Conservatoire National des Arts et Métiers
FOD Ile De France

MANUEL LINUX
Prise en main de Linux

10 mai 2020

Version	*Auteur*	*Commentaires*
10 octobre 2008	*Emile Geahchan*	*Version Initiale*

SOMMAIRE

UNIX vs LINUX

Le Système Unix

Unix est un système d'exploitation Multitâche / Multiplateformes conçu par Ken Thompson en 1969, dans le cadre d'un projet des "Bells Laboratories" du groupe AT&T[1], projet destiné à contrer le monopole d'IBM dans le monde des systèmes d'exploitation.

Son nom de baptême "Unics" (Uniplexed Information and Computing Service) est un clin d'œil à l'ambitieux projet temps partagé "Multics" (Multiplexed Information and Computing Service), mené conjointement par le MIT[2], les Bells Laboratories et la General Electric.

Pour des raisons de marketing Unics deviendra "Unix". La première version commercialisée "Unix-v6[3]" parait en 1975, la dernière version aboutie "Unix System V - version 4" paraitra en 1989.

Les Unix-Like

Unix est protégé par un brevet américain de type "secret de fabrication", vous avez donc le droit de développer un système Unix compatible[4] ou "Unix-like", à partir du moment ou vous n'utilisez pas les sources d'ATT.

C'est ce qui a été fait par Richard Stallman, Andrew Tanenbaum, Linus Torvalds et l'Université de Berkeley aux Etats Unis :

Richard Stallman, un des pionniers du logiciel libre, démarre les travaux de son système GNU (Gnu is Not Unix) en 1883 dans le but de fournir une alternative libre à Unix.

En 1987 Andrew S. Tanenbaum publie son livre "Systèmes d'exploitation : Design et Implémentation", livre qui comporte le code source du système Minix, mais son système ambitieux, autour d'un concept de micronoyau aura du mal à convaincre.

En 1991, Linus Torvalds crée un noyau Linux. A ce moment GNU était quasiment prêt à l'exception de son noyau, l'association de Linux avec la panoplie d'utilitaires GNU va permettre de créer un système d'exploitation libre et complet.

---

[1] American Telgraph & Telephone
[2] Massachusetts Institute of Technology
[3] Développé en langage "C"
[4] Tout comme Pepsi-Cola a créé une boisson se rapprochant du Coca-Cola

L'Université de Berkeley qui fait partie des tous premiers utilisateurs d'Unix - elle a même participé à un certain nombre de ses évolutions - décide en 1991 de réécrire les sources d'Unix afin de créer un système libre baptisé BSD (Berkeley Software Distributions), mais un long procès l'opposera à ATT ce qui va en freiner les développements, même si Berkeley finira par gagner le procès en 1994.

Les Distributions Linux

Finalement c'est le noyau Linux avec une large panoplie d'utilitaires GNU le tout principalement sur plateforme PC qui s'est imposé sur le marché. Aujourd'hui Il existe de nombreuses distributions[5] Linux : Slackware, RedHat, Debian, Ubuntu, Suse, ... Cependant le noyau est unique, c'est celui maintenu par Linus Torvalds (appelé Vanilla Kernel).

Linux est un logiciel libre sous licence GPL[6], cependant certains éditeurs rajoutent des composants propriétaires à leur distribution afin d'en faire des packages commerciaux.

Les Unix de souche

Les sources d'Unix sont disponibles commercialement (autrefois propriété d'ATT et appartenant aujourd'hui à l'Open-Group) SUN, HP, IBM et Bull ont construit leur système d'exploitation à partir des sources d'Unix.

Berkeley Software Distributions

Les distributions BSD, proches de l'Unix ATT vont également aboutir avec 3 variantes : FreeBSD, NetBSD et OpenBSD.

Apple a basé son nouveau système d'exploitation "Darwin" sur FreeBSD.

La marque Unix

La marque "Unix" a été déposée par ATT et appartient aujourd'hui à l'Open Group, d'où les appellations : GNU, Minix, Linux, BSD, Solaris (Sun), HP-UX (HP), AIX (IBM, Bull), Darwin etc.

Les communautés Linux

« Étant donné un ensemble de bêta-testeurs et de co-développeurs suffisamment grand, chaque problème sera rapidement isolé, et sa solution semblera évidente à quelqu'un ».

Eric Raymond[7] "The Art of UNIX Programming"

[5] Par distribution on entend un package complet : Système d'exploitation + Utilitaires et comportant une procédure d'installation de l'ensemble.
[6] "General Public Licence" qui implique que tout logiciel dérivé d'un logiciel libre est un logiciel libre
[7] *Les 2 grands pionniers du logiciel libre : Richard Stallman et Eric Raymond*

I- INSTALLATION DE LINUX

Nous vous préconisons Open Suse pour suivre ce cours. Open Suse est la version libre de la distribution Suse laquelle fait partie des distributions ayant acquis une certaine notoriété en entreprise.

Si vous souhaitez utiliser une autre distribution Unix-like, les commandes en mode terminal présentées dans ce document seront le plus souvent compatibles.

- Les commandes présentées qui ne seront pas compatible avec l'ensemble des distributions Unix-like seront signalées avec le symbole ci-contre.

I.1- Installation d'Open Suse

Nous vous conseillons une installation d'Open Suse en tant que machine virtuelle pour ne pas remettre en question le partitionnement de vos disques durs.

Nous vous préconisons VMware Player comme moteur de virtualisation. VMware Player est une version gratuite de WMware, logiciel ayant tout comme Suse, acquis une certaine notoriété en entreprise.

Ci-dessous les étapes de l'installation :

1) Installation du moniteur VMware Player :

a) Télécharger le moniteur de machines virtuelles VMware Player.

b) Installer VMware Player.

2) Téléchargement d'Open Suse :

Télécharger l'image ISO d'OpenSuse.

3) Création de la machine virtuelle Open Suse :

a) Lancez VMware Player

b) "Create a new virtual machine"

c) "Installer Disc Image File" et pointez sur votre image ISO

Poursuivre l'installation, OpenSuse sera installé dans un répertoire sur votre disque dur.

Si vous souhaitez être synchrone avec les manips décrites dans ce document :
- le mot de passe de `root` est `formateur`.
- créez un utilisateur `cnam` avec comme mot de passe `auditeur`.

Au niveau de la configuration je vous conseille :
- processeurs : tous
- mémoire : 2MO
- espace disque : 10 MO

I.2- Lancement / Arrêt d'Open Suse

Lancement de la machine virtuelle Open Suse :

a) Lancez VMware Player

b) Double cliquez sur votre machine virtuelle dans la colonne de gauche.

** Vous pouvez également cliquer sur le fichier avec l'extension `".vmx"` dans le répertoire correspondant à votre machine virtuelle.*

c) Attendez que le système se lance, puis agrandissez la fenêtre pour avoir le bureau au complet.

❖ Pour affecter le clavier à la machine physique (hôte) utilisez : `Ctrl/Alt` et pour le réaffecter à la machine virtuelle (invitée) utilisez `Ctrl/G`.

Attention ne fermez pas brutalement la fenêtre de la machine virtuelle, mais arrêtez d'abord la machine virtuelle.

Pour arrêter la machine virtuelle : `Icône Ordinateur -> Arrêter.`

Pour changer d'utilisateur : `Icône Ordinateur -> Se déconnecter.`

I.3- Linux mode Terminal

Pour ouvrir un "Terminal*" :

`Clic droit sur le bureau -> Ouvrir un terminal.`

L'ouverture d'un terminal peut être différente selon les distributions Linux

❖ Le répertoire personnel de l'utilisateur `cnam` est `/home/cnam`.

❖ Le répertoire personnel de `root` est `/root`.

