# TP1 : Shell

Il existe de nombreux shells : `sh`, `zsh`, `ksh`, `bash`, `csh`, `tcsh`, etc. Leur syntaxe varie beaucoup. L'énoncer de ce TP et les corrections seront donnés pour le shell `bash` uniquement. Il est assez répandu (c'est le shell par défaut de GNU/Linux et MacOS, entre autres). Vous pouvez, bien sûr, faire le TP avec n'importe quel shell.

## 1 Utilisation du shell

### 1.1 La commande `man`

La commande `man` permet d'afficher les pages de manuel. Toutes les pages de manuel on la même structure :

- le nom de la commande, fonction, avec une courte description ;
- la syntaxe d'utilisation :
- une description détaillée ;
- ...
- une rubrique "SEE ALSO".

Pour plus d'information et apprendre à naviguer dans les pages, vous pouvez faire : `man man` et `man less`.

**Question 1.** Quelles sont les différences entre les deux commandes suivantes :

```
man 1 unlink
man 2 unlink
```

### 1.2 Le mécanisme d'expansion

En shell, les caractères suivants sont des méta-caractères : `?`, `*`, `[`, `!`. Ils permettent de définir des motifs qui serons expansés par la liste des fichiers les satisfaisant. Ces caractères ont la sémantique suivante :

| | |
|---|---|
| `?` | Un caractère quelconque sauf `/` et sauf `.` en première position ou derrière un `/` |
| `*` | Une chaîne de caractères quelconque (éventuellement vide) mais ne commençant pas par `.` et sans `/` |
| `[` | Début de définition d'un ensemble |
| `[!` | Début de définition d'un ensemble par complémentation |
| `]` | Fin de définition d'un ensemble |
| `-` | Marque d'intervalle dans un ensemble |

**Question 2.** Utiliser la commande `echo` et le mécanisme d'expansion pour afficher successivement tous les fichiers du répertoire `/bin/` qui :

- commencent par un `c` ;
- ont exactement trois caractères ;
- ont soit la lettre `a` soit la lettre `e` comme avant dernière lettre.

### 1.3 Exécution en tache de fond

**Question 3.** Exécuter la commande suivante, décrire ce qui se passe et pourquoi cela se passe ainsi.

```
echo Debut & seq 1000 & echo Fin &
```

### 1.4 Redirections et pipes

**Question 4.** Que fait la commande suivante ?

```
cat < /etc/passwd | tee /tmp/a | sort >> /tmp/aa
```

**Question 5.** De même, commenter l'exécution de :

```
ls /etc/*.conf | cat
ls /etc/*.conf | xargs cat
```

### 1.5 Les variables

En `bash`, on peut définir des variables et les consulter de la façon suivante :

```
export VAR="ma variable"
echo $VAR
```

Le shell définit un ensemble de variables. On peut afficher la liste des variables exportées avec la commande `env`.

**Question 6.** Exécuter les commandes suivantes et en déduire le rôle des variables `PS1`, `PWD`, `OLDPWD` et `HOME`.

```
export PS1=":-) "
cd /tmp
cd /etc
echo $PWD
echo $OLDPWD
echo $HOME
cd
echo $PWD
cd -
echo $PWD
```

**Question 7.** Exécuter les commandes suivantes et en déduire le rôle de la variable `PATH`.

```
bash
cd /tmp
ls
echo $PATH
export PATH=/tmp:$PATH
echo $PATH
echo "echo Hello" > ls
chmod u+x ls
ls
rm ls
export PATH=""
ls
exit
```

### 1.6 Les guillemets

**Question 8.** Exécuter les commandes suivantes et en déduire la différence entre guillemets : `"..."`, `'...'` et `` `...` ``.

```
echo "ls $HOME"
echo 'ls $HOME'
echo `ls $HOME`
```

## 2 Programmation en shell

On rappelle qu'un script shell est un fichier texte d'extension `.sh`, comme `toto.sh`. Un tel script peut être exécuté par la commande `sh toto.sh`. Une autre alternative est de faire commencer le script par une ligne précisant l'interpréteur :

```
#!/bin/sh
```

Le fichier est ensuite rendu exécutable par la commande `chmod +x toto.sh` et puis exécuté par `./toto.sh`.

**Question 9.** Écrire un script shell qui affiche `Hello World!` à l'écran.

**Question 10.** Définir un script shell `est_pair` qui attend un argument et teste si cet argument est pair. Si l'argument est pair, il affiche sur la sortie standard `pair` et termine avec le code de retour `0`. Si l'argument est impair, il affiche sur la sortie standard `impair` et termine avec le code de retour `1`.

Dans tous les autres cas, il affiche `Entree incorrecte` sur la sortie d'erreur standard et termine avec le code de retour `2`.

**Question 11.** Nous voulons maintenant tester notre script. Pour cela, on veut créer des répertoires `good` et `bad` dans lesquels il y a des fichiers vides qui ont pour nom des entiers.

À l'aide des commandes `seq` et `touch` créer 100 fichiers dont le nom est un nombre pair dans le répertoire `good` et 100 fichiers dont le nom est un nombre impair dans le répertoire `bad`.

**Question 12.** Écrire un script qui teste `est_pair` sur tous les fichiers qui sont dans `good` et `bad`. Ce script doit seulement afficher le nombre de tests réussis et les noms de fichier sur lesquels le programme a échoué.

**Question 13.** Lorsqu'une commande prend en argument un chemin mais que celui-ci n'existe pas, un message d'erreur est affiché, par exemple :

```
--> ls /usr/lib/thissoft/config/Makefile
/usr/lib/thissoft/config/Makefile: No such file or directory.
```

Écrire un script shell verif qui affichera la partie gauche du chemin qui est valide et la première entrée inexistante :

```
--> verif /usr/lib/thissoft/config/Makefile
/usr/lib/thissoft: No such directory.
```

**Question 14.** Écrire un script shell permettant d'afficher l'arborescence d'un répertoire. Pour plus de lisibilité, on affichera ici uniquement les noms des répertoires et on omettra les noms des fichiers.

**Question 15.** Écrire un script `jeter` qui permet de manipuler une poubelle à fichier (un répertoire nommé `poubelle` situé à votre racine). La commande accepte trois options :

- `jeter -l` pour lister le contenu de la poubelle ;
- `jeter -s fichier chemin` pour sortir le fichier de la poubelle et le placer à l'emplacement chemin ;
- `jeter -v` pour vider la poubelle ;
- `jeter fichier1 fichier2 ...` pour déplacer les fichiers considérés vers la poubelle ;

Si la poubelle n'existe pas, elle est créée à l'appel de la commande.