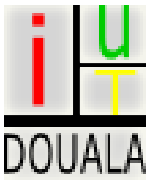




# Initiation au System UNIX

(Cours et exercices)

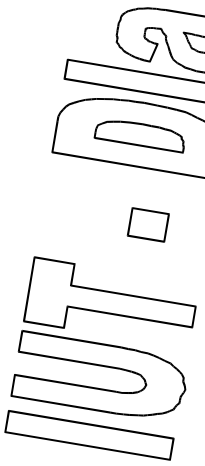
**YOMBI Jean Armand**
Département Génie Informatique
IUT de Douala

Année 2005

Ce cours a été élaboré avec l'aide de la coopération française

# Initiation au Système UNIX

## Table des Matières


IUT-Dla

# Chapitre I : Introduction

Cette introduction donne les principes généraux du système Unix et de ses dérivés, qu'ils soient commerciaux ou open source.

## *I-1 Rappels sur les Systèmes d'exploitations*

En général, quand on parle d'ordinateurs, on distingue deux éléments: le matériel (Hardware) et les logiciels (Software).

- Le hardware est l'ensemble des organes physiques qui composent l'ordinateur, comme le processeur, la RAM, les Disques, les cartes graphiques etc .. tout comme les périphériques (qui sont des composantes externes à l'ordinateur) à savoir l'écran, le clavier ainsi de suite.
- Les logiciels sont les programmes, qui s'exécutent à l'intérieur des ordinateurs.

Le plus important logiciel dans un ordinateur est le système d'exploitation, qui est en quelque sorte le programme qui contrôle l'ordinateur.

Ce système d'exploitation se charge automatiquement au démarrage de l'ordinateur, ensuite il prend le contrôle de ce dernier et gère: la mémoire, le Disque dur les lecteurs, le moniteur, l'imprimante etc..
Il met à la disposition des utilisateurs les commandes qui permettent à ces derniers de travailler avec l'ordinateur.

Les systèmes d'exploitations les plus connus sont les suivants: MS-DOS, Windows 95/98 et Windows NT/2000 de Microsoft, OS/2 de IBM et Unix/Linux, le système dont il sera question ici.

## *I-2 Généralités sur les systèmes Unix*

Le développement du système Unix commence à la fin des années 60 dans les laboratoires Bell de AT&T aux USA. L'objectif était d'avoir un système d'exploitation réseau, puissant, proche du matériel et capable de faire fonctionner les gros ordinateurs et les stations de travail. Pour ce faire, un langage de programmation fut aussi conçu à cet effet, le langage **C**. C'est pourquoi 95% du système **Unix** est écrit en **C**.

L'utilisation d'**Unix** dans les gros ordinateurs et les systèmes connectés se justifie par sa stabilité. C'est ce qui justifie aussi les nombreuses variantes de ce système d'exploitation. Heureusement l'existence des standards (POSIX, ANSI) permet le passage d'une variante à une autre sans grosse difficultés.

Parmi ces variantes, beaucoup sont commerciales qui dans la plupart des cas ont les propre nom, comme **Solaris** (Sun Microsystems), **IRIX** (Silicon Graphics), **AIX** (IBM), **HP-UX** (Hewlett-

Packard) ou **SCO Unix** (Santa Cruz operations).

Mais toutes ces variantes restent à l'usage professionnel et des professionnels et ne sont pas par conséquent répandus dans le grand public jusqu'à la fin des années 80.

En 1990 un étudiant finlandais du nom de Linus TORVALD met le feu aux poudres en développant une version d'Unix pour son PC. C'est la naissance de **Linux** et le début de la démocratisation des systèmes Unix autrefois reconnus comme "Systèmes d'exploitations pour informaticiens". Ici aussi le code source de **Linux** est écrit en C, mais il est disponible sur Internet permettant ainsi à des centaines de milliers de programmeurs d'apporter leur pierre à l'édifice.

**Linux** qui est d'ailleurs conforme au standard POSIX et respecte les principes d'Unix. La large diffusion du système **linux** doit beaucoup à son environnement graphique qui permet de l'utiliser comme s'il s'agissait d'un système Windows. Mais en plus cette simplicité d'utilisation, **Linux** ajoute la stabilité et la robustesse des systèmes Unix. Si on y ajoute des programmes populaire comme open office et autres, tous les ingrédients sont réunis pour faire de lui le système d'exploitation le plus répandu d'ici quelques années.

Dans ce cours nous allons illustrer les concepts de **Unix**, mais les Travaux Pratiques se feront sur un système Linux installé (**Red Hat 9.0**).



## *I-3 Les grands principes des systèmes Unix*

Le système d'exploitation Unix gère les **processus** encore appelés **tâches**. Ces derniers sont des instances de programmes en cour d'exécution.

**Tâche = Processus = instance de programme en cours d'exécution**

Unix est un système d'exploitation **Réseau**, **Multitâche**, **Multi-Utilisateur**, **ouvert**, **indépendant** du matériel et **sécurisé**.

### I-3-1 Système Multitâches

Le système gère les différentes tâches au même moment:
Il n'attend pas la fin d'une tâche pour prendre en charge une nouvelle tâche.
Le processeur fonctionne en temps partagé entre les différentes tâches créées.
Un utilisateur peut demander l'exécution de plusieurs tâches.

### I-3-2 Système Multi-utilisateurs

Le système gère plusieurs utilisateurs au même moment:
Identification,
Authentification,
Sécurité et droits,
Partage des ressources,
Chaque utilisateur peut demander l'exécution de plusieurs tâches.

### I-3-3 Système d'exploitation réseau

Un utilisateur peut directement travailler sur la machine Unix comme il peut travailler à distance sur un terminal ou un autre ordinateur relié à la machine Unix à travers un réseau local ou étendu.

Tout utilisateur d'un système Unix doit s'authentifier en donnant un login (ou identifiant) et un mot de passe. Les deux sont attribués par un administrateur du système et composent ce qu'on appelle un **compte utilisateur**.

