FIP SIT151 Introduction à Linux

Christophe Lohr

Automne 2023

Grace Hopper - 1960

Dennis Ritchie & Ken Thompson - 1972

Sommaire

1 Introduction **2**
1.1 Le système d'exploitation . . . 2
1.2 Historique . . . 3
1.3 De Unix à Linux . . . 4

2 Le système de fichiers **5**
2.1 Structure, nommage, droits . . . 5
2.2 Organisation sur disques . . . 14

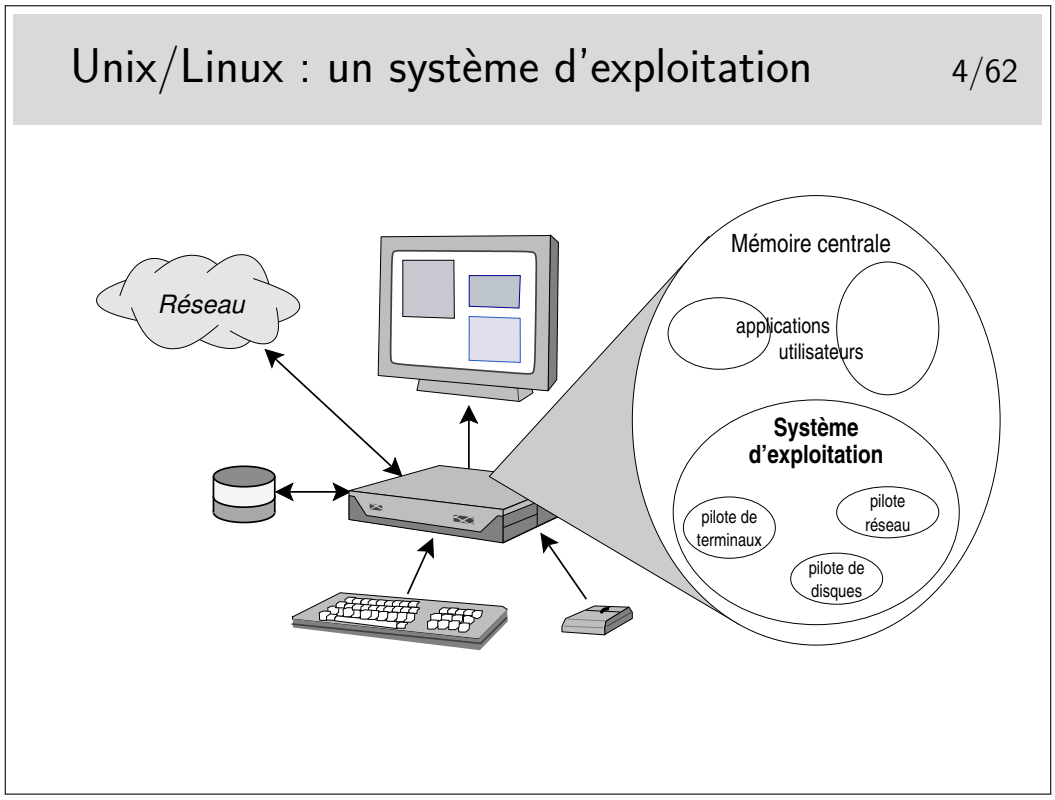
3 Utilisation courante **17**
3.1 Les commandes et leur syntaxe . . . 17
3.2 La documentation . . . 21

4 Les processus **23**
4.1 Environnement, cycle de vie . . . 23

5 L'interface graphique X-Window **30**
5.1 Client-serveur, authentification, bureau . . . 30

1 Introduction

1.1 Le système d'exploitation



Le système d'exploitation (SE, OS en anglais) est une application chargée en mémoire centrale peu après la mise sous tension de la machine (juste après que que programme résidant en mémoire EEPROM se soit exécuté, ce qu'on appelle le BIOS dans l'architecture PC).

Le SE gère les applications qui s'y déroulent, et sert d'interface entre elles et les périphériques matériels. La mémoire centrale est chargée avec le (1) Système d'exploitation et avec (2) les programmes applicatifs lancés par les utilisateurs. Le système d'exploitation contient des modules spécifiques aux périphériques qu'il sait contrôler, ce sont les pilotes (*drivers* en anglais)

1.2 Historique

Historique 6/62

- Unix naît officiellement le 1er janvier 1970 dans les laboratoires Bell AT&T : Ken Thompson et Dennis Ritchie
- Années 1970 : développement d'Unix, langage C (1973)
- Années 1980 : deux filières
  - Univ. Berkeley : système Unix BSD (Berkeley Software Development)
  - AT&T : Unix Système V, version commerciale standard
  - 1984 : Richard Stallman crée la Free Software Foundation et la Licence Publique Générale (GNU GPL)
- Années 1990 :
  - 1994 : Linus Torvalds écrit le noyau Linux
  - Les versions BSD continuent en logiciel libre : FreeBSD, OpenBSD...

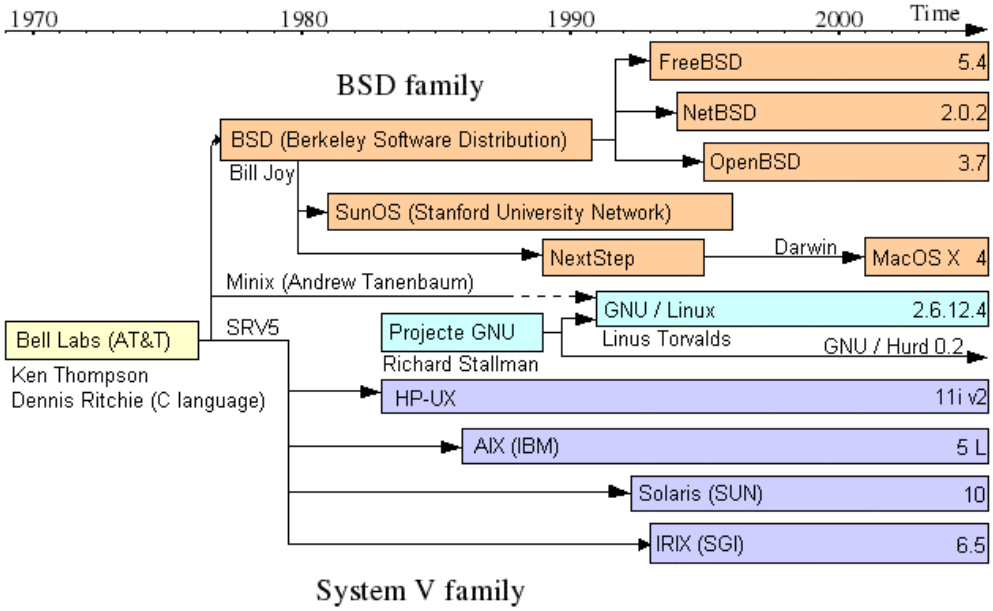
Pointeurs :

— `http://en.wikipedia.org/wiki/Unix`

— `http://virtual.park.uga.edu/hc/unixhistory.html`

— `http://www.princeton.edu/~hos/mike/transcripts/thompson.htm` (interview de Ken Thompson)

— `http://www.unix.org/what_is_unix/history_timeline.html`



1.3 De Unix à Linux

Linux : un système Unix 8/62

- Philosophie d'Unix :
  - (presque) tout s'utilise comme un fichier
  - "Do one thing, do it well" (Doug McIlroy, l'inventeur des *pipes Unix*) :
    - Write programs that do one thing and do it well.
    - Write programs to work together.
    - Write programs that handle text streams, because that is a universal interface.
- Caractéristiques d'un système d'exploitation Unix :
  - Multitâche (multi processus)
  - Multi utilisateurs
- Spécificités (de Linux et de tous les Unix) :
  - Son Système de Gestion de Fichiers
  - La gestion des processus
- → Linux c'est (une implémentation) Unix

...vocabulaire... 9/62

Linux (p.ex. Linux 5.7.0)

Le noyau, uniquement !

