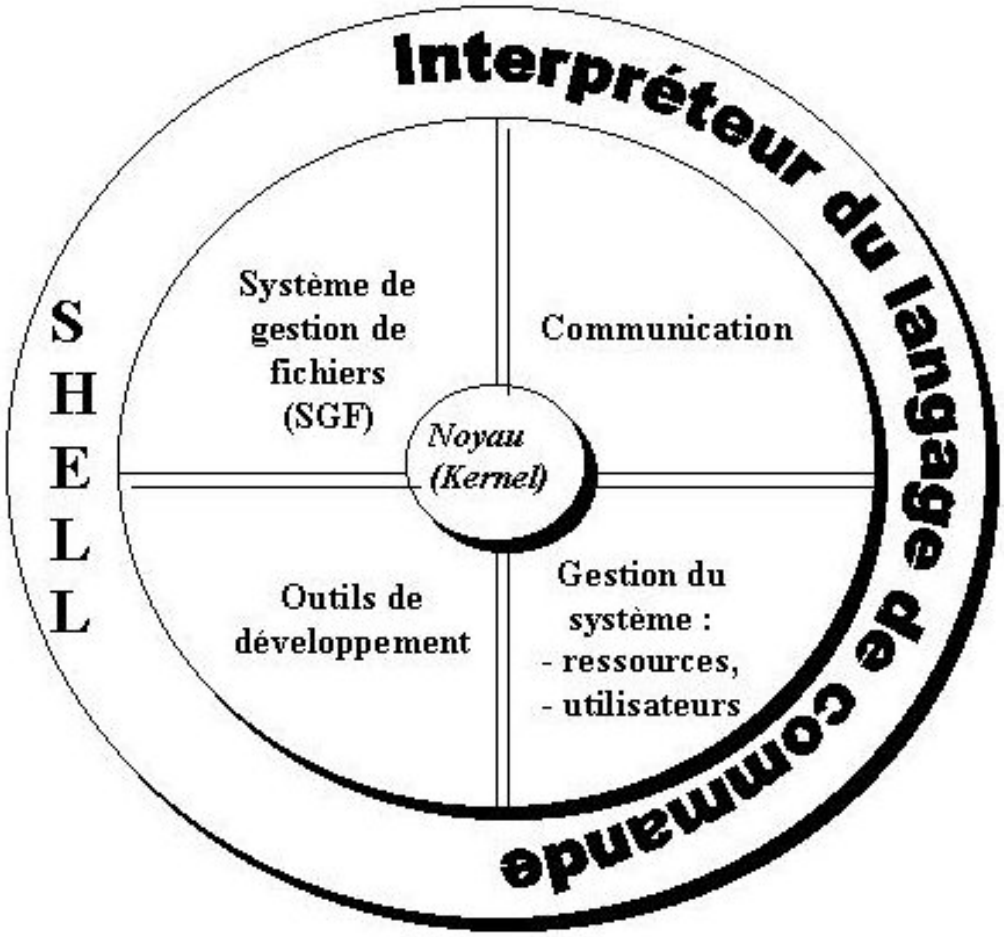
2.1 Un peu d'histoire

- UNIX créé au Laboratoire BELL, USA, en 1969.
- Destiné à la gestion d'un mini-ordinateur pour une petite équipe de programmeurs.
- Intéresse rapidement de nombreuses universités puis des constructeurs.
- Deux principales familles de systèmes UNIX (1983): **Berkeley(BSD)** et **System V** de Bell.
- De nombreux efforts de normalisation : norme System V , **POSIX**(1988), OSF
- De nombreuses versions d'UNIX sont donc apparues :
  - ULTRIX (BSD) puis OSF sur DIGITAL, IRIX(System V) sur Silicon Graphics,
  - LINUX(POSIX - 1991) sur PC, et bien d'autres.



2.2 Caractéristiques et architecture du système Unix

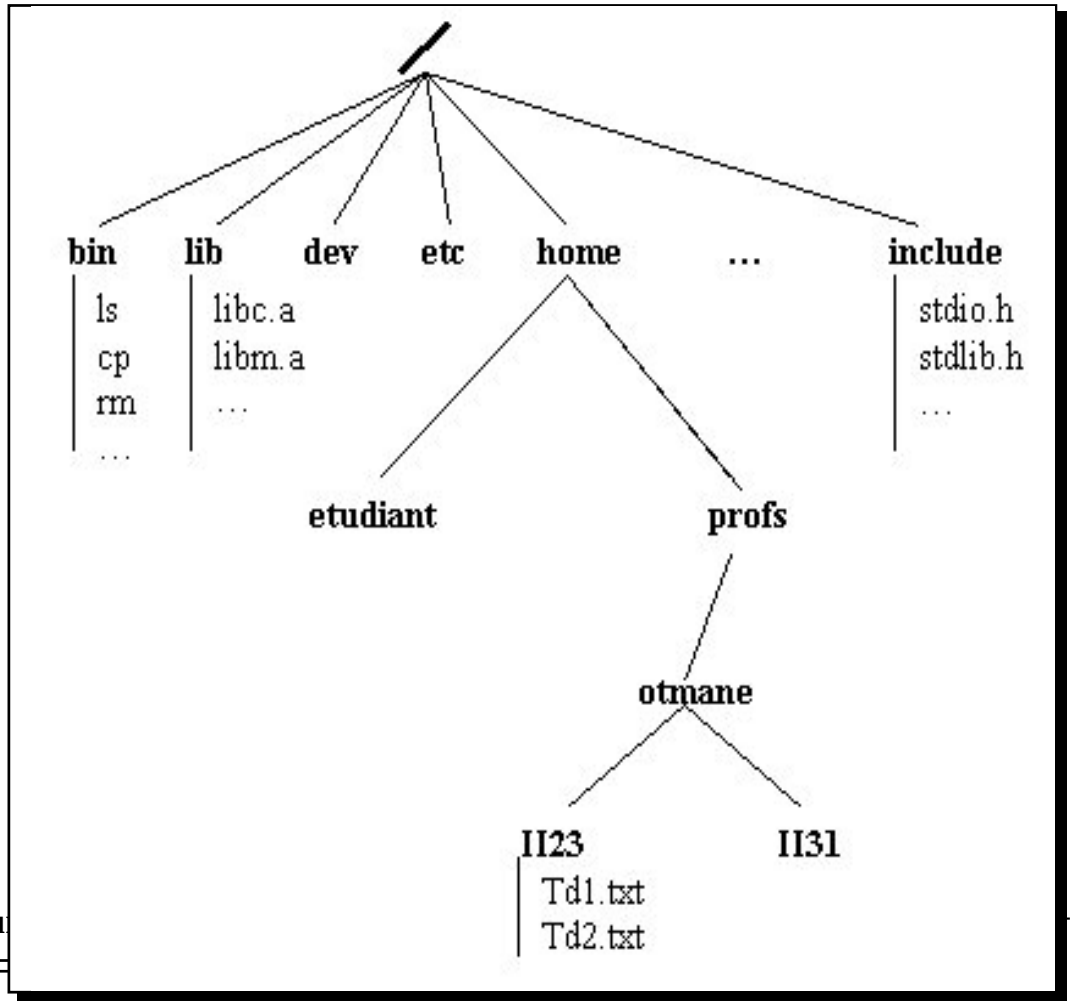
- Système ***multi-utilisateurs et multi-tâches***,
- Indépendant de toute architecture matérielle.
- Répartition des ressources entre les utilisateurs et les tâches.
- Unix est constitué d'un ***gestionnaire de fichiers*** et d'un ***gestionnaire de processus***.
- Chaque utilisateur peut exécuter plusieurs programmes simultanément.
- Fournit des primitives pour construire des applications complexes à partir d'autres plus simples.
- Il est possible de ***rediriger*** les entrées et sorties des processus.
- Un mécanisme de ***communication par tubes*** permet de synchroniser des processus et de leur faire échanger des informations.



2.3 Le Système de Gestion de Fichiers (SGF)

2.3.1 Arborescence, fichier et références :

- Il s'agit d'un **système de fichiers arborescent** dont les nœuds sont les noms des catalogues (répertoires) non vides et les feuilles sont les noms des répertoire vides et des fichiers (programmes et ou données).
- Chaque utilisateur dispose de sa propre sous-arborescence.
- Les fichiers sont désignés par des "références" qui doivent désigner de façon unique un seul fichier dans une arborescence.
- La racine du système de fichiers est unique et est référencée par /.
- Deux types de références :
  - Référence absolue,
  - Référence relative.
- Deux références particulières :
  - . : le répertoire lui-même,
  - .. : le répertoire prédécesseur.



