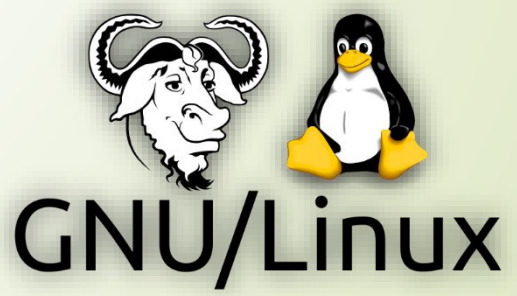
Ministère de L'enseignement supérieur et de La recherche scientifique
Institut Supérieur des études technologiques de Nabeul

Système d'exploitation 2 Linux

Sabri Challouf



Plan

1. Introduction
2. Chapitre 1 : Présentation générale de LINUX
3. Chapitre 2 : Système de Gestion de fichiers sous LINUX
4. Chapitre 3 : Programmation Shell
5. Chapitre 4 : Gestion des processus
    1. Gestion des processus en langage C
    2. Communication Interprocessus
    3. Synchronisation entre processus

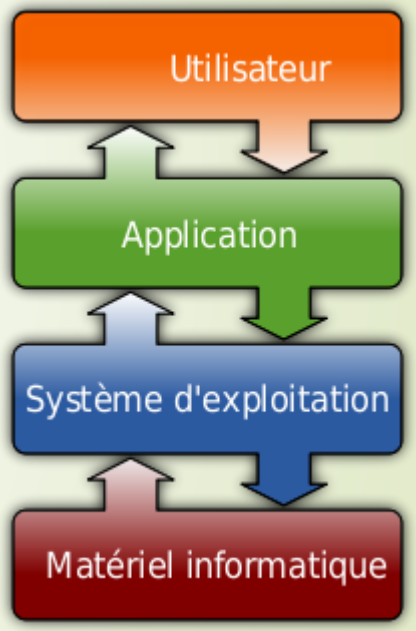
Chapitre 1

Introduction

Introduction

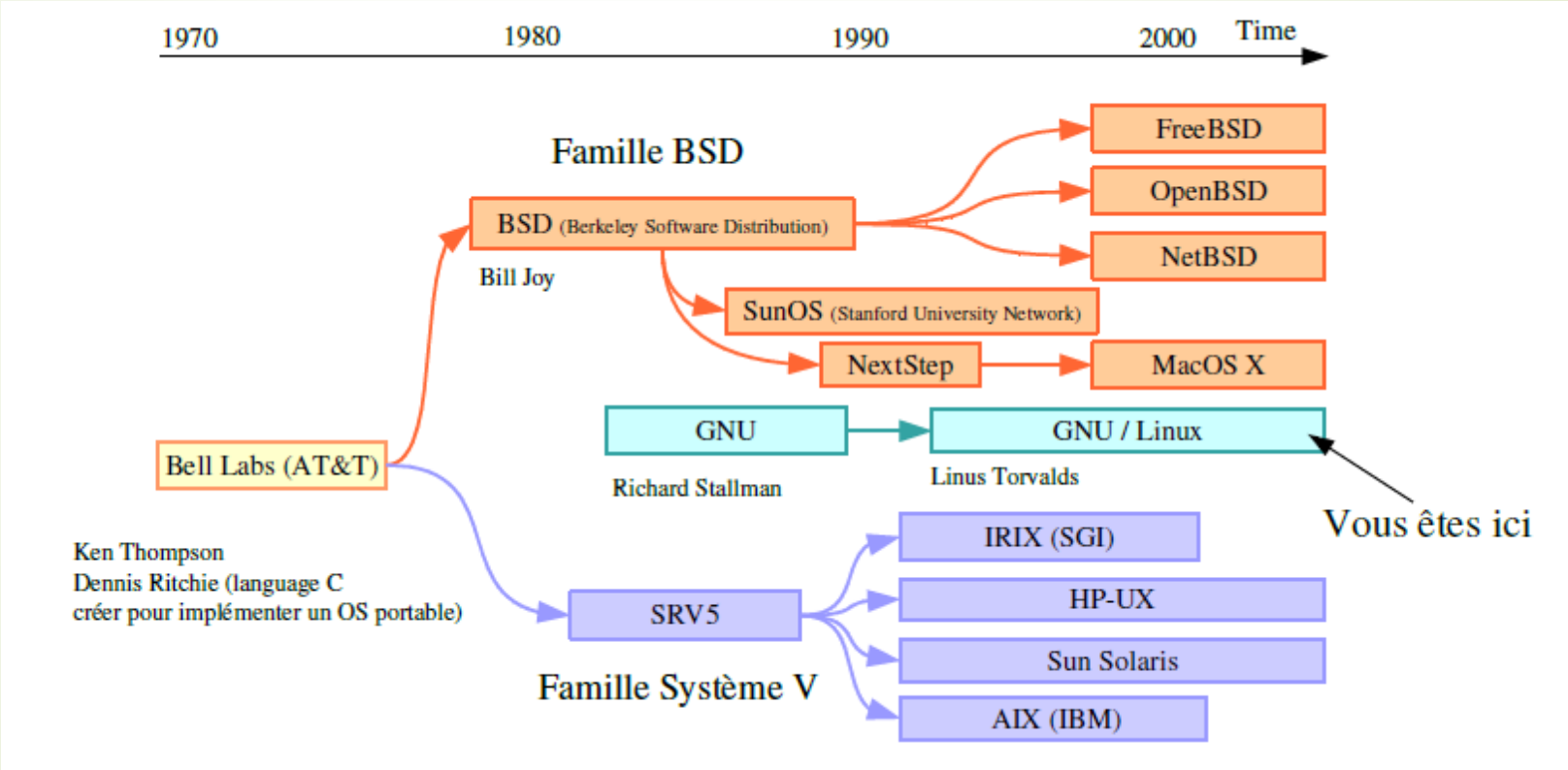
- Système d'exploitation ?
  - Un **système** d'**exploitation**, ou logiciel **système**, ou Operating System (OS), est un logiciel qui pilote les dispositifs matériels et reçoit des instructions de l'utilisateur ou d'autres logiciels (ou applications).

- Plusieurs catégories de SE :
  - Libre (Linux)/ Propriétaire (Windows)
  - Mono-tâche (MS Dos)/ Multi-tâches (Windows 10)
  - …



Chapitre 1:
Présentation générale de LINUX

Histoire - Arbre généalogique d'Unix



Histoire de GNU/Linux
Richard Matthew Stallman

- Le premier personnage important dans cette histoire n'est autre que Richard Matthew Stallman ou appelé par ses initiales RMS.
- C'est en 1984 que RMS a décidé de créer un système d'exploitation (SE ou OS) entièrement libre nommé système GNU.
- Le but du projet GNU était de créer un OS complet et entièrement libre. C'est-à-dire gratuit, que l'on peut modifier et redistribuer. C'est dans cet état d'esprit que RMS créa la licence GPL (General Public Licence).

Histoire de GNU/Linux

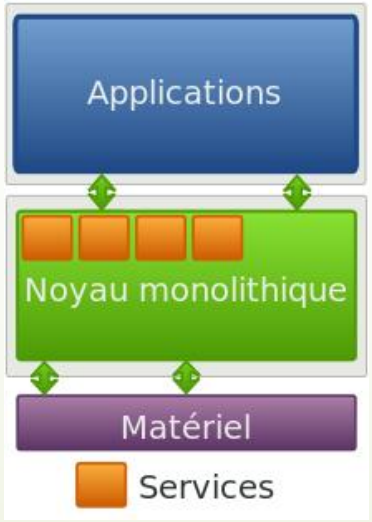
- Il commença par programmer la plupart des commandes Unix nécessaires à l'utilisation du système. Puis il chercha à pourvoir GNU d'un compilateur C.

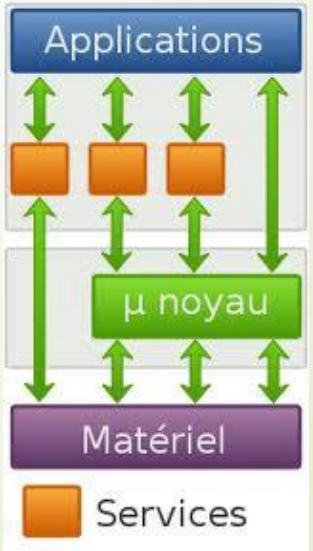
- Malheureusement, les compilateurs les plus proches de ce qu'il cherchait n'étaient pas libres et ceux qui étaient libres nécessitaient trop de modifications pour que le projet soit viable (durable).

- Il commença alors l'écriture depuis 0 (from scratch) qu'il nomma GCC acronyme de GNU C Compiler. GCC était assez flexible pour permettre de lui ajouter de nouveaux langages comme le C++, objective C, Java ,etc, et l'acronyme devint ensuite GNU Compiler Collection.

Histoire de GNU/Linux

- Cependant il manquait au projet GNU la pièce maîtresse d'un OS : le noyau.
- Stallman ne voulait pas d'un noyau monolithique et s'orienta alors vers un micro noyau. La conception d'un tel noyau est plus compliqué et surtout le débogage est d'une incroyable complexité.





Histoire de GNU/Linux
Linus Trovalds

- Le second personnage clé est Linus Torvalds, créateur du noyau Linux.
- En 1991 Linus était étudiant à l'université d'Helsinki en Finlande. Il utilisait **Minix**, un système non libre, basé sur le modèle d'**Unix**, créé par le professeur Andrew Tanenbaum a des fins didactiques (educatifs).
- Il manquait dans Minix **un émulateur** de terminal ce qui aurait permis à Linus de se connecter sur les machines de son université depuis chez lui.

