1 Objectifs du cours

Ce cours se veut très pragmatique : il s'agit *en pratique* d'être capable de gérer, en tant que *super-user*, un système informatique en préservant ses utilisateurs des problèmes courants et en protégeant raisonnablement le système des tentatives de piratage. À l'issue du cours, on saura :
- choisir son matériel en fonction des fonctionnalités attendues du système informatique,
- installer un système *ex nihilo*, qui peut être une machine individuelle sécurisée tout comme un serveur sur le réseau, avec un bon niveau de sécurité (système de type Unix/Linux),
- le maintenir (configuration personnalisée, réalisation d'utilitaires dédiés à des tâches routinières, sauvegardes, conseils aux utilisateurs, *etc.*),
- on saura aussi (et surtout) comment chercher et trouver les informations utiles pour réussir une nouvelle tâche de gestion de la machine, et par conséquent augmenter progressivement ses compétences tout en gérant un système sans mettre en péril la sécurité des données et le confort des utilisateurs.

Nota : le poly contient aussi des annexes qui récapitulent les commandes et les fichiers importants vus durant le cours.

2 Les 3 concepts fondamentaux d'un système

Un système informatique bien structuré met en œuvre trois concepts clefs :
- les *utilisateurs*, qui identifient de manière non ambiguë les acteurs qui peuvent utiliser le système, leurs droits et ce que le système est autorisé à faire en leur nom,
- les *fichiers*, qui sauvegardent de manière fiable à la fois les données, les programmes qui les manipulent et les paramètres qui déterminent les comportements du système et de ses services,
- enfin les *processus*, qui sont les programmes en train de « tourner » sur la machine à un moment donné, services généraux du système et « bras agissants » des utilisateurs du système.

Un utilisateur virtuel particulier, appelé `root`, a tous les droits sur le système... et ce cours vous apprend à être un `root` compétent. Certains fichiers *appartiennent* à `root` ; ce sont typiquement ceux qui gèrent la liste des utilisateurs du système et leurs droits, ceux qui contrôlent le comportement global du système, *etc.* Par exemple `/etc/passwd`.

Un fichier appartient toujours à un utilisateur du système. De même un processus appartient et agit toujours au nom d'un utilisateur du système et l'utilisateur est *responsable* de toutes les actions de ses processus. À l'exception de `root`, seul l'utilisateur propriétaire d'un fichier ou d'un processus peut lui appliquer toutes les modifications qu'il souhaite. Les processus appartenant à un utilisateur disposent eux aussi de tous les droits de l'utilisateur ; ils sont en fait ses « bras agissant » au sein du système.

Le système repose donc sur des « boucles de rétroactions » systématiques entre fichiers et processus : les fichiers dits *exécutables* peuvent être chargés en mémoire pour lancer les processus... et les processus agissent entre autres en créant ou modifiant les fichiers du système, ou en lançant d'autres fichiers exécutables... tout cela sous contrôle des droits utilisateurs dont ils disposent.

> Au démarrage du système, il est donc nécessaire de lancer un premier processus, qui en lancera ensuite un certain nombre d'autres (dont l'écran d'accueil pour les « login », la surveillance du réseau, *etc*) ; eux-mêmes lanceront d'autres processus et ainsi de suite. Le premier processus lancé au boot s'appelle `init` et il tourne ensuite jusqu'à l'arrêt complet du système (par exemple il relance un écran d'accueil chaque fois qu'un utilisateur se « délogue », il gère les demandes d'arrêt du système, *etc*). Le processus `init` agit au nom de `root`, naturellement.

« Booter » une machine signifie donc en fait lancer `init`, le laisser lancer d'autres processus, qui en lanceront eux-mêmes d'autres... jusqu'à l'arrivée d'un processus en mesure de présenter un écran de login.

3 Le concept d'utilisateur

Le système identifie les utilisateurs autorisés par leur *identifiant* qui est simplement un nombre entier positif : l'UID (User IDentification). C'est la combinaison du nom de login et du mot de passe associé qui permet au système de « reconnaître » un utilisateur autorisé et de lui délivrer l'accès à son UID.

Lorsque vous êtes connectés au système, vous pouvez connaître votre UID avec la commande « `echo $UID` ». L'UID de `root` est 0.