Un utilisateur ne doit jamais étendre une machine Unix. Les raisons pour cela nous les verrons plus loin dans ce cours.
Un système d'exploitation moderne dispose d'une interface graphique, comme celle de Windows. Dans Unix celle-ci s'appelle **X11**et dans Linux **Xfree86**. ainsi les utilisateurs ont la possibilité de manipuler le système à travers cette interface ou de le faire directement à l'invite (prompt, qui est le mode le plus robuste).

## Versions d'Unix

System V.R4
BSD
...
Linux

# Chapitre II : Architecture du système Unix

L'architecture du système Unix est organisé en cercle concentrique autour de son noyau qui lui est en contact direct avec le matériel. Comme l'indique la figure suivante, on distingue par ordre:

- Le noyau (**kernel**),
- Les appels système (**primitives**),
- L'interpréteur de commande (**shell**),
- Les utilitaires et outils (**éditeur, compilateur**),
- L'interface graphique (**X11, Motif**) et
- Les applicatifs (**progiciels, SGBD**)

(voir arborescence du système Unix)



## *II-1 Noyau Unix*

Le noyau Unix administre les ressources telles que:

Le processeur (ordonnancement, scheduling)
La mémoire centrale (pagination, swaping)

Et rend les services comme:

la gestion des fichiers
la création de processus
la gestion des échanges d'informations (E/S)
la protection des objets, contrôle, comptabilité

## *II-2 Interpréteur de commandes : shell*

Le SHELL ne fait pas partie du système (cf. schéma de l'architecture du système Unix). Il est interactif, il interprète les commandes et démarre les processus de l'utilisateur. Il intègre les commandes internes et se décline en de nombreuses versions et variantes : **sh, csh, ksh, bash**...

## *II-3 Unix et les outils de communication*

Unix offre de nombreux services en réseau comme:

- Le transfert de fichiers avec **ftp, tftp, rcp** ,
- La connexion sur ordinateurs distants avec **telnet, rlogin, rsh** ,
- La communication entre utilisateurs avec **mail, write, talk,**
- L'accès transparent à des fichiers distants en utilisant **NFS, AFS,**
- Les services Internet tel que **news, www,**
- Etc..

## *II-4 Utilitaires*

Le système Unix met à la disposition des utilisateurs un très grand nombre d'outils destinés à leur faciliter la tâche au quotidien. Il s'agit :

- Des gestionnaire de fichiers,
- Des éditeurs comme **vi, nedit, emacs,**
- Des environnements graphiques,
- Des compilateurs (**C, C++, java, Eiffel**..),
- Des outils du domaine public.

Ces utilitaires concourent au renforcement de la puissance du système d'exploitation Unix.

# Chapitre III : Les Utilisateurs

Les utilisateurs sont répartis dans des groupes, et comme nous l'avons dit plus haut, pour être utilisateur du système Unix, il faut disposer d'un compte utilisateur. C'est à dire avoir un identifiant et un mot de passe qui donne accès au système.

## *III-1 Mot de passe*

Il ne doit pas être choisit au hasard, il doit obéir à certains critères:

C'est la clef d'accès au compte (fichiers, courrier...),
Il doit être fiable (pas de prénom, nom, mot courant, anagramme...), suffisamment long et doit comporter des lettres et des chiffres,
Il faut impérativement le changer régulièrement.



## *III-3 Ouverture d'une session de travail*

Pour travailler dans Unix il faut savoir entrer et sortir du système:

1. Commencer la session en s'identifiant : donner son nom d'utilisateur (login) et son mot de passe (passwd)

```
login: yombi
passwd: ********
```

2. Sortir de la session
   en tapant la commande **exit,**
   ou Bouton gauche/Fin de session en mode graphique,

**Ne jamais éteindre le poste client !**

## *III-4 Fichier des utilisateurs*

Chaque utilisateur possède :
un nom d'utilisateur, attribué par l'administrateur système,
un mot de passe confidentiel,
un répertoire de travail réservé. En principe**:** ***/home/nom_utilisateur***

Le nombre d'utilisateurs n'est limité pratiquement que par la puissance de la machine. Tout le monde peut consulter la liste des utilisateurs du système.

**Exemple de fichier `/etc/passwd`**

```
root:3fG7yCJIvSrS6:0:1:Operator:/:/bin/csh
...
yombi:Cf8vfUVy5LQ:309:300:Armand yombi:/users/profs/yombi:/bin/csh
Dorvor:6V1.zvInJcY:310:300:Patrick
Darvor:/users/profs/Darvor:/bin/csh
...
```

# Chapitre IV : L'interpréteur de commande *Shell*

Le shell est l'interpréteur de commande des systèmes Unix. C'est lui le premier processus à être lancé au démarrage du système. Une fois démarré, il attend les commandes de l'utilisateur.

## *IV-1 Syntaxe d'une commande*

Une commande est un ordre, une tâche que l'on donne au système pur exécution. Après exécution, il doit renvoyer une réponse résultat du traitement effectué.

Une commande Unix doit obéir à la syntaxe suivante:

```
$ cmde [-options] [arguments]
```

Les arguments et options sont facultatifs,
Les options sont précédées de '**-**',
L'espace sépare les arguments.

Exemples :

```
$ ls
$ ls -a /home
```

**Attention ! MAJUSCULE ≠ minuscule**
On dit aussi que Unix respecte la casse, c'est à dire la commande ***mkdir*** est differente de ***mkDir***.

Par ailleurs, Unix est "peu bavard". Il n'affiche que les réponses demandées ou les messages d'erreur mais rien lorsque l'exécution est correcte.

## *IV-2 Touches de survie*

Pendant une session de travail dans un système Unix, il peut arriver qu'un utilisateur soit bloqué d'une manière ou d'une autre. les touches suivantes permettent de le débloquer:

- Effacement d'un caractère **Ctrl-H**
- Effacement d'une ligne **Ctrl-U**
- **Ctrl-C** interrompt la plupart des commandes
- **Ctrl-S** arrête le défilement à l'écran
- **Ctrl-Q** reprise du défilement

## *IV-3 Aide en ligne*

C'est l'équivalent du tutoriel dans les systèmes Windows et leurs applications. Elle permet de se renseigner sur une commande quelconque du système: l'orthographe, la syntaxe, les arguments et les options.
Pour faire appel à cet aide il faut saisir la commande: **man** suivi du nom de la commande sur laquelle on veut se renseigner.

```
$ man cmde
```

La commande man affiche toute la documentation relative à une commande (syntaxe, utilisation, options...). Elle est sans doute la commande la plus importante à connaître.

