Initiation à UNIX

Olivier LOUISNARD

5 janvier 2004

Table des matières

Chapitre 1

Généralités

1.1 La philosophie d'UNIX

On peut résumer très simplement le problème par le petit dicton suivant : **chez UNIX, tout est fichier**. Ça veut dire qu'un programme, une librairie, un périphérique extérieur (disquette, disque, CDROM, imprimante) est vue par le système comme un fichier. Pour l'utilisateur débutant, les implications de ce principe sont peu importantes, mais cette unité de représentation fait d'UNIX un système très sûr et très modulable, contrairement à un certain système dont le nom commence par W, qui est un fouillis inextricable de bouts de scotch qui font tenir tant bien que mal un ensemble peu fiable.

1.2 Le système de fichiers

Puisque tout est fichier, il faut bien organiser le rangement de ces fichiers, et ceci est fait sous forme d'**arborescence**.

Le principe est le même que pour un Macintosh. On y retrouve la notion de répertoire qui en Anglais s'appelle **directory**, que l'on peut voir comme des tiroirs. Les fichiers sont rangés dans ces répertoires ou tiroirs.

Ce qui change c'est que pour copier, déplacer un fichier ou un répertoire dans cette arborescence, on se sert de commandes tapées au clavier plutôt que la souris [1]. L'interpréteur de commandes qui permet ce dialogue s'appelle le **shell**.

L'arborescence a un sommet, qui correspondrait au bureau sur un Macintosh, et qu'on appelle paradoxalement **la racine**. Différentes branches partent de cette racine. Certaines sont requises par le système, d'autres sont prévues pour les utilisateurs.

[1] Note de mise à jour : la plupart des environnements fenêtrés sous UNIX proposent désormais un "gestionnaire de fichiers" à la manière de l'explorateur Windows. Les commandes tapées au clavier sont cela dit toujours plus souples et plus rapides une fois que l'on s'est donné la peine de les apprendre.

1.3 Bref aperçu de l'arborescence

Commençons notre excursion à partir du sommet. Celui-ci est noté /. Dans ce grand tiroir qui contient tout, on trouve les directories suivants (liste non exhaustive) :

- **etc** : les fichiers de configuration du système.
- **bin** : les programmes constituant le système
- **lib** : les librairies système
- **usr** : les programmes et librairies (et parfois aussi la doc) du système qui ne sont pas dans les deux précédents.
- **tmp** : le directory des fichiers temporaires.
  Ces directories, qui contiennent eux-mêmes d'autres directories sont présents sur tout système UNIX et suffisent à priori pour faire tourner le système.
- **home** : le tiroir qui regroupe tous les fichiers des utilisateurs.

1.4 Fonctionnement d'UNIX

Une petite comparaison qui vaut ce qu'elle vaut : lorsque vous allez au marché, observez le vendeur. Il sert les personnes qui font la queue les unes après les autres, en commençant (dans la mesure du possible) par servir les premières arrivées.

C'est ce que fait UNIX. Il doit gérer les requêtes de plusieurs utilisateurs, et les traite les unes après les autres. Le cerveau d'UNIX (ou du vendeur) qui organise tout ça s'appelle **le noyau**.

L'échange entre vous et le vendeur se fait par le toucher et l'ouïe : il vous annonce le prix, vous lui donnez des sous, il vous donne vos poireaux et vous rend la monnaie. Si le vendeur n'avait que son cerveau, ce ne marcherait pas. Il faut donc une interface entre son cerveau et ses clients. En UNIX, cette interface de dialogue entre le noyau et vous s'appelle **le shell**. C'est un langage commun entre vous et le noyau.

De temps en temps, le vendeur perd un peu de temps : il va faire de la monnaie, enlève les cageots vides, c'est-à-dire des choses qui ne vous regardent pas, si ce n'est que ces différentes actions assurent le bon fonctionnement de son commerce. UNIX fait la même chose, et utilise un peu de son temps pour gérer des tâches internes : mettre à jour les disques, répondre à une requête du réseau, dialoguer avec une imprimante ...

Ces actions s'appellent **des démons**[2] («daemon» en anglais), et tournent en permanence sans que vous vous en aperceviez[3].

Ce qu'il faut retenir, c'est que le **shell** (coquille en anglais) constitue en fait les 5 sens du système UNIX, et permet à l'utilisateur de dialoguer avec le cerveau d'UNIX qu'est le **noyau**.

[2] La terminologie d'UNIX est très imagée !

[3] mais il est possible de les voir, car UNIX ne cache rien de son fonctionnement, contrairement à d'autres...

Rappelons enfin que tout ceci se fait avec des fichiers, et même le noyau en est un.

1.5 Utilisateurs

Une machine UNIX est par définition multi-utilisateurs. Le concept va bien au-delà du fait que plusieurs utilisateurs puissent utiliser une même machine. Un utilisateur donné est clairement identifié par UNIX, et les choses qu'il a le droit de faire sont parfaitement définies.

Une autre comparaison : pour faire partie d'un club de sports, vous devez vous inscrire, et ceci fait, on vous donne un casier pour ranger vos affaires. Vous n'avez pas le droit d'aller chambouler le casier des autres, et si vous n'êtes pas inscrits, vous n'avez même pas le droit d'entrer.