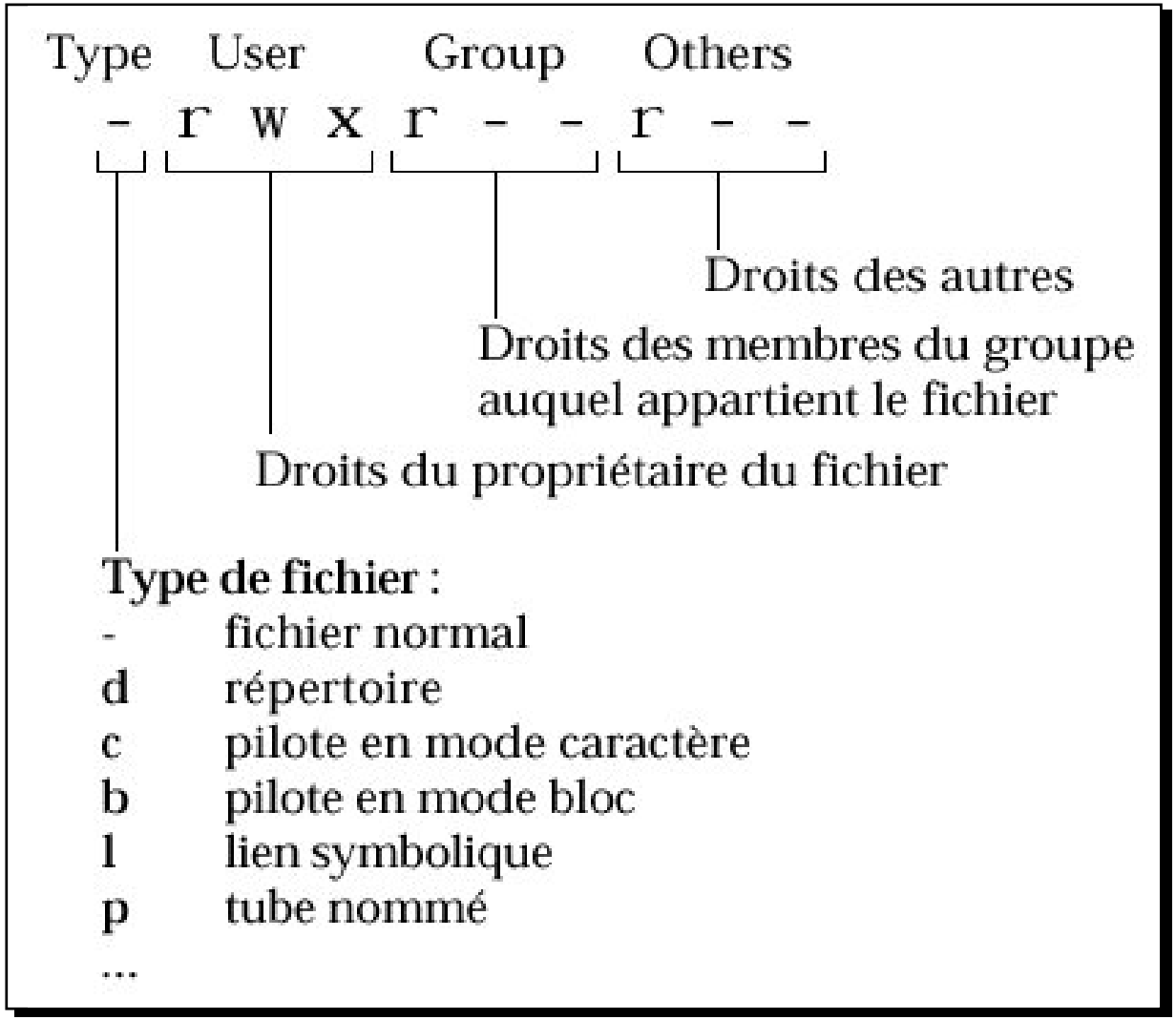
2.3.2 Protection des fichiers et répertoires

Tout utilisateur possède un **numéro d'utilisateur** et le **numéro du groupe** auquel il appartient. On distingue trois types d'utilisateurs potentiels :

- le **propriétaire** du fichier ("user", ou en abrégé **u**);
- les utilisateurs appartenant au même **groupe** ("group", en abrégé **g**);
- les **autres** utilisateurs ("other", en abrégé **o**).

Trois types d'opérations sur les fichiers sont possibles :

- la **lecture** ("**r**ead", en abrégé **r**);
- l'**écriture** ("**w**rite", en abrégé **w**);
- l'**exécution** ("e**x**ecute", en abrégé **x**).



2.3.3 Modification des droits

chown, chgrp et chmod

chown propriétaire fic1 fic2 ... (*change owner*)

chgrp groupe fic1 fic2 ... (*change group*)

chmod mode fic1 fic2 ... (*change mode*)

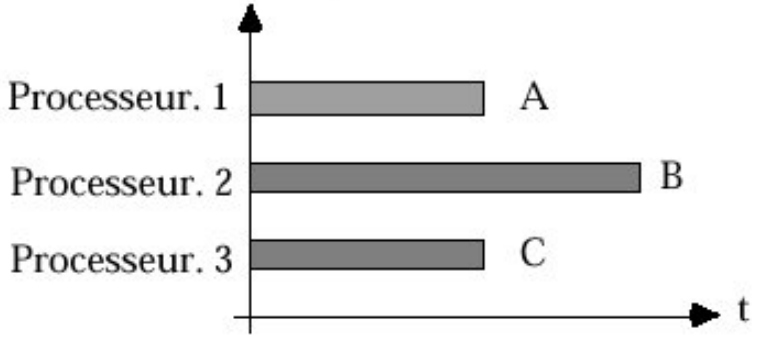
mode = 3 chiffres octaux (changement absolu)

mode = [**u** / **g** / **o**] [+ / -] [**r** / **w** / **x**] (changement relatif)

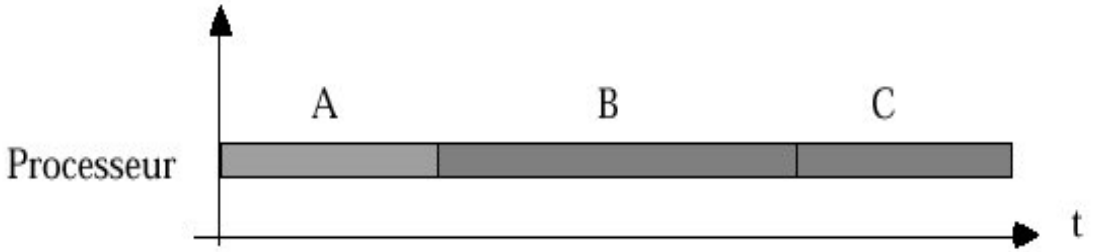
2.4 Processus et temps partagé :

