Chapitre 1

UNIX et le shell

1.1 UNIX

UNIX est une famille de systèmes d'exploitations initialement développés dans les années 70 et dont le but principal était d'être

- **multitâches** : capables de faire tourner plusieurs programmes simultanément
- **multi-utilisateurs** : capables de pouvoir gérer plusieurs utilisateur sur la même installation du système d'exploitation

Même si d'autres systèmes d'exploitations (comme WINDOWS pas exemple) tendent maintenant vers ce même but, ces deux aspects étaient à l'époque originaux.

Un autre point original des systèmes d'exploitation UNIX est une organisation claire où tout est séparé en couches bien distinctes et qui ne communiquent que très peu entre elles :

- *Le noyau* correspond à la couche la plus profonde, où sont gérées les ressources principales comme la mémoire ou les entrées/sorties de bas niveau.
- *le shell* est un interpréteur qui permet à un utilisateur ou à des programmes d'exécuter des instructions plus élaborées (en ligne de commandes).
- *le système graphique* est une sur-couche qui permet un environnement graphique plus facile à manipuler pour l'utilisateur
- *les programmes* n'apparaissent que dans la couche le plus superficielle

Cette séparation en couches isolées évite en principe qu'un programme puisse faire planter l'ensemble du système d'exploitation.

Les systèmes UNIX incluent des systèmes d'exploitation libres et gratuits (les systèmes LINUX) ainsi que des systèmes propriétaires et payants (OSX de Apple par exemple). Au contraire, WINDOWS n'est pas un système UNIX.

1.2 Le shell

Le shell est un interpréteur de commande. Il permet en particulier à un utilisateur d'envoyer des instructions de base au système d'exploitation. On y accède via un **terminal** ou une **console**. Une fois ouvert, ce terminal affiche un **prompt** qui invite l'utilisateur à taper une commande :

```
>
```

Il existe plusieurs langages différents (par ex. bash, csh, tcsh, ksh...) mais les commandes le plus élémentaires sont communes à tous ces langages. De manière générale, on exécute une commande en tapant simplement son nom puis en enfonçant la touche *"entrer"* :

```
>commande
```

Par exemple, la commande `>whoami` affiche le nom de l'utilisateur courant.

Certaines commandes plus complexes acceptent des arguments séparés par des espaces ainsi que des options signalées par le symbole "-". Ainsi par exemple :

```
>commande argument
>commande arg2 arg2 arg2
>commande -option
>commande -opt1 -opt2
>commande arg1 -opt1 arg2 -opt2
```

Parmi les commandes générales, on peut retenir l'instruction `man` dont la syntaxe est la suivante :

```
>man commande
```

Cette instruction permet d'afficher une page de manuel sur la commande indiquée. Y sont en particulier mentionnés le but de cette commande, sa syntaxe, ainsi que les différents arguments et options qu'elle admet. La suite de ce chapitre est consacrée à la description de quelques commandes très utiles pour l'utilisation que nous allons faire du shell.

1.3 Arborescence

Comme tous les systèmes d'exploitation, les systèmes UNIX possèdent une arborescence de répertoires et de fichiers hiérarchisés.

• Les noms de fichiers possèdent souvent une extension qui permet d'identifier facilement le type de fichier dont il s'agit (par ex : .c pour un programme C, .f, .f77, .f90 pour un programme fortran, .txt, .dat, .data pour des fichiers ascii, .x, .out, .exe pour des exécutables...).

• Chaque fichier ou répertoire est repéré par une chaîne de caractères qui contient sa localisation dans l'arborescence (son chemin ou **path**) et son nom. Le symbole `/` est utilisé pour marquer la hiérarchie de répertoires. Ainsi, `rep1/rep2/rep3/fich` désigne le fichier de nom `fich` contenu dans le répertoire de nom `rep3`, lui même contenu dans `rep2`, lui même contenu dans `rep1`. En outre, certains répertoires particuliers sont repérés par des symboles spécifiques :

`./`	le répertoire courant
`../`	le répertoire en amont (celui qui contient le répertoire courant)
`/`	la racine (commune à tous les utilisateurs)
`~/`	le home (le répertoire de base de l'utilisateur courant)

• Lorsqu'un terminal est ouvert, il se trouve par défaut dans le répertoire principal de l'utilisateur courant (le home `~/`). Ensuite, on peut naviguer dans les répertoires avec les commandes adaptées :

`>pwd` *print working directory*
Affiche le nom du répertoire courant et son chemin.