## *IV-4 Organisation de l'aide*

Le manuel UNIX est décomposé en 8 sections principales :

1. commandes UNIX
2. appels système
3. sous-programmes de bibliothèques
4. format de fichiers UNIX: a.out, dir, fs
5. Divers (table ASCII ...)
6. jeux
7. fichiers spéciaux
8. administration (démarrage, génération...)

## *IV-5 Commandes utilisateurs*

**who, users** : qui est connecté
**finger** : qui est qui
**tty** : nom du terminal
**id, logname, whoami, who am i, groups** : identité de l'utilisateur
**yppasswd** : changement du mot de passe

# Chapitre V : Gestion des fichiers

## *V-1 Arborescence Windows*

Il existe plusieurs arborescences pour chaque unité logique :

- **A**: lecteur de disquette
- **C**: disque dur (partition principale)
- **D:** lecteur de CD-Rom
- **E**: partition temporaire (lecture/ écriture)
- **P**: public
- **Z**: répertoire de base de l'utilisateur

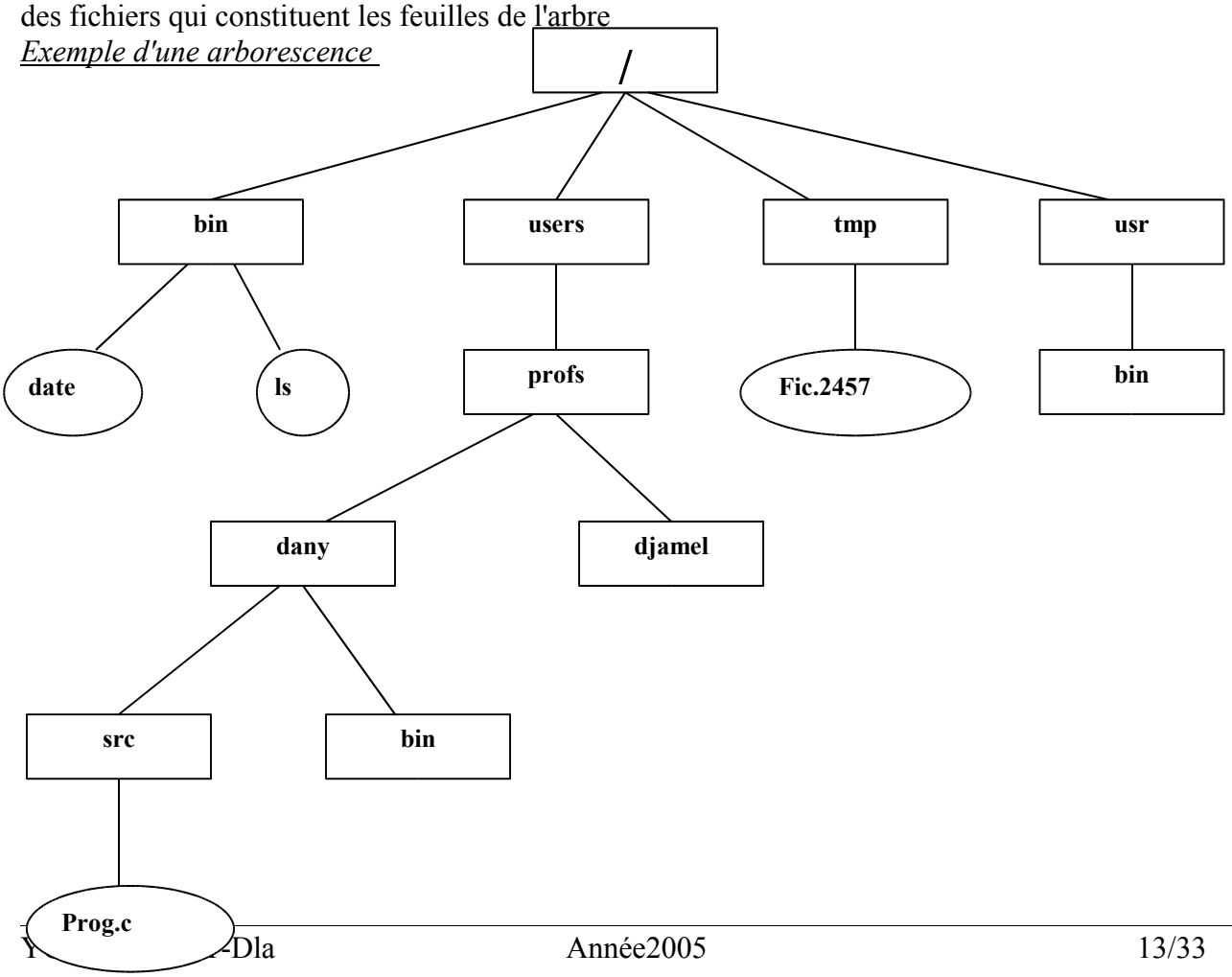
## *V-2 Unix et les données*

Structure hiérarchique des informations
Les informations (données, programmes, images ...) sont stockées dans des fichiers
Les fichiers sont regroupés dans des fichiers spécifiques appelés répertoires

## *V-3 Unix et les fichiers*

La structure de données d'UNIX se présente sous la forme d'une arborescence inversée contenant :
un point d'entrée unique appelé racine et noté /
des répertoires (nœuds de l'arbre) pouvant contenir d'autres noeuds et des feuilles
des fichiers qui constituent les feuilles de l'arbre

*Exemple d'une arborescence*



## V-4 Arborescence des fichiers

Une arborescence UNIX comporte différents types d'objets :
des fichiers ordinaires (suite d'octets),
des répertoires (catalogues),
des liens symboliques (alias),
des fichiers dits spéciaux,

- périphériques
- outils de communication

L'utilisateur a une vision uniforme (commandes et syntaxe identiques) de tous les fichiers.