Unix est multi-tâches et multi-utilisateurs

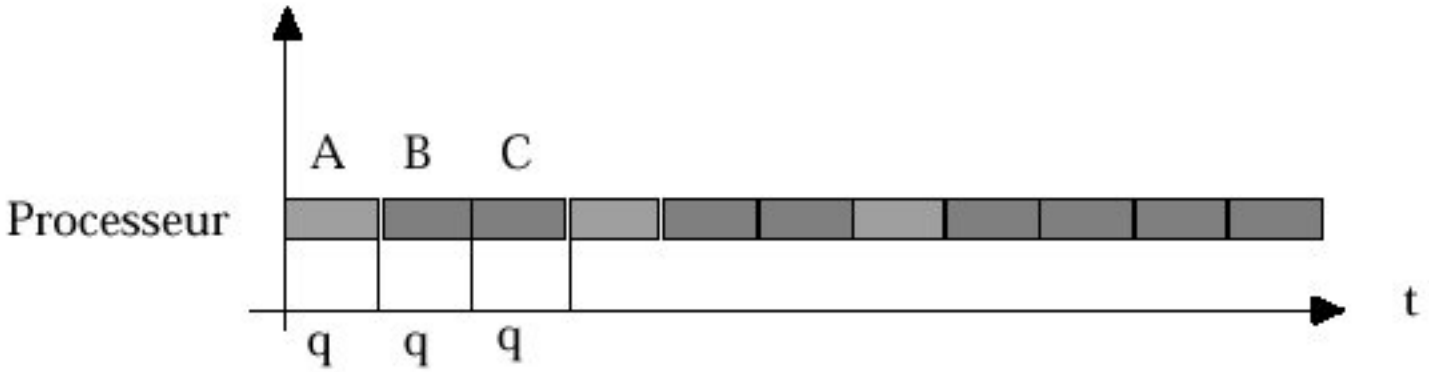
Multi-processus – Multi-processeurs



Mono-processus – Mono processeur



Multi-processus – Mono-processeur



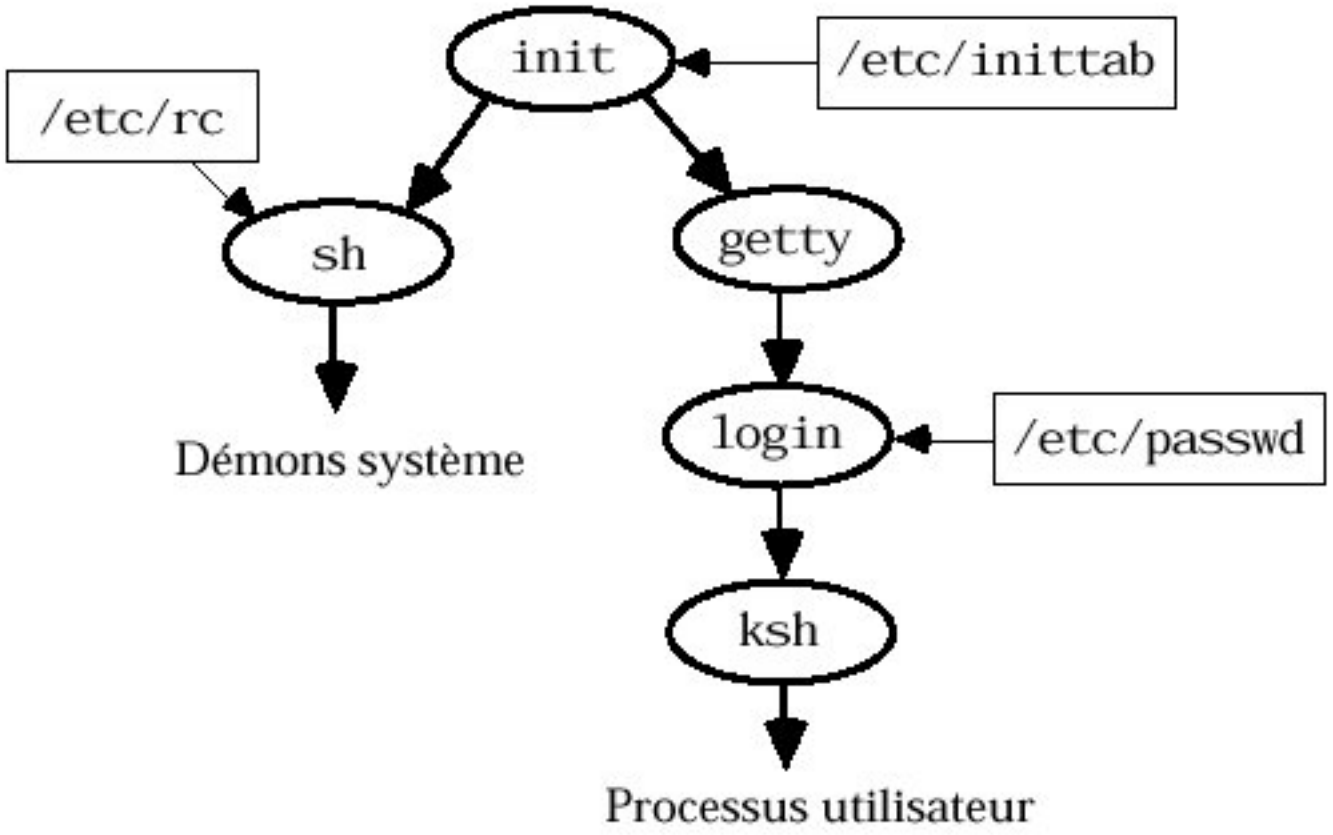
Processus Unix

Tout processus est créé par un autre processus (son père)

Démarrage (*bootstrap*) ➡ création de deux processus fondamentaux :

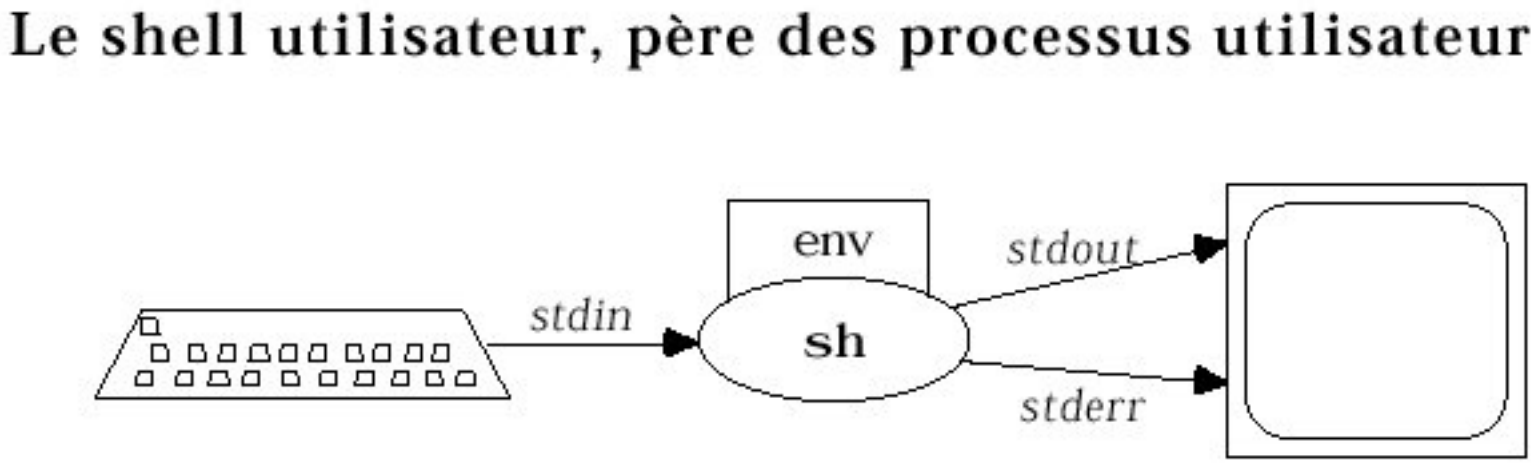
swap PID 0

init PID 1



3. Interpréteur de commande : SHELL

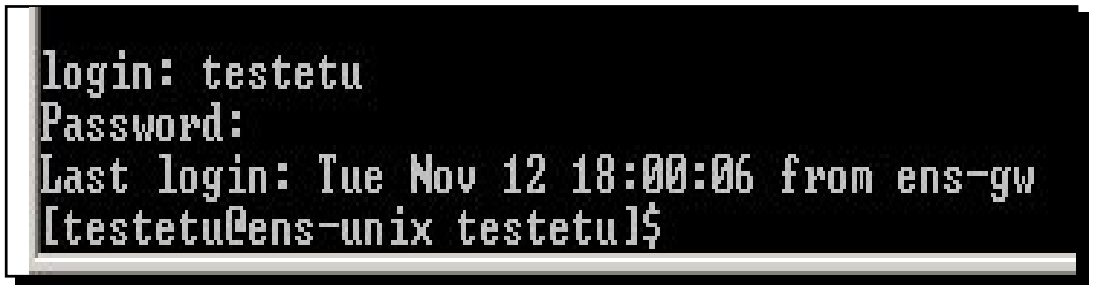
Un interpréteur de commande (ou "shell" en terminologie UNIX) est un processus lancé par le système UNIX lorsqu'un utilisateur se connecte. Il est attaché au terminal qui sert à établir la connexion. Il est chargé de recueillir les commandes émises par l'utilisateur et si possible de les exécuter



3.1 Connexion / Déconnexion

Une fois le terminal allumé, le message suivant apparaît à l'écran :
login :. taper son nom d'utilisateur suivi de la touche **<CR>**.
En réponse à la question suivante, i.e.
Password : le mot de passe de l'utilisateur doit être composé.

- Le mot de passe est évidemment tapé en aveugle, pour que personne ne puisse le lire.
- Ensuite, un certain nombre d'informations sont affichées, suivies d'un caractère d'invite ("prompt") qui peut être le caractère $, le nom de la machine, ou autre chose, dépendant du site sur lequel on se trouve.
  - Le "prompt" est envoyé par l'interpréteur du langage de commandes shell pour informer l'utilisateur qu'il est en attente de lecture de commandes.
  - Le système différencie les minuscules et les majuscules.



- La connexion permet la création d'une session de travail sous le SHELL.

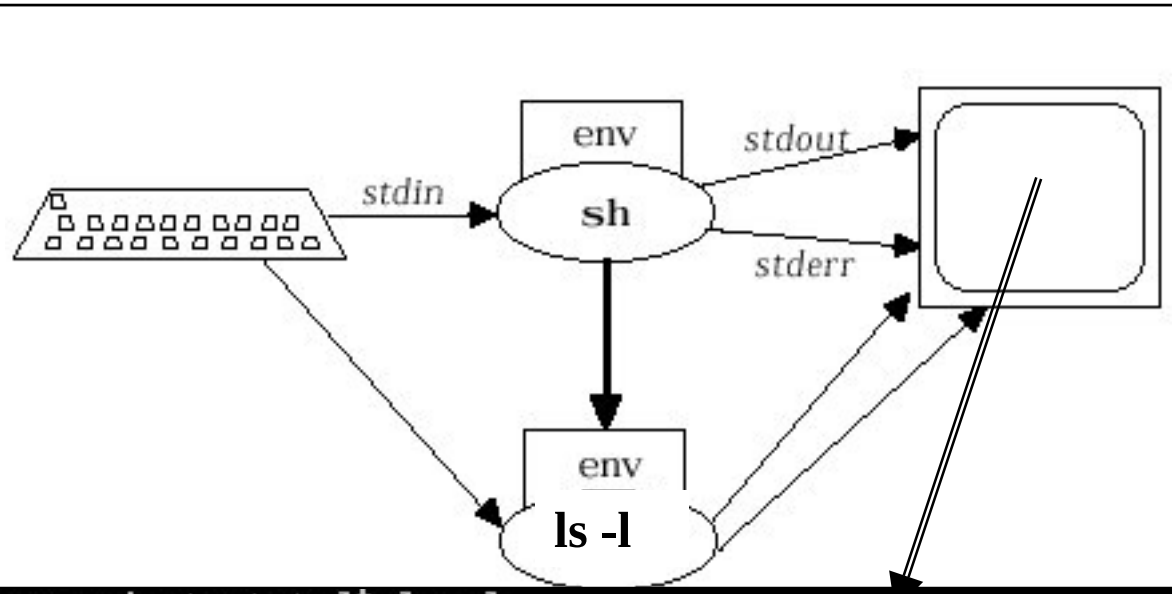
3.2 Fonctionnement du Shell

Forme générale d'une commande :

nom [option] arg1 arg2 …

Pour l'exécution d'une commande deux cas sont possibles :

- soit le nom de l'action est le nom d'un fichier contenant un programme exécutable (commande externe),
- soit l'action doit être exécutée par l'interpréteur de commandes lui-même(commandes interne).




```
[testetu@ens-unix testetu]$ ls -l
total 28
drwxrwxrwx    3 testetu  etudiant     4096 oct 21 14:59 Desktop/
drwxr-xr-x    2 testetu  etudiant     4096 nov 14 21:53 ii21/
drwxr-xr-x    2 testetu  etudiant     4096 nov 14 21:53 ii23/
drwxr-xr-x    6 testetu  etudiant     4096 nov 12 20:49 ii31/
drwx------    3 testetu  etudiant     4096 oct  8 17:31 otmane/
-rwxrwxrwx    1 testetu  etudiant      365 sep 17 22:24 procmail.log*
drwxr-xr-x    2 testetu  etudiant     4096 oct 31 11:47 public_html/
[testetu@ens-unix testetu]$
```

3.3 Caractères spéciaux

Certains caractères jouent un rôle particulier :

- ***** représente toute chaîne de caractères ;
- **?** désigne un caractère quelconque ;
- **[...]** désigne un caractère quelconque de l'ensemble spécifié dans **...**. Par exemple, [abc1] représente un caractère appartenant à {a, b, c, 1}. Des abréviations permettent de ne pas décrire explicitement tout l'ensemble : [0-9], [a-z], [A-Z] ;
- > et < représentent des redirections d'entrées/sorties standard ;
- | désigne un tube ;
- **;** permet de construire une séquence ;
- **<CR>** passe à la ligne de commandes suivante.