Histoire de GNU/Linux

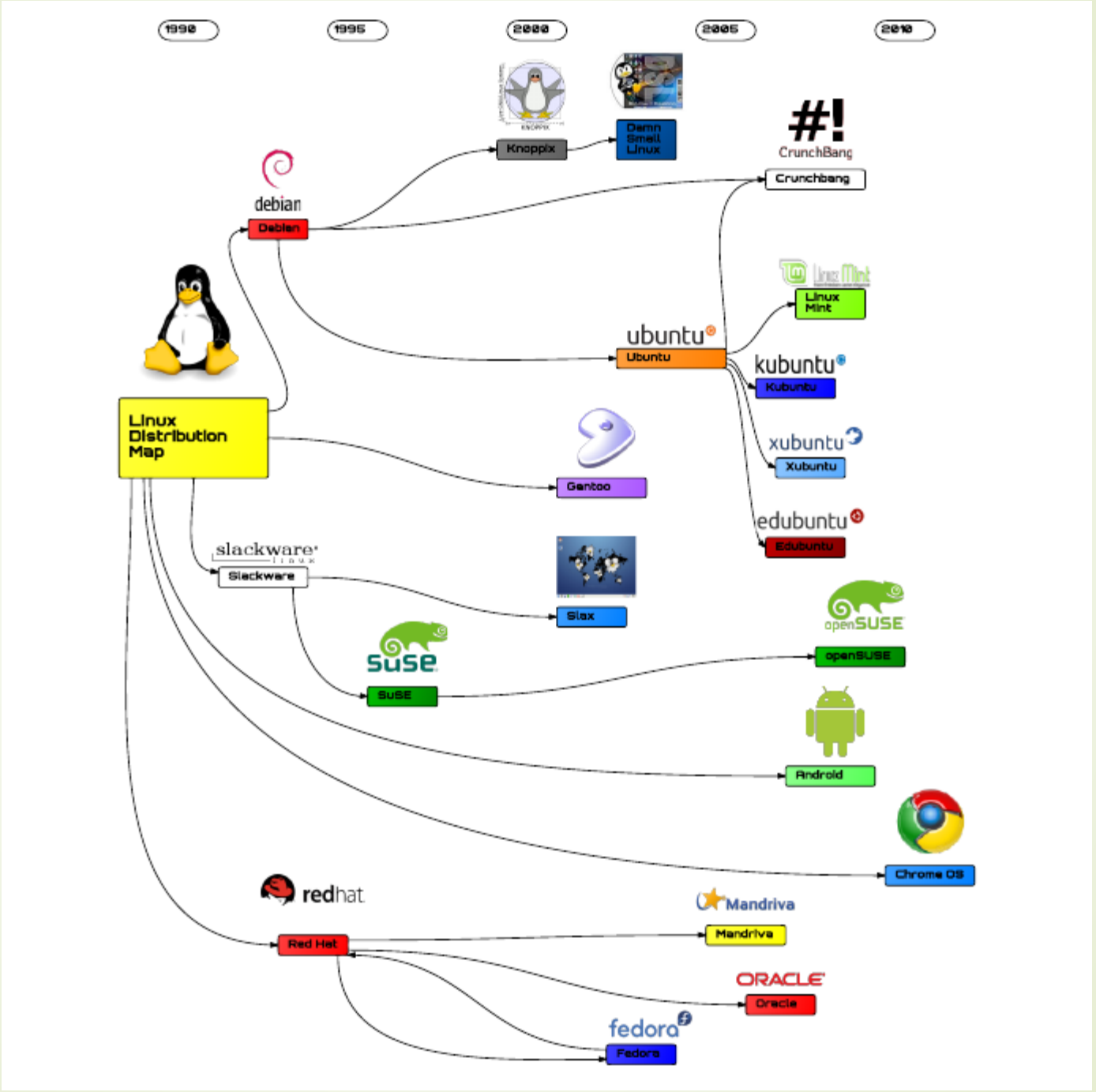
- Linus entrepris alors de le coder mais il le fit indépendant de Minix. Il continua d'ajouter des fonctionnalités à son émulateur de terminal tant et si bien qu'il en avait fait un embryon de noyau. Il porta dessus bash et GCC, tous deux issus du projet GNU.

- Il posta le fruit de son travail sur des listes de discussions sur internet et rapidement beaucoup de personnes commencèrent à le modifier et à ajouter leurs propres fonctionnalités.

- Le choix de Linus de rendre son noyau libre, et le ralliement par internet d'une armée de développeurs permit dès 1994 d'avoir la version 1.0 de Linux sur lequel fonctionnait la majorité des programmes GNU.

Evolution de GNU/Linux

- Installer un système Linux revenait à installer le noyau Linux, les outils GNU ainsi que d'autres logiciels tiers. => C'était réellement une tâche compliquée.
- Des personnes eurent l'idée géniale de faire ce travail et de redistribuer ensuite ce système complet en tant que Distribution Linux.
- C'est dans les années 1993/1994 que deux distributions, qui existent encore aujourd'hui, sont nées : **Slackware Linux** et **Debian GNU/Linux**.


1990
1995
2000
2005
2010
Knoppix
Damn Small Linux
#!
CrunchBang
Crunchbang
debian
Debian
Linux Mint
Linux Mint
ubuntu
Ubuntu
kubuntu
Kubuntu
xubuntu
Xubuntu
edubuntu
Edubuntu
Linux Distribution Map
Gentoo
slackware
Slackware
Slax
openSUSE
openSUSE
SuSE
SuSE
Android
Chrome OS
redhat
Red Hat
Mandriva
Mandriva
ORACLE
Oracle
fedora
Fedora

- Si aujourd'hui **Slackware** est peu utilisée c'est parce qu'elle est restée très proche de ce qu'elle a été au début : un ensemble cohérent comprenant le noyau, les outils GNU et des logiciels tiers.

- Alors que des distributions comme **Debian** (puis **Red Hat**, **Suse**, Mandrake devenue **Mandriva**) apportent une vraie valeur ajoutée par la possibilité d'utiliser des paquets : c'est-à-dire que tous les logiciels disponibles pour la distribution ne sont pas compris sur le support d'installation mais sont disponibles soit sur des supports annexes soit sur des serveurs internet.

- Ces paquets sont installables via des outils dédiés et permettent la résolution des dépendances.

- Par exemple si vous voulez installer le logiciel **foo** et qu'il nécessite la bibliothèque **libbar**, le **gestionnaire de paquet** va vérifier que vous avez libbar, dans le cas contraire il va le télécharger et l'installer puis faire de même pour l'application foo.

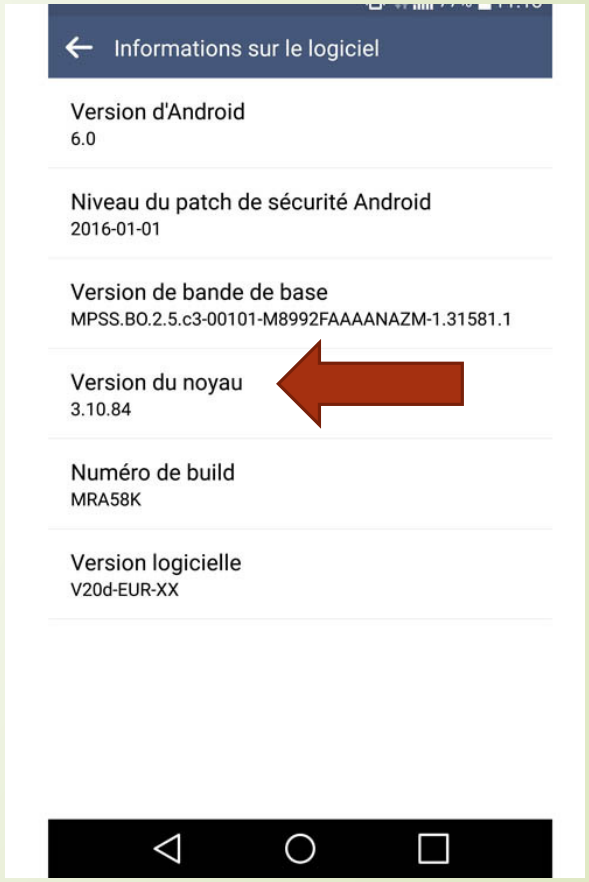
- De plus ils ont souvent des outils graphiques permettant une configuration simplifiée de diverses parties du système.

- Par exemple la distribution qui est actuellement la plus populaire c'est **Ubuntu** (et ses dérivés Kubuntu / Xubuntu), si vous utilisez du matériel dont les drivers ne sont pas libres ou qu'ils nécessitent un firmware non libre, le petit outil graphique va se charger de tout télécharger et tout installer tout seul.

- Ce genre de choses doivent être faites manuellement lorsqu'on utilise une Slackware.

- Tout cela concerne les distributions de bureaux (ou de serveurs). Mais l'utilisation de GNU/Linux ne s'arrête pas là, on le retrouve de plus en plus dans des systèmes embarqués.

- Par exemple de plus en plus de téléphones portables utilisent le noyau Linux comme tous les téléphones utilisant Android de Google. A noter que Chrome OS est aussi bâti sur un noyau Linux. La plupart des routeurs, Box ainsi que les systèmes fonctionnent grâce aussi à Linux.



- Depuis sa constitution Linux suit les normes et les standards comme POSIX ou ANSI ainsi que BSD.

- C'est un énorme avantage puisque lorsqu'on écrit un programme fonctionnant sur GNU/Linux, on peut normalement le porter facilement, voire sans modifications, sur d'autres systèmes respectant aussi ces normes comme les BSD (NetBSD, OpenBSD, FreeBSD), les solaris (Solaris, OpenSolaris) et dans une moindre mesure MacOS.

- En revanche il sera souvent beaucoup plus difficile de le porter sur un OS ne respectant rien à l'instar de **windows** de Microsoft dont l'implémentation des normes et standards est très limitée, partielle et parfois ne respecte justement pas les standards.

- Un autre point important que nous avons vu, le noyau Linux ainsi que la grande majorité des logiciels fournis avec (outils GNU inclus) sont libres.

  Vous pouvez accéder aux sources, les modifier, vous en inspirer, les redistribuer et tout ça gratuitement. C'est une sorte de gigantesque bibliothèque de codes sources entièrement libre.