UNIX fait la même chose (sauf que c'est gratuit). Tant que vous n'êtes pas inscrit, vous ne pouvez pas accéder au système. Le chef du club, c'est l'administrateur système, ou **super-utilisateur** ou **root** et lui seul peut vous inscrire. Si vous vous comportez mal, vous pouvez même être exclu du club. Rassurez-vous, ça n'arrive pas souvent `:-)`

Votre inscription consiste en l'attribution de :

- un **nom de login** ou **username** :
  c'est votre identifiant pour la machine UNIX. Souvent c'est votre nom complet tronqué à 8 lettres en minuscules.
- un **password** :
  c'est votre mot de passe ou code d'accès au club. Au début on vous en donne un par défaut, mais vous devrez le changer rapidement. Nous y reviendrons, mais de toutes façons, ne donnez votre mot de passe a PERSONNE.
- un **home-directory** :
  c'est un directory pour ranger vos fichiers (c'est le casier). Dans l'arborescence, ce directory s'appelle généralement :
  `/home/username`
- votre appartenance à **un groupe** :
  Les groupes permettent de classer les utilisateurs en fonction de leurs centres d'intérêt.

On dit alors que vous avez **un compte UNIX**.

Les seuls utilisateurs connus par le système sont donc ceux qui sont inscrits, **root** qui a tous les droits, plus quelques autres qui ne correspondent pas à des personnes physiques, et sont utilisés par le système (webmaster pour le WEB, postmaster pour le Email etc...).

1.6 Se connecter

Tout ce qui a été dit ci-dessus s'applique à une machine UNIX toute seule. Dans ce cas, pour se connecter à la machine (on dit aussi **se logguer**), vous vous mettez devant, et vous voyez qu'elle vous propose l'invitation

```
login:
```

On tape son nom de login, et apparait ensuite l'invite

```
password:
```

Là, on tape son mot de passe. *ATTENTION : lorsque vous tapez votre mot de passe, aucun caractère n'apparaît à l'écran !*, sans quoi tout le monde pourrait le voir.

Ca y est : vous êtes connecté, le shell attend que vous lui entriez des commandes.

L'invite qui apparaît à l'écran dépend du shell et de la configuration faite par l'administrateur système. A l'EMAC, vous voyez apparaître :

```
username-machine[~]
```

où username est votre nom de login et machine le nom de la machine sur laquelle vous êtes connecté. Le [~] sera expliqué plus tard.

Rappelez-vous que lorsque vous vous connectez, vous arrivez toujours dans votre home-directory (votre casier dans le club «machine»), qui est

```
/home/username
```

On peut également se connecter depuis un Macintosh ou un PC, en utilisant un programme appelé TELNET (voir aussi section 3.3.2). Il faut alors indiquer le nom de la machine à laquelle on veut se connecter, puis TELNET propose l'invitation

```
login:
```

1.7 Machines en réseau

En général les machine UNIX ne sont pas isolées du monde. Elles sont connectées entre elles par un réseau.

Reprenons la comparaison précédente : imaginez qu'il y ait plusieurs clubs de sport dans votre ville, et que vous ayiez envie d'aller tantôt dans l'un, tantôt dans l'autre. Le plus simple c'est de vous inscrire une fois pour toutes, et d'être connu ensuite

dans tous les clubs. UNIX vous offre cette possibilité et une fois que vous avez un compte, toutes les machines d'un réseau donné vous connaissent. Vous pouvez alors vous connecter sur l'une ou l'autre de ces machines.

La où ça devient fort (et où la comparaison précédente s'arrête...), c'est que vous retrouvez votre casier identique dans chaque club, avec les affaires que vous aviez laissé la dernière fois. Votre home-directory est le même sur toutes les machines du réseau, et vous le retrouvez dans l'état ou vous l'avez laissé la dernière fois, même si vous étiez connecté sur une autre machine.

Ceci est réalisé grâce au système **NFS** (Network File System) qui permet de mettre des bouts d'arborescence (`/home` dans l'exemple qui nous préoccupe) en commun entre plusieurs machine.

Chapitre 2

Concepts et commandes de base

Vous savez maintenant vous logguer. Nous allons apprendre ici à se déplacer dans l'arborescence, à copier, déplacer, renommer des fichiers.

Avant d'aborder les commandes permettant de faire tout ça, la première chose que vous devez savoir faire, c'est vous y retrouver dans l'arborescence. Quand on a compris ce qui suit, on a plus de problèmes avec UNIX.

Remarque importante : n'hésitez pas à vous déplacer partout dans l'arborescence, même dans les fichiers systèmes. De toutes façons, vous ne pouvez pas faire de bêtise, hormis sur vos propres fichiers.

2.1 Préliminaire

2.1.1 Référencer un point de l'arborescence.

Commençons par de la terminologie : un point de l'arborescence s'appelle *PATH*.

La racine de l'arborescence se note / Pour référencer un nom de manière absolue, il suffit de descendre les étages à partir de la racine en les séparant par des / Exemple :

```
/home
/home/louisnar
/usr
/usr/local
/usr/local/bin
```

On voit bien que la notation précédente est longue. Le shell propose les raccourcis suivants :
– La notation `~` représente mon home-directory
– La notation `~username` représente le home-directory de l'utilisateur username.
On peut plus également référencer un point de l'arborescence en tenant compte du directory où l'on se trouve.

