Cours n°1

Nota : poly disponible sur `http://www.i3s.unice.fr/~bernot` rubrique *Enseignements*. Il contient aussi une fiche en annexe qui récapitule les principales commandes vues durant le cours.

1 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 ambigüe les acteurs qui peuvent utiliser le système, leurs droits et ce que le système peut faire en leur nom,
- les *fichiers*, qui sauvegardent de manière (*a priori*) 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.

Un utilisateur virtuel particulier, appelé `root`, a tous les droits sur le système. Certains fichiers *appartiennent* à `root` ; ce sont typiquement ceux qui contrôlent le comportement global du système.

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 donc **responsable** des actions de ces processus. Hormis `root` qui a tous les droits, 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.

2 Les utilisateurs

Le système identifie ses utilisateurs autorisés par leur *nom de login* et c'est la combinaison du nom de login et du mot de passe associé qui permet au système de « reconnaître » ses utilisateurs autorisés et de leur donner un accès identifié.

> Lorsque vous êtes connectés au système, vous pouvez connaître votre nom de login en lançant une fenêtre dite *de terminal*, et en y tapant la commande « `whoami` » indique alors votre nom de login.

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).

Un utilisateur possède également un répertoire où le système le place par défaut au moment du « login » ; 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 $HOME` ». C'est généralement quelquechose du genre `/home/monEquipe/monLoginName`.
> La commande `echo` est très primitive : elle écrit à l'écran ce qu'on lui donne en « arguments » (après le premier espace). Le `$` accolé à un nom transforme ce nom en sa valeur. Les noms qui possèdent une valeur sont appelés des variables. Ainsi « `echo $HOME` » écrit à l'écran la valeur de la variable `HOME`.

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 généralement 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 en réalité un simple shell qui est exécuté au moment du login et ce dernier lance immédiatement les processus de l'environnement graphique au lieu d'attendre que l'utilisateur tape une commande.

3 Les fichiers

Les fichiers sont les lieux de mémorisation durable des données de tout le système. Ils sont généralement placés physiquement sur un ou plusieurs *disques durs* ou sur tout autre dispositif de grande capacité qui ne perd pas ses données lorsque le courant est éteint (disque SSD, mémoire flash, clef USB, DVD, ...).

3.1 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 mémoriser une liste d'autres fichiers, on les appelle des *répertoires* ou *directories* (ou encore « dossiers » pour les utilisateurs naïfs) ; les répertoires sont bel et bien des fichiers comme les autres
- il y a des fichiers dont le rôle est de pointer vers un seul autre fichier, on les appelle des *liens symboliques* (ou encore « raccourcis » pour ces mêmes utilisateurs naïfs)
- il y a des fichiers *exécutables* qui contiennent un programme que l'ordinateur peut exécuter si on le lui demande
- *etc.*

Les fichers auxquels les gens pensent habituellement, c'est-à-dire ceux qui ont un contenu avec des données standard, 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, ...) et la commande `mimetype` fait cette traduction mais ce n'est pas très fiable puisqu'on peut sans problème renommer un fichier à sa guise. La commande `file`, en revanche, analyse le contenu du fichier qu'on lui donne en argument pour déterminer son type.

