Les scripts BASH

Introduction

Les taches d'un administrateur sont variées, et dans un souci de moindre effort, il est indispensable de pouvoir écrire des scripts qui vont nous permettre de résoudre bien des problèmes relativement aisément.

On peut citer par exemple :
Créer les utilisateurs à partir d'un fichier excel contenant les coordonnées des personnels
Effectuer un nettoyage des fichiers temporaires de chacun des utilisateurs
Etc…

Mais qu'est-ce qu'un script BASH ?

On peut dire qu'un script BASH est une suite de commandes shell regroupées au sein d'un fichier ASCII exécutable.
Ce fichier peut soit être exécuté

- à partir de l'interpréteur de commande shell
- à partir d'un appel issu d'un autre script BASH

Mais le shell bash est aussi un langage interprété qui possède :

- Une déclaration de variables
- Des structures conditionnelles
- Des instructions de boucle qui permettent un traitement itératif
- La création de fonctions
- Des instructions d'E/S interactives

Comment créer un script

On crée un script en utilisant un éditeur de texte simple. Le contenu de ce fichier devra être interprété, il s'agit donc de ne pas utiliser de caractères de formatage particulier.
Vi est l'éditeur présent sur toute machine unix. Des éditeurs de prise en main plus simple sont disponibles (pico, gedit) mais vi est un incontournable.

Un script commence généralement en indiquant quel sera l'interpréteur à utiliser ainsi que son chemin dans l'arborescence. Pour cela, on commence par :
#!/bin/bash

Une ligne commençant par # est une ligne de commentaire. Ne pas hésiter à utiliser le maximum de commentaires possibles, permettant ainsi d'augmenter la qualité du code.

Exemple :

```
#!/bin/bash
```

```
#mon script sera interprété par bash
echo bonjour
# echo affiche sur le périphérique de sortie standard la \
chaine bonjour
```

Lorsque l'on veux debugger un script, on peut lui demander d'afficher les commandes simples lors de leur exécution en insérant une commande set +x en début de script et en terminant le script par un set +x.

Les fichiers d'initialisation

3 fichiers sont exécutables et contiennent des instructions en shell bash:

- .login : ce fichier est exécuté à la connexion. Il permet de définir des variables d'environnement.
- .bashrc : ce fichier est exécuté à chaque exécution d'un shell (sh). Il permet de définir des variables internes au shell bash ainsi que des alias.
- .logout : exécuté à la sortie de la session

Les fonctions

Le bash permet d'utiliser des fonctions. En général, on utilise des fonctions pour réaliser des taches particulières qui sont issues d'un découpage fonctionnel. Une fonction rend donc un service au sens programmation.
On distingue la déclaration de la fonction et son corps (ce qu'elle fait) de son appel.

Déclaration
2 manières, soit avec ou sans le mot clé **function**

```
Function nom_fonction {
     Bloc d'instructions
}
nom_fonction() {
     Bloc d'instructions
}
```

Appel

```
Nom_fonction()
```

Il est bien évidement possible de passer des paramètres. Ceci s'effectue de la même manière que dans un script.
Comme en C, il est possible de déclarer des variables à l'intérieur de la fonction, si elles sont précédées du mot clé **local**, alors ces variables seront locales à la fonction (inconnues dans le programme appelant).

Boucle for

Syntaxe :

```
for x [ in list ]
do
     Commandes
done
```

exemple :
for x in 1 2 3
do
echo $i
done

for x in `ls /home`
do
echo "sauvegarder $x"
done

Boucle while

syntaxe

```
while
     test
do
     commandes
done
```

Exemple :

```
while [ $reponse != "oui" ]
do
      echo "repondre oui"
      read reponse
done
echo "ok"
```

Instruction break et continue
Dans le corps d'une boucle, l'utilisation de l'instruction break a pour effet de sortir de la boucle.
Dans le corps d'une boucle, l'utilisation de l'instruction continue pour effet de passer à l'itération suivante.
Exemple :

```
for x in 1 2 3
do
if [ i eq 1 ] continue
```

```
echo $i
done
```
Seul seront affiché les valeurs 2 et 3