Si l'on désire utiliser ces caractères en tant que caractères normaux, il faut les faire précéder d'un caractère \ qui perd lui-même sa particularité lorsqu'il est doublé. Lorsqu'une référence de fichier commence par un **.**, ce premier caractère doit être spécifié explicitement car cette occurrence n'est pas couverte par la convention *****.

3.4 Composition des commandes

Toute commande Unix retourne une valeur :
0 si tout s'est déroulé normalement
≠ 0 en cas d'erreur

Une commande peut être vue comme un prédicat : **VRAI (0)** si tout c'est bien passé et **FAUX (≠0)** en cas d'erreur.

Composition séquentielle simple (;)
$ cp f1 f2 ; mv f2 toto
Composition conditionnelle EtAlors (&&)
$ cc -o prog prog.c && prog
Composition conditionnelle OuSinon (||)
$ cc -o prog prog.c || echo "Il y a des erreurs !"
Lancement en arrière plan *(backgroun*d) (&)
$ cc -o prog prog.c &
[1] 3427

Pour afficher les processus en cours
$ ps
Pour arrêter un processus lancé en arrière-plan
$ kill -9 3427

3.5 Redirection des Entrées/Sorties (E/S)

Entrées (<, <<) : Flot stdin, descripteur de fichier n° 0

$ sort < annuaire	*$ mail otmane < message*

Redirection jusqu'à une certaine chaîne

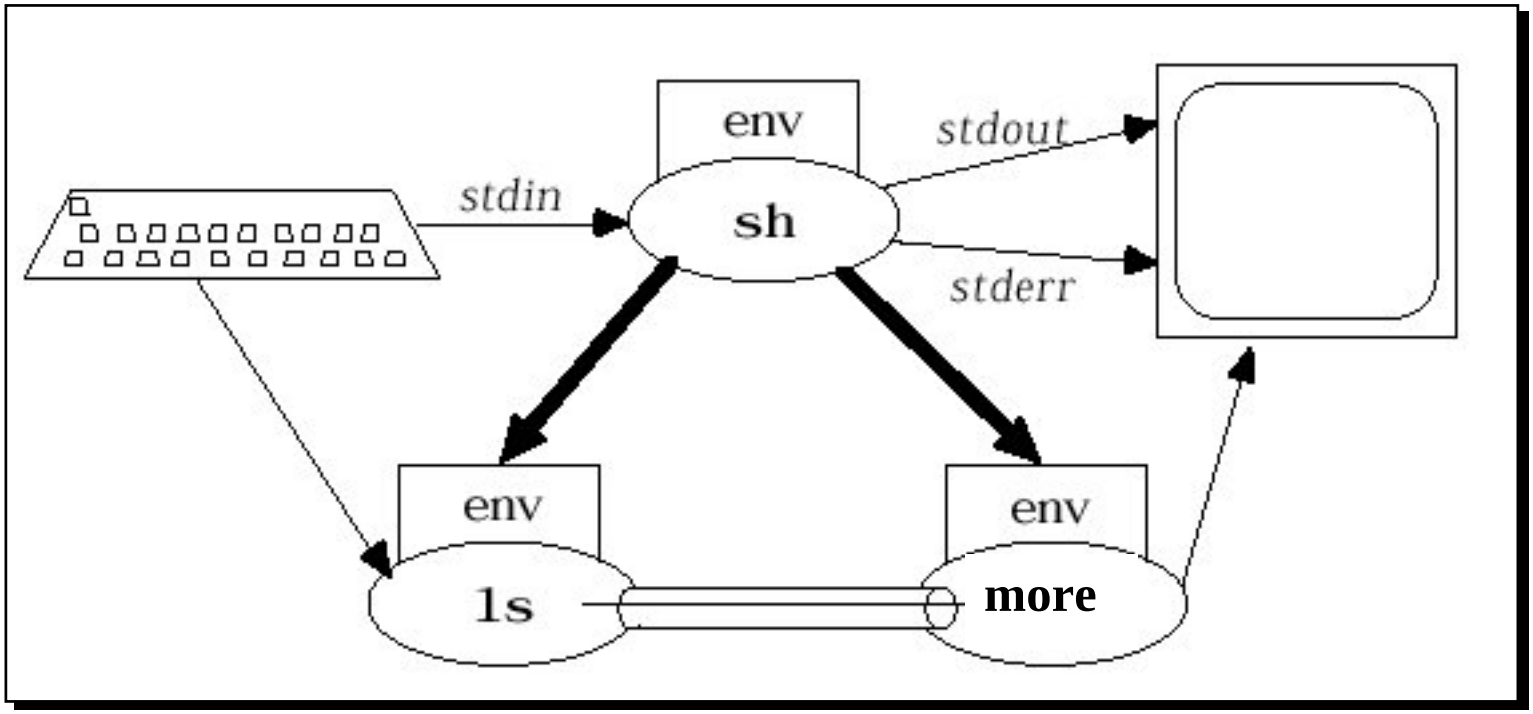
```
$ mail otmane << --FIN--
Bonjour,
Ceci est le corps du message, il se
termine avec la ligne suivante :
--FIN—
```

Sorties (>, 2>, >>, 2>>) : Flots stdout (descripteur 1) et stderr (descripteur 2)

*$ ls *.c > liste* *$ cat *.c > tous_les_programmes*	*$ cc -o tp tp.c 2> erreurs* *$ sort < liste > liste_triee* *$ cc -c *.c >> erreurs_cumulees*

Tubes (|) : Redirection de la sortie d'une commande sur l'entrée d'une autre

$ ls | more



3.6 Quelques commandes utiles

man [section] titre

affiche page à page le chapitre correspondant au *titre* donné dans le manuel standard d'Unix. Certains titres se trouvent dans plusieurs sections.

- Les commandes externes sont en section 1,
- les appels système en section 2 et
- les fonctions des bibliothèques standard en section 3.

man commande

permet d'obtenir de la documentation sur la commande.

man -k sujet (file/process)

permet d'obtenir de la documentation sur le sujet, si il y en a
La touche **h** ("help"), on obtient une aide pour l'affichage page à page.

Quelques Commandes Utiles (suite)

* **who** affiche la liste des utilisateurs connectés.
* **more *nom_fichier*** affiche page à page le contenu de *nom_fichier*.
* **find** permet de trouver des fichiers dans l'arborescence.
  - Critères de recherche : nom, permissions d'accès, type, nombre de références, propriétaire, groupe du propriétaire, taille, numéro de i-noeud, dates de création, de modification.
  - Ces critères peuvent être combinés à l'aide d'opérateurs logiques "et" et "ou".
  - Lorsqu'un fichiers a été trouvé , la commande find peut lui appliquer certaines opérations.
    - find / -name titi -print
    cherche récursivement à partir de la racine tous les fichiers de nom titi et affiche leur référence absolue au fur et à mesure.

Quelques Commandes Utiles (suite)

* **passwd** permet de créer ou de changer le mot de passe.
* **tty** affiche le nom du terminal utilisé.
* **wc [-lwc] [*liste_noms_fichiers*]** ("word count") compte et affiche, pour tous les fichiers référencés dans *liste_noms_fichiers* le nombre de lignes, mots et caractères. Si aucun nom de fichier n'est précisé, la commande utilise l'entrée standard. L'option **-l** ("line") permet de n'afficher que le décompte des lignes, l'option **-w** ("word") celui des mots et l'option **-c** ("character") celui des caractères.
* **grep *chaîne_caractères liste_noms_fichiers*** affiche, pour chaque fichier référencé dans *liste_noms_fichiers*, les lignes contenant *chaîne_caractères*.

```
[testetu@ens-gw testetu]$ ls -l | grep ii23
drwxr-xr-x    2 testetu  etudiant     4096 nov 14 21:53 ii23/
[testetu@ens-gw testetu]$ who | grep testetu
testetu  pts/0    Nov 15 13:28 (iupc96)
[testetu@ens-gw testetu]$
```

Quelques Commandes Utiles (suite)

* **pwd** ("print working directory") affiche la référence absolue du catalogue de travail.
* **cd [*nom_catalogue*]** ("change directory") change de catalogue de travail. Si aucun paramètre n'est fourni, le catalogue de travail devient le catalogue privé de l'utilisateur, sinon le catalogue de travail devient celui dont la référence est *nom_catalogue*.
*** cp [-i] *ancien nouveau*** ("copy") effectue une copie physique d'un fichier dans un autre : création d'un nouvel i-noeud, d'une nouvelle entrée dans un catalogue et recopie effective du contenu du fichier *ancien* dans *nouveau*

Quelques Commandes Utiles (suite)

ln [-s] ***fichier_existant fichier_nouveau*** ("link") crée un lien de nom *fichier_nouveau* sur *fichier_existant* .