Les utilisateurs sont structurés en *groupes* : chaque utilisateur appartient à un groupe par défaut (souvent le nom de l'équipe ou du service où il travaille). De plus, s'il est autorisé à rejoindre d'autres groupes, il peut le faire en utilisant la commande `newgrp` (à condition de connaître le mot de passe du groupe ou d'y être admis dans le fichier `/etc/group`). Chaque groupe est identifié par un GID.

L'utilisateur `root` appartient au groupe également appelé `root` (GID=0) et certains autres utilisateurs virtuels peuvent appartenir au groupe `root` (ils servent à gérer le système sans prendre le risque de lancer des processus ayant tous les droits de `root`). Il y a également un groupe `wheel` (de GID non nul) qui joue un rôle similaire pour gouverner certaines parties du système.

Un utilisateur possède un répertoire où le système le place par défaut au moment du « login », sauf exception l'utilisateur a tous les droits sur ce répertoire (créer des sous-répertoires, y placer des fichiers de son choix, *etc*). Il s'agit de « *la home directory* » de l'utilisateur.

Lorsque vous êtes connectés au système, vous pouvez connaître votre home directory avec la commande « `echo $HOME` ». C'est généralement quelquechose du genre `/home/monEquipe/monLoginName`.

Enfin un utilisateur possède un processus d'accueil : c'est le processus qui est lancé par le système au moment où il se connecte avec succès (nom de login et mot de passe reconnus). C'est aussi celui qui est lancé lorsque l'utilisateur ouvre une fenêtre de « terminal ». Ce genre de processus est appelé *un shell*. Il s'agit le plus souvent de `/bin/bash` et ce processus attend tout simplement que l'utilisateur tape une commande (donc un nom de fichier) pour l'exécuter. Lorsque l'utilisateur bénéficie d'un environnement graphique (c'est le cas en général), c'est simplement que le shell lance immédiatement les processus de l'environnement graphique au lieu d'attendre que l'utilisateur tape une commande.

Certains utilisateurs virtuels ont un processus d'accueil plus restreint que `bash`, par exemple l'utilisateur `shutdown`, qui appartient au groupe `root`, a pour « login shell » un processus qui se contente d'éteindre l'ordinateur (en le demandant à `init` qui va alors « tuer » tous les processus qu'il a lancés).

Le fichier `/etc/passwd` contient la liste des utilisateurs reconnus par le système (dont `root` lui-même), à raison de un par ligne, et il contient les informations suivantes : nom de login, mot de passe (encrypté ! et souvent déporté vers `/etc/shadow`), UID, GID (celui du groupe par défaut), vrai nom, home directory et shell de login.

Les utilisateurs virtuels : on constate que `/etc/passwd` définit aussi des utilisateurs qui ne sont pas attachés à des personnes. C'est le cas de `root` bien sûr mais aussi par exemple `ntp` (net time protocol, pour mettre à l'heure l'ordinateur), `smtp` (send mail transfer protocol, pour distribuer le courrier électronique), `lpd` ou `cups` (line printer deamon ou common unix printing system, pour gérer les files d'attende des l'imprimantes), `nfs` (network file system, pour mettre à disposition *via* l'intranet une partition d'un serveur), *etc.*

Si l'ordinateur est connecté au réseau (c'est généralement le cas), alors le mot de passe encrypté est remplacé par un « `x` » dans `/etc/passwd` et les encryptations sont déportées dans `/etc/shadow`. Par ailleurs, si les utilisateurs doivent pouvoir utiliser plusieurs ou toutes les machines d'un réseau donné, alors seulement l'un des ordinateurs contient ces informations et il les met à disposition des autres ordinateurs du réseau *via* un protocole appelé NIS (Network Information Service). Les ordinateurs « clients » de NIS sur le réseau y trouvent alors la liste des utilisateurs qui ne sont pas virtuels. [1]

4 Le concept de fichier

Les fichiers sont les lieux de mémorisation durable des données et des programmes de tout le système. Ils sont généralement placés physiquement sur un *disque dur* ou sur tout autre dispositif de grande capacité qui ne perd pas ses données lorsque le courant est éteint : SSD (solid-state drive), mémoire flash (clef USB), DVD, CDROM, *etc.*

4.1 Les types de fichier

Les fichiers ne sont pas seulement ceux qui mémorisent des textes, des images, de la musique ou des vidéos :
- Il y a des fichiers dont le rôle est de pointer sur une liste d'autres fichiers, on les appelle des *répertoires* ou *directories* (ou encore « dossiers » pour les utilisateurs naïfs). Oui, les répertoires sont des fichiers comme les autres !
- Il y a des fichiers dont le rôle est de rediriger vers un seul autre fichier, on les appelle des *liens symboliques* (ou encore « raccourcis » pour ces mêmes utilisateurs naïfs).