Attention :
- les systèmes Unix-like font la différence entre les lettres minuscules et les lettres majuscules (case sensitive).
- les commandes se terminent (se valident) par un retour chariot

Premiers pas

`$ date`	afficher la date et l'heure
`$ uname -a`	afficher la version du noyau Linux
`$ pwd`	afficher le nom du répertoire courant
`$ ls`	afficher la liste des fichiers du répertoire courant
`$ cd /`	aller dans le répertoire racine
`$ cd`	retourner dans votre répertoire personnel
`$ cd /home/Durand`	aller dans le répertoire "/home/Durand"
`$ echo Hello`	afficher "Hello"
`$ echo Hello > toto`	créer le fichier "toto" contenant le texte "Hello" (cf. "redirection du fichier de sortie" plus bas)

`$ cat toto`	afficher le fichier "toto"
`$ cp toto tata`	recopier "toto" dans "tata"
`$ mv toto titi`	renommer le fichier "toto" en "titi"
`$ rm titi`	supprimer le fichier"titi"
`$ mkdir srep`	créer le sous-répertoire "srep"
`$ cd srep`	aller dans le sous-répertoire "srep"
`$ cd ..`	aller dans le répertoire père
`$ rmdir srep`	supprimer le répertoire "srep"
`$ grep Hello tata`	rechercher la chaine "Hello" dans le fichier "tata"
`$ find / -name tata`	rechercher le fichier "tata" à partir de la racine
`$ find . -name "*.c"`	rechercher les fichiers `"*.c"` à partir du répertoire courant
`$ sed* s/Hello/Bonjour/ toto`	afficher le contenu du fichier "toto" en remplaçant "Hello" par "Bonjour" (éditeur de texte en mode ligne `sed`)
`$ vi* tata`	Editer le fichier "tata" et le créer s'il n'existe pas (éditeur de texte `vi` en mode pleine page)
`$ more tata`	afficher le fichier "tata" page par page (`Q` pour quitter)

** En dehors des éditeurs historiques Unix `sed` et `vi`, les différentes distributions proposent des éditeurs plus conviviaux en mode graphique : `gedit`, `nano`, `atom`, `vim` …*

Le manuel en ligne `man` permet d'obtenir le mode d'emploi des commandes (cf. chapitre "La Documentation Unix-like") :

`$ man ls`	affichage de la page `ls` du manuel Linux

Jokers

Il s'agit de jokers pour les noms de fichiers

`$ ls *`	liste de tous les fichiers du répertoire courant (quelles que soient les extensions)
`$ ls *.bash`	liste de tous les fichiers avec l'extension ".bash"
`$ ls t?t?`	"?" représente n'importe quel caractère cela pourrait donc correspondre à "toto" ou "tata" …
`$ grep Hello *.html`	rechercher la chaine de caractère "Hello" dans tous les pages HTML du répertoire courant
Attention :	
`$ rm -r *`	Remove récursif (option `-r`) Vous allez détruire tous les fichiers du répertoire courant et tous les sous-répertoires

Changement de mot de passe

L'utilisateur peut changer son mot de passe avec la commande `passwd`

```
$ passwd
  enter new password :
  enter new password again :
```

Substitution d'utilisateur

L'utilisateur peut ouvrir une session de travail imbriquée en se substituant à un autre utilisateur avec la commande "su", par exemple pour se transformer en super-utilisateur root :

```
$ id                        Identifiant de l'utilisateur
uid=1000(cnam) ...
$ su root                   Si votre système Linux refuse cette commande
Mot de Passe ...            cf. "la directive "sudo" ( gestion des utilisateurs)
# id
uid=0(root) ...
# exit                      Quitter la session imbriquée
$ id
uid=1000(cnam) ...
```

- L'invite de commande du super-utilisateur (ou administrateur ou root) est # et l'invite de commande de l'utilisateur normal est $

- La commande su avec l'option "-" permet de se récupérer l'environnement du nouveau compte (répertoire personnel etc.) : $su - root

Redirection du fichier de sortie standard

La redirection de la sortie consiste à rediriger vers un fichier la sortie standard d'une commande qui est par défaut l'écran.

```
$ ls > listing      liste des fichiers du répertoire courant dans le fichier listing
```

Une variante la redirection du fichier de sortie en "ajout" :

```
$ echo "LISTING" > listing     On met le titre LISTING dans le fichier listing
$ ls >> listing                puis la liste des fichiers courants
```

Enchainement des commandes

Il s'agit de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande (l'entrée standard d'une commande est par défaut le clavier), par exemple l'extraction des images jpeg de la liste des fichiers du répertoire courant:

```
$ ls | grep *.jpg       L'opérateur "|" s'appelle "pipe"
```

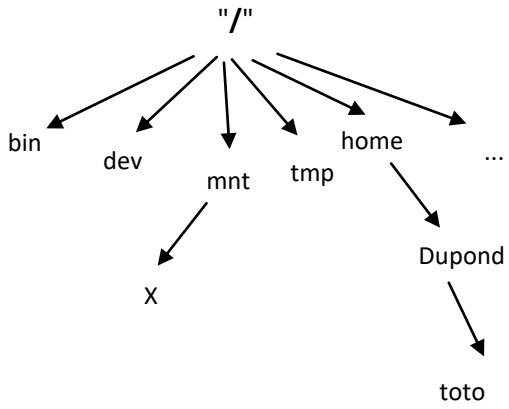
- Les enchainements de commandes sont à la base de la philosophie d'UNIX, ils permettent la réalisation de commandes élaborées à partir de commandes élémentaires :

Nous invitons le lecteur à se familiariser avec les commandes du présent chapitre en répétant les exemples présentés et en effectuant des recherches complémentaires à l'aide du manuel ***man*** *ou sur l'Internet.*

II- LE SYSTEME DE FICHIERS

II.1- L'arborescence

Linux utilise un système de fichiers arborescent. Par exemple le fichier `/home/Dupond/toto` ci-dessous :



La commande `"ls -l"` permet d'afficher les attributs d'un fichier :

```
$touch toto          création d'un fichier vide toto
$ls -l toto
-rw-r--r--           1 cnam cnam 0 2016-12-09 10:05 toto
```

- Le premier caractère à gauche indique le type du fichier :
  - `-` fichier normal (par opposition à fichier spécial, cf. plus bas)
  - `d` répertoire
  - `l` lien symbolique (cf. chapitre correspondant)
  - `b,c` fichier spécial (cf. paragraphe suivant)

  ** Dans notre exemple `"-"` = fichier normal*
- Les 9 caractères suivants indiquent les protections associées : `rw-r--r--` (cf. chapitre droits d'accès aux fichiers)
- Le nombre suivant indique le nombre de liens physiques sur le fichier (cf. chapitre correspondant) :
- Ensuite le nom du propriétaire et du groupe d'appartenance : `cnam, cnam`
- Ensuite la longueur du fichier en octets : `0` *(fichier vide)*
- Ensuite la date de dernière modification : `2006-12-09 10:05`
- Enfin le nom du fichier : `toto`

Remarques :

- Comme pour les commandes Linux les noms des fichiers sont sensibles à la casse.
- Le nom ou chemin absolu d'un fichier commence à la racine : `/home/Dupond/toto`
- On peut aussi identifier un fichier par un chemin relatif :

  `Dupond/toto` fichier "toto" vu du répertoire `"`/home"

  `toto` fichier "toto" vu du répertoire `"`/home/Dupond"
- Les éventuels suffixes des noms de fichiers (`".exe"` ou `".obj"` par exemple), n'ont aucune signification particulière pour le système Linux et notamment les exécutables ne portent pas de suffixe particulier.
- Les fichiers cachés sont ceux dont le nom commence par un point, pour afficher les fichiers cachés il faut utiliser l'option `-a` :

  `$ ls -a` afficher la liste de tous les fichiers du répertoire courant, y compris les fichiers cachés

Fichiers spéciaux

Un fichier spécial est un logiciel pilote (driver), il s'agit d'un mécanisme Unix/Linux de banalisation des entrées/sorties, à chaque périphérique (écran/clavier, disques, voie de communication …) correspond un fichier spécial, après ouverture de ce fichier et éventuelle configuration, les entrées/sorties s'effectuent au travers de primitives **"read()"** **"write()"**de la même façon que l'on lit et écrit dans un fichier normal.

Par exemple les fichiers spéciaux correspondant aux accès disques durs* :
- **/dev/sda1, sda2 ...** **"a"** correspond au premier disque dur, partition** **2.**
- **/dev/sdb1, sdb2 ...** **"b"** correspond au second disque dur partition** **2.**
…

** le préfixe* **sd** *correspond aux disques "sata".*
*** Les partitions* **1** *à* **4** *correspondent aux* **4** *partitions principales des disque DOS et les partitions* **5** *à* **n** *aux partitions secondaires.*

Création d'un Système de Fichiers

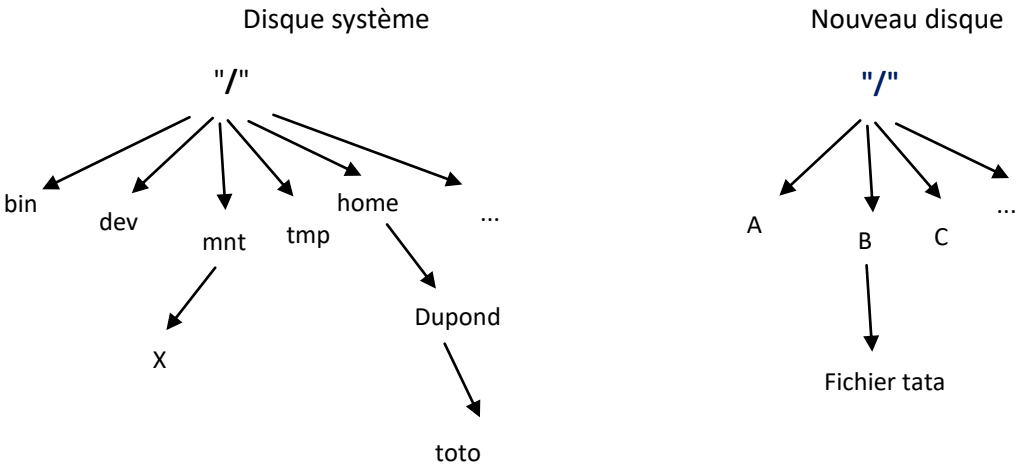
L'utilitaire Linux de création d'un système de fichiers à l'intérieur d'une partition est la commande **"mkfs"** (make file system) :

mkfs /dev/sda3 créer un système de fichiers sur la 3ème partition du premier disque dur de type par défaut ext2