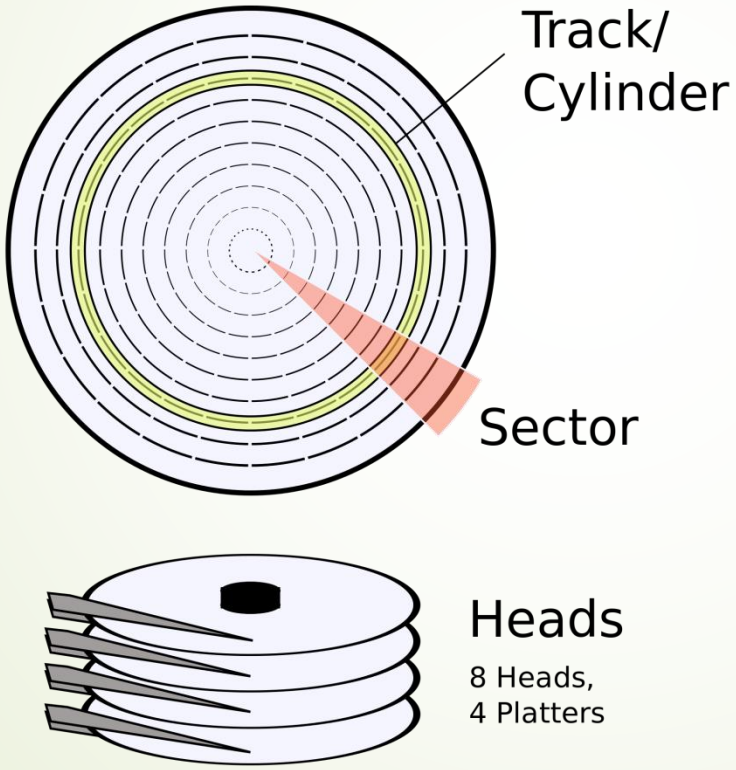
Chapitre 2

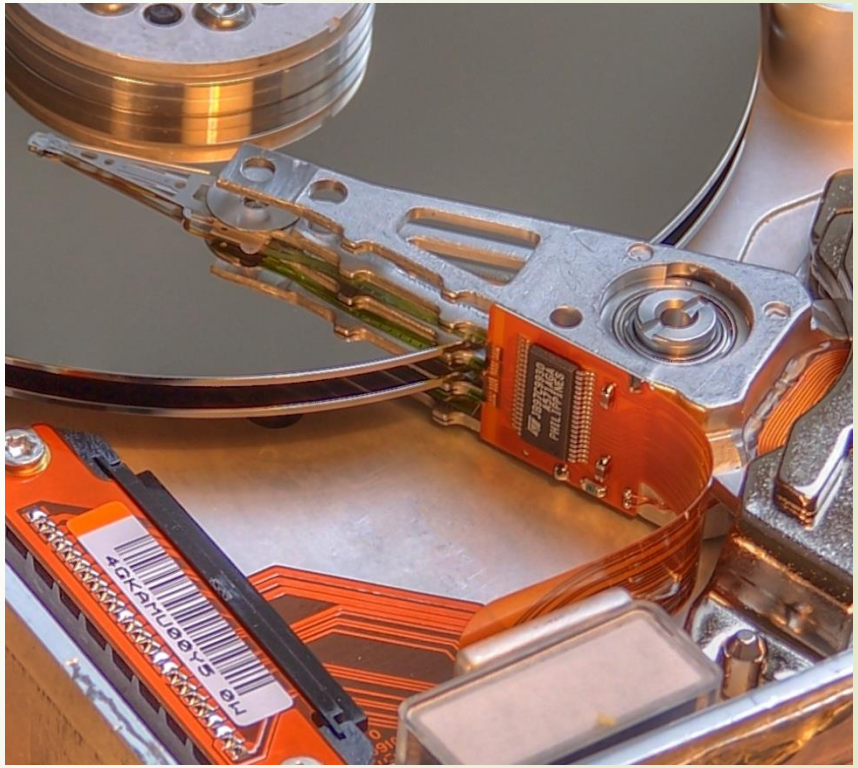
Le Système de Gestion de fichiers

Disques durs et partitionnement

- Un disque dur est composé de plateaux reliés à un moteur central, avec des têtes de lecture de part et d'autre de chacun des plateaux.
- Sur chaque plateau se trouvent des pistes cylindriques découpées en secteurs.
- L'adressage d'un secteur est une référence au cylindre, à la tête de lecture utilisée, à la piste, et enfin au secteur.

Exemple





- À l'installation, un disque dur n'est ni partitionné, ni formaté.
- **Partitionner,** signifie définir sur le disque un ou plusieurs espaces, ou **partitions**.
- **Formater,** signifie préparer une partition à recevoir des informations en utilisant un système de fichiers défini.

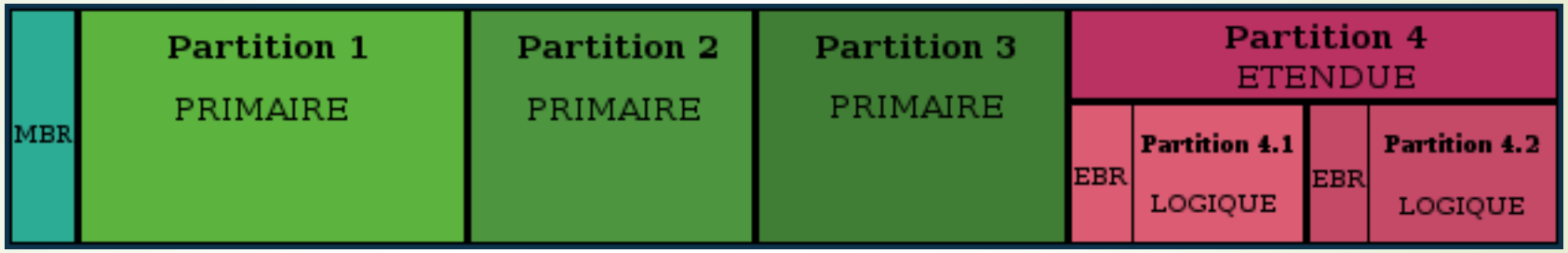
Les partitions

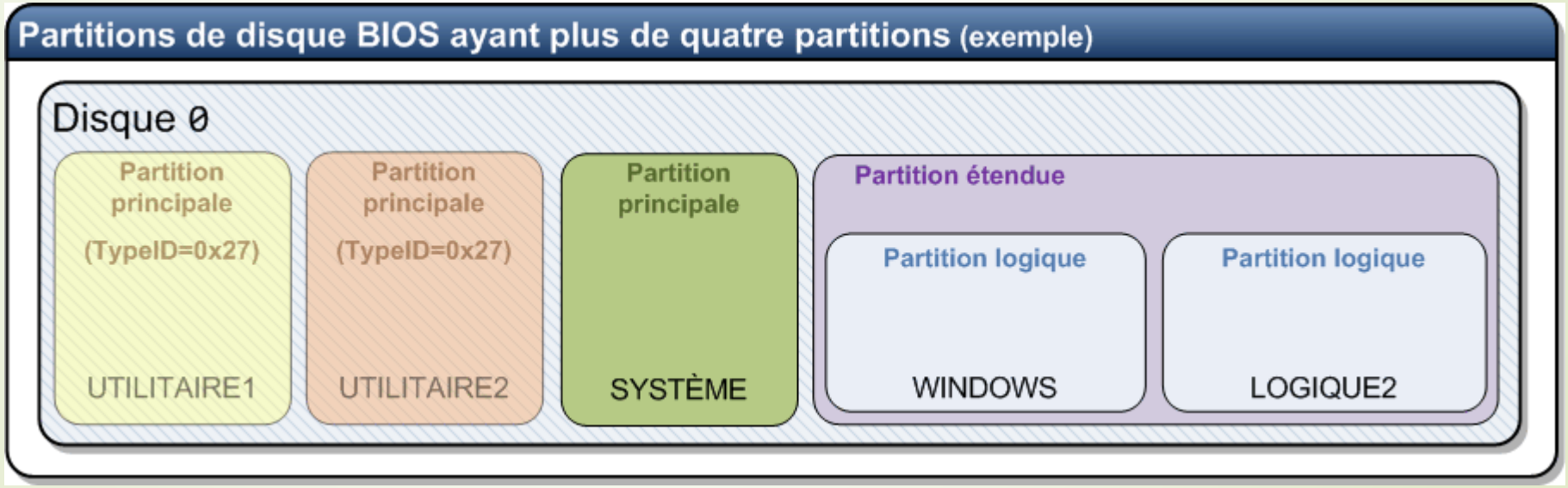
- Une partition est définie par son type, son emplacement de début de partition et enfin soit sa taille, soit son emplacement de fin de partition. Un partitionnement est réversible (non physique).

- Une seule partition est activée à la fois au niveau du BIOS : cette activation indique où le BIOS doit aller chercher le noyau du système d'exploitation pour le démarrage.

- Il existe trois sortes de partitions :
  - **Les partitions principales :** leur nombre est limité à quatre et elles supportent tous types de systèmes de fichiers ;
  - **La partition étendue :** elle ne peut contenir que des partitions logiques et ne peut pas recevoir de systèmes de fichiers. Elle ne peut exister que s'il existe une partition principale ;
  - **Les partitions logiques :** elles sont contenues dans une partition étendue. Elles ne sont pas limitées en nombre et acceptent tous types de systèmes de fichiers.

Exemples


MBR
Partition 1
PRIMAIRE
Partition 2
PRIMAIRE
Partition 3
PRIMAIRE
Partition 4
ETENDUE
EBR
Partition 4.1
LOGIQUE
EBR
Partition 4.2
LOGIQUE


Partitions de disque BIOS ayant plus de quatre partitions (exemple)
Disque 0
Partition principale
(TypeID=0x27)
UTILITAIRE1
Partition principale
(TypeID=0x27)
UTILITAIRE2
Partition principale
SYSTÈME
Partition étendue
Partition logique
WINDOWS
Partition logique
LOGIQUE2

Organisation des partitions sous Linux

- Les descripteurs de disques durs dans le répertoire **/dev** commencent par **hd** pour les périphériques de type **IDE** ou par **sd** pour les périphériques de type **SCSI**.
- Une lettre additionnelle est ajoutée au descripteur pour désigner le périphérique.
- Il y a généralement deux contrôleurs IDE en standard sur un PC, chaque contrôleur supportant deux périphériques (disques, lecteur de cédérom/DVD, lecteur ZIP...).

Bus IDE

- Désignation des périphériques IDE

	PRIMAIRE	SECONDAIRE
Maître	a	c
Esclave	b	d

- Pour le périphérique maître sur le contrôleur primaire : **hda**
- Pour le périphérique esclave sur le contrôleur secondaire : **hdd**.

À noter :

- Les périphériques SCSI sont désignés en fonction de leur position dans la chaîne SCSI (sda, sdb, sdc, etc.).