GNU/Linux

+ commandes Unix de base (implémentation de GNU)
copier un fichier, répertoire, permissions utilisateur..

une distribution Linux (p.ex. Ubuntu 22.04, Debian Buster)

+ organisation des fichiers, outils d'administration, applications

Sur clef USB :

— LinuxLive USB Creator `http://www.linuxliveusb.com/` (sous Windows)

— PenDriveLinux YUMI `http://www.pendrivelinux.com/yumi-multiboot-usb-creator/` (sous Windows)

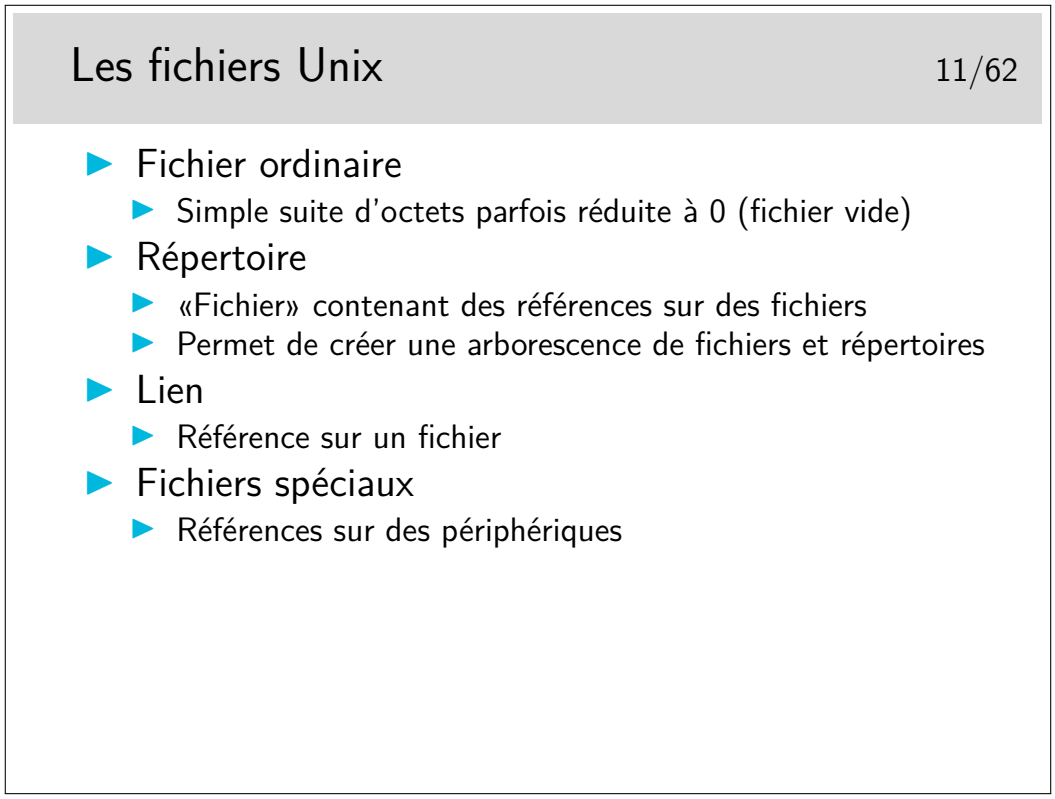
— Xboot `https://sites.google.com/site/shamurxboot/` (sous Windows)

— MultiBoot `LiveUSBhttp://liveusb.info/dotclear/` (sous Linux)

— etc.

2 Le système de fichiers

2.1 Structure, nommage, droits

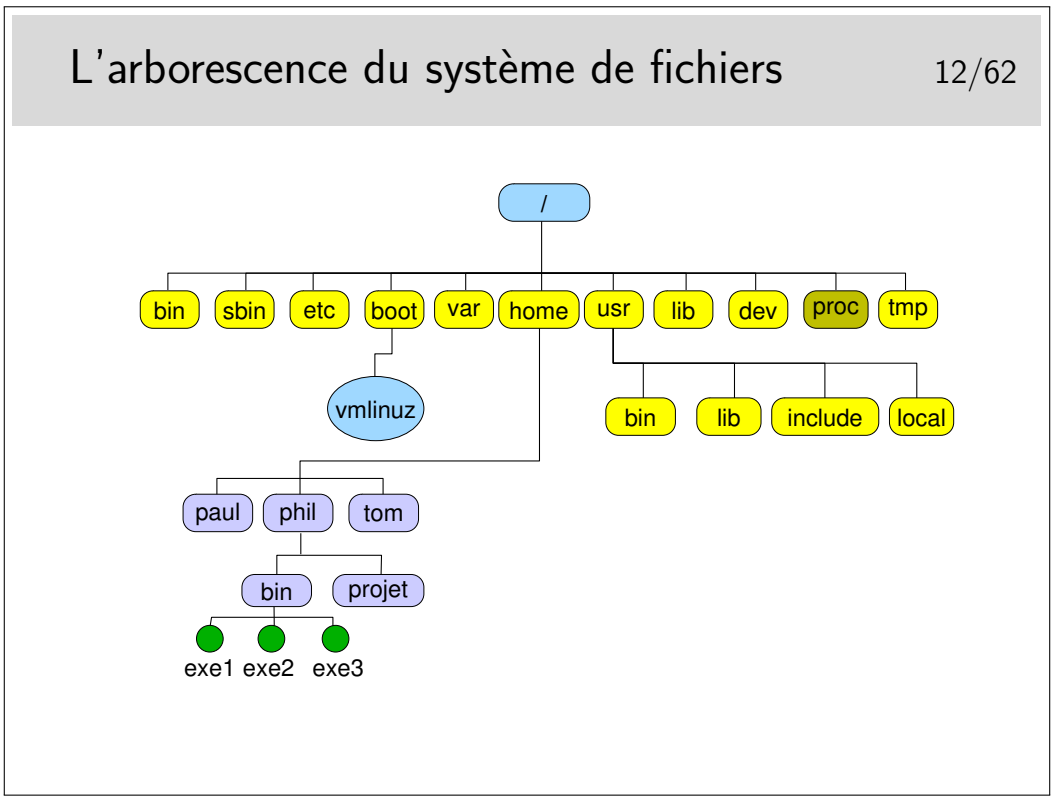


Il existe quelques autres types de fichiers pouvant apparaître dans l'arborescence :

— les sockets (type `s`) permettant la communication entre processus

— les tubes nommés (type `p` comme *pipe*) permettant aussi la communication entre processus

Les liens peuvent être de deux types : liens durs (version originale des liens sur Unix) et les liens symboliques (type l), version apportée par l'Unix de Berkeley (BSD) pour augmenter la portée des liens natifs (durs). Les liens durs ne peuvent être réalisés qu'entre références (nom dans un répertoire, on dit aussi entrée dans un répertoire) sur un même système de gestion de fichiers. Les liens symboliques peuvent passer les frontières physiques des système de fichiers.



— `/bin` (binaires) et `/usr/bin` les commandes standards (usr : UNIX System Resources)

— `/etc` fichiers d'administration (editable text configuration)

— `/sbin` commandes d'administration (binaires système)

— `/lib` et `/usr/lib` les bibliothèques (libraries ; fichiers contenant les fonctions appelées par les commandes pour réaliser les opérations avec le système, exemple libc.a est la bibliothèque standard du langage C).

— `/dev` les fichiers spéciaux, référencent les périphériques du système (les *devices*).

— `/tmp` répertoire où sont créés les fichiers temporaires, il peut exister aussi /usr/tmp et /var/tmp.

— `/var/spool` répertoire où sont créées les files d'attente pour différents services tels que l'impression, le courrier électronique, etc. (var : variable)

— `/var/spool/cron` contient les travaux du service *cron* (voir cette référence dans le manuel de référence, faire man cron).

— `/usr/include` contient les fichiers d'entête standards des programmes en langage C.

— `/boot` répertoire contenant les composants du noyau du système (bootstrap : amorçage du système).

— `/boot/vmlinuz` le noyau (le système-lui même en quelque sorte).

— etc.

Une organisation standardisée sous Unix : Filesystem Hierarchy Standard (`http://www.pathname.com/fhs/`).

Les fichiers - Le nommage 13/62

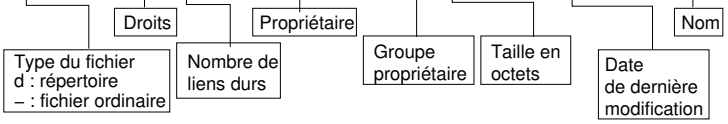
- Nommage absolu
  - par rapport à la racine, le nom commence par /
  - /home/phil/bin/exe1
- Nommage relatif
  - relatif au répertoire dans lequel on est :

    home/phil/bin/exe1 si on est dans /
    phil/bin/exe1 si on est dans /home
    bin/exe1 si on est dans /home/phil
    exe1 si on est dans /home/phil/bin

Visualiser le contenu d'un répertoire 14/62

- La commande ls
  - Exemple :

```
[bash]$ ls -l
total
-rwxr-xr-x    1   clohr   ens-rec     7790   avr 11 2020 essai
-rw-rw-r--    1   clohr   ens-rec     1122   avr 24 2020 essai.c
-rw-rw-r--    1   clohr   ens-rec  9869052   avr 25 2020 test
```



Les fichiers cachés 15/62

- Ce sont les fichiers dont le nom commence par un point
  - Exemple : .bashrc
- Par défaut les outils d'affichage du contenu des répertoires n'affichent pas les fichiers cachés
- Ce sont généralement des fichiers de configuration d'applications
- On peut les comparer à la base de registres sur des systèmes concurrents à Unix/Linux
- Il existe aussi des répertoires cachés, leur nom commence aussi par un point

Pour visualiser la liste des fichiers cachés il faut utiliser l'option `-a` de `ls`. Pour les outils graphiques de gestion de fichiers (les «explorateurs» dirions nous sous un autre système d'exploitation bien connu), il existe une option de paramétrage dans les menus de configuration (parfois appelés «*préférences*»).