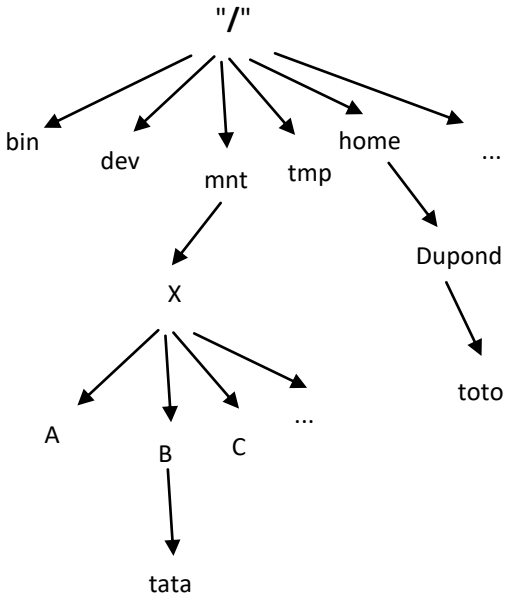
II.2- Le montage des partitions

La grande différence entre le système de fichier Unix-like et celui de Windows, c'est qu'il n'y a qu'une seule arborescence de fichiers dans les systèmes Unix-like : le disque système `"/"` ; alors qu'il y a plusieurs arborescences possibles dans le système Windows (une par lecteur logique) : `"C:\", "D:\", "E:\"` etc.

Ce qui veut dire que si on installe un nouveau disque (ou une nouvelle partition), la racine du nouveau disque sera vue comme un répertoire du disque système `"/"`, on appelle cela le "montage" du nouveau disque sous le disque système :



Après montage du nouveau disque sous le répertoire de montage `/mnt/X` le fichier `/B/tata` sur la partition externe devient le fichier `/mnt/X/B/tata`



Remarque :
Les éventuels fichiers préexistant sous un répertoire de montage (`/mnt/X` dans l'exemple précédent) seront masqués durant tout le montage.

Pour lister les partitions actives sur votre système Linux :

```
$ df -k
```

```
(Fichier spécial)                                          (rep. de montage)
Filesystem 1024-blocks  Free %Used Inused %Inused Mounted on
/dev/sda1      32768     16016  52%  2271      14%    /
/dev/sda2      524288    395848 25%  421       1%     /home
...
```

Commandes de Montage

La partition système (disque système) est montée automatiquement lors du lancement du système Linux, pour monter une autre partition utiliser la commande **mount :**

```
# mount device  /montage  -t  type -o  size=1024m
```

- **device** représente le fichier spécial de la partition
- **/montage** représente le point de montage

```
# mount                    liste des partitions montées

# umount device            démontage de la partition
# umount    /montage       idem
```

- Le fichier système **/etc/mtab** contient la liste des partitions montées.
- Le fichier système **/etc/fstab** permet de programmer le montage automatique de partitions au lancement de Linux.

II.3- Les liens physiques

Créer un lien physique consiste à donner un second nom ou chemin d'accès complet à un fichier.

Ce lien dit physique n'est qu'un "alias" permettant de faire référence au même fichier avec un nom ou un chemin d'accès différent. Il n'y a pas de duplication de fichier.

Par exemple :

```
$ ln toto alias-toto
$ ls -l toto
- rw-r--r--       2 cnam cnam 0 2006-12-09 10:05 toto
```

Le `2` dans les attributs de `toto` et de `alias-toto` indique que ce fichier possède `2` noms, pour supprimer ce fichier il me faudra supprimer les `2` noms (c'est la suppression du dernier nom qui supprime physiquement le fichier).

```
$ ln toto /home/Durand/toto
$ ls -l toto
- rw-r--r--       3 cnam cnam 0 2006-12-09 10:05 toto
```

J'ai créé un lien pour le fichier "toto" dans le répertoire de l'utilisateur "Durand".

Mais alors comment Linux identifie-t-il les fichiers puisque ce n'est pas le nom ou le chemin qui sert d'identifiant ?

Linux identifie les fichiers par un numéro d'inode. Un numéro d'inode est relatif à une partition : Sur une même partition le numéro d'inode est unique.

C'est l'option `"i"` de la commande `"ls"` qui permet d'afficher le numéro d'inode :

```
$ ls –i toto
 1254876
```

Pour trouver tous les fichiers de numéro d'inode `1254876` s'ils existent :

```
$ find  /  -inum  1254876
```

- *Attention on ne peut pas créer de liens physiques vers un répertoire ou sur un fichier dans une partition différente.*

II.4- Les liens symboliques

Un lien symbolique est un fichier qui contient une référence à (c'est un pointeur vers) un autre fichier.

Les liens symboliques peuvent s'appliquer aux répertoires et à des partitions différentes.

```
# ln -s  /home/Dupond  /home/perso
# ls -l /home/perso
lrwxrwxrwx 1 root root ... -> /home/Dupond
```

/home/perso est un lien symbolique qui pointe sur /home/Dupond

Remarques :
- L'accès à un fichier au travers d'un lien symbolique (en lecture/écriture/exécution) est transparent sauf dans le cas de la suppression : La suppression d'un lien symbolique n'entraine pas la suppression du fichier référencé.
- La suppression du fichier référencé sans supprimer le lien symbolique aboutira à un lien symbolique inutilisable.

Attention : Toujours utiliser un chemin absolu lors de la création d'un lien symbolique, autrement ce lien serait contextuellement dépendant.

III- LA GESTION DES UTILISATEURS ET LES DROITS D'ACCES

III.1- Les Utilisateurs

Il y a 3 types d'utilisateurs sur une machine Unix :
- le super-utilisateur **"root".**
- les administrateurs ou **"sudoers"** qui sont les utilisateurs faisant partie du groupe **"admin"**
- les utilisateurs de base

Le Super Utilisateur

Le super utilisateur **root** possède tous les privilèges c'est à dire qu'il possède implicitement tous les droits d'accès à toutes les commandes et à tous les fichiers données ou exécutables.

Les Sudoers

Pour des raisons de sécurité il est préférable de ne pas se maintenir trop longtemps dans une session **root**, car si un pirate prend la main sur votre ordinateur pendant une telle session, il bénéficiera automatiquement des privilèges **root**. La commande **sudo** (exécuter en tant que **root**) va permettre de n'exécuter qu'une commande **root** à la fois, elle est réservée aux "**sudoers**" c'est à dire aux utilisateurs du groupe **admin** lesquels devront s'authentifier avec leur propre mot de passe :

Par exemple

$ sudo ls /root	lister les fichier du répertoire personnel de root
enter password :	demande du mot de passe du sudoer
	affichage des fichier ...

Les Utilisateurs de base

Les utilisateurs de base ne disposent que d'un jeu de commande système réduit, et bien entendu ils n'ont pas accès à l'ensemble des fichiers données ou exécutables (cf. chapitre "Les Droits d'Accès aux Fichiers").

Compte root désactivé

Sur certains systèmes Linux, pour des raisons de sécurité le compte **"root"** est désactivé et l'accès aux privilèges **root** se fait uniquement au travers de la commande **sudo**.

Dans ce cas, si vous souhaitez réactiver le compte **root** il faut faudra lui affecter un mot de passe :

$ sudo passwd root	attribuer un mot de passe à root
enter password :	votre propre mot de passe
root password :	le mot de passe choisi pour root

Fichier système des utilisateurs

Les utilisateurs sont enregistrés dans le fichier système **"/etc/passwd"**, chaque ligne de ce fichier correspond à un utilisateur et contient 7 champs d'information séparés par le caractère **":"** :

- Nom d'utilisateur :
  - Il doit être unique et composé de lettres minuscules ou majuscules et de chiffres (le : est interdit ainsi qu'un chiffre en première lettre)

- Mot de passe :
  - Le mot de passe n'apparait pas directement dans le fichier **/etc/passwd** (on trouve en lieu et place un x), il figure de façon cryptée dans un fichier **/etc/shadow** accessible en lecture uniquement par le super-utilisateur **root**.

- UID :
  - Numéro identifiant d'utilisateur, **"root"** a pour identifiant **"0"**. En fait c'est l'identifiant numéro qui identifie un utilisateur et non pas son nom, on peut donner 2 ou plusieurs noms à un utilisateur (c'est à dire créer plusieurs utilisateur de même identifiant numéro mais de noms différents).