Fdisk

- On utilise la commande **fdisk** pour configurer une nouvelle partition. Par exemple, pour le premier disque IDE :

 # fdisk /dev/hda

- Voici une liste des différentes commandes internes de **fdisk** :
  - a : (dés)active un indicateur « *bootable* » ;
  - b : édite le libellé de disque;
  - c : (dés)active l'indicateur de compatibilité DOS ;
  - d : supprime une partition ;
  - l : répertorie les types de partition connus ;
  - m : affiche la liste des commandes ;

Otions de Fdisk

- n : ajoute une nouvelle partition ;
- o : crée une nouvelle table de partition DOS vide ;
- p : affiche la table de partition ;
- q : quitte le programme sans enregistrer les modifications ;
- s : crée un nouveau libellé de disque Sun vide ;
- t : change l'ID système d'une partition ;
- u : change l'unité d'affichage/saisie ;
- v : vérifie la table de partition ;
- w : écrit la table sur le disque et quitte le programme ;
- x : fonctions supplémentaires (experts seulement).

- L'image suivante est un exemple de table de partitionnement obtenu avec l'option « l » de **fdisk** :

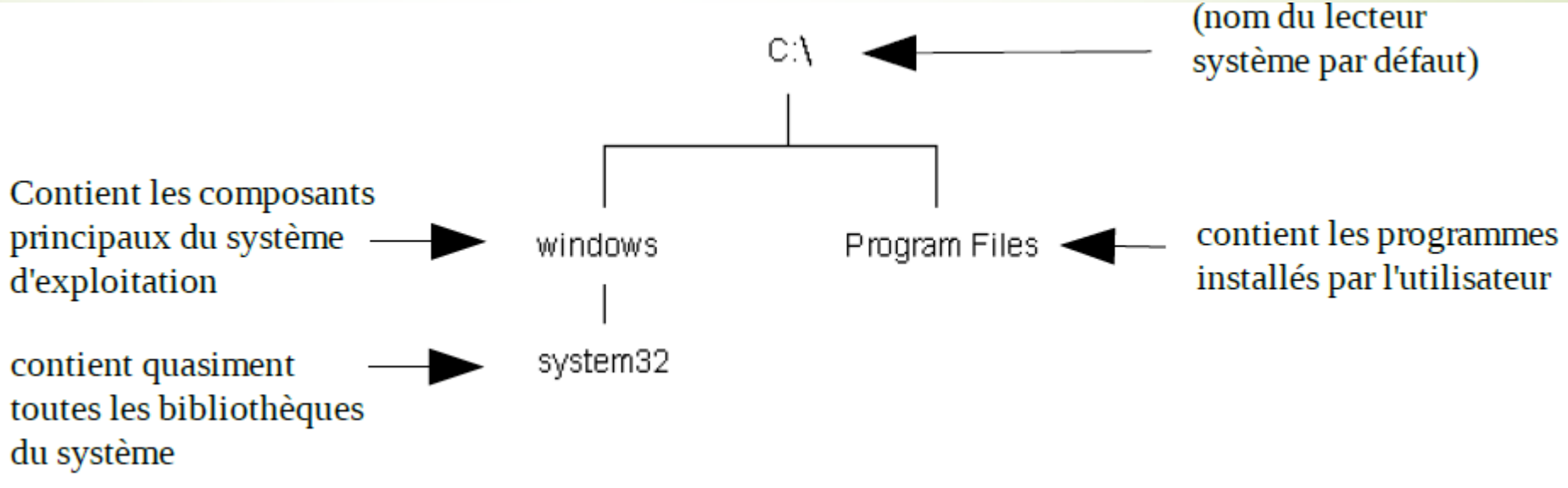
```
Disque /dev/hda : 255 têtes, 63 secteurs, 3647 cylindres
Unités = cylindres sur 16065 * 512 octets

Périphérique Amorce    Début        Fin    Blocs   Id  Système
/dev/hda1              1          255   2048256    b  Win95 FAT32
/dev/hda2   *        256          386   1052257+  83  Linux
/dev/hda3            387         1597   9727357+  83  Linux
/dev/hda4           1598         3647  16466625    f  Win95 Etdue (LBA)
/dev/hda5           1598         2744   9213246   83  Linux
/dev/hda6           2745         3509   6144831   83  Linux
/dev/hda7           3510         3647   1108453+  82  Echange Linux
```

SWAP

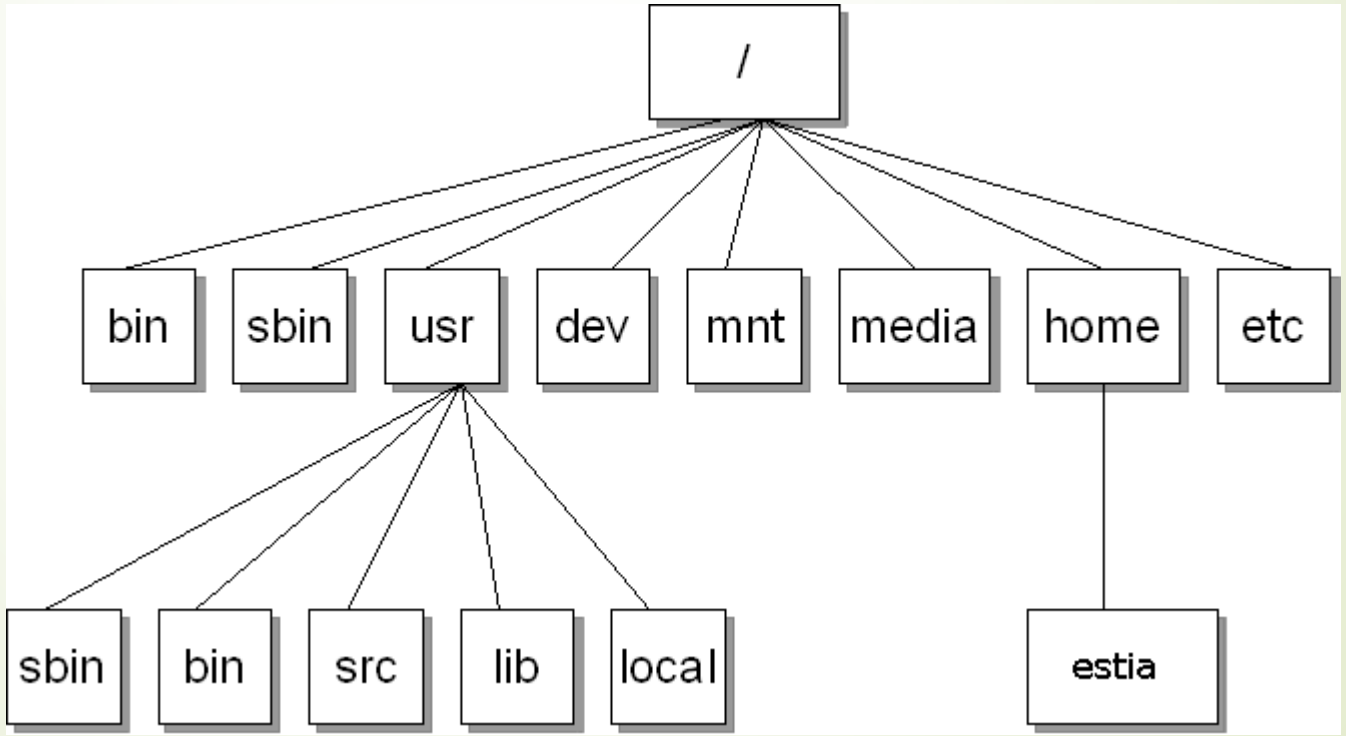
- Le système d'exploitation utilise une zone d'échange (**swap**) sur le disque comme une extension de la mémoire physique.
- Selon les besoins, il y aura donc un échange entre la mémoire physique et la zone swap.
- Linux utilise deux types de partitions : Linux (**Linux native**) et Echange Linux (**swap**) comme on peut le constater sur la figure précédente.
- La première partition est une partition qui peut contenir un système Windows et la quatrième une partition de type étendu qui permet de créer des partitions logiques .

Arborescence des fichiers sous Linux

- L'arborescence est l'ensemble des répertoires du système.
- Par exemple sous **Windows** c'est très simple, nous avons quelque chose qui ressemble à cela :



- Sous GNU/Linux c'est un peu plus compliqué car les fichiers du système ont chacun une place définie.



- Les répertoires de base de l'arborescence standard de fichiers sont les suivants (N. B. la racine du système est représentée par « / ») :
  - **/boot :** contient principalement le fichier binaire du noyau ainsi que les ressources nécessaires à son lancement au démarrage ;
  - **/dev :** contient les fichiers des périphériques (devices) de la machine ainsi que des fichiers spéciaux ;

- **/etc :** répertoire très important où sont stockés tous les fichiers de configuration du système en général et des différents démons (services) en particulier. Il s'agit du répertoire à sauvegarder pour pouvoir restaurer la configuration d'une machine ;

- **/home :** répertoire où sont stockés par défaut les répertoires home des utilisateurs du système ;

- **/proc :** contient les informations nécessaires au noyau. C'est une arborescence virtuelle généralement en lecture seule sauf proc/sys ;

- **/root :** répertoire **home** du super-utilisateur (*root) ;*
- **/tmp :** permet aux applications et aux utilisateurs d'avoir un espace d'échange où ils peuvent stocker leurs fichiers temporaires. Il est effacé à chaque redémarrage de la machine (« *reboot* ») ;
- **/usr :** contient les fichiers nécessaires aux applications, la documentation, les manuels, les fichiers sources ainsi que des librairies généralement statiques et générées à l'installation des logiciels standards de la distribution ;

- **/usr/local :** arborescence qui sert à installer les logiciels supplémentaires ;

- **/var :** contient les fichiers journaux des différents démons (donc variable) ainsi que les spools de mail, d'impression, de cron, etc.
- **/bin** et **/sbin** : contiennent l'ensemble des binaires indispensables au démarrage de la machine et les commandes essentielles d'administration ;

- **/lib** et **/usr/lib** : contiennent les librairies nécessaires aux commandes précédentes.

Formatage & types des SGF

- Les principaux types de système de fichiers supportés par Linux sont présentés dans le tableau suivant:

Système de fichiers	Commande de création
Ext2	mke2fs ou mkfs.ext2
Ext3	mke2fs -j ou mkfs.ext3
Reiserfs	Mkreiserfs
Xfs	mkfs.xfs
Vfat	Mkfs.vfat

- Le système de fichiers ext3 est une simple extension du format standard ext2 de Linux : il intègre un **journal** qui enregistre toutes les opérations effectuées sur le disque. Ceci permet une récupération plus rapide et sûre du système en cas d'arrêt brutal de la machine.