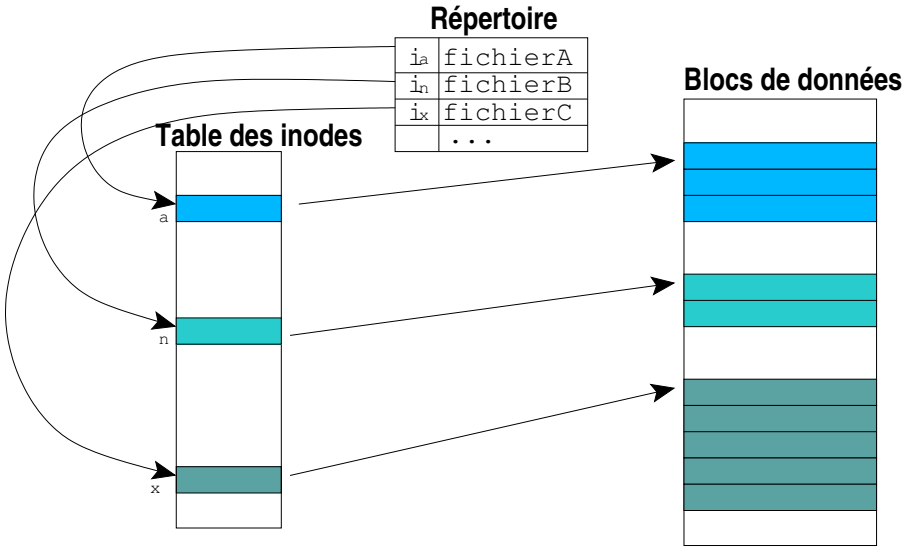
Structure et contenu d'un répertoire 16/62

- Un répertoire est avant tout un fichier
- Ce fichier, de type particulier, contient des références à des fichiers ou sous-répertoires
- Une référence est composée de :
  - la longueur de la référence
  - la longueur du nom du fichier (< 256)
  - le nom du fichier
  - un nombre index appelé **numéro d'inode** qui identifie une structure dans une table appelée ***table des inodes***
  - Cette structure contient toutes les informations sur le fichier

Système de fichiers - Contenu d'un inode 17/62

- mode et type du fichier (voir droits et protections)
- nombre de liens sur le fichier
- numéro d'identification du propriétaire du fichier (uid)
- numéro de groupe du propriétaire du fichier (gid)
- taille du fichier en octets
- adresse des blocs constituant le fichier
- date du dernier accès à ce fichier
- date de la dernière modification de mode
- date de création

Répertoire / inodes / bloc de données 18/62



Un lien est un nom de fichier, une référence dans un répertoire. Un fichier peut être référencé plusieurs fois, on dit alors qu'il a plusieurs liens. Ces liens peuvent être symboliques ou physiques (on dit alors «lien dur»). Un lien dur est tout simplement une référence dans un répertoire sur un fichier déjà référencé. Les deux références sont associées au même inode. Dans le schéma ci-dessus on pourrait par exemple créer un lien sur `fichierB` dans un répertoire quelconque. `fichierB` serait alors accessible par deux chemins. Les deux noms auraient le même inode.

La commande `ln` permet de créer des liens durs ou symboliques.

Les liens symboliques sont des fichiers presque normaux, seul leur type est spécifique. Leur contenu est le chemin d'accès au fichier réel.

Les liens durs ne peuvent être effectués qu'à l'intérieur d'un même système de fichiers (point commun : l'inode). Les liens symboliques peuvent sans problème franchir les frontières physiques des systèmes de fichiers.

Conceptuellement, un lien peut être considéré comme un raccourci sous Windows (si vous connaissez W...)

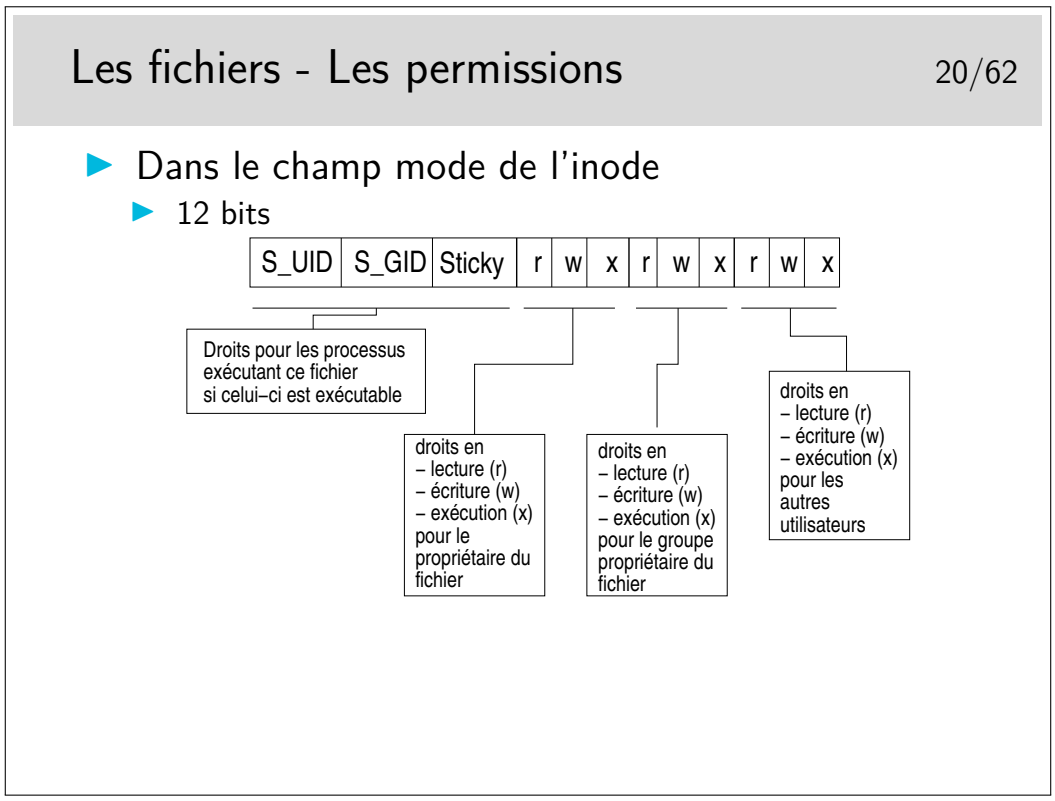
Les répertoires «.» et «..» 19/62

- Un répertoire n'est jamais vide, même à sa création, il contient déjà deux références sur des répertoires de nom «.» et «..»
  - Le répertoire «.» (point) constitue une référence sur le répertoire lui-même
  - Le répertoire «..» constitue une référence sur le répertoire immédiatement au dessus dans l'arborescence (le répertoire *père* en quelque sorte)
- Utilisation
  - Nommage sans ambiguïté d'un fichier local : `./test` par exemple
  - Nommage rapide d'un fichier au dessus : `../fichier` par exemple

Il arrive à beaucoup de gens d'écrire un petit fichier d'essai et de le nommer `test`. Après qu'il soit rendu exécutable, il est essayé... Et...Surprise ! le résultat n'est pas du tout celui attendu... Car la commande lancée, de nom `test`, n'est pas celle qu'on croit, il s'agit de `/bin/test` et non le test du répertoire courant (si la variable d'environnement `PATH` ne contient pas le caractère «.» dans sa liste avant `/bin`).

Il suffit pour remédier à cela d'entrer alors la ligne de commande `./test` et le tour est joué.

En général, par défaut, le «.» ne figure pas dans la variable `PATH` pour des raisons de sécurité élémentaire et toute commande se situant dans le répertoire de travail (celui dans lequel on se trouve) doit se lancer avec la séquence `./nomFichier` (à moins que le répertoire de travail ne soit listé naturellement dans la variable `PATH`).



Si les droits en écriture/lecture/exécution sont assez évidents à comprendre il n'en va pas de même pour les trois premiers : `S_UID`, `S_GID` et le `Sticky bit` :

— Ces droits concernent les processus d'exécution du fichier, ils ne sont actifs qu'au moment où le fichier est en exécution. Ils précisent ce que le fichier (si l'on peut dire) a le droit de faire lorqu'il s'exécute.

— Lorsqu'une commande est lancée, si celle-ci correspond à un fichier exécutable, ces droits agissent comme suit :

  — si le bit *S_UID* est positionné, le processus d'exécution a les droits du propriétaire du fichier et non ceux de l'utilisateur qui a lancé la commande,

  — si le bit *S_GID* est positionné, le processus a les droits du groupe propriétaire du fichier

  — si le *sticky bit* est positionné, le processus ne sera pas purement et simplement effacé de la mémoire, il sera recopié en zone de swap et si la commande correspondant est rappelée, son rechargement en mémoire sera plus rapide. Ce n'est plus beaucoup utilisé.

Remarque importante : le *sticky bit* est maintenant utilisé sur des répertoires ouverts en lecture/écriture à tous pour restreindre le droit d'effacement des fichiers qui s'y trouvent au seul propriétaire de ces fichiers. Cas des répertoires `/tmp`, `/var/tmp`, `/usr/tmp`. Voir aussi la notion d'attribut avec la commande `chattr`.