- GID primaire :
  - Numéro du groupe primaire dont cet utilisateur est membre, les autres groupes dont cet utilisateur serait éventuellement un membre figureront dans le fichier système **/etc/group.**

- Commentaire :
  - champ commentaire

- Répertoire personnel :
  - C'est le répertoire qui appartient à l'utilisateur et où il se trouve placé à chaque ouverture de session.

- Shell :
  - Interpréteur de commandes affecté à l'utilisateur :
    - **/bin/sh** (Bourne shell)
    - **/bin/bash** (Bourne again shell)...

```
$ more /etc/passwd
root:x:0:0:root:Root:/root:/bin/bash
...
cnam:x:1000:1000:cnam User:/home/cnam:/bin/bash
...
```

Fichier système des groupes

Le fichier `/etc/group` contient la liste des groupes valides, chaque ligne de ce fichier correspond à un groupe et contient 4 champs d'information séparés par le caractère `":"` :

- Nom de groupe :
  - Limité à huit caractères.

- Mot de passe crypté :
  - Non utilisé.

- GID :
  - Numéro du groupe.

- Liste des membres :
  - Chaque utilisateur membre du groupe y apparait, les noms étant séparés par des virgules.

```
$ more /etc/group
root:x:0:
cnam:x:1000
admin:x:115:cnam
...
```

- "cnam" est membre du groupe `"admin".`
- dans `/etc/passwd "cnam"` *est membre du groupe primaire* `"cnam"`
- dans `/etc/passwd "root"` *est membre du groupe primaire* `"root"`

(`/etc/group` *ne consigne pas les groupes primaires des utilisateurs qui figurent déjà dans* `/etc/passwd`*).*

Gestion des Utilisateurs

Vous devez avoir les privilèges **root** pour utiliser ces commandes.

Ajout d'un groupe

Commande **groupadd***

```
$ groupadd emile
$ more /etc/group
...
emile:x:1003:
...
```

avec :

1003	identifiant attribué au groupe **emile**

Ajout d'un utilisateur

Commande **useradd***

```
$ useradd --create-home -g emile emile
$ more /etc/passwd
...
emile:x:1002:1003::/home/emile:/bin/bash
...
```

avec :

-g	gid du groupe primaire
--create-home	nécessaire pour la création du répertoire personnel
1002	identifiant attribué à l'utilisateur **emile**
1003	identifiant du groupe **emile**

Changer le mot de passe

$ passwd	changer son propre mot de passe
# passwd emile	changer le mot de passe d'un autre utilisateur (commande réservée à **root**)

Autres commandes

usermod*	modification des attributs d'un utilisateur
userdel*	suppression d'un utilisateur
groupmod*	modification des attributs d'un groupe
groupdel*	suppression d'un groupe

* Cette commande possède un grand nombre d'options, nous renvoyons le lecteur vers le chapitre "Documentation".

III.2- Les droits d'accès aux Fichiers

Attributs de base

```
$ ls -l toto
- rw-r--r--       1 cnam cnam 0 2008-12-09 10:05 toto
```

Le premier caractère indique le type de fichier (ici "-" fichier normal)

Les 9 caractères qui suivent **"rw-r--r--"** sont séparés en 3 sections de 3 caractères :
- la section 1 **rw-** indique les privilèges du propriétaire **owner**
- la section 2 **r--** indique les privilèges du groupe **group**
- la section 3 **r--** indique les privilèges des autres utilisateurs **others**

Les droits d'accès associés à un fichier (normal ou spécial) sont les suivants :

r droit de lecture
Vous donne le droit de lire le fichier (more toto ...).

w droit d'écriture
Vous donne le droit de modifier le fichier ou ses attributs (mais pas de le supprimer).

x droit d'exécution
Vous donne le droit d'exécuter le fichier mais à la condition que le répertoire qui contient ce fichier possède lui-aussi le droit "**x**".

dans l'exemple ci-dessus :
- le propriétaire est autorisé à lire et modifier le fichier
- les utilisateurs du groupe sont autorisés à lire le fichier
- les autres utilisateurs sont autorisés à lire le fichier

Les droits d'accès associés à un répertoire sont les suivants :

r droit de lecture
Vous donne le droit de visualiser le contenu du répertoire, c.a.d la liste des fichiers qu'il contient.

w droit d'écriture
- Vous donne le droit de créer un nouveau fichier dans ce répertoire.
- Vous donne le droit de supprimer un fichier contenu dans le répertoire, mais avec les contraintes imposées par le sticky bit (cf. ci-dessous) si celui-ci est positionné.

x droit d'exécution
Ce droit indique que les fichiers programmes qu'il contient peuvent être exécutés, s'ils possèdent eux-mêmes le droit "**x**".

Attributs complémentaires

SetUid / SetGid :

On distingue le propriétaire du fichier programme et l'utilisateur qui exécute ce programme (s'il en a le privilège). Un programme travaille avec les privilèges (droits d'accès aux fichiers) de l'utilisateur qui l'exécute, sauf dans le cas particulier ou il possède l'attribut **SETUID** qui lui permet de s'exécuter avec les privilèges du propriétaire.

Par exemple le super-utilisateur développe un utilitaire permettant d'afficher les noms des utilisateurs (noms qui se trouvent dans le fichier **/etc/paswwd**, et que le super-utilisateur a interdit en lecture/écriture aux autres utilisateurs). Le super-utilisateur positionne l'indicateur **SETUID** pour qu'un utilisateur normal puisse utiliser cet utilitaire efficacement.

Le droit d'accès **SETGUID** est l'équivalent de **SETUID** pour les utilisateurs du groupe du fichier.

Sticky :

La suppression d'un fichier est liée au droit d'accès en écriture du répertoire correspondant, si le droit accès **sticky** est positionné sur un répertoire, il faudra en plus pour supprimer un fichier de ce répertoire :

- Avoir le droit d'écriture sur le fichier
- ou être propriétaire du fichier

Modification des droits d'accès

Seul le propriétaire ou **root** sont autorisés à changer les droits d'accès d'un fichier. Par contre un propriétaire peut donner un fichier à un autre utilisateur qui en deviendra donc à son tour le propriétaire.

Changement de propriétaire

```
# echo ttt > toto          root créé toto
# ls -l toto
  -rw-r--r-- 1 root root 3 mars 13 19:03 toto
# chown cnam toto          root donne toto à cnam
$ ls -l toto
  -rw-r--r-- 1 cnam root 3 mars 13 19:04 toto
# mv toto /home/cnam       et le dépose dans son répertoire presonnel
# su - cnam                nouvel utilsateur cnam
$ pwd
  /home/cnam
$ ls -l toto
  -rw-r--r-- 1 cnam root 3 mars 13 19:04 toto
$ chgrp cnam toto          cnam change le groupe primaire de toto
$ ls -l toto
  -rw-r--r-- 1 cnam cnam 3 mars 13 19:05 toto
```

Bits de modification des droits

On représente les droits d'accès à un fichier par une chaine de 12 bits :

attributs complémentaires			owner			group			others		
setuid	setgid	sticky	r	w	x	r	w	x	r	w	x

On transforme ensuite ces 12 bits en 4 chiffres octaux `oooo` (de 3 bits chacun)
- le 1er chiffre octal représente les attributs complémentaires
- le 2ème chiffre octal représente les droits `owner`
- le 3ème chiffre octal représente les droits `group`
- le 4ème chiffre octal représente les droits `others`

On utilise alors la commande `chmod oooo`, par exemple :

```
$ chmod 777 toto   Je donne les attributs rwx à tout le monde
$ ls -l toto
  -rwxrwxrwx 1 cnam cnam 3 mars 13 19:05 toto

$ chmod 644 toto          Je donne les attributs rw à owner
                          et r à tout le monde
$ ls -l toto
  -rw-r--r-- 1 cnam cnam 3 mars 13 19:05 toto

$ chmod 7777 toto Je donne tous les droits à tout le monde
                          (attributs rwx  et attributs complémentaires)
$ ls -l toto
  -rwsrwsrwt 1 cnam cnam 3 mars 13 19:05 toto
```

Règle d'affichage des attributs complémentaires:

`SetUid :` On affiche un `s` à la place du `x` de `owner` si ce dernier possède `x`
s'il ne possède pas `x` alors on affiche `S`

`SetGid :` On affiche un `s` à la place du `x` de `group` si ce dernier possède `x`
s'il ne possède pas `x` alors on affiche `S`

`Sticky :` On affiche un `t` à la place du `x` de `others` si ce dernier possède `x`
s'il ne possède pas `x` alors on affiche `T`

Rechercher des droits d'accès :

Par exemple pour trouver les logiciels exécutables ayant le droit d'accès `SETUID` :

```
# find  /  -perm  -4000
```

Le signe `"-"` devant les droits d'accès `4000` signifie qu'il faut avoir au moins ces droits donc ici avoir au moins le droit `SetUid`.