(-s lien symbolique)

Exemple :

```
[testetu@ens-gw otmane]$ ls -l
total 12
-rw-r--r--    1 testetu  etudiant        0 nov 16 16:52 fichier
drwxr-xr-x    2 testetu  etudiant     4096 nov 16 16:50 ii21
drwxr-xr-x    2 testetu  etudiant     4096 nov 16 16:51 ii23
drwxr-xr-x    4 testetu  etudiant     4096 nov 16 16:50 ii31
[testetu@ens-gw otmane]$ ln -s fichier fichier2
[testetu@ens-gw otmane]$ ls -l
total 12
-rw-r--r--    1 testetu  etudiant        0 nov 16 16:52 fichier
lrwxrwxrwx    1 testetu  etudiant        7 nov 16 16:53 fichier2 -> fichier
drwxr-xr-x    2 testetu  etudiant     4096 nov 16 16:50 ii21
drwxr-xr-x    2 testetu  etudiant     4096 nov 16 16:51 ii23
drwxr-xr-x    4 testetu  etudiant     4096 nov 16 16:50 ii31
[testetu@ens-gw otmane]$
```

Quelques Commandes Utiles (suite)

- **mv [-i]** ***ancien_nom nouveau_nom*** ("move") renomme le fichier *ancien_nom* en *nouveau_nom*. L'option **-i** demande confirmation si *nouveau_nom* est le nom d'un fichier existant déjà (sinon il est écrasé).

- **rm [-ir]** ***liste_noms_fichiers*** ("remove") supprime, pour chaque nom de fichier dans *liste_noms_fichiers*, la référence du fichier. Le compteur de références de l'i-noeud correspondant est décrémenté de 1. L'i-noeud et le fichier physique ne sont détruits que si le compteur de références devient nul. L'option **-i** demande confirmation avant chaque suppression. L'option **-r**, permet de supprimer récursivement le contenu du catalogue puis le catalogue lui-même.
- **mkdir** ***liste_noms_catalogues*** ("make directory") crée les catalogues vides référencés dans *liste_noms_catalogues*.

- **rmdir** ***liste_noms_catalogues*** ("remove directory") pour chaque catalogue vide référencé dans l*iste_noms_catalogues*, supprime le catalogue.

Quelques Commandes Utiles (suite)

- **chown [-R]** ***nom_utilisateur*****[.*nom_groupe*] *liste_noms_fichiers***
  ("change owner") pour chaque fichier référencé dans *liste_noms_fichiers*, le propriétaire du fichier devient *nom_utilisateur*. L'option **-R** permet d'appliquer récursivement la commande aux catalogues de la liste. Chown est réservée à l'administrateur.

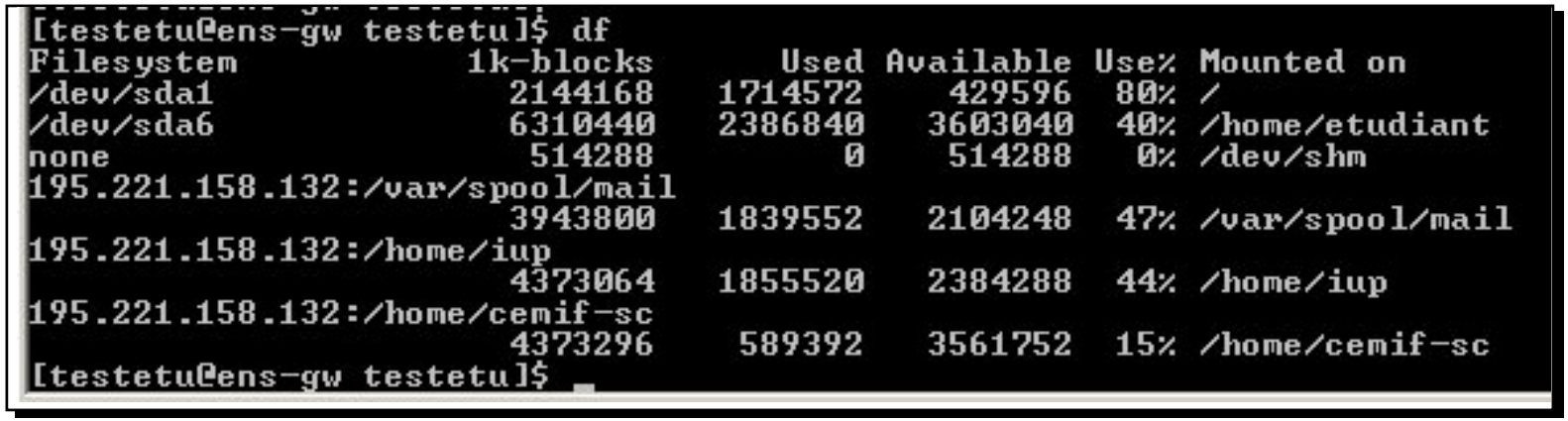
- **chmod [-R]** ***nouvelles_protections liste_noms_fichiers*** ("change mode") pour chaque fichier référencé dans *liste_noms_fichiers*, le mode de protection devient *nouvelles_protections*.

  → Protections définies par rapport à : *user*, *group*, *other*

Quelques Commandes Utiles (suite)

df [-i] [liste_noms_disques_logiques] ("display filesystem") affiche la taille des disques logiques montés, la place utilisée et disponible, le pourcentage de la capacité d'un disque logique déjà utilisé. L'option **-i** permet d'afficher le nombre d'i-noeuds utilisés et disponibles. On peut noter que la somme de la taille disque utilisée et de la taille disque disponible est plus petite que la taille totale du disque logique. Ceci est dû au fait que le système réserve une partie du disque logique (en général 10%) pour ne pas avoir de problème lors des allocations. Seuls un super-utilisateur et le système lui-même peuvent se servir de cette marge.

Exemple :



Quelques Commandes Utiles (suite)

ps [[-]aef] [-t*term*] affiche des informations sur les processus sous forme de tableau. **-aef** (System V) permet de voir les informations complètes sur tous les processus en cours.

- colonne **UID** ("user identification") : numéro de l'utilisateur concerné.
- colonne **PID** ("process identification") : numéro du processus concerné.
- colonne **TT** ("terminal") : nom du terminal .
- colonne **TIME :**temps CPU utilisé par le processus
- colonne **COMMAND :** nom complet de la commande
- colonne **STAT** ("status") : état du processus décrit par 4 caractères
  - première lettre (indique l'état de fonctionnement du processus) :
  - **R** ("runable") processus en cours d'exécution;
  - **T** processus arrêté;
  - **S** ("sleep") processus en sommeil pour moins d'environ 20 secondes;
  - **I** ("idle") processus en sommeil pour plus d'environ 20 secondes;
  - **Z** ("zombie") processus ayant terminé qui attend que son père exécute une attente ("wait");

Quelques Commandes Utiles (suite)

- La seconde lettre (indique si un processus est swappé) :
- La troisième lettre (indications sur la priorité du processus) :
- La quatrième lettre indique des traitements particuliers mém. virtuelle.

Exemple :

$ ps –aef | more

```
[testetu@ens-gw testetu]$ ps -aef|more
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Sep17 ?        00:01:02 init [5]
root         3     1  0 Sep17 ?        00:00:00 [keventd]
root         4     0  0 Sep17 ?        00:00:01 [ksoftirqd_CPU0]
root         5     0  0 Sep17 ?        00:00:01 [ksoftirqd_CPU1]
root         6     0  0 Sep17 ?        00:23:36 [kswapd]
```

$ ps –aef | grep testetu

```
[testetu@ens-gw testetu]$ ps -aef| grep testetu
root     26569 26568  0 16:47 pts/0    00:00:00 login -- testetu
testetu  26570 26569  0 16:47 pts/0    00:00:00 -bash
testetu  26802 26570  0 19:27 pts/0    00:00:00 ps -aef
testetu  26803 26570  0 19:27 pts/0    00:00:00 grep testetu
[testetu@ens-gw testetu]$
```

Quelques Commandes Utiles (suite)

kill [-*num*] *num_proc1 num_proc2 ...* envoie le "signal de terminaison" aux processus de numéros *num_proc1*, *num_proc2*, ... L'option ***-num*** permet d'envoyer un autre signal dont la nature est spécifiée par le numéro est *num* plutôt que le "signal de terminaison". On utilise souvent le signal 9 ("signal tueur" ou SIGKILL) qui permet de tuer les processus en toutes circonstances.
Sur un grand nombre de systèmes UNIX, le même effet est obtenu, pour les processus qui ne s'exécutent pas en tâche de fond, en frappant CTRL-C (respectivement CTRL-\) qui provoque l'envoi d'un "signal de terminaison" (respectivement d'un "signal tueur"). Mais certains processus sont récalcitrants. Il n'y a alors pas d'autre possibilité que de se reloger pour utiliser la commande kill.
Pour des raisons évidentes, l'utilisation de la commande kill sur les processus d'un autre utilisateur est réservée à l'administrateur du système.

Quelques Commandes Utiles (suite)

Exemple :

```
root     26569 26568  0 16:47 pts/0    00:00:00 login -- testetu
testetu  26570 26569  0 16:47 pts/0    00:00:00 -bash
testetu  26827 26570  0 19:33 pts/0    00:00:00 vi fichier
testetu  26830 26570  0 19:33 pts/0    00:00:00 ps -aef
testetu  26831 26570  0 19:33 pts/0    00:00:00 grep testetu
[testetu@ens-gw testetu]$ kill -9 26827
[testetu@ens-gw testetu]$ ps -aef¦ grep testetu
root     26569 26568  0 16:47 pts/0    00:00:00 login -- testetu
testetu  26570 26569  0 16:47 pts/0    00:00:00 -bash
testetu  26832 26570  0 19:34 pts/0    00:00:00 ps -aef
testetu  26833 26570  0 19:34 pts/0    00:00:00 grep testetu
[1]+  Processus arrÛtÚ          vi fichier
[testetu@ens-gw testetu]$
```

4 Langage de Commande (Shell)

4.1 Shell, Login-Shell

Le Shell est à la fois un langage de commandes et l'interpréteur de ce langage.
Il assure l'interface externe entre les utilisateurs et le système, mais ne fait pas partie du noyau.
Principaux langages de commandes sont disponibles sur les systèmes UNIX :

- le Bourne shell (**sh**),
- le C-shell (**csh**) basé sur **sh**,
- **tcsh**, basé sur **csh**, qui est du domaine public,
- **ksh** (korn shell) ,
- **bash** (Linux).