## V-5 Chemins d'accès

On peut référencer chaque élément de l'arborescence de deux façons :

**1. Par son nom absolu (par rapport à la racine)**
Le chemin absolu désigne la succession des répertoires à parcourir depuis la racine pour accéder au fichier spécifié
Celui-ci est unique et commence toujours par le caractère / .

Exemple : ***/users/profs/yombi/Cours/S1/html/index.htm***

**2. Par son nom relatif (par rapport au répertoire courant)**
Le chemin relatif désigne la succession des répertoires à parcourir depuis le répertoire courant pour accéder au fichier spécifié.

Exemples : ***S1/html/index.htm, index.htm****, ...*

Le caractère « / » a un double rôle :

- il symbolise la racine ( / = racine),
- il sert de séparateur dans un chemin d'accès.

## V-6 Répertoires particuliers

Le **répertoire courant** est le répertoire où l'on se trouve, Il est noté **.**

Le **répertoire père** est au dessus du répertoire courant, Il est noté **..**

***Exemple de chemins d'accès :***

D'après la figure précédente, le chemin d'accès au fichier **prog.c**.
Absolu : ***/users/profs/dany/src/prog.c***

Relatif à djamel : ***../dany/src/prog.c***

## V-7 Arborescence du système Unix

Il s'agit d'une arborescence standard qui varie très peut d'un système Unix à un autre. Cet arborescence comprend des répertoires qui ont une signification particulière :

**/bin** : contient les commandes de base
**/dev** : contient les fichiers spéciaux ( en général les périphériques)
**/etc** : contient les fichiers d'administration du système
**/tmp** : contient les fichiers temporaires de travail
**/usr** : contient les fichiers partageables
**/users** ou **/home**: contient les données des utilisateurs.



## V-8 Commandes répertoires

Il s'agit des commandes usuelles se rapportant aux répertoires. Elles sont à retenir :
**pwd** : affiche le répertoire courant, c'est dire celui dans lequel on se trouve actuellement.
**ls** : liste le contenu d'un répertoire
**mkdir** : création d'un répertoire
**rmdir** : suppression de répertoire
**cd** : déplacement dans un répertoire
**find** : recherche d'un fichier.

### V-8-1 Utilisation de la commande cd

| *Exemple* | *Description* |
|---|---|
| **cd .** | Se déplace vers le répertoire courant c'est-à-dire ne change pas de place! |
| **cd ..** | Va dans le répertoire parent. |
| **cd /** | Saute vers le répertoire racine, à la base de tout le système. |
| **cd /home** | Se déplace vers le répertoire home fils de la racine. |
| **cd /home/h-etie00** | Parcours l'arbre jusqu'au répertoire h-etie00 en passant par la racine, puis par home. |
| **cd ../Mail** | Remonte l'arbre d'un cran, puis va dans le répertoire Mail |
| **cd ../..** | Remonte de deux crans. |

### V-8-2 Options de `ls`

**-a** : liste toutes les entrées, y compris celles préfixées par .
**-l** : format long, caractéristiques détaillées de chaque fichier (type, droits d'accès, nb de liens, propriétaire, groupe, taille en octets, date de dernière modification)
**-t** : présentation par ordre chronologique des modifications (plus récent en tête)

**-r** : affiche dans l'ordre inverse
**-i** : affiche l'inode en première colonne
**-R** : applique récursivement la commande ls à tous les sous-répertoires rencontrés
**-F** : suffixe respectivement par /, * et @ les répertoires, les fichiers exécutables, et les liens symboliques
...

Pour plus d'informations se référer à la commande :
$ **man** ls

### V-8-3 Utilisation de `find`

La commande **find** est ultra puissante, elle permet de faire une recherche sur le système de fichier et d'afficher la liste des fichiers satisfaisant à une combinaison de critères très variés.

*Syntaxe* **: find** ***répertoire critères*** **[-print]**

*Exemple* : **find . -name "*.c" -print**
Cet exemple lance la recherche depuis le répertoire courant (`.`) et affiche le résultat de la recherche (`print`). Le critère de recherche porte sur le nom (`name`) et doit satisfaire le motif (expression régulière) suivant : "`*.c`" c'est-à-dire tous les fichiers d'extension `.c` (autrement dit les programmes sources écris en langage C).

**Options de** find

-name **`chaîne`** : vrai si l'entrée courante s'apparie à la chaîne.
La chaîne peut être générique **si les méta caractères sont banalisés.**

-type **`x`** : vrai si l'entrée est de type x : f fichier, d répertoire, l lien symbolique, b bloc, c caractère, p file, s socket

-print : affiche le chemin de l'entrée courante

-exec **`commande`** : vrai si la commande exécutée renvoie un code d'achèvement 0.
La fin de la commande est marquée par `\;` et elle peut comporter en argument
l'entrée courante symbolisée par {}

...

Pour plus d'informations voir :
**$ man find**

# Chapitre VI : Fichiers et droits **Utilisateurs**

## *VI -1 Convention de noms de fichiers*

Un nom de fichier sous Unix obéit à certaines conventions. Il doit être noté sous la forme suivante :

***Nom . Ext***

La Différenciation entre majuscules et minuscule est faite : **essai.c ≠ Essai.c**
Par ailleurs ces noms de fichier doivent avoir moins de 128 caractères**.**
Contrairement à d'autres systèmes d'exploitations, le '.' n'est pas un caractère spécial.
Dans ces noms ne doivent pas figurer les caractères : **? / * | ! &** ~ **> <**
Dans un répertoire, un nom de fichier est unique**.**



## *VI -2 Les droits d'utilisation d'un fichier*

Dans un système Multi-utilisateurs, il se pose des problèmes de confidentialité pour les données de chaque utilisateur. Un utilisateur ne doit pas avoir accès aux données d'un autre sans l'autorisation de ce dernier.
C'est pourquoi les fichiers sont munis de droits d'accès.

D'après ce principe, Les utilisateurs sont scindés en 3 catégories :
**user** : le propriétaire du fichier.
**group** : les utilisateurs étant du même groupe.
**other** : tous les autres.