- L'instruction générale de création d'un système de fichiers est : **mkfs -t** type-de-fichier *partition*

  mkfs –t ext3 /dev/hda

- Si le système de fichiers est endommagé ou corrompu, l'utilitaire **fsck** est utilisé pour vérifier et corriger le système.
- L'instruction générale de vérification du système de fichiers est :

  fsck -t type-de-fichier *partition*

- **Remarque**:

  Une vérification de toutes les partitions est faite au démarrage du système par la commande **fsck**.

Contrôle de système de fichiers

- La commande **df** permet de connaître le taux d'utilisation de toutes les partitions montées du système. L'option -h (human readable) facilite la lecture en utilisant des unités de taille plus commodes (Mo, Go, To …).

- La commande **du** (disk usage) est très pratique pour connaître l'espace occupé par une arborescence. L'option -s permet d'afficher le total pour chaque élément et l'option -k de l'afficher en kilo-octets.

Montage / Démontage d'un SGF

1. Montage et démontage manuel
2. Montage et démontage automatique

Montage manuel

- Pour pouvoir utiliser un système de fichiers, celui-ci doit être monté sur un point de montage de l'arborescence Linux : son contenu est alors accessible comme un simple répertoire.

- Le système d'exploitation réalise alors diverses tâches de vérification afin de s'assurer que tout fonctionne correctement.

- La commande **mount** accepte deux arguments :
  - le premier est le fichier spécial correspondant à la partition contenant le système de fichiers ;
  - le second est le répertoire sous lequel il sera monté (point de montage).

- Il peut être nécessaire de spécifier le type de fichiers avec l'option -t au cas où Linux ne parviendrait pas à le déterminer automatiquement.

- Voici un exemple de montage et de démontage d'une clé USB de type «**flashdisk**» décrite par le fichier device **sda** :

  $> **mount** /dev/sda1 /mnt/flashdisk

  $> **umount** /mnt/flashdisk

Montage automatique

- Le fichier **/etc/fstab** est utilisé pour le montage automatique des systèmes de fichiers au moment du démarrage du système.

- Chaque ligne du fichier **fstab** décrit la manière de montage d'un système de fichiers, et ceci à travers six champs séparés par des espaces.

/etc/fstab

- Champs 1 : Nom du périphérique
- Champs 2 : Point de montage
- Champs 3 : Type de système de fichier
- Champs 4 : Options de montage
- Champs 7 : Pour dump , réellement inutilisable (0)
- Champs 6 : Vérification de l'intégrité du système

  0 -> Pas de vérification

  1 -> Vérifier avant de monter

  2 -> Vérifier après le montage

Les droits d'accés

- Linux permet de spécifier les **droits** dont disposent les utilisateurs sur un fichier ou un répertoire par la commande **chmod**.
- On distingue trois catégories d'utilisateurs :
  - u : le propriétaire (*user) ;*
  - g : le groupe ;
  - o : les autres (*others)*.
- Ainsi que trois types de droits :
  - r : lecture (*read) ;*
  - w : écriture (*write) ;*
  - x : exécution ;
  - - : aucun droit.

Exemples

- Pour donner le droit de lire le fichier liste.txt au propriétaire du fichier :

  $> chmod u+r liste.txt

- Pour autoriser une personne du groupe propriétaire du fichier à modifier le fichier :

  $> chmod g+w liste.txt

- Pour autoriser les autres utilisateurs à exécuter le fichier commande.sh :

  $> chmod o+x commande.sh

- Les droits d'accès peuvent aussi s'exprimer en notation octale.
- Les correspondances sont indiquées dans le tableau suivant:

Droit	Notation Octale
- - -	0
- - x	1
- - w	2
- w x	3
r - -	4
r - x	5
r w -	6
r w x	7

- On peut utiliser la **notation octale** pour les droits avec la commande **chmod**, cela permet de positionner plusieurs types de droits en une seule commande.
- Exemples:
  - Attribuer les droits **rw-------** à tous les fichiers .
  - Attribuer les droits **rw-r--r--** à tous les fichiers textes.
  - Attribuer les droits **rwxr-x---** à tous les fichiers.

- Lorsqu'on crée un nouveau fichier, par exemple avec la commande **touch**, ce fichier possède certains droits par défaut.

- La commande **umask** permet de changer ces droits par défaut. Les droits sont indiqués de façon « inverse» en notation octale pour les droits de type r et w seulement.

umask digit	default file permissions	default directory permissions
0	rw	rwx
1	rw	rw
2	r	rx
3	r	r
4	w	wx
5	w	w
6	x	x
7	(no permission allowed)	(no permission allowed)

Umask - exemples

- Pour créer des fichiers en mode rw-r--r-- : umask 022
- Pour créer des fichiers en mode ------- : umask 666
- Si la valeur de umask est 022 alors :
  - Chaque nouveau fichier aura comme droits par défaut 644

    644 = 666 – 022
  - Chaque nouveau répertoire aura comme droits par défaut 755

    755 = 777 – 022
- umask a-x,g+w => ?

Modifier le propriétaire et le groupe des fichiers

- Linux permet de spécifier le propriétaire d'un fichier ou d'un répertoire par la commande **chown**.
  - **$> chown** le_propriétaire /home/user/monfichier

- Linux permet de spécifier le groupe d'un fichier ou d'un répertoire par la commande **chgrp**.
  - **$> chgrp** le_groupe /home/user/monfichier

Principaux commandes systèmes

- Mkdir
- Rmdir
- Cd
- Ls
- Touch
- Cat
- Ln
- …

Mkdir

- **Mkdir** : Créer un dossier

  Exemple : mkdir test

  - On peut créer deux dossiers (ou plus !) en même temps en les séparant là aussi par des espaces

    Exemple : mkdir dossier_1 dossier_2

  - L'option –p permet de créer les dossier intermédiaires :

    mkdir –p dossier_1 /dossier_intermédiaire/dossier_2

- **Rmdir** : Supprimer des répertoires **vides**.

  Exemple: rmdir dossier_1

CD

- Cd chemin

  Changer de répertoire

  - Cd chemin : change le répertoire courant pour celui spécifié par le chemin.

    Exemple : cd /home/user/Bureau

  - Cd : change le répertoire courant pour le home directory.
  - Cd - : change le répertoire courant pour le répertoire précédent.
  - Cd .. : se déplacer au répertoire père.
  - Cd . : rester dans le répertoire courant.

LS

- **ls** file_name

  Liste le contenu d'un répertoire quelques options :

  - **-a** : Affiche tous les fichiers, y compris les fichiers cachés.
  - **-R** : Affichage récursif
  - **-l** : Description complète du contenu d'un répertoire (une ligne par fichier).
  - **-d** :Evite de lister le contenu d'un répertoire : si rep est un repertoire, *ls -l rep* listera le contenu du répertoire rep, alors que *ls -ld rep* listera la description du répertoire

Touch - Cat

- **Touch** : L'intérêt de **touch** , c'est que si le fichier n'existe pas, il sera créé .

  Exemple: $> touch fichier_1

- **Cat** fich1 fich2 : concatène et affiche (sur la sortie standard) le contenu des fichiers

Less - More

- **LESS** file_name

  Afficher le fichier progressivement page par page.

- **MORE** file_name

  Visualise le contenu du ou des fichiers par page. Pour un fichier contenant plus d'une page :

  - **q** ou **Q** :pour terminer la visualisation
  - **RETURN** :pour visualiser une ligne supplémentaire
  - **ESPACE** :pour visualiser la page suivante
  - **h** :pour obtenir de l'aide

- Remarque: Pour faire simple, la différence entre **more** et **less** c'est que more est vieux et possède peu de fonctionnalités, tandis que **less** est beaucoup plus puissant et rapide.

Head - Tail

- **HEAD** file_name

  La commande **head** (« tête » en anglais) affiche seulement les premières lignes du fichier. Elle ne permet pas de se déplacer dans le fichier comme **less**, mais juste de récupérer les premières lignes.

- **TAIL** file_name

  Afficher la fin du fichier.

  Exemple : $> tail –n 3 file_name

Tail - Head

- Prendre les neuf premières lignes, et n'en garder que les cinq dernières (de 5 à 9 inclus).
  - head -9 fichier | tail -5
- Prendre les 8 dernières lignes (de 5 à 12 inclus) et n'en garder que les 5 premières.
  - tail -8 fichier | head -5
- Afficher la cinquième ligne d'un fichier
  - head -5 fichier | tail -1

Mv

- **MV** src dest

  Renomme ou déplace le fichier source en destination. Si la destination est un répertoire, alors la source peut être une liste de fichiers.

Rm

- **RM** [**-irf**] fichiers

  Efface les fichiers(attention, on ne peut pas récuperer un fichier qui a été effacé)

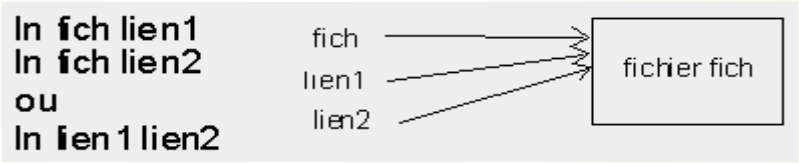
  - **-i** :interactif, demande une confirmation sur chaque fichier
  - **-f** :force la suppression du fichier
  - **-r** :récursivité. permet d'effacer un répertoire et son contenu

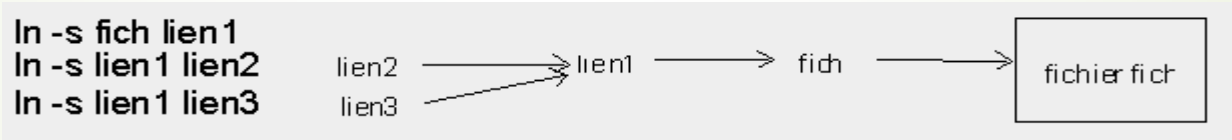
Ln

- **ln** [**-s**] source destination

  Création un lien sur un fichier ou un répertoire. Un lien est un moyen d'accéder à un même fichier ou un répertoire sous plusieurs noms ou à partir de plusieurs répertoires.

  - Attention un lien n'est pas une copie: si vous modifiez le fichier alors tous les liens sur ce fichier seront modifiés.
  - Il existe deux sortes de liens: le lien physique et le lien symbolique (avec l'option **-s**). Le lien physique ne peut adresser que des fichiers, alors que le lien symbolique peut aussi lier des répertoires.





- Dans le cas de lien physique, pour effacer le fichier, vous devez effacer tous les liens qui pointent sur ce fichier.
- Par contre pour des liens symboliques, vous pouvez effacer le fichier sans effacer les liens, mais alors ceux-ci seront invalides.