1. L'ancien nom de NIS est YP (Yellow Pages) et vous trouverez parfois cette ancienne terminologie... devenue illégale à la suite d'un procès gagné par British Telecom...

— Il y a des fichiers dont le rôle est de transmettre vers un matériel périphérique ce qu'on écrit dedans, et également de transmettre (en lecture cette fois) les informations transmises par le périphérique. Ces fichiers sont appelés des « *devices* » ou encore des « fichiers de caractères spéciaux ».
— *etc.*

Les fichers auxquels les gens pensent habituellement, c'est-à-dire ceux qui ont un contenu avec des données, sont souvent appelés des fichiers *plats* par opposition aux fichiers répertoires, liens symboliques et autres.

La commande `file` permet de savoir quel est le type d'un fichier. Le nom de fichier est souvent choisi pour indiquer son type *via* sont suffixe (par exemple « `.txt` » pour du texte, « `.mp3` » pour du son, ...) mais cette indication n'est pas fiable puisqu'on peut sans problème renommer un fichier à sa guise. La commande `file`, en revanche, analyse le début du contenu du fichier qu'on lui donne en argument pour déterminer son type.

```
$ file /home
/home: directory
$ file /etc/passwd
/etc/passwd: ASCII text
$ file /dev/audio
/dev/audio: character special
```

Pour obtenir des informations plus précises à propos du contenu d'un fichier plat, on peut également utiliser la commande `mimetype`. Les informations obtenues sont assez précises pour qu'une interface graphique sache quel logiciel il faut lancer pour l'exploiter lorsqu'on clique dessus, mais *attention*, `mimetype` « croit » le suffixe du fichier...

Plusieurs informations sont attachées aux fichiers. On en verra plusieurs dans cette section (propriétaire du fichier, droits d'accès, *etc*) et l'on peut mentionner également deux informations souvent utiles :
— la *date de dernière modification*, qui est la date exacte (à quelques fractions de secondes près) où le fichier a vu son contenu être modifié pour la dernière fois (ou créé),
— la *date de dernière utilisation*, qui est la date où les données du fichier ont été lues (ou utilisées de quelque façon) pour la dernière fois.

A ce propos, on peut mentionner la commande `touch` qui positionne ces date à l'instant exact où elle est exécutée. Plus précisément, si *`fichier`* n'existe pas alors la commande « `touch` *`fichier`* » crée un fichier plat vide dont les dates de dernière modification et de dernière utilisation sont l'instant présent, et s'il préexistait, elle ne modifie pas le fichier mais positionne néanmoins ces deux dates à l'instant présent.

4.2 L'arborescence du système de fichiers

Les fichiers n'ont pas de nom !
Parler du « nom d'un fichier » est en réalité un abus de langage (que nous faisons tous) ; *explication* :
Les fichiers, quel que soit leur type, sont identifiés par un numéro qui indique leur emplacement sur le disque dur (ou autre support). Il s'agit du *numéro de i-node*. Le système de gestion de fichiers gère un *arbre* dont la racine est un fichier de type répertoire qui appartient à `root`. Chaque répertoire est en fait un ensemble de *liens* (des « vrais » liens, pas des liens symboliques) vers les fichiers fils de ce répertoire. Ce sont ces liens qui portent des noms, pas les fichiers eux-mêmes (Figure 1). C'est le nom du lien qui pointe vers un fichier que l'on appelle abusivement le nom de ce fichier (que le fichier soit plat, un répertoire ou de tout autre type).

Un fichier (quel que soit son type) est donc caractérisé par son numéro de i-node. Il peut être, comme on le voit, également désigné sans ambiguité par une *adresse* qui est le chemin suivi depuis la racine de l'arbre jusqu'à lui. Les noms portés par les liens des répertoires sont séparés par des « / » (exemple : `/home/GB4/paul`). Par conséquent, un nom n'a pas le droit de contenir de « / ».