A tout élément de l'arborescence (fichier et répertoire) sont associés les droits d'accès :

- de lecture **r** (read). L'accès en lecture autorise la lecture du fichier, c'est-à-dire qu'il est possible d'éditer ce fichier avec une application quelconque pour en voir le contenu. Cet accès est désigné par la lettre r (read). Alloué à un répertoire, ce droit permet de lister les fichiers qu'il contient.
- d'écriture **w** (write). L'accès en écriture permet de modifier un fichier et de le supprimer. Il est désigné par la lettre w (write). Alloué à un répertoire, il autorise la modification et la suppression des fichiers qu'il contient quelques soient les droits d'accès des fichiers de ce répertoire (mêmes s'ils ne possèdent pas eux-même le droit en écriture). Donc attention!
- d'exécution **x** (execute)
  Pour qu'un programme puisse être exécuté, il est indispensable que le droit en exécution sur ce fichier soit autorisé pour l'utilisateur qui souhaite le lancer. Quant à un répertoire, il est tout aussi indispensable que son droit en exécution soit autorisé pour qu'on puisse accéder aux fichiers qu'il contient. Ce droit en exécution est sans effet lorsqu'il est affecté à un fichier qui n'est pas un exécutable.

Pour chaque classe d'utilisateur : **propriétaire, groupe et autres**. Les droits d'un fichier apparaissent avec la commande : **$ ls -l**.

*Exemple :*

**[ Hirondelle]$** `ls -l`

```
-rwxrw-rw- 1 grad 4056 Sep 9 12:27 essai.c
drwxr-xr-x 2 grad 512  Sep 9 12:22 Cours
```
**[ Hirondelle]$**

*Remarques importantes :*
**- Exécutable**
Un programme ne peut être exécuté que si le fichier exécutable correspondant possède le droit d'exécution dans la catégorie à laquelle appartient l'utilisateur.
**- Répertoire**
On ne peut accéder à un fichier que si les répertoires successifs constitutifs du chemin absolu de ce fichier possèdent le droit en exécution. Pour pouvoir lister les fichiers d'un répertoire, ce dernier doit être accessible en lecture.

**- Fichier**
Le droit en exécution n'a aucune incidence sur un fichier non exécutable. Par contre, un script (c'est-à-dire un fichier texte contenant des commandes du Shell) doit avoir les droits en lecture et en exécution pour pouvoir être interprété et exécuté par le Shell



*Exemple de droits* **:**

**- rwx r-x r-- (754)**
lecture, écriture et exécution autorisées pour le propriétaire,
lecture et exécution autorisées pour le groupe,
lecture seule pour les autres.
Le premier caractère indique :

- **d** répertoire
- **-** fichier
- **b** périphérique en mode bloc
- **l** lien symbolique.

## *VI -3 Droits par défaut d'un fichier*

A la création d'un fichier, les droits associés par défaut sont ceux spécifiés par la variable *umask*. C'est une variable d'environnement. A la création d'un fichier, avec la commande *touch* par exemple, ce fichier par défaut possède certains droits. Ce sont 666 pour un fichier **(-rw-rw-rw-)** et 777 pour un répertoire **(-rwxrwxrwx**). Ce sont des droits maximum. Pour changer ces droits par défaut, la commande **umask** est là.

Si on tape la commande : **$ umask 022** on part des droits 666 et on retranche 022, on obtient 644, Par défaut les fichiers auront comme droits 644(-rw-r--r--).

Pour un répertoire, la même commande appliquée permet de retrancher 022 de 777, on obtient 755. ainsi, un répertoire créé aura par défaut les droits 755 soit (-rwx r-x r-x).

**umask** n'est utilisable que si on est propriétaire du fichier.

## *VI-4 Modification de propriété*

Elle doit obéir aux principes suivants :
La modification de propriété et des droits d'accès n'est autorisée qu'au propriétaire du fichier.
Seul le propriétaire peut offrir un fichier à un autre utilisateur.

- **chown** est la commande qui permet le changement de propriétaire
- **chgrp** est celle qui permet le changement de groupe

L'ancien propriétaire n'a plus l'autorisation de modifier les droits...

## VI-5 Modification des droits

La modification des droits d'accès s'effectue avec **chmod** selon 2 méthodes :

1 Valeur numérique calculée sur le poids de **rwx** (avec **r=4**, **w=2**, **x=1**)

2 Valeur littérale **[catégorie opération droit]**
- Catégories : **u=user**, **g=groupe**, **o=others**, **a=all**
- Opérations : ajout (+), retrait (-), affectation (=)
- Droits : r, w et x

***Exemples de modifications*** **:**

```
chmod 741  fich  →  rwx r-- --x
chmod 600  fich  →  rw- --- ---
chmod g+r  fich  →  rw- r-- ---
chmod ug+x fich  →  rwx r-x ---
chmod a=r  fich  →  r-- r-- r--
chmod u+wx fich  →  rwx r-- r--
chmod go-r fich  →  rwx --- ---
```

## *VI-6 Accès à un répertoire*

Pour accéder à un répertoire, les principes suivants sont respectés :

- Toute opération sur un fichier est contrôlée à partir des seuls droits **r, w et x**.
- La création d'un fichier correspond à l'ajout du nom du fichier dans un répertoire.
- La création est contrôlée par le droit **w** du répertoire d'accueil.
- La suppression du fichier est donc conditionnée par les droits du répertoire et non pas par ceux du fichier.
- Pour un répertoire, le droit **x** autorise la traversée et le positionnement sous celui-ci.
- Pour un fichier spécial, **x** n'a pas de signification.