Cp

- **cp** source destination

  Copie le fichier source dans le fichier destination.

  - Si le fichier destination n'existe pas, il est créé . Sinon son contenu est écrasé sans avertissement.
  - Si la destination est un répertoire, alors la source peut être une liste de fichiers.

Recherche de fichiers

La recherche dans l'arborescence d'un système de fichiers peut se faire grâce à des utilitaires tel que **find**, **locate**, **which**, **whereis**, **whatis** et **apropos**.

Find

- Syntaxe : **find <chemin> <critères>**.
- La commande **find** est la plus ancienne commande de recherche d'Unix, elle n'utilise pas de base indexée et son exécution peut donc parfois être longue car elle est très complète par ses critères de recherche.
- Les principales options de recherche sont les suivantes :
  - **-name** : par nom de fichiers ;
  - **-type** : par type du fichier (f : fichier, d : répertoire ...) ;
  - **-user** : utilisateur auquel appartiennent les fichiers recherchés ;
  - **-atime** : par date de dernier accès aux fichiers ;
  - **-mtime** : par date de dernière modification du contenu des fichiers ;
  - **-ctime** : par date de dernier changement des fichiers : contenu mais aussi droits d'accès, propriétaire....

- Pour rechercher le fichier *Xinitrc* dans tout le système (à partir de la racine) :

find / -name Xinitrc

- Pour rechercher les fichiers de l'utilisateur userSTD1 dans tout le système (à partir de la racine) :

find / -user userSTD1

Find

- L'option **–iregex** permet de spècifier une expression régulière pour la recherche.

- Pour exécuter une commande sur les résultat de la recherche on utilise la fonction **–exec**

exemple : **find Desktop/ -iregex ".*.txt" -exec echo Bonjour \{\} \;**

Find - exemple

- **Exemple :** Nous allons chercher les fichiers ".ogg" dans un répertoire "~/Music", etles copier dans un répertoire "~/baladeur" :

  $ find ~/Music -name *.ogg -exec cp "{}" ~/baladeur \;

  - on indique la commande à effectuer (ici "cp" pour "copier") ;
  - on indique que l'on souhaite exécuter "cp" sur le résultat de "find" : ce résultat est symbolisé par deux accolades
  - pour éviter que ces accolades soient interprétées par le shell, on les encadre par des guillemets doubles ou simples
  - on indique le chemin de destination (ici ~/baladeur) ;
  - enfin, on termine la commande avec un point virgule ; que l'on protège par un backslash, comme ceci \;

locate et slocate

- La commande **locate** cherche tous les types de fichiers dans l'intégralité des systèmes de fichiers comme **find**, mais elle utilise une base de données.
- La base de données est automatiquement mise à jour par une commande de type **cron**, généralement la nuit, lorsque la machine est peu sollicitée.
- On peut mettre à jour manuellement la base de données en utilisant la commande **updatedb** (on doit être *root* pour lancer cette commande).
- Les options de fonctionnement de la commande **updatedb** sont décrites dans le fichier **/etc/updatedb.conf**. On peut y décrire la racine de l'arborescence à indexer, les fichiers à exclure, l'emplacement de la base de données, etc.

- La recherche est donc très rapide et peut se faire à partir de fragments du nom :

```
$> locate monfichier_perdu
/home/nicolas/trucs/monfichier_perdu
```

- La commande **slocate** est la version sécurisée de **locate**. Elle fonctionne de la même manière, mais stocke également les droits d'accès associés aux fichiers ainsi que les informations de propriété (propriétaire et groupe) du fichier de façon à ne pas afficher dans le résultat de la recherche les fichiers auxquels l'utilisateur n'aurait pas accès.

Which

- La commande **which** est utilisée pour trouver l'emplacement d'une commande : elle effectue sa recherche par rapport au contenu de la variable **PATH**, et retourne le chemin du premier fichier correspondant.

$> **which** bash

/bin/bash

- Elle est très commode pour vérifier quelle version de la commande s'exécute réellement lorsqu'on l'appelle par son nom relatif.

Whereis

- La commande **whereis** fonctionne de façon similaire à **which**, mais elle peut aussi chercher dans les pages de manuel (man) et les codes sources.

bash: /bin/bash /usr/share/man/man1/bash.1.bz2

Whatis

- La commande **whatis** cherche des commandes dans l'intégralité des systèmes de fichiers comme **which**, mais elle utilise une base de données qui contient une courte description ainsi que des mots clés.
- La base de données est créée en utilisant la commande **makewhatis** (on doit être *root* pour lancer cette commande).
- La recherche est donc plus rapide et peut se faire à partir du nom ou d'un mot clé. La réponse contient une description rapide de la commande

$> **whatis** who

who – show who is logged on

Les filtres

cat, cut, expand, fmt, head, join, nl, od, paste, pr, sed, sort, split, tac, tail, tr, unexpand, uniq, wc, grep, regexp

- Les **filtres** sont des commandes qui, à partir d'un flux d'entrées donné, effectuent des traitements avant d'afficher un résultat en sortie. On les nomme également **commandes de traitement de flux**.
- Exemples :
  - La commande **cat** affiche seulement le contenu du fichier /etc/passwd sans altérer le fichier original :

    $> **cat** /etc/passwd

  - La commande **nl** affiche en sortie les lignes du fichier /etc/passwd en les numérotant sans altérer le fichier original :

    $> **nl** /etc/passwd

Cat

- cat : pour concaténer le contenu d'un fichier et l'afficher ensuite sur la sortie standard, qui est par défaut l'écran.

- La commande cat sans argument (nom de fichier) attend les suites de caractères tapés sur l'entrée standard (le clavier) :

$> **cat** /etc/passwd

Cut

- Pour n'afficher que certaines colonnes (champs) d'un fichier donné. Cette commande utilise alors comme option le séparateur de champs et le numéro du champ à afficher pour chaque ligne. S'il n'est pas spécifié, le séparateur de champs par défaut est le caractère de tabulation.
- La commande ci-dessous permet d'afficher les noms d'utilisateurs du système. On utilise comme séparateur de colonnes le caractère « : » et on sélectionne la première colonne du fichier /etc/password.

$> **cut** -d : -f1 /etc/passwd

Expand - Fmt

- **Expand** :

  Pour convertir les tabulations d'un fichier en espaces. Le processus inverse, conversion des espaces en tabulations, peut être effectué avec la commande **unexpand** ;

- **Fmt** :

  Pour formater les paragraphes dans un fichier ;

Grep

- **grep** [-iv] CHAINE fichiers

  CHAINE peut être une expression régulière.

  Sans option :recherche dans les fichiers les lignes contenant l'expression

  -i :pour ne pas tenir compte des majuscules/minuscules

  -v :pour afficher les lignes ne contenant pas l'expression spécifiée.

GREP - Expression Régulière

- Définition
  - – Formule qui représente une chaîne de caractères
  - – Composée de caractères et d'opérateurs
- Utilisation
  - – On recherche alors non pas un mot ou une simple chaîne de caractères mais une suite de caractères qui correspondent au critères énoncés par la formule
  - – Certains opérateurs doivent être précédés d'un \ pour ne pas entrer en conflit avec le shell, ainsi : **{**, **}**, **<**, **>**, **(**, **)** et | seront écrits **\{**, **\}**, **\<**, **\>**, **\(**, **\)** et **\|**. C'est aussi le cas de l'espace

- Comment représenter
  - Un caractère quelconque : **.**
  - Une ou une infinité d'occurrences : **+**
  - Zéro ou une infinité d'occurrences : *****
  - Un choix parmi un ensemble : **[**<liste>**]**
  - Tout sauf un certain caractère : **[^**<caractère>**]**
- Evidement, on peut combiner les expressions
  - **.*** : Zéro ou une infinité de caractères quelconques
  - **a+b*** : Au moins un 'a' suivi de 0 ou une infinité de 'b'
  - **[ab]+** : Au moins un 'a' ou 'b' ou une infinité
  - etc.

- Exemples
  - **v.+** : Les chaînes contenant un **v** suivi de n'importe quelle suite de caractères
    - **v**andale
    - **v**estale
    - la**v**age
  - **[vs].+** : Les chaînes contenant un '**v**' ou un '**s**' suivi de n'importe quelle suite de caractères
    - **v**andale
    - **v**oiture
    - de**s**cendre
    - **s**andale

- **Exemples**
  - **a.*a** : Les chaînes contenant deux 'a'
    - palabres
    - sandale
    - pascale
    - cascade
  - **[ps].*a.*a** : Les chaînes contenant un 'p' ou un 's' suivi d'une sous chaîne contenant deux 'a'
    - **sa**nd**a**le
    - **pa**sc**a**le
    - a**p**prentiss**a**ge linéaire

- Comment représenter
  - Un caractère compris entre 'a' et 'z' : **[a-z]**
  - Un caractère compris entre '0' et '9' : **[0-9]**
  - Le début d'une ligne : **^**
  - La fin d'une ligne : **$**
  - Un choix entre deux chaînes : **\|**
  - Et en combinant avec le choix
    - Caractère compris entre 'a' et 'z' ou 'A' et 'Z' : **[a-zA-Z]**
- Répétitions d'une occurrence
  - Exactement **2** répétitions de 'x' : **x\{2\}**
  - Entre **2** et **5** répétitions de 'x' : **x\{2, 5\}**
  - Au moins **2** répétitions de 'x' : **x\{2, \}**

- Pour afficher toutes les lignes contenant la chaîne de caractères « false » :

$> **grep** false /etc/passwd

- Pour afficher toutes les lignes qui commencent par la chaîne « root » :

$> **grep** ^root /etc/passwd

- Pour afficher les lignes ne commençant pas par la chaîne « root » :

$> **grep** –v ^root /etc/passwd

	Avec `ls` seul	Avec `ls` et `grep`
Commence par «a» et dont la deuxième lettre est «s» ou «t»	`ls a[st]*`	`ls | grep '^a[st].*'`
Contient «un» et se termine par «t»	`ls *un*t`	`ls | grep '.*un.*t$'`
Contient «gre» ou «st»	`ls *(gre|st)*`	`ls | grep '\(gre|st\)'`
Contient exactement deux lettres «m»		`ls | grep '[^m]*m[^m]*m[^m]*'`
Contient au moins deux lettres «m»		`ls | grep '.*m.*m.*'`
Contient au moins quatre caractères et aucun chiffre		`ls | grep '^[^0-9]\{4,\}$'`
Est constitué de deux lettres exactement	`ls ??`	`ls | grep '^..$'`
Commence et finit par un chiffre	`ls [0-9]*[0-9]`	`ls | grep '^[0-9].*[0-9]$'`

- **Comment éliminer les lignes vides dans un fichier ?**
  - Grep -v '^$' fichier
- **Comment calaculer le nombre de lignes vides dans un fichier.**
  - Grep -v '^$' fichier PIPE wc -l
- **Comment calculer le nombre de lignes contenant au moins un espace.**
  - Grep '.*[[:space:]].*' fichier PIPE wc -l
- **Comment éliminer les lignes ne contenant que des blancs ?** on utilise une classe de caractère préexistante : [:space:]
  - Grep -v '^[[:space:]]$' fichier
- **Comment éliminer toutes les lignes blanches ?**
  - grep -v '\(^[[:space:]]$\|^$\)' fichier1 > fichier2
- **Comment afficher uniquement les fichiers du répertoire courant qui sont des liens symboliques ?**
  - ls -l | grep '^l'

Regexp

- Elle permet de tester une expression régulière en lui fournissant une chaîne de test. Cela permet de vérifier les expressions régulières employées avec les commandes classiques ls, sed, awk, etc.