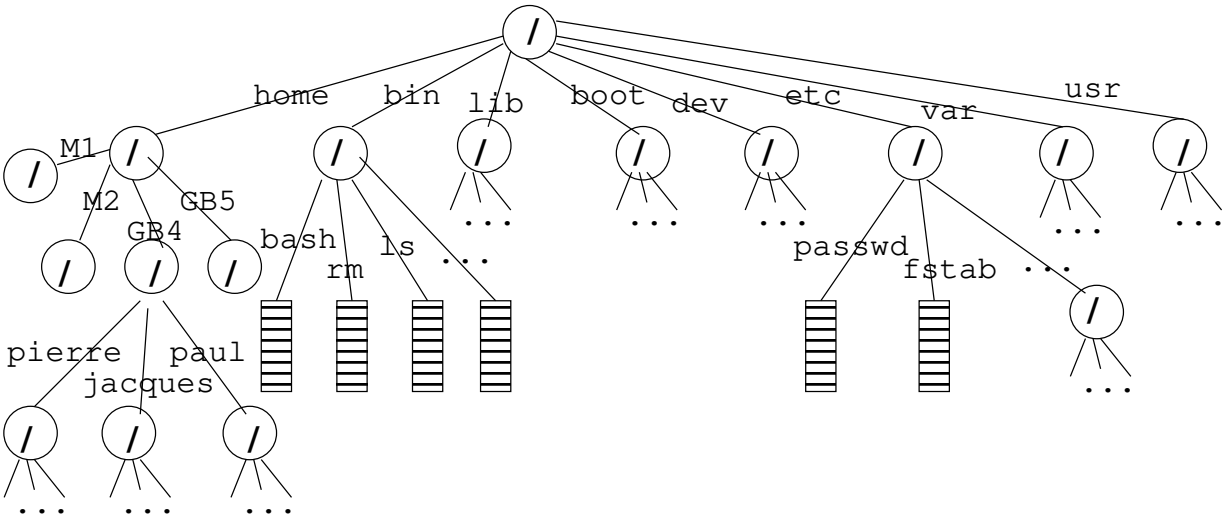
> La racine de l'arbre a simplement `/` pour adresse.

Lorsque l'on « déplace un fichier », avec le gestionnaire de fichier ou avec la commande « `mv` *`ancien nouveau`* », on ne déplace rien en réalité : le fichier reste à sa place sur le disque dur et ne change pas de numéro de inode ; ce qui change, c'est l'arborescence, donc seulement les contenus des fichiers de type répertoire qui la constituent.

> En particulier, la commande `mv` prend un temps très court quelle que soit la taille du fichier. Il y a une exception lorsque le déplacement impose un changement de support physique (e.g. d'une clef USB à un disque dur).

Un répertoire n'est jamais vide car il contient toujours au moins deux liens fils :
— le lien « `.` » qui pointe sur lui-même
— et le lien « `..` » qui pointe sur le répertoire père dans l'arbre. Exception : dans le répertoire `/` les liens `.` et `..` pointent tous deux sur lui-même.

Figure 1 – Arborescence UNIX



Lorsqu'un processus tourne, il possède un *répertoire de travail*. Ceci évite de répéter tout le chemin depuis la racine car l'expérience montre que la plupart des processus utilisent ou modifient des fichiers qui sont presque tous dans le même répertoire. De cette façon, un processus (dont le shell) peut définir une adresse de fichier de deux façons :

— par son *adresse absolue*, c'est-à-dire le chemin depuis la racine, donc une adresse qui commence par un « / »

— ou par son *adresse relative*, c'est-à-dire le chemin depuis le répertoire de travail du processus ; une adresse relative ne commence jamais par un « / ».

C'est donc par son premier caractère qu'on distingue une adresse relative d'une adresse absolue.