Sans ce signe moins on ne rechercherait que les fichiers ayant exactement les droits d'accès `4000`.

IV- LES PROCESSUS

Un processus[8] est à une entité d'exécution atomique, Une application est composée d'un ou de plusieurs processus qui communiquent entre eux.

Un processus est aussi une entité d'exécution isolée (dans le sens protégée) : Un processus qui se plante (ou un processus malveillant) ne peut pas planter un autre processus en écrasant le code ou les données de ce dernier ; un processus ne peut pas non plus lire les données d'un autre processus.

L'architecture des processus est une arborescence, tout processus possède un processus père. Un processus est identifié par un numéro de processus PID[9].

Pour connaitre le PID d'un processus : `$ pidof <nom de processus>`

Arborescence Système

Le premier processus qui est le père de tous les processus est le processus d'initialisation **`init`** qui a pour identifiant **`"1"`** et pour père le processus virtuel **`"0".`**

- Sous la majorité des distributions Unix-like le processus d'initialisation a pour nom **`init.`** Ubuntu a introduit une variante **`upstart`** et de plus en plus c'est la processus d'initialisation **`systemd`** qui s'impose (cf.chapitre "Les cripts d'initialisation") .

```
$ ps -ef      lister tous les processus du système
UID         PID  PPID  C STIME TTY          TIME CMD
...
root          1     0  0 17:02 ?        00:00:01 init [5]
...
```

- l'option **`-e`** permet de lister tous les processus du système
- l'option **`-f`** permet de lister en full-format

`UID :` utilisateur qui a lancé le processus
`PID :` identifiant du processus
`PPID :` identifiant du processus père
`CMD :` commande associée
`TTY :` Identifiant du terminal (ou fenêtre) attaché à la commande

Sous-arborescence Utilisateur

```
$ ps -f
UID         PID  PPID  C STIME TTY          TIME CMD
cnam       5417  5416  0 17:14 pts/1    00:00:00 bash
cnam       5424  5417  0 17:14 pts/1    00:00:00 ps -f
```

- Par défaut (pas d'option **`-e`**) seuls les processus appartenant à l'utilisateur sont listés
- Le premier processus de l'utilisateur **`cnam`** est son interpréteur de commande **`bash`**.
- Le second processus de l'utilisateur **`cnam`** est la commande **`ps -f`** qu'il vient de lancer. Ce processus est le fils du premier (**`PPID = 5284`**).

[8] Process en anglais

[9] Process ID

Si je recherche le père de l'interpréteur de commande de l'utilisateur `cnam` :

```
ps -ef | grep 9415
root      9415  9412  0 17:20 pts/1    00:00:00 bash
```

- Il s'agit du processus `9412`, si je recherche de qui il s'agit je trouve le processus de gestion de terminal (qui sous OpenSuse est `gnome-terminal`) lequel est directement rattaché au processus d'initialisation (`PPID = 1`) :

```
ps -ef | grep 9412
root      9412     1  0 17:20 ?        00:00:03 gnome-terminal
```

Tuer un processus

Pour tuer le processus courant (en avant plan) il suffit d'utiliser la combinaison `"Ctrl/C"`, qui envoie un signal `kill` au processus en avant plan :

```
$ find / -name "toto"
...
Crtl/C
```

Pour tuer un processus en tâche de fond, il me faut connaître son identifiant `PID` (utiliser **`ps`**) puis utiliser la commande `kill PID` :

```
$ find / -name "toto" &
```

Le `&` en fin de commande permet de reléguer la dernière commande en tâche de fond et de récupérer la main

```
$ ps -ef | grep find
cnam      9415  9412  0 17:20 pts/1    00:00:00 find
cnam     11126  9415  0 17:52 pts/1    00:00:00 grep find

$ kill 9415
```

attention a bien saisir le numéro de processus a tuer

Vous ne pouvez tuer que vos processus fils, seul le super-utilisateur peut tuer n'importe quel processus.

Les processus se protègent contre le signal `kill` de base, dans ce cas il faut utiliser l'options `-9 : kill -9 PID`

Une application est généralement composée d'un certain nombre de processus affiliés, pour stopper une application il suffit de tuer le processus père de la famille (cela entrainera la mort de tous les processus fils).

En particulier tuer le processus d'initialisation revient à stopper le système (tuer absolument tous les processus).

Les Daemons

Un daemon (Disk And Execution MONitor) est un processus qui n'est pas rattaché à un utilisateur mais qui est directement rattaché au processus d'initialisation (pour un daemon `PPID = 1`).

Un daemon est donc un processus en arrière plan qui ne pourra pas être interrompu par la fermeture d'une session utilisateur.

Le nom des daemons se termine généralement par un `"d"` (`httpd` par exemple pour un serveur `HTTP`), mais cela est une simple convention.

Les daemons sont activés par le processus d'initialisation au travers des scripts d'initialisation (cf. chapitre correspondant).

V- LES SHELL-SCRIPTS

Le Shell représente le langage de commande interprété Unix-like et les "Schell-script" représentent les fichiers commandes. Il y a plusieurs interpréteurs de commandes sous les Unix-like :

- **sh** (Bourne Shell), c'est l'interpréteur de référence Unix, toujours présent sous Linux.
- **ksh** (Korn Shell), dernière génération d'interpréteur de commandes Unix ATT avec possibilité de créer des sous-programmes.
- **bash** (Bourne Again Shell), interpréteur de commandes du projet GNU, adopté par Linux, proche de **ksh**.

...

Les commandes présentées dans ce document sont celles de l'interpréteur GNU **bash**.

Un shell par défaut est affecté à votre compte utilisateur (variable système **$SHELL**), mais par sécurité, vous pouvez préciser le shell à utiliser dans votre fichier commande grâce à un pseudo commentaire utilisable uniquement sur la première ligne du fichier :

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

Exemple de shell-script :

1) Créer le fichier **myScript** :

```
#!/bin/bash
echo "Bonjour tout le monde"
```

2) Rendre **myScript** exécutable ;

```
$ chmod 744 myScript :
```

3) Exécuter **myScript**

./myScript préciser **"./"** si **myScript** n'est pas dans le **PATH**

L'appel à l'interpréteur permet de court-circuiter les étapes 2 et 3 :

```
$ bash myScript
```

V.1- Syntaxe des commandes

Une commande est composée d'un mnémonique suivi d'un certain nombre d'arguments, tous ces champs étant séparées par un ou plusieurs "blanc" (ou tabulations ou sauts de ligne)

cp toto titi copier le fichier toto dans le fichier titi.

On utilise généralement une commande par ligne, mais on peut aussi mettre plusieurs commandes sur une même ligne en les séparant par des **";"**.

cp toto titi; cp toto tata copier toto dans titi puis toto dans tata

Pour certaines commandes le premier ou le dernier argument peut être une liste d'arguments

`cp tata titi toto truc` copier tata puis titi puis toto dans truc

Les options dans une commande se présentent sous la forme d'un argument préfixé par un tiret `"-"`, cependant il ne s'agit pas là d'une règle générale.

`cp -r rep1 rep2` recopier récursivement le répertoire rep1 dans rep2

Commentaires

Les commentaires sont précédés par un `"#"` sur la ligne

```
# ceci est un commentaire
```

Code retour

Chaque commande Linux renvoi un compte rendu ou code retour après son exécution, il s'agit d'une valeur numérique et par convention `"0"` signifie exécution correcte (sans fautes).

Votre shell-script doit respecter cette règle. C'est la commande `exit n` qui vous permet de spécifier le code retour.

Si vous quittez le shell-script sans préciser de code retour il sera mis à `0` par défaut par le système.

Reprenons notre premier Script :

```
$ bash myScript
$ echo $?          => 0
                   $?  est la variable système qui contient le code retour
                   du dernier processus exécuté dans votre session de travail.
```

Modifions le script :

```
#!/bin/bash
echo "Bonjour tout le monde"
exit 2
```

```
$ bash myScript
$ echo $?          => 2
```

Pensez à bien gérer vos codes retour dans vos shell-scripts

Les commandes imbriquées

Pour les commandes imbriquées nous utiliserons la syntaxe **`$(commande)`** propre à **`bash`**, notation plus élégante que la syntaxe **`` `commande` ``** propre à **`sh`** et **`ksh`** (quotes inverses)

Toute commande imbriquée dans une autre commande est remplacée par son résultat, c'est-à-dire sa sortie standard.

```
$ date
Fri Jan 30 15:03:36 GMT 1998
$ echo $(date)            => Fri Jan 30 15:03:47 GMT 1998
```

V.2- Les variables

Les noms de variables sont préfixés par un **`"$"`**.

Types de variables

Les variables sont toutes du type chaine de caractères même si certaines commandes les traitent numériquement (dans ce cas cela ne fonctionnera que si les chaines de caractères représentent bien des nombres).

