

0.1 Bash : Approfondir

Ce tutoriel fait suite à celui-ci¹, qui avait pour objectif de vous présenter les bases de la programmation Bash. Ici, je vais vous présenter quelques notions plus avancées qui vous permettront d'améliorer vos scripts et d'élargir leur champ d'action.

0.1.1 1. Redirections :

Avant qu'une commande ne soit exécutée, il est possible de rediriger son entrée et sa sortie en utilisant une notation spéciale interprétée par le shell. Les redirections peuvent également servir à ouvrir ou fermer des fichiers dans l'environnement actuel du shell. Ici, le mieux est de vous montrer quelques exemples. Vous comprendrez tout de suite.

- **Redirection d'entrée (<)** : ici la commande "grep" prend en *entrée standard* le contenu du fichier "/etc/passwd" et y recherche la ligne contenant le motif "root".

```
$ grep root < /etc/passwd
root :x :0 :0 :root :/root :/bin/bash
```

- **Redirection de sortie (>)** : dans le premier exemple, la commande "echo" ajoute à "mon_fichier" (qui est créé s'il n'existe pas) une chaîne de caractère. S'il n'y a qu'un seul symbole '>', le contenu du fichier est écrasé, et s'il y en a deux ('>>'), le flux est ajouté à la fin du fichier. Ainsi, les deux exécutions du "cat" mettent le contenu de deux fichiers dans un seul.

```
$ echo "Bonjour" > mon_fichier
$ cat fichier_1 > fichier
$ cat fichier_2 » fichier
```

- **Redirection des erreurs (2>)** : on peut aussi rediriger les messages d'erreurs vers un fichier, ou, dans notre cas, vers la poubelle (/dev/null).

```
$ mozilla 2> /dev/null
```

0.1.2 2. Les expressions arithmétiques :

Bash inclut en natif, un système de calcul arithmétique (donc uniquement des calculs avec des nombres entiers) très simple à utiliser :

```
$(( nombre_1 opération nombre_2))
```

La liste des opérations est la suivante :

- +, -, *, / : addition, soustraction, multiplication, division
- % : reste de la division entière
- ** : exponentiel

Exemple :

```
#!/bin/bash
a=2
b=3
c=$(( a*b +b ))
#retourne "2 puissance 3 plus 3" = "11"
echo $c
```

¹<http://www.trustonme.net/didactels/148.html>

Remarque : à l'intérieur des doubles parenthèses, les noms des variables n'ont pas besoin d'être prefixés par le caractère \$. Vous pouvez néanmoins le faire quand même, les deux syntaxes seront acceptés.

0.1.3 3. Les expressions booléennes :

La liste des opérations booléennes accessibles sous Bash :

- ==, !=, <, >, >=, <=, ! : comparaisons
- &, | : ET, OU binaire
- &&, || : ET, OU logique
- ^ : OU exclusif binaire
- «, » : décalage arithmétique à gauche et à droite

Exemple :

```
#!/bin/bash
nombre_1=25
nombre_2=390
if (( nombre_1 >= nombre_2 )) ; then
echo "nombre_1 est superieur ou egale a nombre_2"
fi
```

Plus couramment, il sera intéressant d'utiliser les connecteurs logiques pour lancer des commandes. Ainsi :

- **cmd_1 && cmd_2** : "cmd_2" ne sera exécuté que si "cmd_1" se termine avec succès.
- **cmd_1 || cmd_2** : "cmd_2" ne sera exécuté que si "cmd_1" se termine par un échec.
- **cmd_1 | cmd_2** : l'entrée de "cmd_2" est la sortie de "cmd_1"

Remarque : à l'intérieur des doubles parenthèses, les noms des variables n'ont pas besoin d'être prefixés par le caractère \$. Vous pouvez néanmoins le faire quand même, les deux syntaxes seront acceptés.

0.1.4 4. Les expressions régulières :

- * désigne une chaîne de caractères quelconque ne commençant pas par un metacaractère .
- ? désigne un caractère quelconque.
- [aA] désigne les caractères A et a
- [0-9a-zA-Z] désigne un caractère alphanumérique quelconque.
- [!0-9] désigne l'ensemble des caractères qui ne sont pas des chiffres.
- ; sépare deux commandes (ou plus) situées sur une même ligne.

Exemple :

```
# afficher tous les fichiers ayant pour extension .avi ou .AVI ou .aVi ...
$ ls *.[aA][vV][iI]
```

Pour la substitution ou la sélection de motifs dans une chaîne de caractère :

- **\${variable :-mot}** retourne le contenu de la variable si cette dernière est non vide, et remplace par le "mot" sinon
- **\${variable :n}** retourne le contenu de la variable en supprimant les "n" premiers caractères

- `${variable:n:longueur}` extrait de la variable la chaîne de caractères commençant au "n-ième" caractère et de longueur "longueur"
- `${variable/motif/chaîne}` permet de faire une substitution dans la variable en remplaçant le "motif" contenu dans la "variable" par la "chaîne" de caractères

Exemples :

```
# affiche "Rien" car la variable n'a pas encore été créée
$ echo ${variable :-Rien}
variable="Bonjour tout le monde"
# affiche "tout le monde"
$ echo ${variable :8}
# affiche "Bonjour"
$ echo ${variable :0 :7}
# affiche "Bonsoir tout le monde"
$ echo ${variable/Bonjour/Bonsoir}
```

0.1.5 5. Divers :

Voici une petite liste de commandes , variables ou de syntaxes pouvant être utiles :

- **exit** vous permet de sortir d'un programme en cours d'exécution.
- **\$IFS** est le Séparateur de Champs Interne (Internal Field Separator) qui est utilisé pour séparer les mots après les développements, et pour découper les lignes en mots avec la commande interne read. La valeur par défaut est "espace"- "tabulation"- "retour-chariot".
- (et) permettent de regrouper un ensemble de commandes et de les exécuter dans un "shell fils"
- **select name [in word] ; do list ; done** permet de créer une liste de sélection automatiquement

Ce tutoriel est loin d'être exhaustif sur le sujet, vous trouverez beaucoup d'autres informations en lisant le manuel utilisateur ("man bash") par exemple. Mais avec toutes ces notions vous pourrez déjà construire des scripts intéressants. Le Bash n'est pas un langage bas ou moyen niveau comme l'assembleur ou le C, mais un langage de script, ne vous méprenez donc pas sur son utilisation et sur ses fonctionnalités. Bon scriptage ...