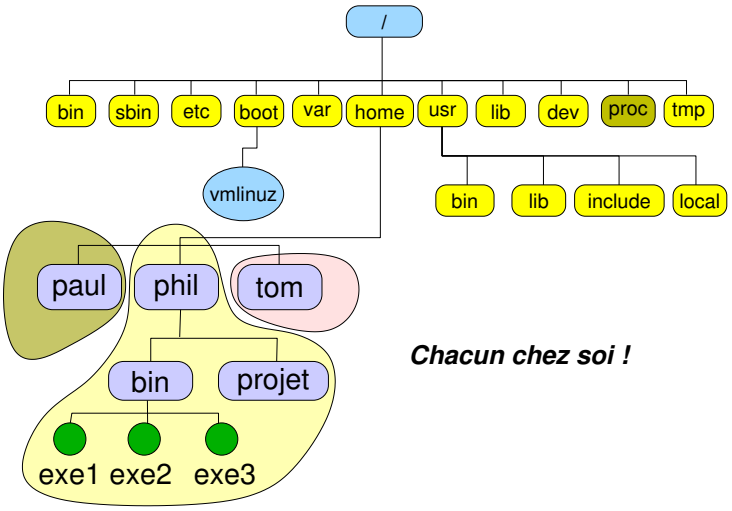
Les droits sur les répertoires 21/62

- Ce sont les mêmes types de droits que pour les fichiers : `r`, `w` et `x` et même `t`
- La sémantique associée est toutefois différente
- Droits :
  - `r` : le répertoire est lisible, on peut lister son contenu
  - `w` : le répertoire est «écrivable», on peut y créer des fichiers ou des répertoires
  - `x` : le répertoire n'est pas exécutable, il est accessible : on peut aller dedans, ou le traverser pour accéder à ce qu'il contient (fichiers, sous-répertoires)
  - `t` : le *sticky bit*, valable pour un répertoire ouvert en `w` à tout le monde, indique que seul un propriétaire de fichier peut supprimer ce fichier.

Si un répertoire est interdit de passage (i.e. le droit `x` est ôté), on ne peut pas s'y déplacer, on ne peut pas non plus descendre dans ses sous-répertoires, même si ceux-ci sont autorisés. Un droit `x` ôté est comme un obstacle infranchissable.

Si un répertoire à le droit `w` pour les utilisateurs non propriétaires, n'importe qui peut y créer un fichier, mais n'importe qui peut aussi supprimer ce fichier. Ce serait le cas pour les répertoires contenant les fichiers temporaires (`/tmp`, `/var/tmp`) si ceux-ci n'étaient pas munis du droit `t`.

Les fichiers et répertoires et les utilisateurs 22/62



Chacun chez soi... Et chacun maître chez soi. Si un utilisateur veut ouvrir son répertoire à tout le monde il en a le droit. Il a aussi le droit de fermer son répertoire à tous (même à lui

même, ce qui est embêtant pour lui sur le moment, mais il peut modifier à nouveau les droits pour se ré-autoriser...).

Les répertoires ouverts en lecture sont visitables pour autant qu'ils aient le droit `x` positionné. Les répertoires `/tmp`, `/var/tmp`, `/usr/tmp` sont ouverts à tous et tous peuvent y créer des fichiers et les supprimer. Généralement ces répertoires sont munis du *sticky bit* afin de restreindre le droit d'effacement aux seuls propriétaires des fichiers.

Il est donc possible de «se promener» dans la plus grande partie du système de fichiers. Seuls certains répertoires et fichiers sensibles sont interdits.

Les fichiers spéciaux 23/62

- Référencent des périphériques
  - Permettent les échanges (lectures/écritures) avec les pilotes des périphériques
  - Permettent le contrôle de ceux-ci
- Deux types
  - Les périphériques en mode bloc
    - les échanges se font par bloc d'octets (par «pages»)
  - Les périphériques en mode caractère appelé encore mode transparent (on dit plutôt mode *raw*)
    - les échanges se font octet par octet
  - Les disques sont plutôt en mode bloc, les terminaux en mode *raw*

Les fichiers spéciaux - le répertoire /dev 24/62

- Exemple d'entrée dans /dev

```
[bash]$ cd /dev; ls -l hda1
brw-rw---- 1 root  disk   3,  1  mar 24 2001  /dev/hda1
```

Indique le mode
b : bloc
c : caractère

Nombre majeur
indique le numéro du pilote (driver) dans le noyau

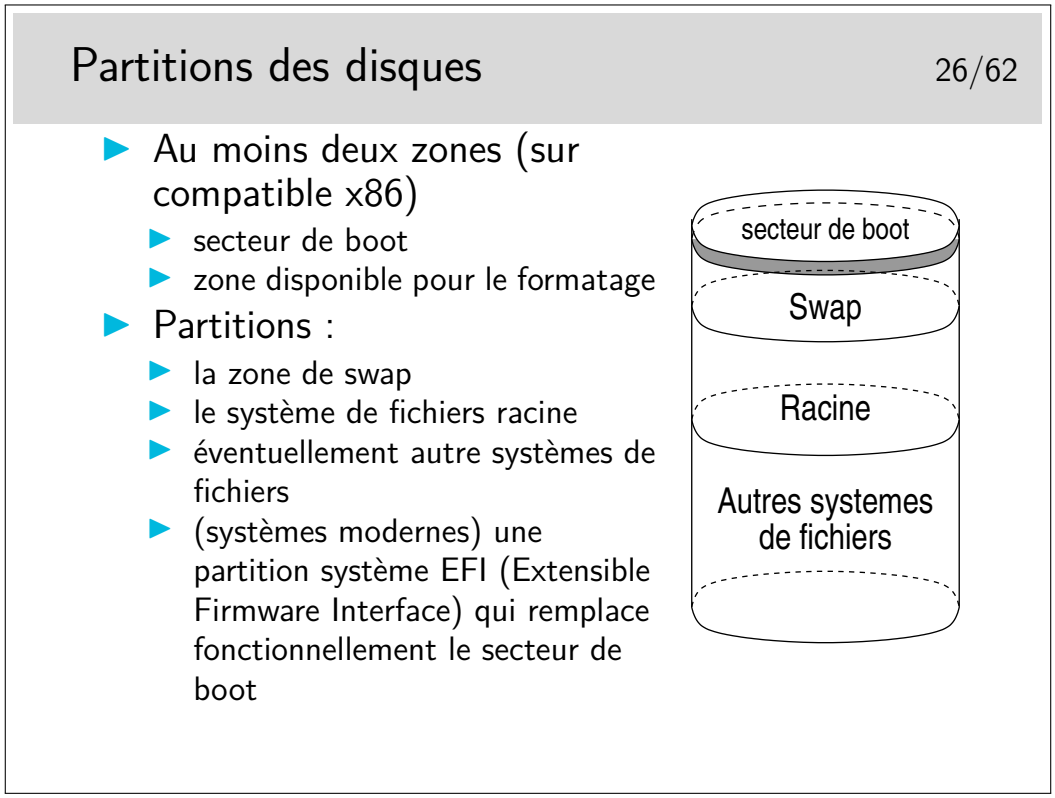
Nombre mineur
indique le numéro du périphérique piloté par le driver

Ces fichiers sont créés par la commande `mknod`, ils peuvent être créés n'importe où mais en

général on les trouve dans `/dev`.

Cette manière de voir les périphériques permet de considérer ceux-ci comme des fichiers (certes spéciaux, mais fichiers quand même), ainsi les échanges entre les applications et les périphériques se font essentiellement par des écritures et des lectures, comme pour des fichiers normaux.

2.2 Organisation sur disques



bootstrap : en français «amorçage», la séquence d'opérations à suivre pour démarrer le SE (OS).

zone de swap : en français «zone d'échange», permet de «décharger» la mémoire centrale si celle-ci est saturée. Les processus endormis (en attente d'un événement quelconque) sont transférés dans cette zone. Sa taille peut influencer le fonctionnement d'applications «gourmandes» en mémoire. Il est recommandé qu'elle soit au moins égale à la taille de la mémoire centrale.

On peut avoir plusieurs systèmes de fichiers de natures différentes : FAT, NT ou linux (ext2, ext3, etc...). Attention danger pour l'intégrité des partitions NT si autorisation en écriture.