On distinguera :

Variables Environnementale

```
$HOME          répertoire personnel
$ echo $HOME
/home/cnam
$PATH          chemin des exécutables
$SHELL         interpréteur de l'utilisateur
...
```

Par convention les noms des variables environnementales sont en majuscules

Variables système

`$$`	contient le numéro du processus en cours (PID).
`$!`	contient le numéro du dernier processus lancé en arrière plan
`$?`	contient le code retour de la dernière commande exécutée. par convention 0 pour OK sinon un numéro de faute.
`$IFS`	séparateur d'entités de la commande (Internal Field Separator) par défaut 3 caractères : blanc, tabulation et saut de ligne

Variables Utilisateur

```
a=Bonjour
i=3;
a="Bonjour tout le monde"      Les blancs sont des séparateurs
                               il faut délimiter les chaines de caractères
```

```
echo $a
```
=> Bonjour tout le monde

Attention :
- Dans le cas de l'initialisation d'une variable, pas de `$` devant le nom de la variable
- pas de blancs autour de l'opérateur `"="`.

- La commande **`unset`** permet de supprimer une variable utilisateur.

Paramètres positionnels

Ce sont les variables **`$1, $2, ... $9`** qui correspondent aux arguments de la commande.

- Cas particulier la variable $0 contient le nom du fichier commande.

Si l'on souhaite disposer de plus de 9 arguments alors la commande **`shift`** permet de passer aux 9 arguments suivants (de 10 à 18 etc.)

La variable système **`$*`** contient tous les arguments séparés par des espaces.
La variable système **`$#`** contient le nombre d'arguments de la commande

```
if [[ $# -lt 2 ]]; then
      echo '2 arguments au moins sont requis'
      exit
fi
```
cf. test "if" plus loin

Manipulation des variables

- Affectation, utilisation

```
a=be; b=au
echo "il fait $a$b"
b=$a
```
=> il fait beau
affectation d'une variable à une autre

- Lecture au clavier

```
read variable
echo $variable
```
lit jusqu'au retour chariot dans **`variable`**
afficher ce qui vient d'être saisi

- Opérations arithmétiques

```
expr $i + $j
i=$(expr $i + 1)
expr $i \* 3
expr $var / 4
expr $k % 7
```

Attention : il faut toujours mettre des blancs autour des opérateurs.

Variables globales / locales

Une variable est globale est une variable qui est visible par les processus fils.

Les variables environnementales sont globales.

Une variable utilisateur est par défaut locale, sauf si on utilise la directive **`export:`**

```
var1=toto
```

```
export var1
var2=titi
bash                shell fils
echo $var1          => toto
echo $var2          => - néant -
```

V.3- Règles de substitution

Compléments Jokers

Joker	Signification
?	représente n'importe quel caractère (un seul caractère).
*	représente n'importe quelle chaine de caractères, y compris la chaîne vide.
[abcdef]	représente un et un seul de la liste dess caractères précisés
[A-Z]	représente un et un seul des caractères de l'intervalle précisé

`echo *` => affiche tous les fichiers du répertoire courant

Ordre de substitution avant exécution

1) Substitution des jokers
2) Substitution des variables
3) Substitution des commandes imbriquées

Inhibition des substitutions

On peut toujours inhiber la substitution :

Le caractère `"\"` inhibe le caractère suivant

```
echo *              => tous les fichiers du répertoire
echo \*             => *
```

Les double quottes inhibent la chaine de caractères délimitée mais pas les variables incluses.

```
echo $HOME          => /home/cnam
echo "$HOME *"      => /home/cnam *
```

Les simples quottes inhibent tous les caractères de la chaine de caractères incluse sans exceptions.

```
echo '$HOME *'      => $HOME *
```

V.4- Les Débranchements et Itérations

Les débranchements conditionnels

```
if condition
then
   liste_de_commandes
else
   liste_de_commandes
fi
```

Conditions

Nous utiliserons la syntaxe **bash** avec des doubles crochets **[[...]]qui** permet d'utiliser les parenthèses internes, les opérateurs **&&** et **||** et qui n'impose pas de délimiter les opérandes caractères par des double quotes.

`[[ $x == ABC ]]`	égalité de chaines de caractères (== est propre à bash, sh et ksh utilisent =)
`[[ $y != ABC ]]`	non égalité de chaines caractères
`[[ $i -eq $j ]]`	égalité numérique
`[[ $i -ne $j ]]`	non égalité numérique

les autres opérateurs numériques sont : **-lt -le -gt -ge**

Attention la syntaxe de la condition est délicate, Il faut absolument mettre des "blanc" autour des crochets et des opérateurs :

`[[ -e $fichier ]]`	vrai si le fichier $fichier existe
`[[ -f $fichier ]]`	vrai si le fichier $fichier existe et est un fichier normal
`[[ -d $fichier ]]`	vrai si le fichier $fichier existe et est un répertoire

On peut combiner les tests avec les opérateurs logiques et les parenthèses :

`[[ ! (-e $fichier) ]]`	"not" : vrai si le fichier $fichier n'existe pas
`[[ ($i -eq $j) && ($i -gt 4) ]]`	"and"
`[[ ($i -eq $j) \|\| ($i -gt 4) ]]`	"or"

Exemples

```
if [[ $i -eq $j ]]; then
  echo i est egal a j
else
    echo i non egal a j
fi
```

Les Itérations

La boucle for

```
for variable in liste_de_mots
do
      liste_de_commandes
done
```

Exemples

```
for i in 2 4 13; do
    echo $i
done
```

```
for i in *; do
    echo $i
done
```

L'interpréteur remplace "*" par la liste de tous les fichiers du répertoire

afficher tous les fichiers du répertoire, 1 par ligne

Attention, dans l'exemple ci-dessus si les noms de fichiers comportent des blancs, il va falloir ruser pour ne pas éclater ces noms de fichiers sur plusieurs lignes :

```
IFS=$'\n'     # on limite IFS à \n c'est à dire le saut de ligne
for i in *; do
  echo $i
done
IFS=$' \t\n'# On restaure la valeur par défaut d'IFS, cad blanc, tabulation et saut de ligne
```

La boucle while

```
while condition
do
   liste_de_commandes
done
```

Exemples

```
i=0
while [[ $i -lt 10 ]]; do
    echo $i
    i=$(expr $i + 1)
done

# Copypairs : Copie file1 dans file2, file3 dans file4, etc
while [[ $2 != "" ]]; do
    cp $1 $2
    shift; shift
done
if [[ $1 != "" ]]; then
    echo "$0: nombre impair de paramètres" >&2
fi
```

La rupture de boucle break

```
for $i in $(ls); do
  if [[ $i == bla_bla ]]; then
      break
else
      # suite du traitement
      ...
fi
done
```

L'instruction `break` permet de sortir d'une boucle itérative (`for` ou `while`).

La commande continue

```
somme=0
for i in 1 2 3 4 5; do
        if [[ $(expr $i % 2) -eq 0 ]]; then
                continue
        fi
        somme=$(expr $i + $somme)
done
echo $somme
```

L'instruction `continue` permet de passer à l'itération + 1 sans exécuter les instructions qui suivent.

V.5- Combinaisons logiques de commandes

et logique :

`cmd1 && cmd2` commande 2 s'exécutera si commande 1 a réussi
(c'est à dire retour = 0)

est équivalent à :

```
cmd1
if [[ $? -eq 0 ]]; then
      cmd2
fi
```

ou logique :

`cmd1 || cmd2` commande 2 s'exécutera si commande 1 a échoué
(c'est à dire retour != 0)

est équivalent à :

```
cmd1
if [[ $? -ne 0 ]]; then
      cmd2
fi
```

V.6- Compléments redirection des fichiers standards

Redirection des fichiers de sortie standard

Il y a 2 fichiers standard de sortie sous Linux :
- Le fichier standard de sortie normale : n° 1
- Le fichier standard de log des fautes : n° 2

Par défaut c'est l'écran qui correspond aux fichiers standard 1 et 2.

Pour rediriger la sortie normale vers un fichier :

```
find . -name "*.obj"    1> resultat
```

Pour rediriger les fautes d'exécution vers un fichier :

```
find . -name "*.obj"    2> fautes
```

Pour rediriger la sortie normale et les fautes d'exécution vers un fichier :

```
find . -name "*.obj"    > resultat
```

Pour supprimer les sorties d'un processus :

```
find . -name "*.obj"    > /dev/null
```

** `/dev/null` est un fichier spécial particulier qui correspond à une poubelle, dont le contenu n'est pas récupérable.*

Redirection du fichier d'entrée standard

Le fichier standard d'entrée de Linux est le clavier.