Sort - Split

- Sort :

  Cette commande permet de trier le contenu d'un fichier par ligne :

  - option -n pour effectuer un tri numérique,
  - option -r pour effectuer un tri décroissant ;
  - Les caractères ``**+**'' et ``**-**'' permettent de spécifier de quelle colonne à quelle colonne le tri doit s'effectuer (1^{ere} colonne pour 0, 2eme colonne pour 1...) : sort +1 -2 /etc/passwd

- Split :

  Permet découper un fichier en plusieurs. On peut spécifier une taille de fichiers en option.

  Pa r exemple, la commande pour créer les fichiers petitfichieraa, petitfichierab… d'une taille maximum de 1,4 Mo (taille d'une disquette) est :

  $> **split** -b 1.4m /home/maitre/grosfichier petitfichier

WC

- Cette commande affiche des statistiques sur un fichier, nombre de caractères, nombre de mots et nombre de lignes.
  - -b : affiche seulement le nombre de caractères ;
  - -w : affiche seulement le nombre de mots ;
  - –l : affiche seulement le nombre de lignes.

- Exemple :

```
$> wc sources.list 31 175 1686 sources.list
```

Commande composée

Redirection

- Linux fonctionne avec trois types de flux de données :
  - l'entrée standard identifiée par le descripteur 0, par exemple le clavier ;
  - la sortie standard identifiée par le descripteur 1, par exemple l'écran ou l'interpréteur de commande ;
  - la sortie d'erreur standard identifiée par le descripteur 2, par exemple l'écran :

```
maitre@maestro maitre$echo 'ceci est un message de test'
ceci est un message de test
maitre@maestro maitre$
```

Entrées/sorties

- Par défaut, tout programme (script ou non) lit du clavier est écrit sur l'écran.
- A chaque programme sont associées une entrée standard, une sortie standard et une erreur standard. Cela peut être change avec :
  - **> fichier** : spécifie que la sortie standard n'est plus l'écran, mais le fichier spécifié.
  - **< fichier** : spécifie que l'entrée standard n'est plus le clavier, mais le fichier spécifié.
  - **2> fichier** : spécifie que l'erreur standard n'est plus l'écran, mais le fichier spécifié.
  - **2>&1** : spécifie que l'erreur standard devient la même que la sortie standard
- Exemple : $> programme <données >resultats 2>erreur

- La commande **echo** permet d'effectuer un simple affichage sur la sortie standard.
- Le mécanisme de redirection permet de changer la sortie, l'entrée ou la sortie d'erreur d'une commande donnée. Le caractère « **>** » est utilisé pour changer la sortie standard.

Syntaxe : commande > chemin.

maitre@maestro maitre$echo 'message test' > /home/maitre/fichier
maitre@maestro maitre$cat /home/maitre/fichier
message test
maitre@maestro maitre$

- Si le fichier de redirection n'existe pas encore, « **>** » permet de le créer. S'il existe, son contenu sera écrasé par la sortie de la dernière commande. Les caractères « **>>** » permettent soit de créer un fichier inexistant soit de rajouter la sortie d'une commande au contenu d'un fichier existant (sans écrasement). **commande < chemin**
- La redirection fonctionne dans les deux sens, le caractère « < » permet de spécifier une autre entrée que l'entrée standard, syntaxe :

- Les caractères « **<<** » permettent de lire le fichier en entrée jusqu'à ce que la commande rencontre une certaine chaîne de caractères.
- Dans l'exemple qui suit, les caractères saisis sur l'entrée standard seront pris en compte jusqu'à ce que la commande cat rencontre la chaîne « FIN ».

cat **<<** FIN

- Enfin, pour rediriger la sortie d'erreur vers un fichier, on utilise le descripteur 2 de la sortie standard :

cat /etc/passwd **2>** fichier_erreur

- Les caractères « **>&** » permettent de rediriger la sortie erreur et la sortie standard vers un fichier :

tail /etc/passwd **>** fichier_sortie **2>&1**
Ou
tail /etc/passwd **>&** fichier_sortie

- Cette commande copie les 10 dernières lignes du fichier /etc/passwd dans le fichier fichier_sortie et y redirige également les éventuels messages d'erreur.

Les tubes

- Le mécanisme de tube (*pipe*) permet de faire en sorte que la sortie d'une commande devienne l'entrée d'une autre. Les tubes utilisent le caractère «|», syntaxe : commande | commande.

```
sort /etc/passwd | head -6
```

- Cette commande affiche les 6 premières lignes du fichier /etc/passwd une fois ce fichier trié par ordre alphabétique croissant.
- Les Tubes et redirections peuvent êtres enchaînés indéfiniment sur une ligne de commande selon les résultats que l'on veut obtenir.

La commande **tee**

- La commande **tee** duplique le flux de données en sortie : elle copie la sortie dans un fichier (simple redirection) et, en même temps, affiche le résultat sur la sortie standard, et permet donc de le renvoyer à une autre commande.
- La commande de l'exemple ci-dessous affiche à l'écran les 6 premières lignes du fichier /etc/passwd et, en même temps, les redirige dans le fichier le_fichier.

```
ls -l /etc | tee le_fichier | wc -l
```

La commande **xargs**

- La commande **xargs** permet de passer en argument d'une commande les flux reçus en entrée.
- La commande de l'exemple ci-dessous prend la sortie de la commande **cat** le_fichier comme argument de la commande **ls**.

```
cat le_fichier | xargs ls
```

- On peut obtenir le code de sortie d'un processus par le biais de la commande **exit**.

  En SH : **exit n**

  - Si n=0, alors le processus s'est déroulé sans erreur, sinon il y a eu une erreur correspondant au code de sortie n.

- $>commande1 **&&** commande2 **&&** commande3 :
  - commande2 s'exécute **ssi** le code de sortie de commande1 vaut **0**.
  - De même commande3 s'exécute **ssi** le code de sortie de commande2 vaut **0**.
- $> commande1 || commande2 || commande3 :
  - Commande2 s'exécute **ssi** le code de sortie de commande1 est diffèrent de 0.
  - De même commande3 s'exécute ssi le code de sortie de commande2 est diffèrent de 0.

PIPE - & - && - ;

- | : Branche la sortie standard de la commande de gauche sur l'entrée standard de la commande de droite.
- **>>** : Change la sortie standard pour l'ajouter à la fin d'un fichier existant.
- || : Exécuter la commande suivante si la première a échoué.
- **&** : Exécuter les commandes d'une manière asynchrone.
- **&&** : N'exécuter la commande suivante que si la première a réussi.
- **;** : exécuter les commandes successivement .

Exemple

- $> commande1 **&** commande2 **|** commande3 **&&** commande4

  on aura :

  - **commande1** exécutée de manière asynchrone
  - **commande2** exécutée de manière asynchrone
  - **commande3** reçoit la sortie de **commande2**
  - **commande4** est exécutée si **commande3** s'est bien passée. Son entrée standard est la sortie standard de **commande2**.

Script SHELL

- Bash est le shell de GNU/Linux, un shell étant l'interface utilisateur d'un système d'exploitation. Il est basé sur le *Bourne Shell* d'Unix, d'où son nom, qui est l'acronyme de *Bourne-again shell*.

Echo - Read

- Pour lire une ligne (un mot) de l'entrée standard, utilisez la commande **read**. Pour écrire une ligne a la sortie standard ou a la erreur standard, utilisez **echo**.

- Exemple :
  - **read lecture** : lit une ligne (un mot) et la met dans la variable lecture
  - **echo Lecture de la chaine $lecture 1>&2** : sur la sortie standard.
  - **echo Lecture de la chaine $lecture 2>&1** : sur l'erreur standard

Environnement BASH

- Pour affecter un contenu à une variable, on utilise la commande = de la manière suivante :

ma_variable = 'Ne pas oublier le chat'

- Attention à ne pas mettre d'espace avant et après le signe « **=** » !!!
- Pour faire référence au contenu d'une variable, on la préfixe par le signe « **$** ».

echo $ma_variable

Ne pas oublier le chat

- Pour effacer le contenu d'une variable, on utilise la commande unset.

unset ma_variable

- Le shell utilise des variables pour tenir compte des paramètres de configuration spécifiques des utilisateurs appelés **variables d'environnement**. Les variables **HOME**, **DISPLAY**, **PWD** en font partie.

echo $HOME

/usr/home/sabri

- Lors de l'utilisation d'un programme, le shell utilise la variable d'environnement **PATH** pour retrouver le chemin d'accès à ce programme.
- On peut afficher le contenu de cette variable par la commande echo :

echo $PATH

/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr/games:/usr/X11R
6/bin:/usr/home/nicolas:/:/usr/bsd:/usr/sbin:/usr/local/bin:/usr
/bin/X11

- Pour qu'une variable soit visible de tous les shells (donc de toutes les commandes lancées depuis le shell courant), il faut l'exporter par la commande export.

export MANPATH="/usr/share/docs"

- Lors du démarrage d'une session shell, la plupart des variables d'environnement sont initialisées et exportées à partir des différents fichiers de configuration, tels que les fichiers **bashrc** et **profile**.