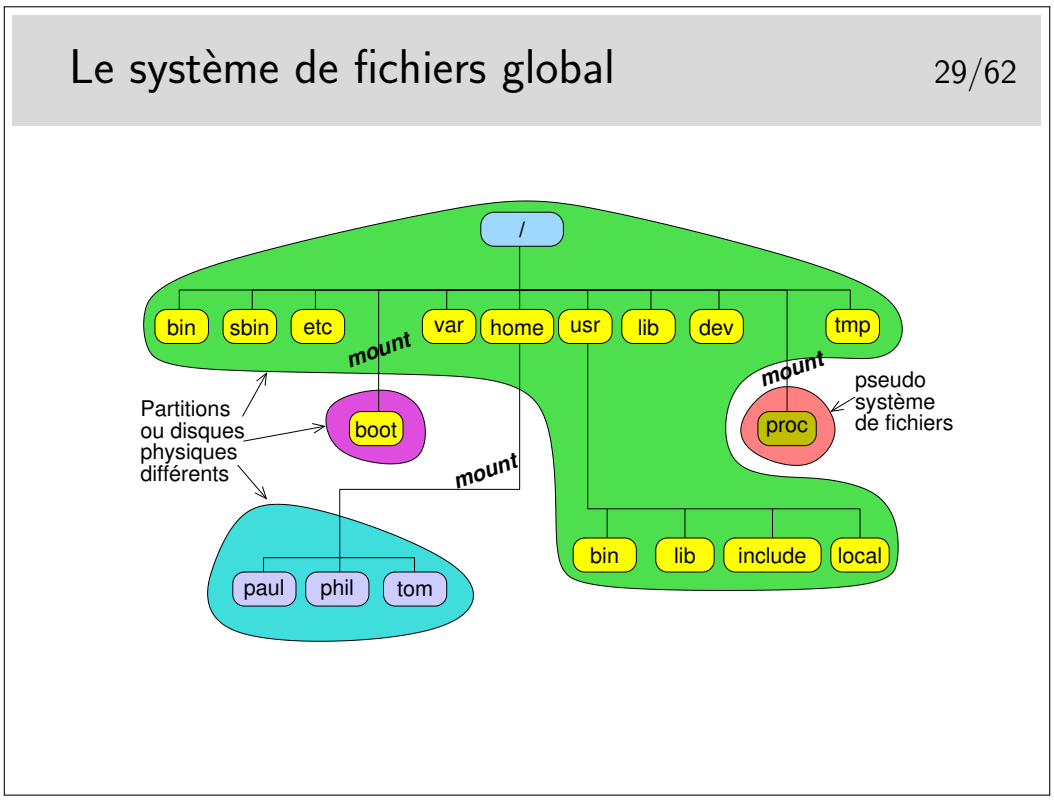
EFI : une evolution du BIOS (Basic input/output system), qui lui rajoute des fonctionnalites

Création d'un système de fichiers 27/62

- Commande `mke2fs` (*make extended 2 File System*)
  - Issue de la commande standard Unix `mkfs`
  - Crée un système de fichiers sur une partition de type Linux (type fixé au formatage du disque via `fdisk` par exemple)
  - Le système créé est de type `ext2`, ou `ext3` (s'il est journalisé), ou `ext4` (s'il est gros, jusqu'à 1024 péta-octets)
  - Permet de fixer la taille des blocs (par défaut 4Ko), le nombre d'inodes, etc.
    Tous ces paramètres peuvent être fournis par l'administrateur sinon leur valeur est estimée en fonction de la taille de la partition (nombre d'inodes par exemple)

Contrôle d'un système de fichiers 28/62

- Commande `fsck` ou similaire (`e2fsck`, `fsck.ext2`)
  - Vérifie la cohérence d'un système de fichiers Linux
    - Vérifie les super blocks, les inodes, les fichiers perdus (inode affecté mais pas de référence du fichier dans un répertoire). Ces fichiers sont recréés dans un répertoire appelle `lost+found` avec un nouveau nom autogénéré comportant le numéro de l'inode
  - Commande lancée automatiquement au boot. Elle vérifie d'abord si la partition a été correctement démontée lors de l'arrêt précédent, si oui, elle se termine, sauf si le nombre de montage maximum est atteint (voir transparent suivant)
  - Utilisable à tout instant, «à la main», sur un système de fichiers non monté de préférence ou monté en mode lecture seule (read only)



Le système de fichier global peut être constitué de plusieurs systèmes de fichiers de natures identiques ou différentes.

Dans le schéma ci-dessus on a choisi de créer 3 partitions formatées en système de fichier Linux (`ext2` ou `ext3`) :

- la partition racine avec `/bin`, `/etc`, `/var`, `/usr`, `/lib`, `/dev`, `/tmp`, ...
- une partition pour `/boot` (fait par défaut sous RedHat ou Debian par exemple)
- une partition pour les utilisateurs
- proc est un système virtuel de fichiers, utilisé par le noyau Linux. il n'existe pas sur disque mais est créé en mémoire vive.
- système de fichiers `/sys` avec le noyau 2.6, donne accès aux informations sur le système (les périphériques (devices) et leurs drivers).

L'utilitaire permettant de raccrocher un système de fichiers à l'arborescence globale est la commande `mount`.

Lorsqu'un utilisateur insère une clé USB, le gestionnaire de fichier utilisé (de façon cachée) `mount`. Le resultat : le volume apparaît alors comme par magie sur votre bureau.

3 Utilisation courante

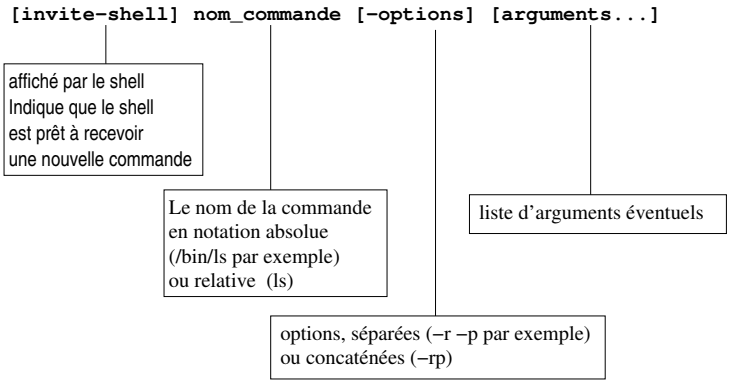
3.1 Les commandes et leur syntaxe

Comment lancer une commande 31/62

- Manière moderne
  - Via l'interface graphique
- Manière moins moderne... mais ô combien efficace !
  - Via un émulateur de terminal (xterm, gnome-terminal, ...)
  - Dans un terminal virtuel de console (accessible via `Ctrl-Alt-F1` à `F6` depuis l'interface graphique)
  - Ces méthodes lancent un processus interpréteur de commandes associé au terminal, appelé *Shell*
  - Le *Shell* est une interface entre le clavier et le système, il offre des facilités pour entrer les commandes mais aussi un vrai langage de programmation

Les commandes 32/62

- Syntaxe générale



Les commandes - La variable PATH 33/62

- Le PATH est une variable d'environnement
- Contient la liste des répertoires où se trouvent les commandes qu'on peut appeler par leur nom relatif le plus court : `ls` au lieu de `/bin/ls` par exemple
- Si le système vous dit : «*command not found*», il se peut que votre variable PATH soit mal configurée...
- Configuration dans le shell directement ou dans `$HOME/.bashrc` :
  - `export PATH=$PATH:/usr/local/bin` (par exemple)

Quand vous appelez une commande, le SE doit savoir ou la chercher. Pour cela, il va regarder la variable PATH.

Si on modifie la variable `PATH` directement dans le Shell par la commande `export` ci-dessus, la modification est prise en compte dans CE shell seulement... Pas dans le shell de la fenêtre terminal d'à côté.

Pour qu'une modification d'environnement soit permanente il suffit qu'elle soit écrite dans un fichier de configuration pris en compte systématiquement lors du lancement du Shell. Le fichier personnel `.bashrc` est le plus approprié pour le shell `bash`; et le fichier `.tcshrc` (ou `.tcsh.PERSO`) pour le shell `tcsh`.

Les commandes - Quelques raffinements 34/62

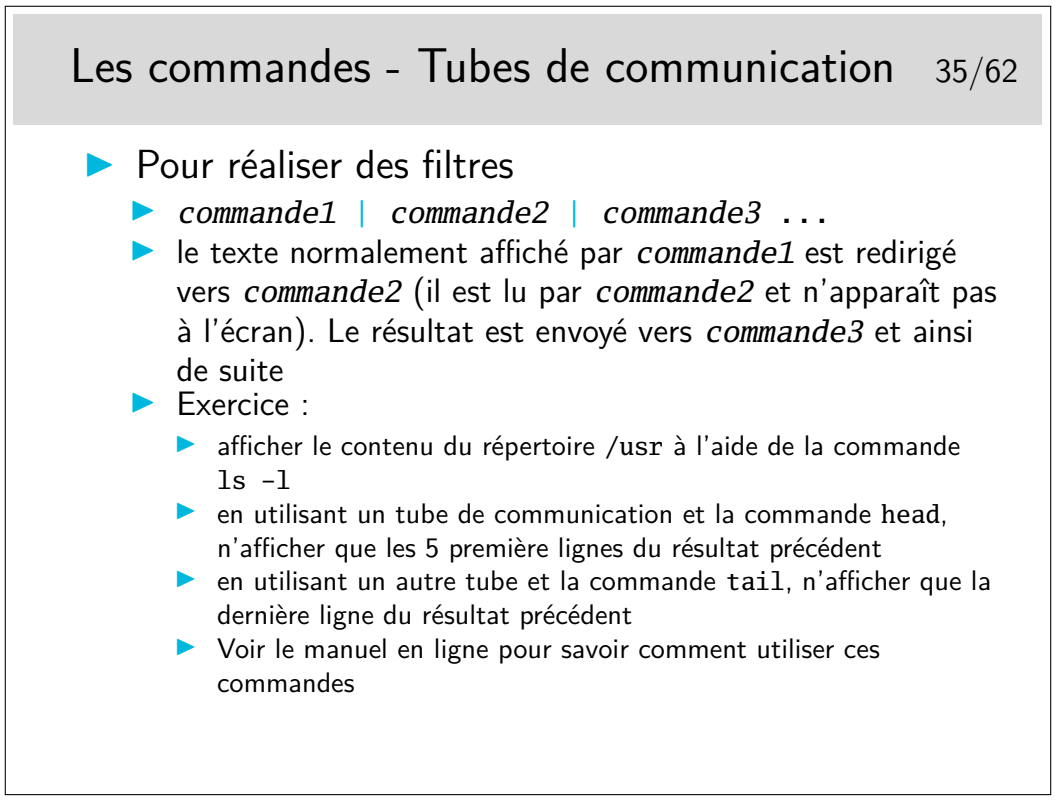
- Les commandes en arrière-plan (*background*)
  - la ligne de commande se termine par le caractère &
  - le shell lance la commande et redonne la main aussitôt sans attendre qu'elle se termine. La commande poursuit son exécution en arrière-plan
- Les redirections
  - redirection du fichier standard de sortie
    - *commande* > *fichier*
  - Redirection du fichier standard d'erreur
    - *commande* 2> *fichier*
  - redirection du fichier standard de sortie et d'erreur
    - *commande* > *fichier* 2>&1

To combine stderr and stdout into the stdout stream, we append this to a command :
2>&1

File descriptor 1 is the standard output (stdout). File descriptor 2 is the standard error (stderr).