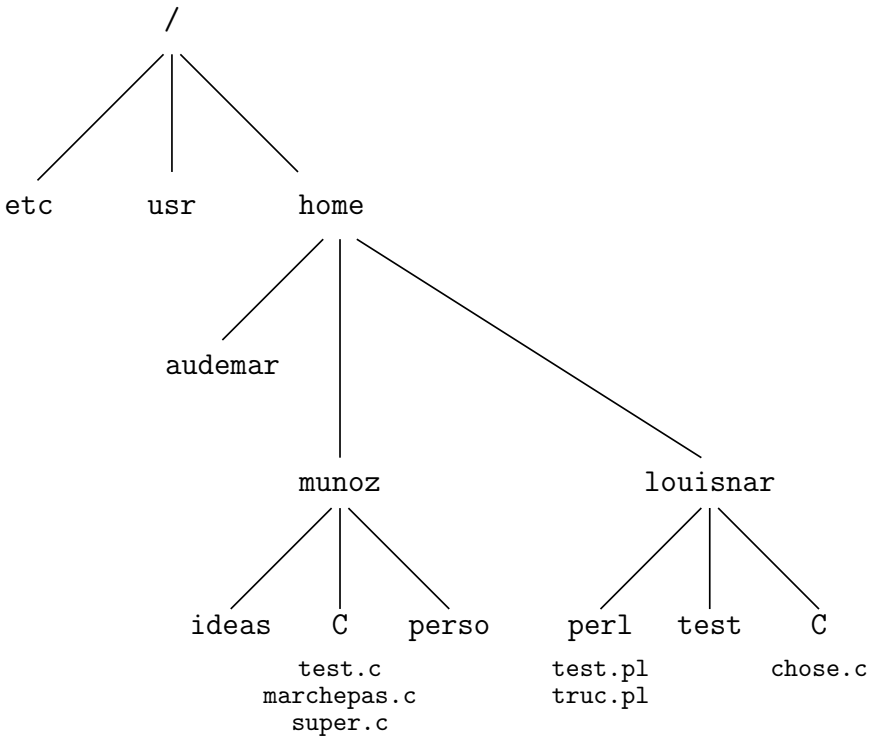


FIG. 2.1 – Exemple d'arborescence.

Voilà en gros la règle :
- Si un fichier ou un directory est à l'endroit où je me trouve, je peux le référencer simplement par son nom. Ensuite je peux descendre dans l'arborescence comme précédemment en nommant les étages successifs séparés par des caractères / (prononcer "slash").
- La notation `.` représente le directory courant
- La notation `..` représente le directory juste au-dessus

Exemples (se reporter à la figure 2.1) :

1. Admettons que je (louisnar) sois dans mon home-directory, et je veux référencer le fichier test.c dans le directory C de munoz. Je peux y accéder par :

   ```
   /home/munoz/C/test.c
   ~munoz/C/test.c
   ../munoz/C/test.c
   ```

2. Je (toujours louisnar) suis toujours dans mon home-directory et je veux accéder au fichier truc.pl dans mon directory perl. Je peux utiliser les notations :

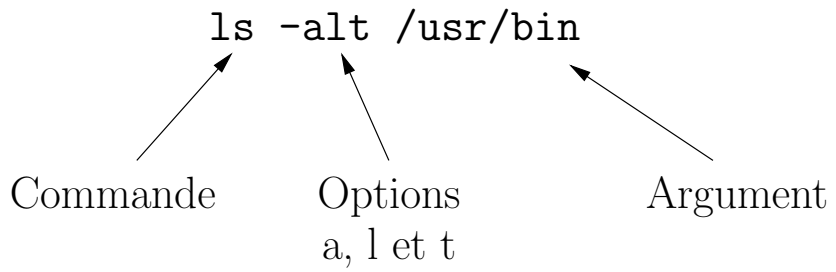
   ```
   /home/louisnar/perl/truc.pl
   ~louisnar/perl/truc.pl
   ~/perl/truc.pl
   perl/truc.pl
   ```

2.1.2 Syntaxe des commandes UNIX

Les commandes UNIX sont suivies de
– zero, une ou plusieurs options.
– zero, un ou plusieurs arguments.
La commande, les options et les arguments doivent être séparées par un ou plusieurs espaces .

En général, les options commencent par le caractère `-`, et sont suivis d'une ou plusieurs lettres, chacune représentant une option.

Un exemple tout de suite :



Mais on peut aussi écrire les options une par une :

```
ls -a -l -t /usr/bin
```

Ce qu'il faut retenir, c'est qu'il faut mettre des espaces entre la commande, ses options et ses arguments. Par contre, **il faut faire attention à ne pas mettre d'espace au milieu d'un nom de chemin**, surtout avec certaines commandes «définitives» telles que `rm`.

2.1.3 Référencer un ensemble de fichiers ou directories

Le caractère `*` a un sens particulier. Il remplace n'importe quelle chaine de caractère de n'importe quelle longueur. Ainsi `truc*` représente tous les fichiers qui commencent par la chaine `truc`.

On l'utilise souvent derrière les commandes UNIX pour que l'action recherchée s'applique à un ensemble de fichiers, plutôt qu'à un seul. Par exemple, on peut vouloir lister tous les fichiers finissant par `.gif`, déplacer tous les fichiers finissant par `.f`, etc...

Il existe d'autres caractères spéciaux du même type, mais ceci dépasserait les objectifs de ce cours.

2.1.4 Le shell `tcsh`

Il existe plusieurs types de shells. L'ancêtre de tous les shells existants s'appelle **sh**, et est toujours disponible par défaut sur tous les systèmes. Avec la division d'UNIX en deux branches, deux shells appelés **ksh** (Korn Shell) et **csh** (C-Shell) virent le jour.

Des fonctionnalités supplémentaires furent ensuite ajoutées à ces deux shells, ce qui déboucha sur **bash** successeur du Korn-Shell et **tcsh** successeur du C-Shell. C'est ce dernier que nous utilisons par défaut à l'EMAC.