`>ls` *list*
Affiche la liste de tous les fichiers et répertoires contenus dans le répertoire courant. Cette fonction admet de nombreuses options (`ls` -l -t -a). Cette commande permet aussi d'afficher le contenu d'un répertoire spécifique lorsque celui-ci est précisé : `>ls répertoire/`

`>cd rep` *change directory*
permet de naviguer vers le répertoire `rep`. En particulier, `>cd ./` ne fait rien, `>cd ../` remonte d'un répertoire dans la hiérarchie, `>cd /` va à la racine, et `>cd ~/` va dans le home de l'utilisateur.

Mise en pratique :
- afficher le chemin du repertoire courant
- remonter d'un répertoire dans l'arborescence
- revenir dans le répertoire de départ

1.4 Manipulation de fichiers et répertoires

Les fichiers et répertoires peuvent être manipulés à volonté. En particulier :

`>mkdir rep` *make directory*
Crée un répertoire de nom `rep` dans le répertoire courant.

`>rmdir rep` *remove directory*
Efface le repertoire de nom `rep` (uniquement s'il est vide)

`>touch fich` *touch*
Crée un fichier ascii (texte) de nom `fich` s'il n'existe pas, change la date de dernière modification s'il existe déjà.

`>rm fich` *remove*
Efface le fichier de nom `fich`. **Attention**, avec le shell, il n'existe pas de corbeille. **Tout fichier effacé est irrémédiablement perdu !!**

`>cp fich1 fich2` *copy*
Fait une copie du fichier `fich1` et la nomme `fich2`

`>mv fich rep/` *move*
Déplace le fichier de nom `fich` dans le répertoire de nom `rep`. Si le deuxième argument est un nom de fichier (`>mv fich1 fich2`), alors cette commande renomme `fich1` avec le nom `fich2`.

Mise en pratique :
- créer un répertoire `M1/` dans le home
- créer un répertoire avec un nom propre au binôme dans le répertoire `M1/`
- créer un fichier texte vide `toto.txt` dans le répertoire du binôme
- créer une copie de `toto.txt` appelée `titi.txt`
- effacer ces deux fichiers et vérifier qu'ils ne sont plus là.

1.5 Astuces indispensables

Un certain nombre de commandes sont souvent (mais pas toujours) interprétées par les différents shell et permettent des gains de temps précieux :

`>clear`	Efface toutes les lignes apparentes du terminal.
>↑	La flèche vers le haut permet de rappeler les dernières commandes exécutées.
→|	La tabulation permet d'auto-compléter les débuts de commande ou de noms de fichiers.
`*`	Le symbole `*` permet d'appliquer des opérations à des ensembles de fichiers qui possèdent un ensemble de caractères communs (par ex, `>ls *.txt` permet de lister tous les fichiers dont le nom finit par `txt`.). **Attention : `>rm *` efface irrémédiablement tout le contenu d'un répertoire !!!**
`ctrl+C`	La combinaison de touches `ctrl+C` permet de stopper arbitrairement une commande même si celle-ci n'a pas fini de tourner. Lorsque la commande appelée correspond à un programme, elle permet de quitter brutalement le programme.
`>commande &`	Le symbole `&` à la fin d'une commande permet de lancer cette commande en *tâche de fond*. Ce faisant, le terminal qui a servi à lancer la commande reste accessible pour lancer de nouvelles commandes même si la première n'a pas fini de tourner. Sans ce symbole, et lorsque que la commande a lancé un programme, il faut quitter le programme pour avoir de nouveau accès au shell.

Mise en pratique :
- créer 2 fichiers `toto1.txt` et `toto2.txt`
- créer 2 fichiers `titi1.txt` et `titi2.txt`
- afficher tous les fichiers qui commencent par `toto`
- effacer tous les fichiers qui contiennent le caractère `2`, et vérifier le résultat
- effacer tous les fichiers du repertoire du binôme

1.6 Editeurs de texte

Les programmes peuvent être lancés en les appelant par leur nom. En particulier, il existe plusieurs éditeurs de texte installés sur les machines à votre disposition (kwrite, gedit ; emacs, xemacs, vi, vim...) qui peuvent être lancés avec leur nom. Par exemple :

`>kwrite &`	ouvre l'éditeur kwrite avec un nouveau fichier
`>kwrite fich &`	ouvre le fichier de nom `fich` avec l'éditeur kwrite