At first, 2>1 may look like a good way to redirect stderr to stdout. However, it will actually be interpreted as "redirect stderr to a file named 1".

& indicates that what follows and precedes is a file descriptor, and not a filename. Thus, we use 2>&1. Consider >& to be a redirect merger operator.



C'est en fait la philosophie de base de l'utilisation d'Unix : une pléiade de petits utilitaires que l'on assemble au moyen de tubes pour réaliser une grande tâche. Le contraire des approches monolithiques.

Note : lors de l'exécution de `commande1 | commande2`, les deux programmes sont exécutés en parallèle. Sur d'autres systèmes mono-tâche comme MS-DOS, il y a d'abord l'exécution du premier programme en sauvegardant sa sortie dans un fichier temporaire, puis ensuite l'exécution du second programme en lisant le fichier temporaire...

Les commandes – Les jokers (*wildcards*) 36/62

- Un caractère spécial qui en remplace plusieurs...
  - Pour les noms de fichiers
  - Un genre d'expression rationnelle (mais non POSIX)
- Exemple : `rm *` (efface les fichiers du répertoire)
- Reconnus par le shell :
  - `?` : n'importe quel caractère
  - `*` : zéro ou plusieurs caractères
  - `[A1x]` : le caractère `A` ou `1` ou `x`
  - `[a-z]` : les caractère de `a` à `z` (code ASCII)
  - `{ab,ac}` : la chaîne `ab` ou `ac`

Les commandes Unix classiques 37/62

(liste non exhaustive...)

- Créer, naviguer parmi les fichiers et répertoires
  - `ls cd pwd cp mv rm mkdir`
- Afficher — éditer des fichiers
  - `more less` — `nano`
- Filtres texte
  - `echo cat grep sort uniq sed tail tee head cut tr split paste printf`
- Comparaison de fichiers
  - `comm cmp diff patch`

Les commandes Unix classiques 38/62

...

- Administration basique (niveau utilisateur)
  - `chmod chown ps su w who`
- Communication
  - `mail telnet ftp finger ssh`
- Shells
  - `sh csh ksh zsh bash tcsh fish`

Ce ne sont quelques commandes classiques que tout utilisateur d'Unix finit par connaître par coeur au bout de quelque temps de pratique...

Même si il est évidemment difficile de les connaître toutes dans le détail, il est bon de savoir qu'elles existent et de savoir retrouver leur documentation en temps utile.

Ces commandes sont rangées typiquement dans `/bin` et `/usr/bin`

`http://en.wikipedia.org/wiki/List_of_Unix_programs`

3.2 La documentation

Les commandes - La documentation 40/62

- La commande `man`... Savoir lire la syntaxe de man...
  - Par exemple : `man rm`

```
RM(1)        Manuel de l'utilisateur Linux    RM(1)
NOM
       rm - Effacer des fichiers.
SYNOPSIS
       rm  [-dfirvR] nom...
```

Nom de la commande

Liste des options
Entre [] signifie que ces opttions ne sont pas obligatoires.
(Si on les indique cependant, on ne mettra pas les crochets)

Le ou les arguments.
Ici les caractères ... indiquent que nom peut être répété plusieurs fois

Les pages du «manuel» Unix, en ligne (Sun fournissait un gros classeur du man imprimé...).

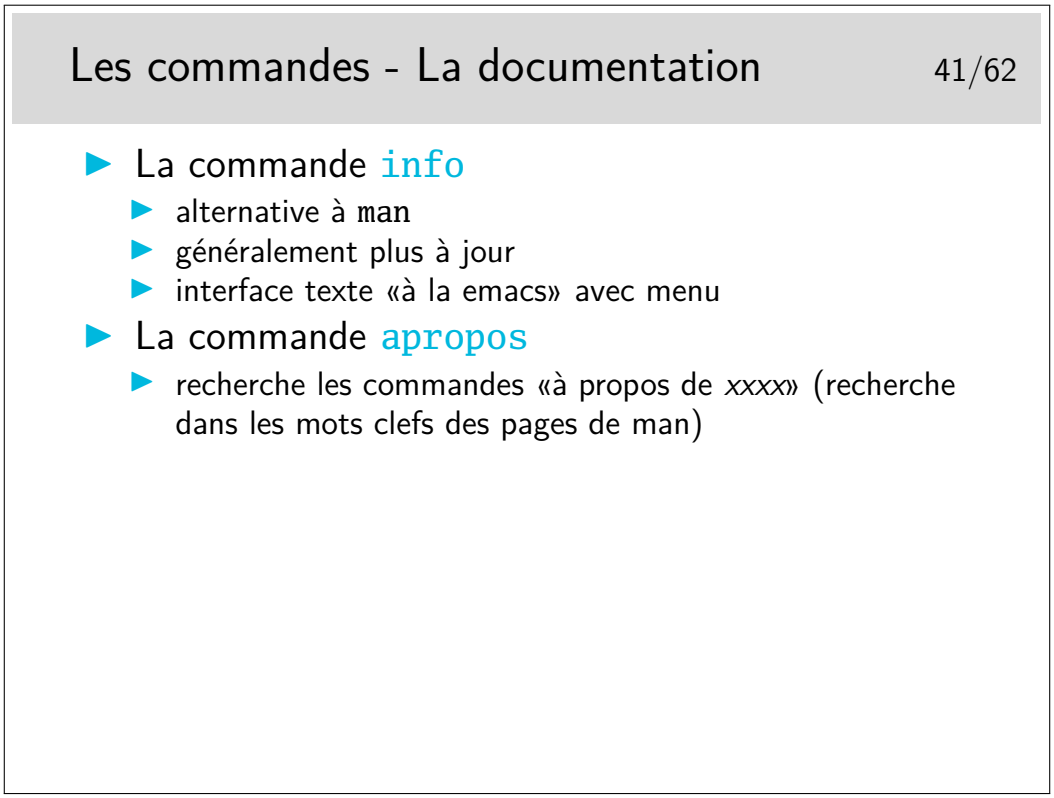
La source d'information de référence incontournable. Ne pas poser une question dans les forums de discussion dont la réponse est dans le man... sous peine d'être bien mal accueilli :-)

Attention : les pages du man ne sont pas des cours ni des tutoriels, mais du condensé d'information. Chaque mot est important. Il est parfois difficile de se plonger dedans, mais les pages de man sont toutes rédigées selon la même manière. relativement facile de s'y retrouver rapidement.

Le man est organisé en sections :

1. Commandes utilisateur
2. Appels système
3. Fonctions de bibliothèque
4. Fichiers spéciaux
5. Formats de fichier
6. Jeux
7. Divers
8. Administration système
9. Interface du noyau Linux

Chaque section possède une page d'introduction qui présente la section, disponible en tapant `man <num_section> intro`.



Note : les commandes *internes* au shell (p.ex. `cd pwd history` etc.) n'ont pas de page de `man` car ce ne sont pas des programmes, mais juste des mots clefs reconnus et interprétés directement par le shell. Ces commandes sont donc expliquées dans la documentation du shell lui-même. Par exemple, si vous cherchez de la documentation sur la commande `cd` et que vous avez *bash* comme shell, regardez dans le `man bash`. Une autre façon de faire est de taper `help cd` (En effet, `help` est une autre commande interne de *bash* qui affiche de la documentation sur les commandes internes de *bash*.)

Les outils de recherche de fichiers 42/62

- La commande `locate`
  - recherche les occurrences de la chaîne de caractères qui lui est passée en arguments dans une base de données mise à jour via la commande `updatedb`
- La commande `find`
  - Recherche n'importe quoi n'importe où EN TEMPS REEL (lent)
- La commande `whereis`
  - Recherche le nom de commande passé en argument dans un certain nombre de répertoires standards
- La commande `which`
  - Recherche dans le `PATH` où se trouve la commande indiquée en argument

4 Les processus

4.1 Environnement, cycle de vie

Les processus 44/62

- Le concept de processus
  - Un processus est un programme en cours d'exécution, dans un environnement donné, dans la mémoire centrale.
- Notion d'environnement
  - Ensemble d'informations complémentaires au programme en exécution qui viennent paramétrer l'exécution
  - Essentiellement trois types d'information d'environnement
    - Les variables d'environnement
    - L'identité de l'utilisateur au nom duquel s'exécute le processus
    - Les fichiers ouverts (notamment ceux d'entrée/sortie standard)

Création d'un processus 45/62

- Un processus est toujours créé par le noyau, à la demande d'un autre processus
- Le processus qui demande la création est appelé le père, le processus créé est appelé le fils
- Le processus fils est créé en mémoire centrale dans une zone mémoire distincte du processus père
- Le processus fils est la copie intégrale du processus père. Mais un détail technique (PID : Process IDentifier) permet au développeur de différencier les deux processus.