Lorsque vous êtes sous un shell, la commande `pwd` (*print working directory*) vous fournit le répertoire de travail dans lequel vous êtes (c'est-à-dire le répertoire de travail de votre processus shell courant). La commande `cd` (*change directory*) vous permet de changer de répertoire de travail. Enfin la commande `ls` (*list*) fournit la liste des liens « fils » d'un répertoire (par défaut son répertoire de travail si on ne lui donne aucun argument).

```
$ pwd
/home/bioinfo/bernot

$ cd bin

$ pwd
/home/bioinfo/bernot/bin

$ ls
RNAplot  hsim  wi  wx  wxd

$ ls -a
. .. RNAplot  hsim  wi  wx  wxd

$ ls /usr
X11R6  etc    include  lib64    local  sbin   src  uclibc
bin    games  lib      libexec  man    share  tmp

$ ls /etc/X11/ /etc/rc.d
/etc/X11/:
app-defaults  proxymngr    xinit.d      Xmodmap     prefdm
fontpath.d    wmsession.d  xorg.conf.d  Xresources  xorg.conf
gdm           xdm          xsetup.d     Xsession
lbxproxy      xinit        X            lookupdm

/etc/rc.d:
```

```
init.d  rc1.d  rc3.d  rc5.d  rc7.d           rc.local
rc0.d   rc2.d  rc4.d  rc6.d  rc.alsa_default  rcS.d
```

La plupart des commandes Unix admettent des *options* qui modifient leur comportement par défaut. Par exemple, le comportement par défaut de `ls` est de donner la liste des fils du répertoire courant en ignorant tous ceux dont le nom commence par un « . » mais l'option « `-a` » de `ls` modifie ce comportement en montrant tous les fils du répertoire.

Les commandes Unix peuvent également recevoir des arguments « standard » qui ne commencent pas par un « - » ; par exemple si l'on indique un ou plusieurs noms de fichiers à `ls`, il fait son travail sur ces fichiers au lieu du répertoire courant.

> *Devinette : c'est généralement une mauvaise idée d'avoir un nom de fichier qui contient un espace ou qui commence par un tiret ; pourquoi ?*

Lorsqu'un processus en lance un autre, il lui transmet son propre répertoire de travail[2].

4.3 Les droits d'accès

Outre son numéro de inode qui le caractérise, tout fichier quel que soit son type a :
- un *propriétaire*, qui est un utilisateur reconnu du système (donc un UID),
- et un *groupe* (donc un GID), qui n'est pas nécessairement celui par défaut de son propriétaire, ni même obligatoirement un groupe auquel son propriétaire a accès... mais en pratique presque tous ont le groupe par défaut de leur propriétaire,
- enfin les « autres » utilisateurs, c'est-à-dire ceux qui ne sont ni propriétaire ni membre du groupe du fichier, peuvent le cas échéant avoir le droit d'utiliser le fichier, si le propriétaire du fichier l'a autorisé.

Tout fichier peut être *lu* ou *écrit/modifié* ou encore *exécuté*. Pour un fichier de type répertoire, le droit de lecture signifie l'autorisation de faire `ls`, le droit d'écriture signifie l'autorisation de modifier la liste des fils (création, suppression, renommage) et le droit d'exécution signifie l'autorisation d'y faire un `cd`. Pour un fichier plat, le droit d'exécution signifie généralement soit qu'il s'agit de code machine qui peut directement être chargé pour lancer un processus, soit qu'il s'agit des sources d'un programme (en python, en shell ou tout autre langage) qui peut être interprété par un programme *ad hoc* pour lancer un processus.

Le propriétaire peut attribuer les droits qu'il veut aux 3 types d'utilisateurs du fichier (lui-même, le groupe et les autres). Il peut même s'interdire des droits (utile s'il craint de faire une fausse manœuvre). Il y a donc 9 droits à préciser pour chaque fichier :

	owner	group	others
read	4	4	4
write	2	2	2
exec	1	1	1
SUM	...	...	...

et l'encodage se fait par puissances de 2 de sorte que la somme de chaque colonne détermine les droits de chacun des trois types d'utilisateur. La commande pour modifier les droits d'un fichier est `chmod` ; par exemple :
- `chmod 640` ***`adresseDeMonFichier`*** donne les droits de lecture et écriture au propriétaire, le droit de lecture aux membres du groupe du fichier, et aucun droit aux autres.
- `chmod 551` ***`adresseDeMonFichier`*** donne les droits de lecture et exécution au propriétaire et au groupe, et seulement le droit d'exécution aux autres.
- `chmod 466` ***`adresseDeMonFichier`*** donne le droit de lecture au propriétaire, mais les droits de lecture et écriture aux membres du groupe ainsi qu'aux autres...