Les fonctionnalités sont impressionnantes, donc voici seulement les principales :

- Vous pouvez utiliser les flèches haut, bas, gauche et droite pour retrouver les précédentes commandes tapées, et les modifier pour en créer une nouvelle.
- Tapez les premières lettres d'une commande, puis appuyez sur ESC P : vous retrouvez toutes les commandes déjà tapées commençant par ces lettres. Chaque nouvelle frappe de ESC P vous fait remonter vers les commandes les plus anciennes commençant par ces lettres.
- CTRL U efface la ligne en cours de frappe.
- Pour simplifier la frappe de noms de fichiers ou directories très long, tapez les premières lettres du nom, puis appuyez sur TAB : tcsh complète automatiquement les dernières lettres, sauf s'il y a ambiguité, auquel cas un BIP est émis, et une liste des différents choix possibles est proposée. Ça s'appelle la **complétion automatique**.
  Ce mécanisme fonctionne aussi pour les commandes. Par exemple la frappe de c puis TAB vous propose toutes les commandes UNIX auxquelles vous avez accès commençant par la lettre «c».

2.2 Les principales commandes UNIX

2.2.1 `man` : Aide en ligne

`man` *nom commande UNIX*

vous affiche le mode d'emploi de la commande spécifiée. Si vous voulez approfondir tout ce qui est écrit ci-dessous, n'hésitez pas ! Petit problème, c'est en anglais... [1]

2.2.2 `cd` : Se déplacer dans l'arborescence

`cd` *nom directory*

vous place dans le directory spécifié.

2.2.3 `pwd` : Voir où l'on est dans l'arborescence

`pwd`

vous montre dans quel directory vous êtes actuellement.

Notons que `tcsh` ainsi que tous les shells modernes, vous permettent d'afficher l'endroit où vous vous trouvez directement dans le prompt. On peut afficher le chemin complet, mais ca serait très lourd, alors en général (et c'est le cas à l'EMAC) on choisit de n'afficher que le dernier élément du chemin où l'on se trouve.

Si on est dans son home-directory, le shell affiche juste «~».

Un exemple : initialement je suis dans mon home-directory.

```
louisnar@stalingrad[~] cd enseignement
```

Le prompt devient :

```
louisnar@stalingrad[enseignement] cd UNIX
louisnar@stalingrad[UNIX]
```

Bref c'est bien pratique !

[1] Sauf sur des versions récentes de LINUX, où c'est traduit, mais ca ne concerne pas (pour l'instant) les machines de l'EMAC.

2.2.4 `ls` : Voir le contenu d'un directory

`ls` *nom directory*
nom fichier

Cette commande vous renvoie la liste des fichiers et sous-directories du (ou des) directory demandé.

Utilisée sans argument, elle s'applique au directory courant.

Avec un (ou plusieurs) nom de fichier, elle liste ces fichiers.

Il existe de nombreuses options pour modifier l'ordre de tri, le style de l'affichage etc... Nous indiquons ci-dessous les plus utiles, voyez les pages man pour les autres.

`-F` mise en évidence exécutables et directories

Affiche le caractère `*` après les nom de fichiers exécutables (les programmes) et le caractère `/` après les noms de directories. Exemple :

```
louisnar@stalingrad[dirtest] ls -F
Makefile   ddriv3.f   opendir.f  ref.pl*
autocad/   mainbul.f  perso/     travail/
```

C'est le fonctionnement par défaut à l'EMAC.

`-l` affichage long

Vous présente toutes les informations possibles relatives aux fichiers mentionnés :

```
total 348K
-rw-r--r--    1 louisnar pcf      234 déc 19 16:10 enonce.aux
-rw-r--r--    1 louisnar pcf       14K déc 19 16:10 enonce.dvi
-rw-r--r--    1 louisnar pcf     9,9K déc 19 16:10 enonce.log
-rw-r--r--    1 louisnar pcf       59K déc 19 16:12 enonce.pdf
-rw-r--r--    1 louisnar pcf      168K déc 19 16:10 enonce.ps
-rw-r--r--    1 louisnar pcf       13K déc 19 16:06 enonce.tex
-rw-r--r--    1 louisnar pcf     3,7K déc 16 09:13 pommeau.fig
-rw-r--r--    1 louisnar pcf     4,1K déc 16 13:35 pommeau.pdf
-rw-r--r--    1 louisnar pcf      12K déc 16 09:13 pommeau.pstex
-rw-r--r--    1 louisnar pcf     1,2K déc 16 13:35 pommeau.pstex_t
```

De gauche à droite, les information présentées sont :

- Les permissions (`-rw-r-r-`, voir section 3.1.2)
- Le nombre d'inodes (`1`)
- Le propriétaire (`louisnar`)

– Le groupe propriétaire (`pcf`)
– La taille (`168K`)
– La date et l'heure de dernière modification ou création (`168K déc 19 16 :10`)
– Le nom (`enonce.ps`)

`-a` affichage fichiers cachés

Affiche aussi les fichiers dont le nom commence par un point. Ceux-ci sont appelés fichiers cachés, ce qui signifie simplement que la commande **l**s classiques ne les liste pas [2]

`-t` tri par date

Affiche les fichiers les plus récemment modifiés en premier.

[2] Ce sont en général des fichiers de configurations du shell, de X-windows, en fait de n'importe quel logiciel.

2.2.5 `cp` : Copier ou dupliquer des fichiers

`cp` *nom fichier* *nom fichier*
nom directory

La commande `cp` admet deux arguments : une *source* qui est ce qu'on veut copier, et une *destination* qui est ce dans quoi on veut le copier. Selon que cette destination est un fichier ou un directory, la commande prend des sens différents :

`cp fich1 fich2`	duplique le fichier fich1 en fich2
`cp fich dir`	copie le fichier source fich dans le directory dir, en conservant le même nom.