Le fils est un clone du père mais il y a une mutation génétique... Le père demande la création du fils via une fonction de bas niveau (un appel système). Cette fonction rend 0 dans le processus fils et une valeur strictement positive dans le processus père. Cette valeur dans le père est en fait le numéro de processus du fils. Le programmeur profite de cette différence pour différentier le code exécuté par les deux processus. Sinon les deux exécuteraient strictement la même chose.

Les variables d'environnement 46/62

- Chaînes de caractères, en majuscules par convention
  - Syntaxe `NOM=valeur`
  - Regroupées dans un espace mémoire appelé «tableau des variables d'environnement»
  - Ce tableau est hérité par les processus fils et résiste à l'exécution d'une commande (voir plus loin)
- Quelques variables standard
  - `PATH`, `HOME`, `USER`, `LOGNAME`, `DISPLAY`, ...

Identité utilisateur associé au processus 47/62

- Deux identités d'utilisateur !
  - Utilisateur réel (qui a lancé le processus), identifié par son numéro d'utilisateur dans `/etc/passwd` (*ruid : real user ID*)
  - Utilisateur effectif :
    - Dans la majorité des cas il s'agit de l'utilisateur réel
    - Si le fichier exécuté (qui a donné naissance au processus) a le bit *S_UID* positionné (une lettre s apparaît à la place du x des droits du propriétaire du fichier) alors l'utilisateur effectif est le propriétaire du fichier exécuté (*euid : effective user ID*)
    - Attention : trou de sécurité potentiel, surtout si le fichier appartient à root
- Deux identités de groupe
  - Concept identique à ci-dessus : groupe réel, groupe effectif (bit *S_GID*)

Pendant l'exécution d'un fichier ayant le bit *S_UID* positionné on a les droits du propriétaire du fichier, les droits d'un autre utilisateur, sans lui demander... Mais si le fichier exécuté possède ce droit c'est que son propriétaire l'y a mis, car lui seul peut le faire, ou l'administrateur, ou une application malicieuse...

Tout utilisateur peut avoir chez lui de tels fichiers pour des raisons qui lui sont propres et pour que des applications qui lui sont personnelles puissent fonctionner pour tout le monde. L'administrateur peut toutefois demander aux utilisateurs de justifier la présence de tels fichiers, c'est son droit et peut être même son devoir.

De tels fichiers existent et appartiennent à root, l'administrateur. S'ils sont vulnérables à des attaques de type débordement de tampon (buffer overflow) alors la sécurité du système entier est gravement compromise.

Les fichiers standards d'entrée-sortie 48/62

- Trois fichiers standards
  - Le fichier standard d'entrée (descripteur 0, FILE Pointer stdin)
  - Le fichier standard de sortie (descripteur 1, FILE Pointer stdout)
  - Le fichier standard de sortie d'erreur (descripteur 2, FILE Pointer stderr)
- Ouverts par défaut lors du lancement d'un exécutable
- Associés virtuellement au clavier pour l'entrée standard et à l'écran pour les deux autres
- Ils peuvent être redirigés vers des fichiers réels ou des tubes de communication

Contrôle sur les processus 49/62

- Lister les processus
  - La commande `ps`
    - Nombreuses options : `ax`, `axl`, `axf`, `-ef`, etc.
  - La commande `top`
    - Comme `ps axu`, avec réaffichage régulier, plus des informations sur la charge et l'occupation mémoire
- Arrêter un processus
  - Si on a le contrôle (processus en premier plan dans un terminal)
    - Sans le tuer (arrêt momentané) : `<Ctrl-Z>` (`fg` pour resumer)
    - En le supprimant : `<Ctrl-C>`
  - La commande `kill` (voir pages suivantes)

Sur certains systèmes les paramétrages des terminaux ou des émulateurs de terminaux sont tels que les associations de touches `<Ctrl-Z>` ou `<Ctrl-C>` ne fonctionnent pas. On peut le vérifier avec la commande `stty -a` qui affiche le paramétrage du terminal.

Exemple :

```
[bash]$ stty -a
speed 38400 baud; rows 25; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
```

```
lnext = ^V; flush = ^O; min = 1; time = 0;
...
```

Remarquer le paramètre `intr = ^C`, l'accent circonflexe indique la touche `<Ctrl>`. En fait, `intr` est le paramètre permettant de tuer rapidement un processus. Voir aussi `susp`. Et faire `man stty`.

On peut changer le paramétrage `intr` avec : `[bash]$ stty intr ^F`

Contrôle sur les processus : les signaux 50/62

- Un signal est une interruption logicielle envoyée à un processus par le noyau après un événement
- L'événement peut être :
  - Une faute logicielle (ex : division par 0, manipulation d'une adresse mémoire interdite, erreur d'alignement de donnée)
  - Terminaison d'un processus fils : par défaut (mais paramétrable) le père est prévenu
  - Intervention de l'utilisateur via le shell ou l'interface graphique pour tuer le processus ou le stopper ou autre (modification de la taille d'une fenêtre par exemple)
- Dans la plupart des cas le signal est fatal au processus

Les principaux signaux 51/62

Signal	Numéro	Fonction
HUP	1	Signal envoyé au processus en premier plan associé à un terminal lorsque celui-ci est fermé
INT	2	Envoyé depuis le clavier avec la combinaison de touches <CTRL-C> (par défaut)
QUIT	3	Envoyé depuis le clavier avec la combinaison de touches <CTRL-|> (par défaut)
KILL	9	Ne peut être intercepté, envoyé depuis le clavier via la commande `kill` (`kill -9` ou `kill -KILL`)
TERM	15	Envoyé via le clavier par la commande `kill` simple
SEGV		Erreur de segmentation, accès à une zone mémoire interdite
CLD		Terminaison d'un fils
WINCH		Modification de la taille de la fenêtre associée à l'application
STOP		Arrêt du processus sans le terminer. Envoyé via la combinaison de touches <CTRL-Z>
URG		Une données urgente a été reçue via le protocole TCP et est en attente de lecture (voir le cours sur la programmation réseau)
IO		Des données réseau sont arrivées et sont en attente de lecture. (voir le cours sur la programmation réseau)
USR1		Nom de signal utilisable par le développeur, à son gré
USR2		Nom de signal utilisable par le développeur, à son gré

Les raccourcis claviers (p.ex. `<CTRL-C>`) pour envoyer les signaux au processus en cours dans le shell sont paramétrés au niveau du terminal : `stty -a`.

La commande kill

52/62

- `kill` *[numéro_ou_nom_de_signal] numéro_processus | numéro_job*
- le numéro ou le nom de signal sera en général omis sauf si le résultat est négatif, auquel cas on pourra essayer le signal `KILL` (-9) qui ne peut pas être intercepté par le processus
- Le numéro de processus sera obtenu par `ps`
- Le numéro de job n'est valable que pour les processus en arrière plan (background) ou les processus stoppés. On peut le connaître avec la commande `jobs`
- Le programmeur de l'application peut gérer l'arrivée des signaux (sauf le signal 9) et les ignorer ou les traiter, pour que le processus se termine proprement ou ne se termine pas

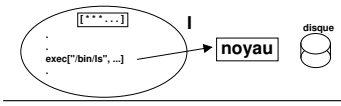
Les processus zombies

53/62

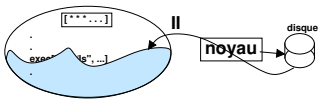
- Lorsqu'un processus fils se termine, son père doit acquitter la terminaison. C'est un problème de programmeur, pas d'utilisateur. Si l'acquittement n'est pas fait, le processus terminé reste dans la liste des processus (état `Z`), il est appelé «zombie»
- Un processus zombie est un processus fils pour lequel son père n'a pas acquitté la terminaison
- Le processus zombie est vidé de sa substance mais reste dans la liste des processus de la machine et peut être listé par `ps`
  - On ne peut plus le supprimer, il faut supprimer le père pour que le zombie disparaisse
  - Il est généralement dû à une erreur de programmation

En Shell (`sh`, `bash`, `ksh`) on peut récupérer le code de retour du exit du fils dans la variable `$?`.