Le nombre à trois chiffre est appelé le *mode* du fichier (et le `ch` de `chmod` est pour « change »). Naturellement seuls le propriétaire d'un fichier (et `root` bien sûr, puisqu'il a tous les droits) peuvent effectuer un `chmod` sur un fichier.

L'option « `-l` » (=format long) de `ls` permet de connaître les droit d'accès d'un fichier :

```
$ chmod 640 toto
$ ls -l toto
-rw-r----- 1 bernot bioinfo 5 avril 25 08:13 toto
$ chmod 755 toto
$ ls -l toto
-rwxr-xr-x 1 bernot bioinfo 5 avril 25 08:13 toto
```

Le résultat de `ls -l` fournit les informations suivantes :

2. mais ce processus fils peut tout à fait décider de commencer par changer de répertoire de travail !

— Le premier caractère indique le type du fichier : un tiret pour un fichier plat (comme dans l'exemple), un `d` pour un répertoire[3] (`d`irectory), un `l` pour un lien symbolique, un `c` pour un device (`c`aractère spécial), *etc.*
— Les 3 caractères suivants indiquent les droit du propriétaire du fichier dans l'ordre `r`ead, `w`rite, `e`xecute : un tiret indique que le droit n'est pas donné, sinon la lettre correspondante apparaît (voir les deux exemples).
— Les 3 caractères suivants indiquent les droit des membres du groupe du fichier selon le même principe.
— Les 3 caractères suivants indiquent les droit des autres utilisateurs, toujours selon le même principe.
— Ensuite vient le nombre de liens qui pointent sur le i-node du fichier. La plupart du temps c'est 1 pour les fichiers qui ne sont pas des répertoires, et de 1 + le nombre de répertoires fils pour un répertoire (il s'agit du lien « standard » sur le répertoire, du lien `.` et du lien `..` de chaque répertoire fils).
— Suivent le nom du propriétaire et le nom du groupe du fichier.
— Puis la taille du fichier (nombre d'octets, 5 dans l'exemple).
— Puis la date et l'heure de dernière modification du fichier (ou la date et l'année si plus d'un an).
— Et enfin le nom du fichier.

Seul `root` peut changer le propriétaire d'un fichier, en utilisant la commande `chown` (change owner) :

```
# whoami
root
# chown untel ceFichier
# chown untel:ungroupe cetAutreFichier
```

(Sous la seconde forme, `chown` permet de modifier aussi le groupe du fichier).

Lorsqu'un utilisateur a le droit d'exécuter un fichier plat, le processus qu'il lance ainsi appartient à l'utilisateur qui l'a lancé, c'est-à-dire qu'***il agit en son nom*** et non pas au nom de l'utilisateur qui possède le fichier. ***Cela suppose donc d'avoir pleinement confiance en l'honnêteté du propriétaire de ce fichier...***

> En fait, le mode d'un fichier contient un quatrième entier rarement utilisé. Il permet au propriétaire du fichier de basculer à vrai trois propriétés dangereuses avec le code suivant : 1 pour un « sticky bit » qui empêche l'exécutable du processus d'être complètement déchargé de la mémoire après utilisation (donc un redémarrage plus rapide mais au prix de surcharger de mémoire), 2 pour donner au processus les droits du groupe du fichier (au lieu de ceux du groupe de l'utilisateur) et enfin 4 pour que tout utilisateur qui exécute ce fichier lance un processus avec les droit du propriétaire du fichier au lieu de ses propres droits. *C'est extrêmement dangereux pour le propriétaire du fichier, qui doit dans ce cas être absolument certain que le programme ne peut pas être détourné de ses buts d'origine !* ; il est alors responsable de tout ce qui sera fait avec. La commande `su`, qui permet de lancer un shell avec les droits de `root` est naturellement dans ce cas :
> `$ ls -l /bin/su`
> `-rwsr-xr-x 1 root root 28872 déc.  28 16:51 /bin/su`
> La commande `ls -l` le montre en mettant un `s` à la place du `x` en face des droits du propriétaire (mode **4**755).