## *VI-7 Commandes fichiers*

**cp** : copie de fichiers
**mv** : déplacement et renommage

**rm** : suppression IRREVERSIBLE
**file** : type du fichier
**ln** : création d'un lien ; syntaxe : **$ ln –s** <*source*> <*destination*>
**chmod** : modification des droits
**cat** : affichage avec défilement
**more** : affichage par page
**df** ou **du** : pour connaître l'espace disque restant
**print** : impression
**head** et **tail** : début et fin du fichier
**vi** et **nedit** ou **emacs**: éditeurs de texte
**grep** : recherche d'expressions dans un fichier
**wc** : comptage des lignes, mots d'un fichier
**sort** : tri des lignes d'un fichier
**uniq** : supprime les répétitions (**uniquement après un tri**)
**cmp** : comparaison de fichiers
**cut** : sélection d'une portion de ligne

*Options de la commande* **`cp`** **:**

**-R** : copie les sous-répertoires récursivement

*Options de la commande* **`rm`** **:**

**-i** : interactif, demande confirmation avant effacement
**-f** : force la suppression
**-R** : récursif

*Options de la commande* **`ln`** **:**
**-s** : lien symbolique

Un lien symbolique est une sorte de pointeur vers un fichier. La création de lien symbolique évite la copie de fichiers identique dans différents répertoires. Par exemple si les utilisateurs d'un groupe ont besoin de certains fichiers, il est inopportun de les copier dans chacun des répertoire des utilisateurs. On crée tout simplement des liens qui pointent vers ces fichiers.

*Options de la commande* **`head`** et **`tail`** :
-n **`i`** : affiche i lignes

*Options de la commande* **`grep`** :
**-e ou -E `expression`** : recherche les lignes contenant l'expression (éventuellement une expression régulière)
**-i** : ignore la casse (majuscule/minuscule)
**-v** : affiche les lignes qui ne sont pas retenues

La commande **grep** permet de retrouver du texte dans un fichier. Sa syntaxe est la suivante :
**$ grep [option]** <*chaîne de caractère*> <*nom de fichier*>

exemple : **$ grep –i -n pacifique *.txt**

cette commande affiche toutes les lignes numérotées (option –n) contenant le mot « pacifique » sans prendre en compte les majuscules (option –i) dans les fichiers finissant par .txt

*Options de la commande* **`wc`** **:**

**-l** : nombre de lignes
**-c** : ..nombre de caractères
**-w** : ombre de mots

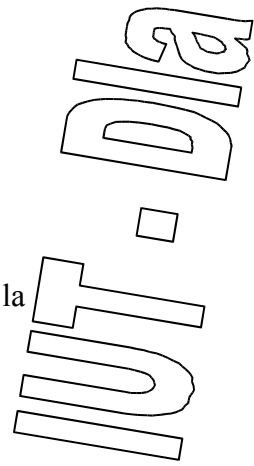
*Options de la commande* **`sort`** **:**

**-r** : inverse l'ordre de tri

*Options de la commande* **`cut`** **:**

-c **`liste`** : sélectionne une liste de caractères
-f **`liste`** : sélectionne une liste de champs
-d **`délimiteur`** : change le délimiteur de champs (tabulation, par défaut)

Une liste est composée d'une ou plusieurs plages séparées par des virgules. Une plage est de la forme :
**N** : Exactement le N-ième
**N-** : Tout à partir du N-ième
**N-M** : Tout entre le N-ième et le M-ième
**-M** : Tout jusqu'au M-ième




## *VI-8 Commandes diverses*

**uname** : affiche le nom du système
**hostname** : affiche le nom de la machine hôte ; celle qui contient le système
**date** : donne ou modifie la date du système
**cal** : affiche un calendrier
**mesg** : accepte/refuse les messages

## *VI-9 Expressions régulières*

Une expression régulière sert à identifier une chaîne de caractère répondant à un certain critère (par exemple chaîne contenant des lettres majuscules uniquement). L'avantage est qu'avec une seule commande, on peut réaliser un grand nombre de tâches qui seraient fastidieuse à faire avec les commandes Unix classiques.

**a) Caractères génériques**

**.** : 1 caractère
**[aeiouy]** : liste
**[^aeiouy]** : liste exclusive
**[A-Z]** : liste par intervalle
^ : début
**$** : fin
\ : banalisation

**b) Répétitions**

***** : de 0 à n
**+** : de 1 à n
**{x,y}** : de x à y
**{x,}** : de x à n
**{,x}** : de 0 à x

Les parenthèses peuvent être utilisées pour grouper les expressions.

*Exemples simples*

**^a.*$** : tous les fichiers commençant par **a**
**^.*e$** : tous les fichiers finissant par **e**
**^[AEIOUYaeiouy].*$** : tous les fichiers commençant par une voyelle
**^.*x{4,}.*$** : tous les fichiers contenant au moins **4 x** successifs
**^a.*e.*z{4}$** : tous les fichiers commençant par un **a** et finissant par **4 z** et contenant au moins un **e**

*Exemples plus complexes*

**^([0-9]{2}/){2}[0-9]{4}$** : une date style JJ/MM/AAAA
**^([0-9]{3}\.){3}[0-9]{3}$** : une adresse IP

# Chapitre VII : Gestion des processus

## *VII-1 Généralités*

On appelle processus une activité du système. Il peut s'agir d'une tâche quelconque que le système effectue ou un programme utilisateur en exécution. Il peut exister des dizaines voire des centaines de processus qui cohabitent dans un système Unix.

- Chaque processus a un identifiant unique pid (Process Identifier)
- Un processus est toujours créé par un autre processus ppid (processus père)
- Les principaux processus du système Unix sont les suivants :

```
  UID   PID  PPID  C    STIME TTY      TIME CMD
 root     0     0  0  sep  16 ?        0:03 sched
 root     1     0  0  sep  16 ?        0:15 /etc/init -
 root     2     0  0  sep  16 ?        0:02 pageout
 root     3     0  1  sep  16 ?        6:48 fsflush
```



## *VII-2 Commandes sur les processus*

**ps** : liste les processus actifs
**time** : donne la durée d'un processus
**nice** : donne la priorité d'un processus
**nohup** : lancement sans interruption
**jobs** : gestion des processus fils
**kill** : émission d'un signal à un processus. Syntaxe : **kill** -numéro pid

**Signaux :**
**1** : SIGHUP fin de session
**2** : SIGINT interruption
**9** : SIGKILL terminaison forcée
**15** : SIGSTOP demande de suspension

## *VII-3 Entrées / Sorties*

Tout processus possède :