4.2 Commandes Externes et Commandes Internes

L'interpréteur du shell a la possibilité, soit d'exécuter lui même la commande demandée par l'utilisateur (*commande interne*), soit de lancer l'exécution de programmes existant par ailleurs pour ce faire (*commandes externes*).
À chacune des commandes externes correspond un fichier exécutable ayant comme nom le nom de la commande. Ce fichier se trouve souvent dans le répertoires */bin*
L'exécution d'une commande externe entraîne la création d'un nouveau processus exécutant le programme correspondant à la commande.
Une commande externe peut être appelée à partir de n'importe lequel des langages de commandes disponibles, mais aussi à partir d'un programme utilisateur.
Chacun des langages de commandes dispose de ses propres commandes qui sont les commandes internes.

4.3 Le Login et l'Environnement Shell

Lorsqu'un utilisateur se logue, un processus shell est exécuté. De plus, certaines commandes, soit communes, soit propres à chacun, sont effectuées :

- **.profile**,
- **.login**
- **.cshrc**

Ce type de commande permet l'initialisation de variables shell, soit pour leur donner une valeur différente de celle par défaut, soit pour en définir de nouvelles.
L'ensemble de ces variables constitue **l'environnement** shell.

4.4 Les variables

Le nom d'une variable shell est une chaîne de caractères contenant des lettres, des chiffres ou le caractère _ et commençant toujours par une lettre.
La valeur d'une variable est une chaîne de caractères quelconque.

Une affectation de la valeur *val* à la variable *var* s'effectue par :

var = val.

Ou

let var=<expression>

ou <expression> peut être :

- statut de retour d'une commande,
- résultat d'une commande entre ``,
- expression arithmétique,
- opération sur les chaînes de caractères.

Les variables (suite)

La valeur d'une variable *var* est obtenue en utilisant *$var*. Si la variable *var* n'a pas été définie, son contenu est la chaîne de caractères vide.

Il est également possible d'exporter une variable(la rendre globale) :

export var

La commande export sans argument donne la liste des variables exportées.

On distingue plusieurs catégories de variables :

- Les variables définies par l'utilisateur,

- Les variables d'environnement,

Les variables réservés au SHELL.

4.5 Les variables d'environnement existant toujours sont :

* **PS1** : premier prompt ;
* **PS2** : second prompt. Celui-ci est utilisé en particulier pour continuer une commande commencée mais pas terminée ;
* **HOME** : référence absolue du catalogue privé ;
* **PATH** : liste de chemins dans lesquels les commandes appelées vont être cherchées ;

- **TERM** : type du terminal utilisé;
- **USER** : Nom de l'utilisateur connecté;
- **SHELL** : Nom du SHELL pris par défaut
- **ENV** : Liste des variables d'environnement.

Ces variables sont consultables par la commande **env** et sont modifiables par **setenv**

Il est possible de définir de nouvelles variables d'environnement :

Ex:

```
set MON_PROJET=~/src/projet/linux
cd $MON_PROJET
```

4.6 Lecture et affichage de variables

read permet la lecture d'une ou plusieurs valeurs à affecter à des variables :

read var1 var2 …
Les valeurs numériques respectives des différentes variables sont séparées par des espaces. L'affectation des variables est effectuée apres la validation(<CR>).

echo permet l'affichage un texte(son argument) sur l'écran, ce texte peut contenir des variables. Dans ce cas la variable est précédée de $. La validation de **echo** provoque un saut de ligne.

echo "texte1 $var texte2"
echo -n "texte" évite un retour à la ligne
echo -e "salut \t à tous " prend en compte les caractères non visualisables

4.7 Interprétation par le SHELL des chaînes de caractères

Des délimiteurs permettent d'effectuer des opérations à l'intérieur de chaînes de caractères :
* '**...**' : **...** est pris tel quel, c'est-à-dire que si cette chaîne contient des appels à des variables, aucune substitution n'est effectuée ;
* "**...**" : les variables contenues dans **...** sont substituées ;
* `**op**` : évalue la commande shell **op**.

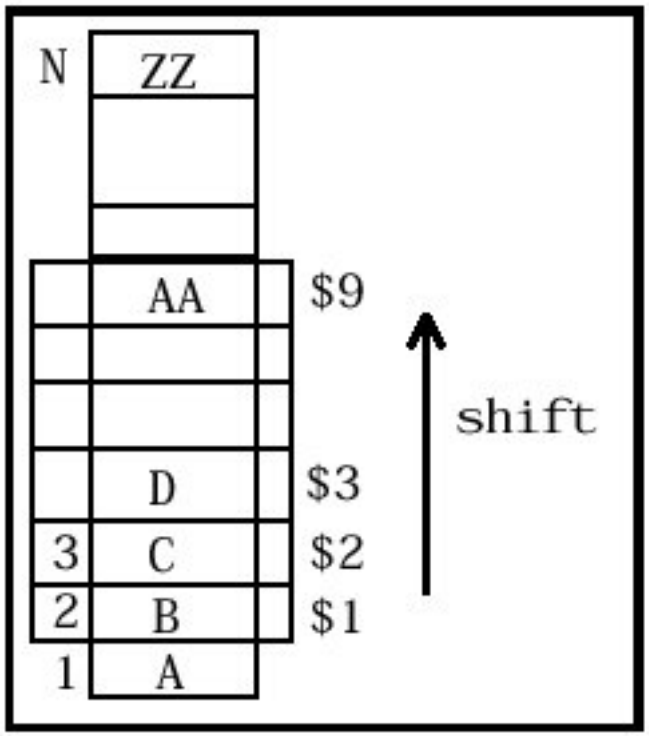
4.8 Les Procédures et leurs Paramètres

Une procédure (ou *script shell*) est une suite de commandes shell. Elle peut accepter des paramètres:
Commande paramètre$_1$ ... paramètre$_n$. Ces variables réservées au SHELL sont accessibles en lecture seule.
Le premier paramètre est référencé par **$1**, le second par **$2**, ..., le neuvième par **$9**.

- **$1**, **$2**, ..., **$9** : paramètres (de position) de la commande (si plus de 9 utiliser $*);
- **$0** : nom de la commande(script) appelée ;
- **$*** : liste des paramètres à partir de $1;
- **$#** : nombre de paramètres passé au script;
- **$$** : numéro du processus shell correspondant à la commande ;
- **$?** : code de retour de la dernière commande exécutée(vaut 0 si la commande s'est bien passé).

Les variables utilisées dans une commande(script) sont locales à celle-ci.

La commande interne **shift** effectue un décalage de pas +1 dans les paramètres de position ($1 prend la valeur de $2) et la variable $# est décrémentée de 1 ainsi de suite …



la commande **set** `commande`permet d'initialiser $1,$2,…

4.9 Structures de Contrôle

Affectation

set <chaîne de caractères> : la <chaîne de caractères> devient la nouvelle liste de paramètres.
read <liste de variables> : les variables prennent les valeurs fournies par l'entrée standard.

Conditionnelles :

if <liste de commandes 1> **then** <liste de commandes 2> **else** <liste de commandes 3> **fi**	**if** <liste de commandes 1> **then** <liste de commandes 2> **elif** <liste de commandes 3> **then**<liste de commandes 4> **else** <liste de commandes 5> **fi**

Structures de Contrôle (suite)

Exemples

```
if test $# -eq 0
then
    echo Pas de paramètres
fi
```

```
if cc -o tp tp.c
then
    tp
else
    echo Erreurs...
fi
```

Itération non bornée classique : tantque

```
while commande
do
  instructions
done
```

```
until commande
do
  instructions
done
```

Exemples :

```
while [ -r "$1" ]
do
  cat $1 >> liste
  shift
done
```

```
until [ ! -r "$1" ]
do
  cat $1 >> liste
  shift
done
```

Structures de Contrôle (suite)

Itération bornée : for

for <variable> **in** <liste de chaînes de caractères> **do** <liste de commandes> **done**	**for** <variable> **in** $* **do** <liste de commandes> **done**

Branchement sélectif : case

case <chaîne de caractères> **in**
<motif1> **)** <liste de commandes 1> **;;**
...
<motif n> **)** < liste de commandes n> **;;**
***)** <liste de commandes pour les autres cas> **;;**
esac

Structures de Contrôle (suite)

test <condition>

L'instruction test permet d'effectuer de nombreux tests aussi bien sur des fichiers, des valeurs numériques ou des chaînes de caractères. Le résultat sous forme booléenne est renvoyé dans le statut (variable ?).
Les options (comparateurs) ci-dessous sont utilisés.

Tests sur les chaînes de caractères :

[<chaîne de caractères 1> **=** <chaîne de caractères 2> **]**

teste si <chaîne de caractères 1> et <chaîne de caractères 2> sont égales.
Attention : les [et] doivent impérativement être entourés de blancs ainsi que les primitives de comparaison.

Autres tests sur les chaînes de caractères

!= (différentes), **-n** (non vide), **-z** (vide)

Structures de Contrôle (suite)

Tests sur les valeurs numériques :	**Tests sur les fichiers :**
• **-eq** (égales), • **-ne** (différentes), • **-gt** (strictement supérieure), • **-ge** (supérieure ou égale), • **-lt** (strictement inférieure), • **-le** (inférieure ou égale) ;	• **-d** (répertoire), • **-f** (fichier), • **-r** (permissions de lire le fichier), • **-w** (permissions d'écrire le fichier), • **-x** (permissions d'exécuter le fichier).