4.4 L'arborescence typique d'un système Unix

Les fichiers qui sont utiles au système Unix sont souvent placés à des adresses communes à tous les systèmes installés, établies principalement par des habitudes des super-users (`root`) dans le monde entier, donc par le poids de l'histoire. D'un système à l'autre, on constate cependant quelques variations qui sont généralement motivées par les objectifs principaux du système considéré. La spécification de ces différences constitue l'essentiel de ce qu'on appelle une « distribution » d'Unix. Parmi ces distributions, on peut citer BSD, Mageia, Fedora, RedHat, Debian, Ubuntu, *etc.* D'une *distribution* à une autre, la structure de l'arborescence peut donc changer mais les répertoires suivants sont généralement présents.

`/home` : contient la quasi totalité des home directories des utilisateurs qui ne sont pas virtuels.

`/etc` : contient les fichiers qui paramètrent les fonctions principales du système (utilisateurs et mots de passe, structure du réseau local, structure générale de l'arborescence, propriétés de l'environnement graphique, *etc*).

`/usr` : contient la majorité des commandes utiles aux utilisateurs ainsi que les bibliothèques et les fichiers de paramètres correspondants.

3. « `ls -l nomRepertoire` » fournit la liste des fils du répertoire. Pour avoir les information du répertoire lui-même au lieu de ses fils, on doit ajouter l'option `-d`

`/var` : contient la plupart des informations à propos de l'état courant du système (messages d'information réguliers, paramètres changeant avec le temps, *etc*).

`/dev` : contient tous les fichiers de type devices et leur gestion. On peut noter en particulier `/dev/null` qui est un fichier très spécial : tout le monde peut le lire et écrire dedans mais il reste toujours vide, c'est une « poubelle de données » souvent utile pour passer sous silence des messages inutiles comme on le verra plus loin.

4.5 Manipulation de l'arborescence

- Pour « se déplacer » dans l'arbre des répertoires, on utilise la commande `cd` en lui donnant pour argument l'adresse (absolue ou relative) du répertoire où l'on veut aller.
- Pour créer un répertoire, on utilise `mkdir` en lui donnant pour argument l'adresse (absolue ou relative) du répertoire que l'on veut créer. Le répertoire père doit préexister.
- Pour supprimer un répertoire *vide*, on utilise `rmdir` en lui donnant pour argument l'adresse du répertoire que l'on veut supprimer.
- Pour supprimer un fichier qui n'est pas un répertoire, on utilise `rm`.
- Pour renommer et/ou déplacer un fichier ou un répertoire dans l'arborescence, on utilise la commande `mv` en lui donnant pour premier argument l'ancienne adresse et pour second argument la nouvelle adresse.
- Pour copier un fichier, on utilise la commande `cp` avec pour premier argument l'adresse du fichier d'origine et pour second argument l'adresse du nouveau fichier.
- Pour copier un répertoire entier, on peut utiliser la commande `cp` avec l'option `-r` (copie récursive) en premier argument, le répertoire d'origine en second argument et l'adresse du nouveau répertoire en troisième argument. (Voir aussi la commande `tar` pour une copie plus avancée de répertoires).
- De même pour supprimer un répertoire non vide et tout son contenu, on peut utiliser l'option `-r` de `rm`... à utiliser avec attention ![4]
- Faire `man` pour toutes ces commandes pour en savoir plus. Faire également `man ls` pour extraire des informations sur les répertoires et fichiers d'une arborescence.

Les partitions : les disques qui contiennent les fichiers sont découpés en morceaux appelés des *partitions* afin de faciliter leur sauvegarde et de limiter les dégâts en cas de défaillance matérielle [5]. La structure interne d'une partition (dont la manière dont sont rangés les numéros de i-node) repose sur une spécification technique qui est généralement appelée `ext4` et il est par ailleurs possible « d'importer » des partitions non Unix, comme FAT, NTFS ou autre.
Une de ces partitions contient le répertoire `/` et les autres sont alors des sous-arbres de l'arborescence globale. Le fichier `/etc/fstab` définit cette arborescence « à gros grains » entre partitions. `man fstab` vous aidera à lire et modifier `/etc/fstab` selon vos besoins (en tant que `root` naturellement) mais les distributions d'Unix fournissent une interface graphique pour le faire plus aisément.

4. par défaut, il n'y a pas de « corbeille » en Unix
5. mais rien n'empêche d'avoir une unique partition qui couvre tout le disque