La redirection du fichier d'entrée consiste à remplacer le clavier par un fichier contenant les données d'entrée.

```
cat < entrees
```

Une variante consiste à conserver le clavier comme fichier d'entrée et à saisir mot par mot (mots séparés par un retour chariot) jusqu'à l'apparition d'un mot particulier :

```
cat << fin
```

Enchainement des commandes

Il s'agit de rediriger la sortie standard du premier processus vers l'entée standard du second :

Les enchainements de processus sont à la base de la philosophie d'UNIX/Linux, ils permettent la réalisation de programmes élaborés correspondant à l'enchaînement de commandes élémentaires :

Exemples :

envoi d'un message à l'administrateur :
`cat message | mail root` message contient le message à envoyer

extraction de la ligne relative à l'utilisateur dupond dans le fichier des mots de passe :
`grep dupond /etc/passwd | wc -l`

V.7- Personnalisation de l'interpréteur de commandes

Chaque interpréteur de commandes possède son propre fichier de personnalisation utilisateur par utilisateur. Pour l'interpréteur `bash` il s'agit du fichier caché `.bashrc` qui se trouve dans le répertoire de l'utilisateur.

Lors de la création d'un nouvel utilisateur le système lui attribue une copie du fichier de personnalisation modèle : `/etc/skel/.bashrc`

Nous invitons le lecteur à se familiariser avec les shell-scripts en répétant les exemples présentés et en effectuant des recherches complémentaires sur l'Internet.

Exercice 1

- Créez un fichier contenant le texte "Soyez les Bienvenus" à l'aide de la commande `"echo".`
- Ajoutez le texte "Pilotes SOGETI" au fichier précédent à l'aide de la commande `"echo".`

Exercice 2

- Créez un fichier texte tt contenant "tttttttttttttttttttttttttttt" (une suite de "t") à l'aide de la commande `"echo".`
- Afficher le fichier "tt" avec `cat`.
- Transcoder les "t" du fichier "tt" en "u" en enchaînant `"cat"` et `"sed",` rangez le résultat dans un fichier `"uu"`.

Exercice 3

Développez un shell-script "somme" qui calcule la somme des valeurs passées en argument.

Exercice 4

Développez un schell-script "compte_mots" qui compte le nombre de mots saisis au clavier jusqu'à ce que l'opérateur tape le mot fin.

```
#!/bin/bash
# ------------------------------------------------------
# Ce programme compte le nombre de mots saisis au clavier
# jusqu'à ce que l'opérateur tape le mot fin
# ------------------------------------------------------
compteur=0
while true; do
   read mot
   if [[ $mot == fin ]]; then
       break
   fi
   compteur=$(expr $compteur + 1)
done
echo "Vous avez tape $compteur mots"
```

Exécution

```
$compte_mots
mot1
mot2
fin
Vous avez tape 2 mots
```

VI - Les Scripts d'Initialisation

- Ce chapitre est relativement dépendant des distributions.

Les scripts d'initialisation permettent de démarrer des services locaux ou réseau automatiquement lors du lancement du système sous la forme de daemons rattachés directement au processus d'initialisation.

Traditionnellement ce sont les scripts d'initialisation Unix System V qui sont implémentés avec des variantes de fonctionnement selon que les distributions s'inspirent d'Unix ATT ou BSD.

Aujourd'hui les scripts alternatifs Systemd (System daemon) conçus par Lennart Poettering (développeur communauté Linux, développeur Red Hat) sous licence GNU ont été adopté par la majorité des distributions Linux.

VI.1 - Les Mécanisme System V

Les niveaux d'exécution

Les systèmes Unix-like possèdent généralement 7 niveaux d'exécution :

Niveau	Fonction
0	Niveau arrêt de la machine
1	Niveau mono-utilisateur ou `mode de maintenance` : seul le disque système et les périphériques de base sont montés
2	Niveau multiutilisateurs : tous les disques et périphériques sont montés
3	Idem niveau 2 + lancement des couches réseau
4	Idem niveau 3, réservé à la personnalisation
5	Idem niveau 3 + lancement de l'interface graphique
6	Niveau de redémarrage

Les différentes distributions de Linux se lancent généralement par défaut au niveau `5`.

```
$ /sbin/runlevel        pour connaître le niveau d'exécution
# /sbin/init n          pour changer de niveau (reboot partiel)
```

Spécification des Scripts d'Initialisation

C'est le répertoire `/etc/init.d` qui héberge les scripts d'initialisation.

Pour être exploitable un script d'initialisation doit admettre les 2 arguments `"start"` et `"stop",` ces scripts admettent aussi souvent les arguments : `"restart"` (stop puis start) `"reload"` (relance après modification de la configuration) et `"status".`

Par exemple le script d'initialisation de lancement du serveur `ssh` :

```
# /etc/init.d/sshd status
Checking for service sshd       running
```

Lancement automatique des Scripts d'Initialisation

A chaque niveau d'exécution **"n"** un certain nombre de scripts d'initialisation seront lancés automatiquement par le processus d'initialisation **init** au travers des répertoires **/etc/init.d/rc<n>.d.**

Les répertoires **rc0.d** à **rc5.d** contiennent les liens symboliques vers les scripts d'initialisation de **/etc/init.d** correspondant à leur niveau par exemple pour le niveau **rc2.d** :

```
lrwxrwxrwx 1 root root  7 nov.  1  2009 K01a11y -> ../a11y
...
```

Les scripts liens sont de type "Start" **S<num><nom>** ou de type "Kill" **K<num>nom**, par exemple pour le niveau **rc2.d** :

```
K01a11y   K02alsasound   S01dbus          S07brld
K01cron   K02fbset       S01fbset         S09a11y
...
```

- A chaque passage d'un niveau **n** vers un niveau **n+1**, le processus d'initialisation va exécuter tous les scripts d'initialisation **rc<n+1>.d** de type Start dans l'ordre de leurs numéros avec l'argument **"start".**
- A chaque passage d'un niveau **n** vers un niveau **n-1**, le processus d'initialisation va exécuter tous les scripts d'initialisation **rc<n>.d** de type Kill dans l'ordre de leurs numéros avec l'argument **"stop".**

Création d'un Script d'Initialisation

Il vous suffit de créer un shell script dans **/etc/init.d** qui accepte les arguments **"start"** et **"stop".**

Si vous souhaitez que votre script soit lancé automatiquement en tant que daemon par le processus d'initialisation alors il vous faudra créer les liens symboliques **S<num><nom>** et **K<num><nom>** dans les répertoires des niveaux concernés. Attention il vous faut choisir un numéro relativement grand pour laisser les scripts nécessaires à ce niveau s'exécuter en priorité.

Par exemple je créé dans **/etc/init.d** le script d'initialisation **myFilter** qui va me permettre de mettre en place mes propres règles de filtrage dans le pare-feu Netfilter et cela à partir du fichier de référence de mes règles **/etc/init.d/mesRegles*** :

```
#!/bin/sh
/sbin/iptables-restore < /etc/init.d/mesRegles
```

** Le fichier* **mesRegles** *a été créé en configurant Netfilter grâce aux commandes* **iptables** *puis en sauvegardant la configuration dans* **mesRegles** *à l'aide de la commande* **iptables-save.**

Pour que ce script se lance automatiquement au niveau **5** (création du répertoire au passage de **4** à **5** et suppression au passage de **5** à **4**) il me faut rajouter les 2 liens symboliques **S99myFilter** et **K99myFilter** dans le répertoire **/etc/init.d/rc5.d** :

```
# ln -s /etc/init.d/myFilter /etc/init.d/rc5.d/S99myFilter
# ln -s /etc/init.d/myFilter /etc/init.d/rc5.d/K99myFilter
```

VI.2 - Le Nouveaux Mécanisme Systemd

Systemd représente à la fois un processus d'initialisation **systemd** et une nouvelle génération de scripts d'initialisations qui remplacent respectivement le processus d'initialisation **init** et les scripts d'initialisation System V.

L'avantage c'est que Systemd est plus performant cela notamment grâce à la possibilité de lancer les scripts d'initialisations en parallèle tout en contrôlant les dépendances.

L'inconvénient c'est que les scripts Systemd ne sont pas des "shell-script" (il faut assimiler une nouvelle syntaxe).

Toutefois les scripts d'initialisation Système V sont maintenus par Systemd, si vous créez un shell script dans **/etc/init.d** et lui créez un liens avec un niveau d'exécution dans un répertoire **/etc/init.d/rc<n>.d**, votre script sera lancé automatiquement par **systemd**.