* **exit** <valeur entière> termine le processus et renvoie la valeur passée en paramètre comme code de retour.

Structures de Contrôle (suite)

Exemple 1 : le fichier *monscript* a le contenu suivant :

```
set 'ls'
for i in $*
do
if [ -d $i ]
then echo "$i est un repertoire"
fi
if [ $i = "toto" ]
then echo "toto trouve. Voulez-vous voir son contenu ?"
read rep
case $rep in
o | O ) cat $i;;
n | N ) echo "pas de visualisation du contenu de toto";;
* ) echo "vous répondez vraiment n'importe quoi"
esac
fi
done
```

4.10 Fonctions

2 syntaxes pour la définition d'une fonction :

Définition d'une fonction = bloc nommé

```
nom ()
{
    instructions ...
    return valeur
}
```

```
function nom
{
    instructions ...
    return valeur
}
```

Utilisation d'une fonction

```
nom param1 param2 ... paramN
```

Paramètres : comme les paramètres du shell, donc ceux du shell englobant ne sont plus accessibles

Fonctions (suite)

syntaxe de l'appel d'une fonction :
valeur_retour = nom-fct <parametres>

Exemple :

```
stat()
{
  if [-d "$1" ]
  then
    echo $1 est un répertoire
    return 0
  else
    echo $1 n'est pas un
répertoire
    return 1
  fi
}
```

`stat /tmp`	**retourne 1**
`stat $dir`	**dépend de la valeur de $dir**
`stat $1`	**$1 est ici le paramètre du shell**

4.11 Les caractères spéciaux et leur fonction

4.11.1 Composants de commande de base

Caractère	Signification	Contexte	Exemple
Espace	Délimiteur de composant de cde	Cde	ls -l f.c
-	Flag désignant une option de commande	Cde	ls -al
/	Nom du répertoire root	Cde	cd /
/	Délimiteur de répertoires dans un chemin	Cde	cd /home/etudiant/user1
\	Inhibe le sens d'un métacaractère	Cde	echo *
&	Exécution d'une commande en background	Cde	emacs &

;	Exécute séquentielle de commandes	Cde	date ; ls -l
()	Exécute de commande(s) en sous shell	Comman de	(date ; ls -l)

4.11.2 Redirection d'entrée sortie

Caractère	**Signification**	**Contexte**	**Exemple**
|	Tube	Commande	who | wc -l
|&	Erreur standard du tube	Commande	cc f.c -o f |& wc -l
>	Redirige la sortie standard	Commande	ls -lR >fic
>>	Redirige la sortie standard avec ajout ds fic	Commande	ls -lR >>fic
2>	Redirige la sortie standard erreur	Commande	cc f.c -o f 2>err
<	Redirige l'entrée standard depuis fichier	Commande	mail user@iup.univ-evry.fr <fic.c

4.11.3 Métacaratères sur nom de fichier

Caractère	Signification	Contexte	Exemple
~	Chemin du répertoire HOME de l'utilisateur	Cde	cp f.c ~/src
.	Lien par défaut avec le répertoire courant	Cde	cp ~/src/f2.c .
..	Lien par défaut avec le répertoire père	Cde	cd ..
*****	Représente une chaîne de caractères	Cde	cp ~/src/*.c .
?	Représente un caractère quelconque	Cde	rm ~/bin/*.?
[ ]	Remplace un des caractères compris entre les bornes	Cde	rm ~/bin/f[0-9].o
[,]	Remplace un des caractères de la liste entre crochets	Cde	rm ~/bin/*.[o,b]
' '	Englobe une chaîne de caractères sans substitution	Cde	echo '*'

" "	Englobe une chaîne de caractères avec substitution	Cde	echo "*"

4.11.4 Composants généraux de scripts

Caractère	Signification	Contexte	Exemple
#	Délimiteur de commentaire	Script sh	# script.sh …
#!	Déclaration de l'exécution d'un shell	Script sh	#! /bin/sh
' '	Exécution de commande	Script sh	echo 'date'
$	Accès à une variable	Script sh	echo $USER
$$	N° du shell courant	Script sh	echo $$
$#	Nombre d'arguments du script	Script sh	if [$# -gt 0] then
$0 … $9	Les 10 premiers	Script sh	echo $0

	arguments du script		
$*	Tous les arguments du script	Script sh	

4.11.5 Opérateurs mathématiques

Caractère	Signification	Contexte	Exemple
=	Affectation	Script sh	Var=2
* , / , %	Multiplication, division, modulo	Script bash	let a=$b*$c
+ , -	Addition , soustraction/operateur unaire	Script sh	
<< , >>	Décalage	Script bash	let d=$c<<2;echo d
&	ET bit à bit	Script bash	
|	OU bit à bit	Script bash	
^	OU EXCLUSIF bit à bit	Script bash	
~	NOT	Script bash	

4.11.6 Opérateurs d'affectation

Caractère	Signification	Contexte	Exemple
+= , -=	Add ou Soustraction + Assignation	Script bash	
*=, /=, %=	Mul/Div/Modulo + Assignation	Script bash	
&=,^=,%=	AND/OR/XOR + Assignation	Script bash	
>>=, <<=	Décalage droit/gauche +assignation	Script bash	
>=	Supérieur ou égal	Script bash	
<	Inférieur	Script bash	
<=	Inférieur ou égal		

4.11.7 Opérateurs de comparaison

Caractère	Signification	Contexte	Exemple
~	Complément	Script bash	If [$a=$b] then
= =	Égalité	Script bash	
!=	Inégalité	Script bash	
>	Supérieur	Script bash	
>=	Supérieur ou égal	Script bash	
<	Inférieur	Script bash	
<=	Inférieur ou égal		

4.11.8 Opérateurs booléens

Caractère	Signification	Contexte	Exemple
!	Négation	Script bash	
||	Ou logique	Script bash	If [$a<9 || $a>0] then
&&	Et logique	Script bash	
!(||)	NOR logique	Script bash	
!(&&)	NAND logique	Script bash	
^	XOR logique	Script bash	
!(^)	NOR Exclusif logique	Script bash	

5. Liste des exercices des travaux dirigés

TD1 : Les commandes externes du SHELL

But : L'objet de ce TD est l'étude des commandes externes du SHELL et certains méta caractères
Prérequis : La connaissance de la syntaxe des principales commandes externes du SHELL

Exercice 1.1 :

Soit un répertoire contenant les fichiers suivants :
f1.c , f2.c , f3.f , f4.c , f5.f , f10.a , f11.a , a.out , e.c , t.f
Utiliser les méta caractères de substitution pour lister les fichiers suivants : (on recherchera l'ecriture minimale)

- les fichiers fortran (suffixe f),
- les fichiers C et fortran,
- les fichiers commençant par la lettre f,
- les fichiers C commençant par la lettre f,
- les fichiers dont le nom contient un chiffre avant '.',
- les fichiers dont le 2nd caractère est un chiffre,
- les fichiers dont le 2nd caractère est un '.' .

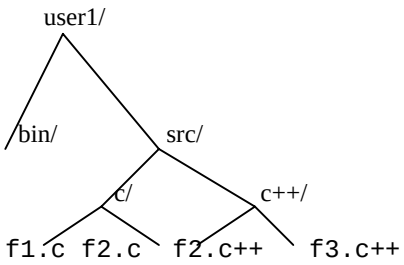
Exercice 1.2 :

Donner la commande qui liste l'ensemble des fichiers du répertoire "*/usr/sbin"* dont le nom commence par i suivi d'un caractère quelconque puis d'un point '.' puis de 2 lettres suivies de la lettre 'l', d'un 'm' ou d'un 't' et qui se terminent par un 'd'.

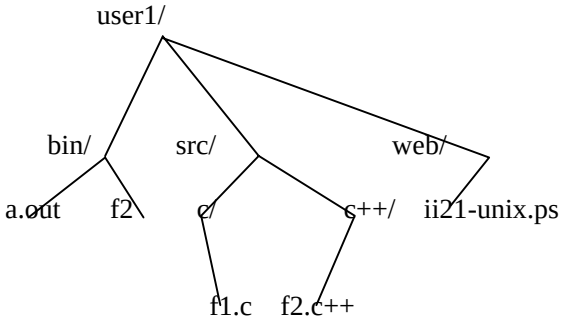
Exercice 1.3 :

Donner une suite de commandes SHELL qui permettent de faire passer l'arborescence de "user1" de l'état i à l'état i+1. On suppose que user1 se trouve dans son "Home Directory" :

Etat i *:*



Etat i+1 :



Exercice 1.4 :

Dessiner l'état i+2 du "Home Directory" de user1 à l'issue de l'exécution des commandes suivantes :
$ mv src/c/f1.c src
$ rm -r src/c
$ mv src/c++/f2.c++ src
$ cc src/f1.c -o bin/f1

TD2 et 3 : Les commandes externes du SHELL

Exercice 2.1 :

Donner la signification des commandes suivantes :
$ sort f.c | head
$ grep printf f.c | wc -l
$ grep printf f.c > sortie
$ cat /etc/passwd | grep user1

Exercice 2.2 :

Déterminer les commandes qui permettent de :
- savoir si l'utilisateur "user1" est connecté,
- afficher le nombre d'utilisateur du système ,
- afficher la liste des utilisateurs par ordre alphabétique,
- connaître le nombre de processus de "user1",
- connaître le nombre de processus de "root"

enregistrer dans le fichier "fuser1" la date et l'ensemble des fichiers de "user1".