Exemples (à partir de la figure 2.1) :

Je suis `louisnar`, je suis dans mon directory `perl` et je veux faire une copie de mon fichier `truc.pl`. Je peux faire :

```
cp truc.pl trucbis.pl
```

Maintenant, je veux copier ce fichier dans mon directory `test`, en conservant le même nom :

```
cp truc.pl ../test
```

ou bien encore :

```
cp truc.pl ~/test
```

Je veux maintenant copier le fichier `super.c` du directory C de munoz dans mon directory `C` :

```
cp ~munoz/C/super.c ~/C
```

ou bien plus simplement, si je suis déjà dans mon directory `C` :

```
cp ~munoz/C/super.c .
```

Je peux également faire cette copie en changeant le nom du fichier :

```
cp ~munoz/C/super.c ./superbis.c
```

Notons que ce dernier exemple peut ne pas fonctionner si `munoz` a protégé son fichier en lecture. C'est la cas par défaut à l'EMAC où tous les fichiers utilisateurs sont par défaut protégés en lecture (paranoïa quand tu nous tiens!!).

De même, et là, c'est normal, on ne peut pas en général[3] copier un fichier dans le compte d'un autre utilisateur, car cela signifierait que l'on pourrait par exemple écraser un de ses fichiers avec un nouveau contenu. Ainsi, vous pouvez essayer la commande suivante :

```
cp photo.gif ~munoz
```

Ca ne marche pas!

On ne peut pas avec la commande de base copier des directories entiers. Ceci n'est possible qu'avec l'option `-r`.

Enfin, on peut spécifier plusieurs sources si la destination est un directory. Par exemple pour copier les fichiers `super.c` et `test.c` du directory `C` de `munoz` dans mon directory `C` :

```
cp ~munoz/C/super.c ~munoz/C/test.c ~/C
```

ou bien carrément pour copier tous les fichiers de son directory `C` :

```
cp ~munoz/C/* ~/C
```

[3]sauf si cet utilisateur autorise explicitement l'accès en écriture sur une partie de son compte... c'est mon cas.

2.2.6 `mv` : Déplacer ou renommer des fichiers

`mv`	*nom fichier*	*nom fichier*
	nom directory	*nom directory*

La commande `mv` admet également deux arguments : une *source* et une *destination*. Ici encore, l'action effectuée dépend du type de la destination :

- `mv fich1 fich2` renomme le fichier fich1 en fich2
- `mv fich dir` déplace le fichier fich dans le directory dir, en conservant le même nom.
- `mv dir1 dir2` déplace toute l'arborescence issue du directory dir1 dans le directory dir2.

Exemples :

Je veux renommer mon fichier `truc.pl` en machin.pl :

```
mv truc.pl machin.pl
```

Je veux déplacer un fichier `netscape.ps` qui serait dansmon home-directory dans mon directory `test` :

```
mv ~/netscape.ps ~/test
```

2.2.7 `rm` : Supprimer des fichiers

`rm` *nom fichier*

Supprime le nom du ou des fichiers spécifiés.

C'est une COMMANDE DANGEREUSE!! Notamment lorsqu'elle est utilisée avec le caractère *. Par exemple :

```
rm *
```

supprime TOUT les fichiers du directory courant. Prenez garde notamment lorsque vous utilisez * pour effacer tous les fichiers d'un type donné. Par exemple pour effacer tous les fichiers finissant par `.gif`, on peut écrire :

```
rm *.gif
```

Si vous mettez malencontreusement un blanc entre `*` et `.gif`, la commande est interprétée comme si vous vouliez d'abord effacer *, c'est-à-dire tout, et ensuite `.gif`, qui n'existe pas, mais peu importe, le mal est fait...

Il n'existe aucun moyen sous UNIX (hormis les sauvegardes effectuées par l'administrateur système) pour récupérer un fichier effacé accidentellement.

La commande `rm` ne peut être utilisée pour effacer des directories entiers, à moins d'utiliser une certaine option, qui est tellement dangereuse que je ne veux pas l'écrire dans ce document !

2.2.8 `mkdir` : Créer un directory

`mkdir` *nom directory*

Pas de difficulté particulière. Rappelez-vous simplement que vous pouvez référencer un directory par un nom absolu ou un nom relatif. Ainsi :

```
mkdir personnel
```

crée un sous-directory `personnel` à l'endroit où vous êtes, et

```
mkdir ~/personnel
```

crée un sous-directory `personnel` juste au dessous de votre home-directory. Il va de soi que vous ne pouvez pas créer un sous-directory chez un autre utilisateur, puisque cela est considéré comme une écriture chez lui.

2.2.9 `rmdir` : Supprimer un directory

`rmdir` *nom directory*

Supprime le directory spécifié, seulement s'il ne contient aucun fichier, ni sous-directory. Commencez donc par vider le directory que vous voulez effacer avant d'utiliser `rmdir`.

2.2.10 `cat` : Afficher le contenu d'un fichier

`cat` *nom fichier*

Cette commande affiche le contenu d'un fichier texte. Un fichier texte est un fichier de caractères visibles organisés en lignes. Si vous appliquez `cat` à un fichier autre que texte, vous verrez apparaître un ensemble de carcatères incompréhensibles, qui correspond aux octets du fichiers interprétés comme des caractères. Vous pouvez essayer d'appliquer cat à `/bin/ls` pour voir ce que ça fait...

2.2.11 `more` : Afficher le contenu d'un fichier page par page

`more` *nom fichier*

Même usage que `cat`, mais la sortie est paginée pour que le contenu du fichier ne défile pas d'un seul coup à l'écran. les différentes commandes sont alors les suivantes :

- ESPACE fait passer à la page suivante
- RETURN fait passer à la ligne suivante
- q quitte
- CTRL B revient à la page précédente
- /chaine pour rechercher une chaine de caractères.

Il existe une version plus récente appelée malicieusement **l**ess, qui est à mon avis plus pratique.