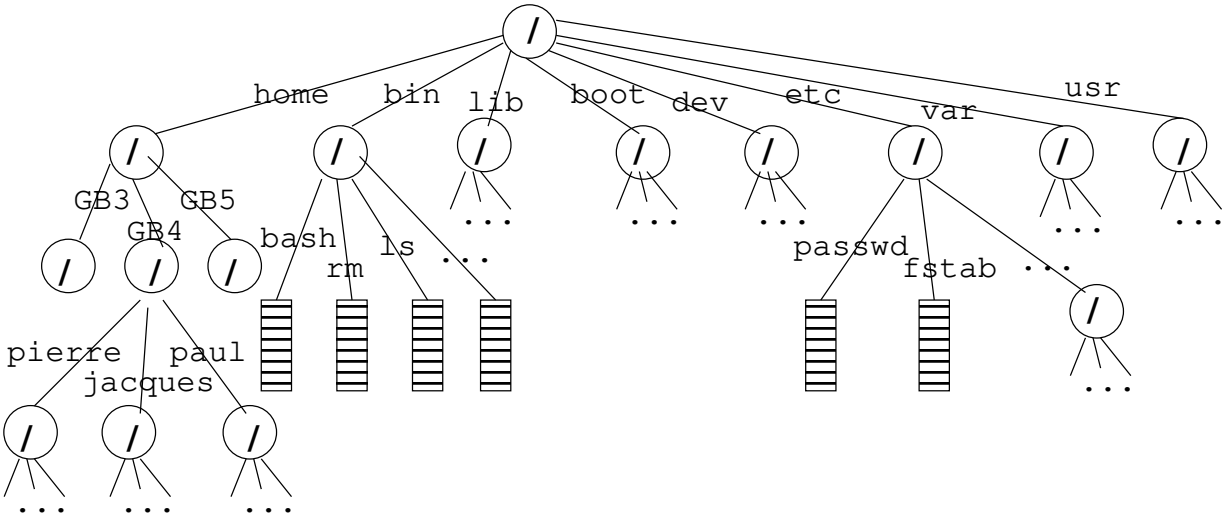
```
$ file /home
/home: directory
$ file /etc/hosts
/etc/hosts: ASCII text
$ file /bin/bash
/bin/bash: ... executable, x86-64, version ...
```

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 (ou créé) pour la dernière fois,
- 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.

3.2 L'arborescence du système de fichiers

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 donc en fait un ensemble de liens vers les fichiers contenus dans ce répertoire (fichiers qui peuvent être eux-même des répertoires, ce qui donne au système de fichiers une structure d'arbre). Ce sont ces liens qui portent le nom des fichiers (que le fichier soit plat, un répertoire ou de tout autre type).



Un fichier (quel que soit son type) est donc 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/etudiants/bio/paul`). Par conséquent, un nom de fichier 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`* », ce qui change, c'est l'arborescence donc seulement les contenus des fichiers de type répertoire qui la constituent. (`mv` pour « move »).

Lorsqu'un processus tourne, en particulier lorsqu'un shell 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. Ainsi, pour un processus (dont le shell) on 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.

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.

Lorsque vous êtes sous un shell, la commande `pwd` (print working directory) vous fournit le répertoire de travail dans lequel vous êtes. La commande `cd` (change directory) vous permet de changer de répertoire de travail. Enfin la commande `ls` (list) fournit la liste des liens d'un répertoire (par défaut son répertoire de travail).

```
$ pwd
/home/bioinfo/bernot

$ cd bin

$ pwd
/home/bioinfo/bernot/bin

$ ls
i686  shells  x86_64

$ ls -a
. .. i686  shells  x86_64

$ ls i686 x86_64
i686:
RNAplot  hsim  wi  wx  wxd

x86_64:
wi  wx  wxd
```

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

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 [1].

4 Les fichiers (suite)

4.1 Les droits d'accès

Tout fichier quel que soit son type a :
- un *propriétaire*, qui est un utilisateur reconnu du système,
- et un *groupe*, qui n'est pas nécessairement celui par défaut de son propriétaire, bien que la plupart des fichiers aient en pratique le groupe par défaut de leur propriétaire ;
- les « autres » utilisateurs, c'est-à-dire ceux qui ne sont ni propriétaire ni membre du groupe du fichier, peuvent néanmoins avoir le droit d'utiliser le fichier si le propriétaire l'accepte.

Tout fichier pourrait *a priori* 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 seulement le droit de lecture au propriétaire, mais les droits de lecture et d'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` peuvent effectuer un `chmod` sur un fichier.

Lorsqu'un utilisateur a le droit d'exécuter un fichier plat, le processusqu'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...*** De même soyez conscients de ce que vous faites si vous utilisez un programme écrit par quelqu'un d'autre ou par une entreprise qui a intérêt à stocker des informations à votre sujet, voire à les divulguer en les revendant...

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