Instruction exit
L'instruction exit permet de sortir du script avec le code retour spécifié. En l'absence de core retour, c'est la valeur de la dernière commande qui est utilisée.
```
exit 0 ou exit
```

La commande exec
Exec [commande]
La commande est exécutée à la place du shell courant (il n'y a pas de création de nouveau processus).
Il est également possible d'utiliser exec pour rediriger les entrées/sorties vers un fichier
```
exec > /tmp/fichier
Ls
…
exec < /tmp/fichier2
Read debut fin
….
exec
```
Le résultat des commandes est redirigé une fois pour toute vers fichier1, et le script lit les informations à partir du fichier fichier2; Lors de l'appel à exec sans paramètre, les entrées sorties reprennent leur valeur standard.

La commande set
Elle permet de remplacer le contenu des variables d'arguments ($0,$1,) par le résultat d'une commande. Ceci permet notamment d'éviter d'avoir à couper les lignes avec cut.

La commande shift
Cette commande (utilisée généralement dans les boucles) permet de décaler le contenu des variables $0, $1 etc. Si les variables $1 contient 1 et $2 2, après l'utilisation de shift, $1 contiendra 2 et $2 sera indéfinie.

Substitution de commande
Il est souvent nécessaire dans un script de traiter le contenu des variables. Pour cela, on dispose du remplacement de variable.
On peut protéger le nom de la variable en le plaçant entre {}
Exemple :
```
Mot=chat
Echo le pluriel de $mot est ${mot}s
Le pluriel de chat est chats
```

Mais il existe bien d'autres options associées aux accolades.

${var:-argument}	Donne une valeur par défaut : si non affectée vaudra argument
${var:=argument}	Idem précédemment avec création de la variable
${#var}	Donne la longueur de la chaine
${var%chaine}	Extrait le début, jusqu'au dernier "chaine". Par exemple, si var vaut /usr/local/bin/local et chaine vaut local*, on obtiendra /usr/local/bin
${var%%chaine}	Extrait le début, jusqu'au premier "chaine". Par exemple, si var vaut /usr/local/bin/local et chaine vaut local*, on obtiendra /usr/
${var#chaine}	Même principe mais en partant de la fin (avec option ## également)

L'utilisation de parenthèses après $ permet d'évaluer les valeurs arithmétiques:
Exemple :
$(($var +2)) permet d'accéder au contenu d'une variable nommée $3 si $var contient la valeur 2

TP script : boucle for et substitution

Objectif : A l'issue de la séance vous saurez :

Parcourir le contenu d'un fichier par une boucle for

Traiter un sous ensemble d'une chaine de caractère

Boucle for

Soit le fichier équipe de France et de nouvelle zélande. Une équipe commende par une ligne contenant Equipe_. Suivent alors les noms des joueurs, un par ligne.

Partie 1 : Afficher chaque ligne

Réaliser un script qui par une boucle for affichera chaque joueur.

1°) quelle est la commande qui permet d'afficher chaque ligne d'un fichier?
2°) Utiliser une boucle for pour affecter à la variable x, le contenu de chaque ligne du fichier
3°) dans le corps de la boucle, afficher le contenu de la variable x

Partie 2 : Ne pas afficher les équipes

Modifier le script précédent pour ne pas afficher le nom des équipes (ligne débutant par equipe_???)
1°) Par une instruction conditionnelle, tester si le contenu de la ligne commence par le mot equipe.
Pour ne comparer que sur un sous ensemble de la chaine, on utilisera la substitution ${}.
2°) En utilisant une instruction continue, n'afficher que les lignes contenant un nom de joueur.

Correction

#!/bin/sh

```
# on parcourt le fichier des equipes/joueurs
for x in $(cat rugby3)
 do
# si c'est une equipe on cree le groupe
if [ ${x%%_*} = 'equipe' ]
      then echo traitement groupe $x
            y=$x
            useradd -D -b /home/$y
            groupadd $y
            mkdir /home/$y
# lorsque l'on a trouvé une equipe, passe a l'utilisateur
            continue
fi
useradd -g $y $x
done
```