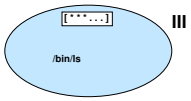
Exécution d'un fichier par un processus 54/62



Étape I : le processus appelle exec en indiquant un fichier exécutable en paramètre

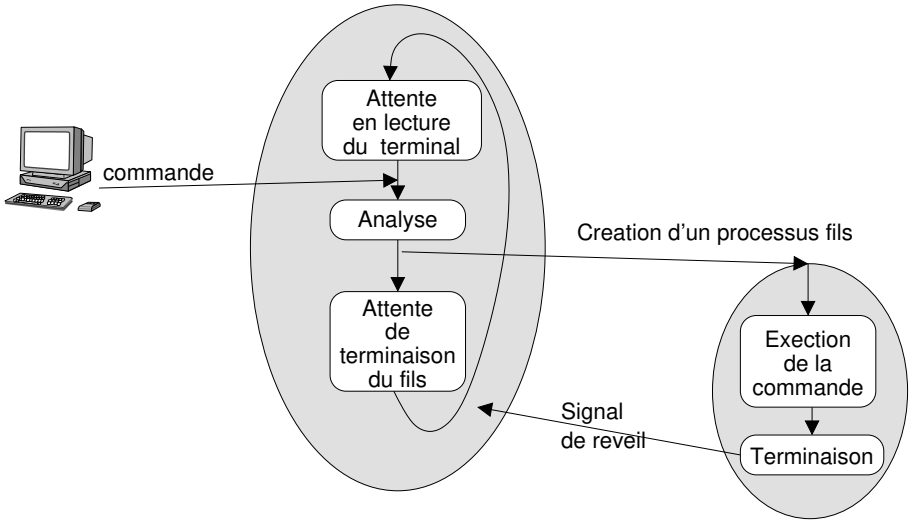


Étape II : le noyau va chercher le fichier sur le disque et le recopie à l'endroit où réside le processus. Le code original de celui-ci est écrasé et remplacé par le code du fichier



Étape III : le processus commence l'exécution de son nouveau code. Il ne peut y avoir retour à l'ancien code
On remarquera qu'un élément a résisté à l'envahisseur... Il s'agit du tableau des variables d'environnement qui n'a pas été altéré. Il est néanmoins possible de remplacer ce tableau par un nouveau.

Le principe du Shell 55/62



— Remplacer les méta caractères (*, ?, etc) par ce qu'il sont censés représenter (donc construire proprement la commande à exécuter) ;

— Vérifier si la commande est un alias, si oui, «désaliasser» ;

— Vérifier si la commande est une fonction interne du Shell (commande interne).

Si la commande correspond à un fichier exécutable, alors un processus fils est créé pour l'exécution. Le Shell ne reste en attente que si la ligne de commande entrée au terminal ne se termine pas par un caractère «&», auquel cas le shell revient sans attendre et renvoie l'invite à l'écran.

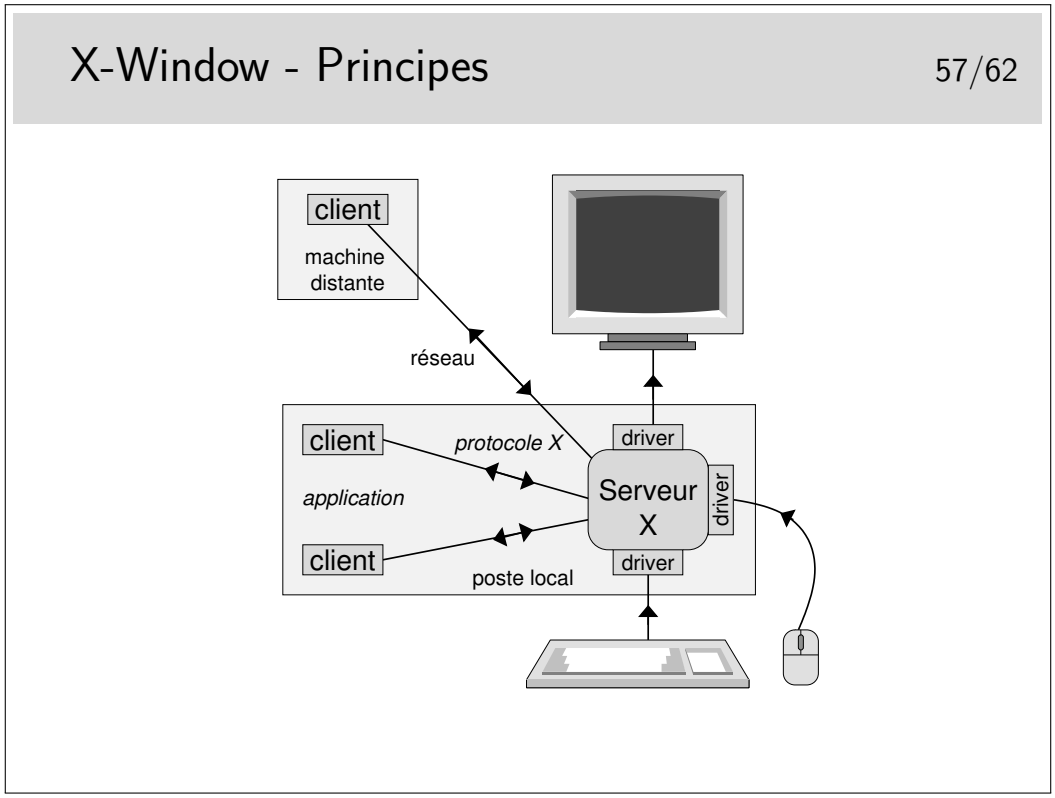
Dans ce dernier cas, il est néanmoins capable de gérer proprement la terminaison de son fils afin que ce dernier ne reste pas zombie.

5 L'interface graphique X-Window

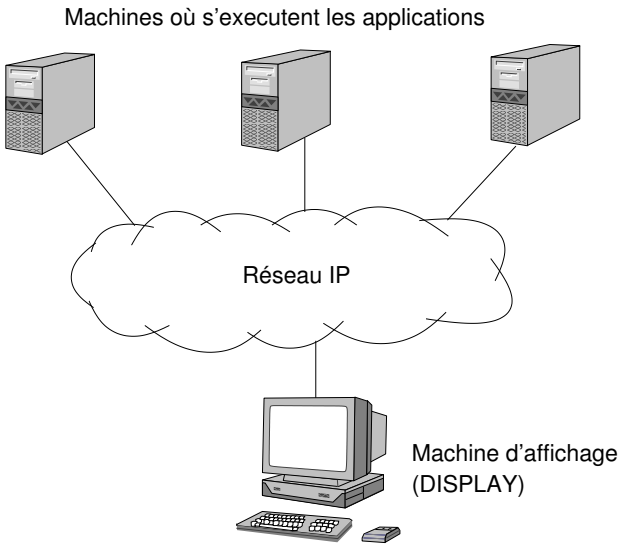
Le principe d'une *interface graphique* émerge à partir des années 80, du fait de l'évolution du matériel, avec différents concepts et architectures logicielles (client-serveur, fenêtres, multi-application, etc.). (Voir `https://en.wikipedia.org/wiki/History_of_the_graphical_user_interface`)

Dans le monde Unix, on utilise un système graphique client-serveur avec une couche d'adaptation X-Window.

5.1 Client-serveur, authentification, bureau



X-Window - Intrinsèquement Réseau



X-Window - Applications en réseau

- Toute application X peut s'exécuter sur une machine et s'afficher sur une autre
- La machine d'affichage est indiquée dans une option (`-display`) ou une variable d'environnement (`DISPLAY`)
  - Exemples :
  - `$ xterm -display lune:0.0`
  - `$ export DISPLAY=lune:0.0`
  - `$ xterm`
  - format du `display` : *adresseOuNomMachine*:*numServ*.*numEcran*

X-Window - Réseau et sécurité 60/62

- Par défaut il n'est pas possible d'afficher des fenêtres sur un «display» utilisé par un autre utilisateur si celui-ci n'a pas donné l'autorisation
- Les autorisations sont possibles avec la commande `xhost`
  - `xhost +terre`
  - autorise les «connexion graphiques» depuis la machine `terre`
  - ⊕ authentification par adresse IP
- Autorisations avec la commande `xauth`
  - plus complexe mais plus *sûr*
  - fichier `.Xauthority`
  - ⊕ autentifie un utilisateur (qui doit posséder le bon *cookie* de 128 bits)

X-Window - Le serveur et les applications 61/62

- Le serveur X sait gérer l'affichage mais il ne sait pas quoi ni comment afficher !
  - ce sont les applications qui l'informent via le protocole X
  - des librairies applicatives pour dessiner boutons, menu, assenceurs, etc. (*graphic toolkits*)
- Le serveur X reçoit tous les événements clavier et souris
  - il informe alors les applications de l'événement si celles-ci ont demandé qu'il leur soit envoyé
  - les applications traitent l'événement et décident de ce qu'il faut faire, elles demandent éventuellement des modifications d'affichage au serveur

Une documentation (particulièrement pédagogique et ludique) sur le fonctionnement interne de X-Windows : `https://magcius.github.io/xplain/article/`

X-Window - Une application très particulière

- Le gestionnaire de fenêtre ou *Window Manager*
  - le serveur X **ne sait pas gérer** les fenêtres
  - application spécifique nécessaire : **Window Manager**
    - permet de déplacer, iconifier, restaurer, supprimer les fenêtres et aussi modifier leur taille
    - offre des menus de fond d'écran
  - Les Window Managers existent en grand nombre
  - Les applications sont normalement compatibles avec tous les Window Managers (vœu pieux !)