# Plus loin avec le bash

## 1. Alias

Un alias est un raccourci d'une commande avec d'éventuels paramètres. Il se définit avec la commande **alias**. Utilisée seule elle liste les alias disponibles.

```
$ alias
alias ..='cd ..'
alias ...='cd ../..'
alias cd..='cd ..'
alias dir='ls -l'
alias l='ls -alF'
alias la='ls -la'
alias ll='ls -l'
alias ls='ls $LS_OPTIONS'
alias ls-l='ls -l'
alias md='mkdir -p'
alias o='less'
alias rd='rmdir'
...
```

Vous pouvez créer vos propres alias.

```
$ alias deltree='rm -rf'
```

## 2. Groupement de commandes

Le chaînage de commande est possible avec « ; ». Il est aussi possible de grouper les commandes. Quand vous exécutez les commandes suivantes :

```
$ uname -a ; pwd ; ls -l >resultat.txt &
```

Seule la dernière commande est exécutée en tâche de fond et seul son résultat est redirigé dans le fichier resultat.txt. Une solution serait :

```
$ uname -a >resultat.txt & ; pwd >>resultat.txt & ; ls -l >>resultat.txt &
[1] 18232
[2] 18238
[3] 18135
```

C'est une solution complexe et qui ne fonctionnera pas toujours. De plus même si les commandes sont lancées séquentiellement, elles tournent toutes en parallèle et c'est la première finie qui écrira en premier dans le fichier. La solution consiste en l'utilisation des parenthèses.

```
$ (uname -a ; pwd ; ls -l) > resultat.txt &
[1] 18239
$
[1]    Done                 ( uname -a; pwd; ls -l ) > resultat.txt
```

Dans ce cas, toutes les commandes placées entre les parenthèses sont lancées par un sous-shell, qui va ensuite exécuter les commandes précisées séquentiellement telles qu'indiquées. Ainsi la redirection concerne l'ensemble des commandes et rien n'empêche de lancer ce sous-shell en arrière-plan. Vous distinguez bien d'ailleurs un seul PID 18239 lors de l'exécution des commandes.

Une seconde possibilité est l'utilisation des accolades {...}. Dans ce cas aucun sous-shell n'est exécuté, et si une commande interne (cd ou autre) est exécutée, elle concerne le shell actif. L'accolade fermante doit être placée juste après un ;.

```
$ { uname -a; pwd; ls -l; } > resultat.txt
```

Vous pouvez faire facilement la différence entre les deux syntaxes avec exit. Le premier exemple semble ne rien faire, alors qu'il quitte le shell fils. Le second sort de votre shell.

© ENI Editions - All rights reserved - Bruno Suarez

```
$ (exit)
$ { exit; }
```

Attention avec les parenthèses, notamment en programmation. Comme le groupement est lancé au sein d'un autre processus, les éventuelles variables modifiées au sein du groupement ne seront pas visibles une fois l'exécution terminée.

## 3. Liaison et exécution conditionnelle

En plus du chaînage classique, les commandes peuvent être liées et exécutées de façon conditionnelle. La condition d'exécution d'une commande est la réussite ou non de la précédente. Chaque commande une fois exécutée renvoie un code de retour, généralement 0 si tout s'est bien passé, 1 ou 2 en cas d'erreur. Le shell peut récupérer cette valeur par la variable $?.

```
$ ls
...
$ echo $?
0
```

Les caractères **&&** et **||** permettent d'effectuer une exécution conditionnelle.

```
commande1 && commande2
commande1 || commande2
```

La commande située après **&&** sera exécutée uniquement si la commande précédente a retourné 0 (réussite). La commande située après **||** ne sera exécutée que si la commande précédente a retourné autre chose que 0.

```
$ grep "souris" liste && echo "Souris trouvee" || echo "Souris introuvable"
souris  optique 30      15
Souris trouvee
$ grep "memoire" liste && echo "Memoire trouvee" || echo "Memoire
introuvable"
Memoire introuvable
```


© ENI Editions - All rights reserved - Bruno Suarez