2.2.12 `grep` : Chercher une chaine de caractères dans un fichier

`grep` *chaine de caractères* *nom fichier*

Permet de trouver toutes les occurrences de la chaine de caractères spécifiée dans le ou les fichiers spécifiés. Il va de soi que ces fichiers doivent être des fichiers texte.

Il est conseillé d'encadrer la chaine de caractères par des doubles-quotes (").

2.2.13 `gzip` : Compression de fichiers

`gzip` *nom fichier*

Certains fichiers sont très volumineux (images, sons, CAO). Si on ne s'en sert pas souvent, il est utile de les compresser. Le fichier obtenu est alors inutilisable tel quel, mais prend moins de place et peut être décompressé le jour où on veut s'en resservir.

Le fichier compressé avec gzip porte le nom du fichier initial avec le suffixe `.gz`. Il peut dans certains cas être entre 10 et 20 fois moins volumineux que le fichier initial.

On peut spécifier les options `-1` à `-9` : plus le chiffre est élevé, plus la compression est efficace, mais plus elle est lente.

2.2.14 `gunzip` : Décompression de fichiers

`gunzip` *nom fichier*`.gz`

Décompresse un fichier compressé par `gzip`.

Chapitre 3

UNIX avancé

3.1 Droits d'accès aux fichiers

3.1.1 Principe général

Puisque tout UNIX est construit sous forme de fichiers, la gestion des droits est effectuée au niveau de ceux-ci. Un fichier appartient forcément à un utilisateur et à un groupe. Chaque fichier a de plus 3 types d'attributs (je mets en face le code correspondant, qui nous servira bientôt) :

lecture	**r**
écriture	**w**
exécution	**x**

Chacun de ces 3 attributs est autorisé ou non pour (encore des codes...) :

le possesseur du fichier	**u**
le groupe auquel appartient le fichier	**g**
le reste du monde	**o**

Le fait de pouvoir lire un fichier signifie que l'on peut en connaître le contenu.

Le fait de pouvoir l'écrire signifie que l'on peut le modifier, le déplacer ou l'effacer.

L'autorisation d'exécution concerne deux types d'entités :
- les programmes : on a le droit ou non de l'exécuter.
  Il est bien évident par exemple que le programme qui arrête une machine est interdit aux utilisateurs.
- les directories : on a le droit ou non d'aller dans ce directory. Il existe peu de directories dans le système où on n'ait pas le droit d'aller[1].

[1]Exception notoire : le directory **.**netscape contenant les fichiers de config de Netscape pour un utilisateur est par exemple interdit d'accès à tout le monde, sauf vous.

3.1.2 Savoir lire les droits d'accès

Le possesseur, le groupe propriétaire et les droits d'accès d'un fichier ou directory vous sont affichés par la commande **ls -l**. L'information est concentrée sur 10 caractères.

- Le caractère de gauche est «d» si c'est un directory, «-» sinon.
- Les trois suivants indiquent les droits du propriétaire du fichier : «r», «w» ou «x» si l'accès en lecture, écriture ou exécution est autorisé pour le propriétaire du fichier, «-» si l'accès correspondant est refusé.
- Les trois suivants indiquent les droits d'un utilisateur du groupe auquel appartient le fichier.
- Les trois suivants indiquent les droits de tous les autres utilisateurs (reste du monde).

Par exemple, si vous tapez :

```
ls -l /bin/ls
```

vous voyez apparaitre :

```
-rwxr-xr-x   1 root     bin          25484 Jul  8  1997 /bin/ls
```

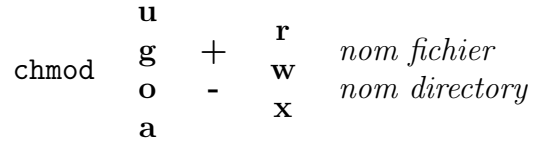
Le fichier appartient à root (le super-utilisateur), et au groupe bin. Tout le monde a le droit de l'éxécuter (heureusement, vu que c'est une commande UNIX !) et de le lire (c'est moins utile...), mais seul root a le droit de le modifier.

Autre exemple :

```
-rwxr-x---   1 louisnar info         9121 Aug 28 02:37 valide.pl
```

Le fichier m'appartient et appartient au groupe info. J'ai le droit de le lire, l'écrire et l'exécuter. Les gens du groupe info ont le droit de le lire et l'exécuter. Tous les autres n'ont aucun droit.

3.1.3 `chmod` : Modifier les droits d'accès



Cette commande[2] ajoute (+) ou supprime (**-**) le droit en lecture(**r**), écriture(**w**) ou exécution(**x**) au propriétaire du fichier (**u**), au groupe auquel appartient le fichier (**g**), au reste du monde(**o**), ou bien à tout le monde (**a**).

[2]Certaines personnes utilisent chmod avec un code octal pour faire la même chose, mais c'est plus compliqué, alors à réserver aux habitués ou à ceux qui veulent se faire passer pour des habitués...

On peut combiner pluseurs lettres ensemble, tant pour les droits (par exemple **rx**) que pour les personnes (par exemple **ug**).