Les niveaux d'exécution

Systemd attribut des mnémoniques au niveaux d'exécution et rajoute un niveau "urgence" **emergency.target** correspondant au mode de maintenance avec protection des fichiers (systèmes de fichiers montés en lecture seule) :

Niveau	Mnémoniques
0	**poweroff.target, runlevel0.target**
1	**rescue.target, runlevel1.target**
2	**multi-user.target, runlevel2.target**
3	**multi-user.target, runlevel3.target**
4	**multi-user.target, runlevel4.target**
5	**graphical.target, runlevel5.target**
6	**reboot.target, runlevel6.target**
-	**emergency.target**

Pour savoir à quel niveau on se trouve, on utilisera : **systemctl get-default**

Pour changer dynamiquement le niveau d'exécution, on utilisera :
systemctl isolate <cible>.target

Par exemple, pour passer en mode urgence :
systemctl isolate emergency.target

Pour changer le niveau d'exécution définitivement (mode par défaut), on utilisera :
systemctl set-default <cible>.target

Fichier de Configuration Systemd

Le fichier de configuration général de Systemd est le fichier **/etc/systemd/system.conf**, en voici les principales directives :

- **LogLevel** Niveau de verbosité.
- **LogTarget** Destination des messages de logs.
- **MountAuto** Honore les montages de systèmes de fichiers listés dans /etc/fstab.
- **MountSwap** Honore l'activation des zones de swap listées dans /etc/fstab.

Les Unités

Systemd est fondé sur des unités "unit". Les unités sont typées, les principaux types sont :

- **`service`** : Service système
- **`socket`** : Démarrage d'un service par sollicitation de son "Socket" de communication
- **`automount`** : Monter un système de fichiers
- **`mount`** : Monter un système de fichiers à partir de **`/etc/fstab`**

Le nom de l'unité est suffixé par son type, par exemple **`Network.service`** est l'unité de type service qui s'occupe de la gestion du réseau.

Chaque unité est spécifiée par un fichier de configuration lequel porte le même nom que l'unité elle-même. Les fichiers de configuration se trouvent dans le répertoire **`/etc/systemd/system`**.

Un fichier de configuration d'unité est composé de 2 ou 3 sections. Vous trouverez ci-dessous les sections et leurs principales directives.

** cf. exemple **`myFilter.service`** plus loin dans le document.*

La section **`[Unit]`**: Information Contextuelle :

- **`Description`** Définition du service
- **`After`** Dépendances, le service ne pourra démarrer que si les services ont il dépend ont bien démarré.

La section **`[Service]`** : Spécifications du Service :

- Cette section ne concerne que les unités de type "service" -

- **`Type=oneshot`** Le service exécute une tâche ponctuelle, Il peut être arrêté ensuite.
- **`RemainAfterExit=yes`** Le service doit être considéré comme actif une fois le script de démarrage terminé (pour les services de type oneshot).
- **`ExecStart=<commande>`** Commande (shell) à exécuter pour la mise en oeuvre du service.
- **`ExecStartPre=<commandes>`** Liste de commandes (shell) à exécuter avant la mise en oeuvre du service.
- **`ExecStartPost=<commandes>`** Liste de commandes (shell) à exécuter après la mise en oeuvre du service.
- **`ExecStop=<commandes>`** Liste de commandes (shell) à exécuter Pour stopper le service.

La section **`[Install]`** : Niveaux de Démarrage :

- **`WantedBy=<mnémo-niveau>`** Niveau de démarrage du service
- **`Also=<nom-d'unité>`** Unités additionnelles à installer.

➢ Pour lister les unités disponibles sur votre système :
`systemctl -t help`

La Gestion des Services

Un service utilise un ou plusieurs démons : par exemple, **Network** gère le réseau et **sshd** gère les connexions sécurisées.

C'est la commande **systemctl** qui permet de gérer les services de Systemd :
systemctl <action> <nom_du_service>.service

Action	Commande
start	démarrer un service
stop	arrêter un service
restart	redémarrer un service
reload	recharger les fichiers de configuration sans redémarrer le service
enable	lancer automatiquement un service au démarrage du système
disable	Ne pas lancer automatiquement un service au démarrage du système
mask	invalider l'activation d'un service (empêcher de démarrer ce service par inadvertance)
kill	envoyer un signal d'arrêt **SIGTERM** à tous les processus d'un service
status	connaitre l'état du service en général
is-active	savoir si le service est démarré
is-enable	savoir si le service démarre automatiquement au lancement du système
is-failed	savoir si le service a eu un problème lors de son démarrage

- Pour connaitre L'état du système Linux lui-même :
  systemctl status

  State running dans la réponse nous indique que le système fonctionne normalement

- Pour connaitre tous les services disponibles et leur statut (y compris les services inactifs) :
  systemctl list-unit-files --type=service --all

Créer son propre service

Comme dans le cas System V, nous allons créer un service pour injecter les bonnes règles dans le pare-feu Netfilter au démarrage du système.

Nous allons créer un fichier **myFilter.service**, que l'on va mettre dans le répertoire de configuration **/etc/systemd/system** :

```
[Unit]
Description=Firewall
After=network.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/bin/sh -c "/sbin/iptables-restore < /etc/init.d/mesRegles"

[Install]
WantedBy=multi-user.target
```

- Pour lancer **myFilter** :
  systemctl start myFilter.service

- Et pour qu'il démarre automatiquement au lancement du système :
  `systemctl enable myFilter.service`

VII- LA DOCUMENTATION UNIX-LIKE

Le manuel en line "man" de System V

Les systèmes Unix sont traditionnellement livrés avec un manuel en ligne comportant 8 sections :

Section I : Utilisateur
Commandes système utilisateur.

Section II : Appels Système
Primitives système.

Section III : Bibliothèques
Primitives bibliothèques standard.

Section IV : Fichiers Système
Cette section décrit le format des fichiers systèmes (ceux qui contiennent les informations de configuration et d'administration).

Section V : Paquetages divers
Cette section décrit des produits complémentaires.

Section VI : Jeux
!

Section VII : Fichiers spéciaux
Cette section décrit le format des fichiers spéciaux (pilotes de périphériques).

Section VIII : Administration
Commandes système administrateur.

** Les commandes ou plutôt les options des commandes peuvent légèrement différer selon que la distribution Linux s'inspire d'Unix ATT ou BSD.*

Utilisation du manuel en ligne :

Utilisez la commande "man" suivie du nom de la commande souhaitée.

```
$ man pwd
$ man ls
$ man man
```

Pour quitter le manuel utilisez `q` (quit).

Chaque commande figurant au manuel comporte une rubrique "Voir aussi" qui vous dirige vers un certain nombre de commandes "complémentaires" ou en "relation".

La commande `man` vous permet de rechercher les entités par mot-clef en utilisant l'option `-k.`

```
$ man -k passwd
```

La commande **man** vous permet également de rechercher des entités qui appartiennent à une section déterminée en utilisant l'option **-S** (ceci est utile en cas d'ambiguïté, par exemple la commande **mkdir** qui appartient à la fois à la section **1** et à la section **2**).

```
$ man -S1 mkdir
$ man -S2 mkdir
```

L'Option --help

L'interpréteur de commandes **bash** auto-documente ses commandes, pour cela il faut utiliser l'option **--help**, par exemple :

```
$ cp --help
```

La Documentation en line "coreutils" de GNU

GNU propose sa propre documentation **"coreutils"** pour ses commandes et utilitaires, elle est native sur un grand nombre de distribution Linux :

Mode d'emploi de **coreutils** :
```
$ info
```

Mode d'emploi d'une commande GNU :
```
$ info <nom-de-commande>
```

Manuel GNU complet :
```
$ info coreutils
```

Annexe A : NORMES ET STANDARDS

Le standard ATT SVID

Le standard ATT[10] "System V Interface Definition" est la référence historique.

Il se décline en 2 sections :
1) Spécification des Interfaces utilisateur (Commandes et Utilitaires Système)
2) Spécification des Interfaces programmeur (Bibliothèques et Appels systèmes).

La norme POSIX

Acronyme pour Portable Operating System Interface et X pour Unix. Norme élaborée par IEEE puis reprise par ISO.

Elle comporte les mêmes 2 parties que le "System V Interface Definition" et comporte en plus un test de conformité : PCTS (Posix Conformance Test Suite)

Windows NT est compatible POSIX.1, ce qui est suffisant pour faire tourner des programmes POSIX. Sur les autres plateformes Windows, des compléments tels que "Services for UNIX" ou "Cygwin" permettent d'obtenir une compatibilité POSIX.

Le standard "Single Unix Specification"

"Single Unix Specification" est le standard de l'OpenGroup (Nouveau propriétaire d'Unix).

Il comporte les mêmes 2 parties que le "System V Interface Definition".

Aujourd'hui les travaux d'ISO et de l'Open Group Posix convergent, ce qui n'a pas toujours été le cas.

Le standard de marché Linux

Très peu de distributions Linux sont certifiées ISO-Posix même si elles sont très proches de cette norme, car la certification coute cher en temps et en argent ce qui est contraire aux principes du logiciel libre.

C'est donc plutôt le standard "Single Unix Specification" qui sert implicitement de référence aux systèmes Linux aujourd'hui.

D'autant plus que la documentation du standard "Single Unix Specification" de "l'Open-group" est libre d'accès (http://www.unix.org/online.html) ce qui n'est pas le cas de la norme POSIX.

[10] American Telephone & Telegraph, société qui a créé le système Unix