- une entrée standard
- une sortie standard
- une sortie d'erreur

| | | |
|---|---|---|
| | | Sortie |
| Entrée | Processus | |
| | | Erreur |

**a) Entrées/sorties standards du shell**

| | | |
|---|---|---|
| | | Sortie : écran |

| | | |
|---|---|---|
| Entrée : clavier | Bash | |
| | | Erreur : écran |

**b) Processus en avant-plan**

```
bash> cmde
```

| | | |
|---|---|---|
| | | Sortie : écran |
| Entrée : clavier | Processus | |
| | | Erreur : écran |
| | | |
| | Bash | |
| | | |

**c) Processus en arrière-plan**

```
bash> cmde &
```

| | | |
|---|---|---|
| | | Sortie : écran |
| | Processus | |
| | | Erreur : écran |
| | | Sortie : écran |
| Entrée : clavier | Bash | |
| | | Erreur : écran |

## *VII-4 Redirections*

Les entrées/sorties d'un processus peuvent être redirigées vers d'autres fichiers ou processus :

**< fichier_entrée** : redirige l'entrée standard vers **fichier_entrée**
**> fichier_sortie** : redirige la sortie standard vers **fichier_sortie**
**2> fichier_erreur** : redirige la sortie d'erreur vers **fichier_erreur**
**>> fichier** : ajoute la sortie en fin du fichier **fichier**
**2>&1** : redirige la sortie d'erreur vers la sortie standard

```
bash$ cmde < fich_ent 2> fich_err
```

| | | |
|---|---|---|
| | | Sortie : écran |
| Entrée : fich_ent | cmde | |
| | | Erreur : fich_err |

## *VII-5 Communication entre processus*

Nous avons dit tantôt que dans un système Unix peuvent cohabiter des dizaines voire des centaines de processus. Certains peuvent soit n'avoir aucun lien avec les autres, soit communiquer avec eux.

Le tube (pipe) permet une communication entre deux processus. Il s'agit d'un canal spécial à travers lequel les processus peuvent engager des échanges.

**cmde1 | cmde2** : La sortie standard du premier est redirigée sur l'entrée du second.
Semblable à : $ **cmde1 > fich_tmp ; cmde2 < fich_tmp ; rm fich_tmp**

```
bash$ cmde1 | cmde2
```

| | | | | |
|---|---|---|---|---|
| | | | | Sortie : écran |
| | | → | cmde2 | |
| Entrée : clavier | cmde1 | | | Erreur : écran |
| | | Erreur : écran | | |

## *VII-6 Enchaînements de commandes*

**cmde1 ; cmde2** : Exécution séquentielle
**cmde1 && cmde2** : Exécution conditionnelle de cmde2. cmde2 est lancée si cmde1 réussit.
**cmde1 || cmde2** : Exécution conditionnelle de cmde2. cmde2 est lancée si cmde1 échoue.

# Chapitre VIII : Automatisation des tâches

Le processus lancé par défaut lors d'une connexion de l'utilisateur est le shell ; l'interpréteur de commandes.
Ce dernier intègre des commandes internes et interprète les commandes de l'utilisateur. Il crée aussi les autres processus de l'utilisateur.
L'interpréteur de commandes utilisé ici est le **bash** (bourne again shell).



## *VIII-1 Les Variables*

Positionnement et portée d'une variable.

**A=bonjour** : Positionnement de la variable A à la valeur bonjour.
A n'est visible que dans le shell courant.

**export A** : A est visible dans les processus fils du shell courant.

**$A** : Lecture du contenu de la variable A.

## *VIII-2 Variables d'environnement*

Les variables d'environnement ou système sont celles définis par le système.

- **$HOME** : Répertoire de base
- **$PATH** : Répertoires de recherche
- **$PS1** : Caractères du prompt
- **$IFS** : Séparateurs de chaînes
- **$TERM** : Type de terminal

## *VIII-3 Caractères spéciaux du* bash

**& > < >> 2>** : ces caractères sont utilisés pour les redirections
**| && || ;** : enchaînements
**. (...) {...} ` ...`** : lancement
*** ? [ ]** : noms de fichiers
**$ { }** : variables
**\ ' ... ' " ... "** : banalisation

## *VIII-4 Substitution de commandes*

Une commande entre apostrophes inverses est remplacée par sa sortie. La ligne de commande est évaluée ensuite.
`*cmde*` est remplacée par la sortie de *cmde*

*Exemple*

```
Bash$  echo le repertoire est : `pwd`
```
le repertoire est : /home/users

## VIII-5 Substitution de noms de fichiers

Dans une commande, les caractères suivants sont remplacés par la liste des noms de fichiers du répertoire courant.
***** : représente toute suite de caractères (même vide)
**?** : représente un seul caractère
**[atv]** : représente un caractère parmi a t v
**[a-d]** : représente un caractère compris entre a et d

## VIII-6 Substitution de variables

**read A** : Lecture des valeurs d'une variable sur l'entrée standard
**echo $A** : Substitution des variables dont le nom commence par $
**echo ${A}jour** : Séparer avec { } en cas de concaténation

## VIII-7 Banalisation de caractères

Les caractères spéciaux sont banalisés et ne sont donc pas interprétés par le shell si
ils sont précédés de \
ils sont situés entre ' '
ils sont situés entre " ", sauf $ \ ? et ??

## VIII-8 Exécution de commandes

```
bash> ( cmde1
> cmde2
> )
```
exécution par un sous-shell .

```
bash> { cmde1
> cmde2
> }
```
exécution par le shell courant .

## VIII-9 Shell scripts

Un script est une suite d'instructions élémentaires qui sont exécutées de façon séquentielle par le langage de script. Il s'agit d'un fichier texte standard pouvant être créé avec n'importe quel éditeur (vi par exemple).

**Entête**

Par défaut, un script est supposé écrit en Bourne-shell.
S'il commence par un commentaire (#les commentaires commencent par un dièse et se terminent en fin de ligne), le shell courant est utilisé à la place de sh.