- La commande **env** permet de démarrer un shell avec des variables d'environnement déjà positionnées à des valeurs données. Ces variables d'environnement ont une durée de vie égale à celle du shell démarré.
- Par exemple, pour lancer la commande ma_commande en positionnant la variable d'environnement ma_variable à la valeur « ma_valeur » :

env ma_variable **=** ma_valeur ma_commande

- La durée de vie et la visibilité de ma_variable sont limitées à la durée d'exécution de ma_commande.

Les fichiers de configuration

- Il y a plusieurs types de fichiers de configuration, ceux qui sont lus au moment de la connexion (*login*) et ceux qui sont lus à chaque lancement d'un shell.
- Les fichiers lus au moment de la connexion au système sont :
  - /etc/profile, commun à tous les utilisateurs (s'il existe) ;
  - ~/.bash_profile ou éventuellement ~/.bash_login ou ~/.profile/, spécifiques à chaque utilisateur.

Ils servent généralement à décrire l'environnement de l'utilisateur.

- Les fichiers lus à chaque lancement de shell sont :
  - /etc/bashrc commun à tous les utilisateurs (s'il existe) ;
  - ~/.bashrc spécifique à chaque utilisateur.

Les scripts – Script Shell

- Un script shell est une liste d'instructions contenues dans un fichier.
  - **#!/bin/bash**
  - **# Un petit script mon_script**
  - echo 'Ne pas oublier le chat'
- Pour pouvoir exécuter ces instructions, deux conditions doivent être remplies :
  1. la première ligne doit contenir #!/bin/bash (pour un shell script utilisant bash) ;
  2. le fichier doit être exécutable (e.g. en utilisant la commande chmod +x) et lisible (e.g. avec les permissions 755)

  chmod +x mon_script

  ./mon_script
- Si toutes ces conditions ne sont pas remplies, il est toujours possible de forcer l'exécution du script avec la commande bash.

  bash mon_script

Script - Passage de paramètres à un script

- Dans le script par les variables réservées **$1** pour le premier argument, **$2** pour le deuxième et ainsi de suite.
- À noter qu'il existe un opérateur **shift** qui décale ces paramètres : la valeur contenue dans $2 passe dans $1, celle contenue dans $3 passe dans $2 et ainsi de suite. Cela permet essentiellement d'utiliser plus de 9 paramètres en entrée.

- D'autres variables réservées sont accessibles à l'intérieur d'un script :
  - **$0** : nom du script ;
  - **$*** : liste des paramètres ;
  - **$#** : nombre de paramètres ;
  - **$$** : numéro du processus en cours d'exécution ;
  - **$?** : valeur de retour de la dernière commande.
- Exemple :

```
#!/bin/bash
# un autre script
echo "mon script est $0"
echo "il y a eu $# paramètres en entrée"
echo "le premier paramètre est $1"
```

Exercices

1. Ecrire un script qui permet d'afficher les 3 paramètres d'un script.

2. Ecrire la commande système permettant d'afficher les différents noms utilisateurs du système.

   $> cut –f1 –d: /etc/passwd

3. Ecrire un script shell qui affiche la ligne descriptif d'un utilisateur passé en paramètre.

Script - Les expressions logiques

- Les expressions logiques sont évaluées à l'aide de la fonction test, qui peut également s'écrire **[ ]**.
- Le résultat de l'évaluation est stocké dans la variable **$?** qui contient :
  - **0** si le résultat est vrai ;
  - une valeur différente de 0 si le résultat est faux.
- Pour vérifier si le fichier /bin/bash existe :

  test -f /bin/bash

  ou

  [-f /bin/bash]
- Pour vérifier si le fichier ~/bin/mon_script est exécutable :

  test -x ~/bin/mon_script

  ou

  [-x /bin/mon_script]

- Les expressions logiques peuvent être combinées par les opérateurs logiques && (ET/AND) et || (OU/OR). Il est également possible d'utiliser les connecteurs -a (ET/AND) et -o (OU/OR).
- Pour vérifier si les fichiers /etc/profile et /etc/bashrc existent :

test -f /etc/profile -a test -f /etc/bashrc

OU

test -f /etc/profile && test -f /etc/bashrc

OU

[-f /etc/profile -a -f /etc/bashrc]

- Quelques options de la commande test :
  - -f : vérifie si le fichier est un fichier standard ;
  - -d : vérifie si le fichier est un répertoire ;
  - -b : vérifie si le fichier est de type bloc ;
  - -e : vérifie si le fichier existe indépendamment de son type ;
  - -r : vérifie si le fichier est lisible ;
  - -w : vérifie si le fichier est inscriptible ;
  - -x : vérifie si le fichier est exécutable.
- D'autres options seront données pour le traitement spécifique des nombres.

Instruction **IF**

- Cette instruction sert pour les tests et branchements.

Syntaxe (la partie « else... » est optionnelle) :

```
If [ conditional
expression1 ] then
   statement1
   statement2
       .
elif [ conditional
expression2 ] then
   statement3
   statement4
      . . .
else
    statement5
fi
```

```
#!/bin/bash
count=99
if [ $count -eq 100 ]
then
   echo "Count is 100"
elif [ $count -gt 100 ]
then
   echo "Count is greater than 100"
else
   echo "Count is less than 100" fi
```

Script - Calculs

- Il est possible de comparer des nombres en utilisant la commande test, vue précédemment, avec les options suivantes :
  - -**lt** pour « inférieur à » (<) ;
  - -**gt** pour « supérieur à » (>) ;
  - -**le** pour « inférieur ou égal à » (<=) ;
  - -**ge** pour « supérieur ou égal à » (>=) ;
  - -**eq** pour « égal à » (=) ;
  - -**ne** pour « différent de » (!=).

- Pour tester l'existence du fichier « monfichier.txt » :

```
#!/bin/sh

if test -f monfichier.txt then

echo " le fichier existe "

fi
```

- Pour tester le nombre de paramètres en entrée :

```
#!/bin/bash
if [ $# -eq 0 ] then
echo " Vous n'avez entré aucun paramètre"
fi
```

- La commande **expr** permet d'effectuer les quatre opérations arithmétiques de base, avec les opérateurs suivants :
  - **+ pour l'addition ;**
  - **- pour la soustraction ;**
  - *** pour la multiplication ;**
  - **/ pour la division.**
- Par exemple, la commande **expr 1 + 2** renvoie « 3 ».

Boucle **FOR**

- Cette boucle sert pour répéter les traitements un nombre de fois connu.
- Syntaxe :

```
for <variable> in <liste> do
commande1
commande2
commande3
...
done
```

- Pour afficher les jours de la semaine :

```
for jour in lundi mardi mercredi jeudi vendredi samedi dimanche
do
echo $jour
done
```

Boucle **WHILE**

- Cette boucle sert pour répéter les traitements un nombre de fois inconnu *a priori*. Le test de continuité se fait au début de la boucle. La boucle continue tant que la condition est vraie.
- Syntaxe :

```
while <condition> do
<commande1>
<commande2>
...
done
```

- Pour faire le test de lecture d'une valeur jusqu'à ce que l'utilisateur entre la valeur « 1 » :

```
#!/bin/sh
i=0
while [ $i -ne "1" ] do
read i
done
```

- Il existe une autre boucle, qui utilise le mot clé until à la place de while. Elle diffère dans le traitement de la condition : la boucle est exécutée jusqu'à ce que la condition soit vraie. Le test est donc effectué en fin de boucle et la boucle est toujours exécutée au moins une fois.

- Pour afficher une table de multiplication :

```
#!/bin/bash
for i in 1 2 3 4 5 6 7 8 9 do
expr $i \* $1
done
```

- Pour compter jusqu'à 100 :

```
#!/bin/bash
i=0
while [ $i -ne 100 ] do
i=`expr $i + 1`
echo $i
done
```

- On peut également écrire expr <expression> sous la forme ((<expression>))

```
#!/bin/bash
i=0
while [ $i -ne 100 ] do
#i=`expr $i + 1`
i=$(($i+1))
echo $i
done
```

Exercice

1. Ecrire un script shell qui prend en paramètre un nom d'utilisateur. Puis il vérifie si ce dernier existe.
2. Ecrire un script shell qui demande l'année du naissance puis calcule et affiche l'âge.

```
$ ./quel-age
Votre année de naissance ?
1990
Vous êtes né en 1990, vous avez donc 19 ans.
$
```

Instruction **CASE**

- Ce test permet de spécifier les commandes à exécuter pour chacune des valeurs prises par la variable passée en argument.

Syntaxe :

```
case <variable> in
valeur1) commande1 ;;
valeur2) commande2 ;;
valeur3) commande3 ;;
...
esac
```

- Pour tester la valeur du premier paramètre :

```
#!/bin/sh

case $1 in

1) echo " un ";;

2) echo " deux ";;

3) echo " trois ";;

esac
```

Les paramètres d'entrée d'un script

- La lecture d'une valeur peut se faire par le passage de paramètres sur la ligne de commande (*cf.* précédemment l'utilisation des variables $1, $2, etc.) ou en lisant une entrée au clavier depuis le script avec la commande read.
- Pour lire le nom d'une personne dans la variable « nom » et afficher son contenu :

```
#!/bin/sh
echo 'Entrez votre nom'
read nom
echo " Vous vous appelez $nom "
```

- L'entrée au clavier lue avec la commande **read** peut ensuite être traitée avec la commande case vue précédemment.

```
#!/bin/sh
echo "entrez votre choix"
read choix
case $choix in
1) echo " menu un ";;
2) echo " menu deux ";;
3) echo " menu trois ";;
esac
```

- La commande **select** est utilisée pour demander à un utilisateur de choisir une valeur et une seule dans une liste de valeurs prédéfinies. L'invite à afficher est à indiquer dans la variable prédéfinie PS3.
- La commande **select** a deux arguments : la liste des valeurs proposées, et une variable dans laquelle sera stockée la valeur choisie. Un numéro séquentiel est automatiquement attribué à chaque valeur proposée. Le numéro de la valeur choisie sera stocké dans la variable prédéfinie REPLY.
- Syntaxe :

select <variable> in <liste de choix> »

- On peut sortir de la boucle avec la commande **break**.

- Exemple utilisant les commandes **select** et **break** :

```
#/bin/bash
PS3="Entrez le numéro de votre commande -> "
echo "Que désirez-vous manger?"
select plat in "Rien, merci" “viande" “salade" “kouskous"
do
echo "Vous avez fait le choix numéro $REPLY..."
if [ "$REPLY" -eq 1 ] then
echo "Au revoir!"
break
else
echo "Votre $plat est servie."
fi
echo
done
```