Exemple : je reprends le fichier `valide.pl` précédent, dont les droits sont initialement :

```
-rwxr-x---   1 louisnar info        9121 Aug 28 02:37 valide.pl
```

et je veux que les gens du groupe info n'aient plus l'accès en lecture :

```
chmod g-r valide.pl
```

Le fichier devient alors

```
-rwx--x---   1 louisnar info        9121 Aug 28 02:37 valide.pl
```

Finalement je veux bien que tout le monde puisse le lire :

```
chmod a+r valide.pl
```

```
-rwxr-xr--   1 louisnar info        9121 Aug 28 02:37 valide.pl
```

et enfin je décide que ni moi, ni les gens de l'info n'ont plus le droit ni de le lire ni de l'exécuter (ce qui est débile, j'en conviens...) :

```
chmod ug-rx valide.pl
```

```
--w----r--   1 louisnar info        9121 Aug 28 02:37 valide.pl
```

Notons enfin l'option `-R` (récursif) qui permet d'exécuter la commande à tous les sous-directories issu du point spécifié. Ainsi une bonne manière de donner les accès en lecture à tout le monde sur tous les fichiers de son compte s'écrit :

```
chmod -R a+r *
```

3.1.4 Droits d'accès par défaut

Reste le problème suivant : quand je crée un fichier ou un directory, quels sont les droits affectés par défaut. Vaste polémique : à l'EMAC, suite à la demande expresse de certains individus, tous les nouveaux fichiers sont par défaut :

```
-rw-------
```

genre je veux pas que mon voisin copie sur moi. Vous l'avez compris, ce n'est pas la solution que j'apprécie le plus, car dès qu'on veut copier les fichiers de quelqu'un d'autre, celui-ci doit faire un chmod. Sur de nombreux sites, les droits par défaut sont :

```
-rw-r--r--
```

qui signifie : par défaut je veux bien que tout le monde vienne voir chez moi. A chacun de se débrouiller ensuite s'il veut protéger certains fichiers particuliers.

En fait chacun peut choisir le comportement par défaut pour ses propres fichiers avec une commande appelée `umask`, qui sort du cadre de ce cours.

3.2 Commandes relatives aux process

3.2.1 Les process

Nous l'avons déjà mentionné, UNIX fait tourner en parallèle plusieurs programmes, qu'on appellera en bon Franglais «process», pour ses besoins propres d'une part, et pour satisfaire chaque utilisateur d'autre part.

Par exemple, le shell qui interprète vos commande est un process. Le truc qui affiche `login :` sur la console en est un autre.

Un process peut éventuellement en lancer un autre, auquel cas ce dernier est appelé «fils» du premier.

Pour se repérer dans tous ces process , UNIX leur affecte un numéro appelé «process ID» (prononcer Aïe Di) ou «PID».

De plus chaque process a un propriétaire. Les process systèmes appartiennent en général à `root`. Si vous lancez un process vous-même, il vous appartient.

3.2.2 ps : lister les process d'une machine

`ps`

`ps` sans options affiche seulement les process liés à la fenêtre dans laquelle vous avez tapé la commande, c'est-à-dire au moins le shell, plus d'autres programmes que vous auriez lancé dans cette fenêtre. Pour chaque process, `ps` affiche plusieurs informations. Les plus utiles sont :

- **PID** : le process ID
- **CMD** : le path complet de l'exécutable correspondant au process et éventuellement ses arguments
- **TIME** : le temps cumulé accordé par le processeur au process.

Il y a un grand nombre d'options, compatibles entre elles, dont voici les plus utiles :

`ps -e`	affiche tous les process qui tournent sur la machine.
`ps -f`	affiche le maximum d'informations pour chaque process, notamment le username de son propriétaire.
`ps -u username`	(incompatible avec -e) affiche seulement les process dont le propriétaire est username.

3.2.3 Lancer un process

Le plus simple, c'est de taper le nom du fichier exécutable. Ainsi quand vous tapez `ls`, vous lancez le fichier exécutable `/bin/ls`. Un process est lancé, et ce process est fils du process tcsh. Une fois terminé, vous reprenez la main dans la fenetre où vous avez tapé `ls` : le shell attend une nouvelle commande.

Dans le cas où le process que vous lancez a une durée de vie non négligeable, par exemple dans le cas d'une application fenêtrée sous X-Windows, mais aussi si le process est par essence très long, le fait de ne pas avoir la main dans la fenêtre où vous lancez ce process peut être gênant. On peut alors l'envoyer *en tâche de fond*, c'est-à-dire que le shell vous rend la main, tout en ayant lancé le process. Pour lancer un programme de cette manière, vous rajoutez le caractère «&» après le nom du programme :

```
nom_programme &
```

Souvent, on oublie ce «&». Rien n'est perdu ! ! Vous tapez alors CTRL Z. Le process est dit suspendu, c'est-à-dire qu'il est toujours chargé en mémoire, mais le processeur ne s'occupe plus de lui (le marchand de légumes laisse un de ses clients de coté momentanément). Ensuite pour le relancer en tâche de fond, vous tapez `bg`, et tout se passe comme si vous aviez lancé le programmme avec «&». La séquence s'écrit ainsi (avec xload comme exemple de programme) :

```
louisnar@stalingrad[UNIX] xload
^Z
Suspended
louisnar@stalingrad[UNIX] bg
[4]    xload &
louisnar@stalingrad[UNIX]
```

Il y a aussi une commande pour voir la liste des process qui ont été lancés en tâche de fond :

```
jobs
```

Cette commande vous renvoie la liste de ces process avec un numéro. Exemple :

```
louisnar@stalingrad[UNIX] jobs
[1]  + Running                 xdvi main
[3]  - Running                 xload
[4]    Running                 xload
```

Pour faire revenir ce process en tâche normale vous tapez «%» suivi du numéro voulu : vous n'avez plus la main jusqu'à ce que ce process soit fini ou que vous le suspendiez.

3.2.4 `kill` : tuer un process

`kill` *process ID*

Tue le process spécifié par le process ID. Cette commande peut être pratique quand un programme plante (ça arrive moins souvent que sous WINDOWS...), ou quand vous avez lancé par erreur un programme qui tourne d'office en tâche de fond (par exemple netscape).

`kill` envoie un signal au process concerné, les signaux pouvant être de différents types. Le signal envoyé par défaut par `kill` peut ne pas être pris en compte par le process à tuer. Dans ce cas, utilisez l'option `-9` qui est radicale !