Si le commentaire est de la forme `#!/bin/xxx`, c'est le shell xxx qui est utilisé.
**Remarque** : Un script qui fonctionnait (en sh) peut ne plus fonctionner lorsqu'on ajoute un commentaire en première ligne. Pour éviter ce problème, on conseille de toujours préciser le shell en première ligne.

### VIII-9.1 Exécution d'un shell script

Soit un script nommé : Mon_script. Pour l'exécuter par un sous shell on lance la commande :
**$ bash** Mon_script : le script est exécuté par le bash.
**$ chmod +x** Mon_script; Mon_script : on confère d'abord le droit d'exécution au script
**$ .** Mon_script ou **$ ./** Mon_script : exécution par le shell courant

### VIII-9.2 Mise au point d'un script

sh -v **`script`** : Mode verbose, recopie sur la sortie chaque ligne avant interprétation
sh -x **`script`** : Mode trace, recopie le résultat de l'interprétation de chaque ligne

**Arguments d'un shellscript**
**$0** : le nom de la commande
**$1** : le premier argument
**$2 ... $9** : le deuxième ... neuvième argument
**$*** : l'ensemble des arguments
**$#** : le nombre d'arguments
**$$** : l'identification du processus
**$?** : le code de retour
*Exemple :*

```
#!/bin/sh
# shell script argum
# affichage des arguments
echo le nombre d arguments est $#
echo la commande est $0
echo le premier argument est $1
echo les arguments sont $*
```

*Décalage des arguments* : `shift`

```
#!/bin/sh
# script echopara illustrant shift
echo $1 $2 $3
P1=$1
shift
echo $1 $2 $3
echo $P1
```

```
$ echopara 1 2 3 4
1 2 3
2 3 4
1
```

*Commandes shellscript :*
**echo** : affichage des arguments

**sleep** : attente pendant n secondes
**test** : expression conditionnelle
**env, set** : variables d'environnement
**expr** : expression arithmétique
**eval** : évaluation d'une expression

*Commande* test
Syntaxe :$`test -option arg` **ou** `[ -option arg ]` **ou** `[ arg1 -comp arg2 ]`

*Tests sur les fichiers*
**-f** : vrai si arg est un fichier ordinaire
**-d** : vrai si arg est un répertoire
**-r** : vrai si arg est accessible en lecture
**-w** : vrai si arg est accessible en écriture
**-x** : vrai si arg est accessible en exécution

*Test sur les entiers*
**-eq** : vrai si arg1 est arg2 sont égaux
**-ne** : vrai si arg1 est arg2 sont différents
**-lt** : vrai si **arg1 < arg2**
**-le** : vrai si **arg1 ≤ arg2**
**-gt** : vrai si **arg1 > arg2**
**-ge** : vrai si **arg1 ≥ arg2**

*Test sur les chaînes*
**-z** : vrai si arg est une chaine vide
**-n** : vrai si arg a une longueur > 0
= : vrai si arg1 et arg2 sont égales
**!=** : vrai si arg1 et arg2 sont différentes

*Opérateurs logiques*
**!** : négation
**-a** : et logique
**-o** : ou logique

Exemple : `[ $# -ne 2 -o -r $1 -a -w $2 ]` Le script a un nombre d'arguments différents de deux, ou son premier argument est un fichier accessible en lecture et son second argument est un fichier accessible en écriture.
**Commande** expr
*1Réalise des calculs arithmétiques
*2Utilisation des 4 opérateurs + - * /
*3Les opérandes sont des entiers ou des variables
*4Exemple : **expr $A * 2 + $B**
**Commande** eval
**eval** *nom_variable* : Exécution de la commande contenue dans une variable

*Exemple*

```
bash>  echo $A
pwd
bash>  eval $A
/home/users/anne1
```

## *VIII-10 Structures de contrôle*

- **Structure conditionnelle if**

**Syntaxe :**

**Première forme :**

```
if expression_logique
then liste_commandes1
fi
```

**deuxième forme :**

```
if expression_logique
then liste_commandes1
else liste_commandes2
fi
```

**troisième forme :**

```
if expression_logique1
then liste_commandes1
elif expression_logique2
     liste_commandes2
...
else liste_commandes0
fi
```

*Exemple*

```
#!/bin/sh
# shell script argdeux
# verifie le deuxieme argument
if [ $2 = deux ]
then
     echo arg 2 OK
else
     echo arg 2 KO
fi
```

- **Liste de cas case**

**Syntaxe:**

```
case variable in
  modele1)liste_commandes1;;
  modele2)liste_commandes2;;
```

```
        *       )liste_commandes_defaut;;
      esac
```

*Exemple*

```
#!/bin/sh
# shell script argcpt
# nombre des arguments
case $# in
  0) echo trop peu ;;
  1) echo un seul ;;
  2) echo deux arguments ;;
  *) echo trop d arguments ;;
esac
```

- **Itérations bornées for**

**Trois syntaxes :**

```
for variable in liste_valeurs
do liste_commandes
done
```

La variable prend successivement chacune des valeurs.

```
for variable in *
do liste_commandes
done
```

La variable parcourt l'ensemble des noms de fichiers (et répertoires, etc.) du répertoire courant.

```
for variable
do liste_commandes
done
```

La variable parcourt l'ensemble des arguments du script.

*Exemple*

```
#!/bin/sh
# shell script fichtyp
# affichage du type des fichiers
for fichier in `ls -d /*[e]*`
do
     file $fichier
done
```

- **Itérations non bornées while et until**

**Syntaxe :**

```
while expression_logique
do liste_commandes
done

until expression_logique
```

**do** *liste_commandes*
**done**

*Exemple*

```
#!/bin/sh
# shell script fichls
# saisie nom de fichier
read f
while [ $f != fin ]
do
ls -l $f
read f
done
```

## Bibliographie

1) Le Système UNIX ; *Steve Bourne* ; Masson
2) Unix : initiation et utilisation : cours et exercices corrigés. *Armspach, J.-P.*, Colin,
3) Unix par la pratique ; *Dominique Bouillet* ; Ellipse.
4) Unix System V ; *AT&T* ; Masson
5) La programmation sous Unix; *J. M. Rifflet* ; Ediscience