Exercice 2.3 :

Ecrire le diagramme SHELL à base de fonctions systèmes (fork, exec, wait, exit) correspondant à chacune des commandes suivantes :
$ sort f.c | head
$ ps -aef | grep root | wc -l

Exercice 2.4 :

2.4.1
Ecrire une commande qui affiche l'ensemble des processus dont vous n'êtes pas propriétaire (votre nom d'utilisateur se trouvant dans la variable d'environnement $USER).

2.4.2
Donner la syntaxe qui lance la commande "sleep" en arrière plan("background") pendant une durée de 5 minutes.

2.4.3
Créer un sous-répertoire de "/tmp" ayant pour nom votre nom de login. Positionnez vous dans "/tmp". Créer dans le sous-répertoire précédent un fichier qui est la copie conforme de votre fichier ".profile" . Ce nouveau fichier doit avoir un nom ayant pour préfixe ".profile" et pour suffixe votre nom d'utilisateur. Protégez le contenu de ce sous-répertoire contre tout regard indiscret y compris le votre.

2.4.4
Ecrire une ligne de commande qui affiche uniquement le nom du port *TTY* courant (terminal) (sans utiliser la commande "tty")

2.4.5
Ecrire une commande "*find"* qui va rechercher à partir de votre répertoire HOME, les fichiers nommés "*core"* ou "*a.out"* et les supprimer.

TD 4 : Les scripts Shell - sh

But :

L'objet de ce TD est l'étude des fonctions des scripts sh.

Prérequis :

Une maîtrise des fonctions des scripts sh est requise.

Exercice 4.1 :

Ecrire un script sh qui réalise la somme de tous les arguments acquis a partir de la ligne de commande.

Exercice 4.2 :

Ecrire un script sh qui réalise l'affichage de tous les arguments de la ligne de commande.

Exercice 4.3 :

Ecrire un script sh qui réalise la copie d'un fichier sous plusieurs repertoires, utile pour l'administrateur. les repertoires file1 et file2 sont supposés exister, sinon fichier est duplique dans des fichiers ordinaires file1 et file2.

Exercice 4.4 :

Ecrire un script sh qui réalise l' affichage de la date en anglosaxon.

Exercice 4.5 :

Ecrire un script sh qui réalise l' affichage de la date en français

Exercice 4.6 :

Ecrire un script sh qui réalise l' Affichage d'un décompteur.

Exercice 4.7 :

Ecrire un script sh qui réalise la somme de tous les arguments acquis a partir de la ligne de commande

TD 5 : Les scripts Shell - sh

But :

L'objet de ce TD est l'étude des fonctions des scripts sh.

Prérequis :

Une maîtrise des fonctions des scripts sh est requise.

Exercice 5.1 :

Ecrire un script sh sous forme de fonction « recherche » qui recherche un nom passe en paramètre dans le fichier annuaire et qui affiche si le nom est trouvé ou pas.

Exercice 5.2 :

Ecrire un script sh sous forme de fonction « ajoute » qui ajoute un nom passe en paramètre dans le fichier annuaire et trie par ordre alphabétique » ce dernier.

Exercice 5.3 :

Ecrire un script sh sous forme de fonction « supprime » qui supprime un nom passe en paramètre dans le fichier annuaire.

Exercice 5.4 :

Ecrire un script sh sous forme de fonction « affiche » qui réalise Affichage de l'annuaire.

Exercice 5.5 :

Ecrire un script sh sous forme qui permet de créer un menu pour gerer un annuaire dans lequel se trouvent les fonctions recherche, ajoute, supprime et affiche déterminées précédemment.

TD 6 : Les scripts Shell - sh

But :

L'objet de ce TD est l'étude des fonctions des scripts sh.

Prérequis :

Une maîtrise des fonctions des scripts sh est requise.

Exercice 6.1 :

Ecrire un script sh "rep" qui affiche la liste des fichiers ordinaires lisibles(par tout utilisateur)se trouvant dans une liste de repertoires passés en paramètre. Si pas de paramètres, traiter le repertoire courant.

Exercice 6.2 :

Ecrire un script sh "rep2" qui à partir du repertoire courant va chercher tous les repertoires et sous repertoires. Pour chaque repertoire rencontré, afficher la liste des fichiers ordinaires qu'il contient(utiliser le script de l'exo. 1).

Exercice 6.3 :

Modifier le script "rep" de l'exo. 1 de façon à ce que celui-ci renvoie la liste des fichiers trouvés.

Exercice 6.4 :

Modifier le script "rep2" de l'exo. 2 de façon à ce que celui-ci affiche pour chaque fichier son nom et son nombre de lignes..

Exercice 6.5 :

Ecrire un script sh qui renomme plusieurs fichiers passes en parametre. Les nouveaux noms ont pour préfixe le 1er paramètre et pour suffixe le rang de renommage.

6. Liste des exercices des travaux pratiques

Université d'Evry-Val d'Essonne
IUP Sciences et Technologie
Unix

TP1 : Les commandes externes du SHELL

But :

L'objet de ce TP est l'étude des commandes externes du SHELL et certains méta caractères

Prérequis : La connaissance de la syntaxe des principales commandes externes du SHELL

PARTIE A : Gestion de Fichiers

Question A.1 :
Soit un répertoire contenant les fichiers suivants que vous aurez crées:
f1.c , f2.c , f3.c, f4.c++ , f5.c , f10.c++ , f11.c++ , a.out , e.c , t.c++

Utiliser les méta caractères de substitution pour lister les fichiers suivants : (on recherchera l'ecriture minimale)
- les fichiers C++ (suffixe c++),
- les fichiers C et C++,
- les fichiers commençant par la lettre f,
- les fichiers C commençant par la lettre f,
- les fichiers dont le nom contient un chiffre avant '.',
- les fichiers dont le 2nd caractère est un chiffre,
- les fichiers dont le 2nd caractère est un '.' .

Question A.2 : (connaissance requise : redirection, pipe)
Donner la signification des commandes suivantes :
$ sort | head

```
$ grep printf  f.c | wc -l
$ grep printf f.c > sortie
$ cat /etc/passwd | grep user1
```

Question A.3 : (connaissance requise : commandes externes)
Déterminer une suite de commandes SHELL qui permettent de faire passer l'arborescence de votre repertoire d'accueil
de l'état i à l'état i+1 (cf. figure). On suppose que le reperoire courant est le même que le répertoire d'accueil.

Dans l'etat i , votre repertoire courant contient les fichiers de l'exo. 1 .
Dans l'etat i+1, 3 repertoires doivent être crées : src_c pour les fichiers C, src_c++ pour les fichiers C++, et bin pour
a.out et les executables correspondants aux sources précedents.

Question A.4 : (connaissance requise : commandes externes)
Dessiner l'arborescence de votre répertoire d'accueil suite à l'execution des commandes suivantes :

```
$ mv src_c/f2.c src_c++/f2/c++
$ rm -r bin
```

PARTIE B : Gestion de Processus

Question B.1 : (connaissance requise : commandes ps et who)
Déterminer les commandes qui permettent de :
- si l'utilisateur "user1" est connecté,
- le nombre d'utilisateur du système ,
- affiche la liste des utilisateurs par ordre alphabétique,
- le nombre de processus de "user1",
- le nombre de processus de "root"
- enregistre dans le fichier "fuser1" la date et l'ensemble des fichiers de "user1".

...

Université d'Evry-Val d'Essonne
IUP Sciences et Technologie
Unix

TP 2 : Les scripts Shell - sh ou bash

But :
L'objet de ce TP est l'étude des fonctions des scripts sh ou bash.
Prérequis :
Une maîtrise des fonctions des scripts sh ou bash est requise.

Question 1 :
Ecrire un script sh sous forme de fonction qui recherche un nom passe en paramètre dans le fichier annuaire

Question 2 :
Ecrire un script sh sous forme de fonction qui ajoute un nom passe en paramètre dans le fichier annuaire et trie
par ordre alphabetique ce dernier.

Question 3 :
Ecrire un script sh sous forme de fonction qui supprime un nom passe en paramètre dans le fichier annuaire.

Question 4 :
Ecrire un script sh sous forme de fonction qui réalise Affichage de l'annuaire.

Question 5 :
Ecrire un script sh sous forme de fonction qui réalise creer un menu pour gerer un annuaire dans lequel se
trouvent les fonctions recherche, ajout, supprime et affiche déterminés précédemment.

Question 6 : modifier la fonction supprime de facon a ce qu'elle puisse supprimer un abonne parmi plusieurs
ayant le meme nom, en fournissant le prenom de celui-ci.

question 7 : modifier la fonction ajoute de maniere a ce que celle-ci puisse ajouter un abonne en precisant
son nom, prenom et numero de telephone..

Université d'Evry-Val d'Essonne
IUP Sciences et Technologie
Unix

TP 3 : Les scripts Shell - sh ou bash

But :

L'objet de ce TP est l'étude des fonctions des scripts sh ou bash.

Prérequis :
Une maîtrise des fonctions des scripts sh ou bash est requise.

Question 1 :
Ecrire un script sh "rep1.sh" qui donne la liste de tous les fichiers ordinaires accessibles par tout utilisateur dans
une liste de repertoires passes en parametre. Si pas de parametre, traiter le repertoire courant.

Question 2 :
Ecrire un script sh "rep2" qui donne tous les sous repertoires (recurssivement) du repertoire passe en parametre.
"rep2" fera appel à "rep1"(qui devient une fonction) afin d'afficher la liste de tous les fichiers ordinaires
accessibles par tout utilisateur dans chaque sous repertoire.

Question 3 :
Modifier "rep2" en "rep3" de manière a ce que le nom et le nombre de lignes de chaque fichier ordinaire renvoyé
par "rep1" soient affichés. "rep1" nécessite également une modification qui consiste à renvoyer la liste des
fichiers ordinaires accessibles.

...