Il va se soi que vous ne pouvez tuer qu'un process qui vous appartient.

3.3 Commandes réseau

A partir du moment où deux machines sont connectées par un réseau, qu'il soit local ou téléphonique, il est logiquement possible d'accéder à une machine à partir de l'autre. Cela signifie que vous pouvez théoriquement accéder à n'importe quelle machine UNIX du réseau Internet.

Evidemment, il faut connaître le nom de la machine, et de toutes façons, on ne vous laissera y accéder que sous certaines conditions. Notons qu'une machine qui est dans le même domaine que celui dans lequel vous travaillez est accessible par son nom simple (picpus, stalingrad). Dans le cas contraire, le nom à spécifier est celui de la machine suivi d'un point, puis du nom de domaine. Ainsi pour des gens de l'extérieur, notre machine picpus doit être référencée par `picpus.enstimac.fr`.

Il existe plusieurs programmes de base permettant d'accéder à une machine distante, d'y ou d'en transférer des fichiers, ou encore d'y exécuter une commande.

`telnet` et `rlogin` permettent d'ouvrir une session sur une machine distante. `ftp` permet de transférer des fichiers vers ou depuis une machine distante. Le protocole **ssh**, et la commande du même nom remplacent avantageusement telnet et rlogin, et sont en passe de devenir le standard.

3.3.1 `rlogin` : se loguer sur une machine distante.

`rlogin` *nom machine*

Cette commande vous permet d'ouvrir directement un shell sous votre nom sur la machine concernée. Evidemment, vous n'aurez le droit de faire un rlogin que sur une machine sur laquelle vous avez un compte (vous n'imaginez pas débarquer au pied levé dans un club sportif en Australie et obtenir directement un casier...).

Elle est donc adaptée pour travailler sur une machine de l'école qui n'est pas celle que l'on a devant soi.

3.3.2 `telnet` : se loguer sur une machine distante.

`telnet` *nom machine*

`telnet` ouvre une fenêtre de connection sur la machine spécifiée et vous affiche l'invite

```
login:
```

Le processus pour se logger est ensuite le même que si vous étiez devant la machine. Notons que de par son principe, cette commande ne doit pas être obligatoirement

lancée depuis une machine UNIX. Ainsi, le programme `telnet` existe sur PC et Macintosh.

Son intérêt principal est donc de pouvoir travailler sur une machine UNIX depuis une machine de type quelconque.

3.3.3 `ssh` : se loguer ou exécuter une commande sur une machine distante.

`ssh [-X]` *nom machine* [commande]

Si l'argument commande est omis, vous serez loggué simplement sur la machine distante (après avoir tapé votre mot de passe sur cette machine). Si vous spécifiez une commande, celle-ci sera exécutée sur la machine distante.

L'option `-X` vous permettra de plus de lancer des commandes fenêtrées sur cette machine, la fenêtre s'ouvrant sur la vôtre. Par exemple si vous comptez lancer matlab sur une machine distante appelée "chatelet" tapez :

```
ssh -X chatelet matlab
```

Outre un niveau de sécurité plus élevé [3], ssh offre de nombreux avantages que vous découvrirez dans la doc.

3.3.4 `ftp` : transfert de fichier.

`ftp` *nom machine*

`ftp` est un programme interactif qui permet de transférer des fichiers entre deux machines. Il est un peu tombé en désuétude du fait que les navigateurs type Netscape permettent de faire la même chose de manière transparente... mais moins rapide.

Les deux commandes de base sont `get` pour ramener un fichier depuis la machine distante et `put` pour envoyer un fichier vers la machine distante.

`ftp` existe sur de nombreuses machines, existe de base en mode MSDOS sous Windows. Il existe de nombreuses implémentations plus conviviales : `fetch` sur Macintosh, WsFTP sous Windows et ncftp sous UNIX.

[3] le premier "s" de ssh signifie "secure"

Chapitre 4

Conclusion

Ce document vous fournit les bases essentielles d'UNIX. Il ne traite pas des points suivants :
- l'utilisation du multi-fenêtrage X-Windows et dérivés (OpenWindows, CDE, KDE, Gnome)
- les redirections d'entrées et sorties, les pipes
- les fichiers de configuration de votre compte
- les tâches d'administration système

Soyez bien conscients qu'UNIX s'apprend en le pratiquant plus ou moins quotidiennement. Sa grande puissance réside dans sa logique simple, sa fiabilité et sa transparence vis-à-vis de l'utilisateur. Profitez de cette dernière et n'hésitez pas à allez voir comment est structuré le système.

Si vous souhaitez aller plus loin, sachez qu'il existe des versions d'UNIX gratuites, installables sur n'importe quel PC ou Mac actuel. La plus connue est LINUX (il y a aussi FreeBSD, OpenBSD,...). Ces systèmes sont disponibles sur Internet, et plus récemment sous forme de CDROM que l'on peut trouver à peu près n'importe où. L'installation de Linux est désormais aussi simple que celle de Windows, notamment au niveau du partitionnement des disques, et de la gestion de sa coexistence avec Windows sur un PC.

C'est un bon moyen de pratiquer, tant au niveau utilisateur qu'au niveau administrateur. A ce sujet, il existe un livre gratuit en Français appelé «Bien débuter sous LINUX», qui est avant tout un excellent bouquin sur UNIX en général.

Note de dernière minute : le nouveau système d'exploitation des MacIntosh, appelé "MacOS X" (X comme 10 parce qu'il succède au 9, mais aussi comme uniX !) est un vrai système UNIX, même si ça ne se voit pas au premier abord.

Bonne route sous UNIX.