Programmation Shell

Introduction

- Le shell peut être utilisé comme un simple *interpréteur de commande*, mais il est aussi possible de l'utiliser comme *langage de programmation interprété* (scripts).
  - **sh** : Bourne Shell. L'ancêtre de tous les shells.
  - **bash** : Bourne Again Shell. Une amélioration du Bourne Shell, disponible par défaut sous Linux et Mac OS X.
  - **ksh** : Korn Shell. Un shell puissant assez présent sur les Unix propriétaires, mais aussi disponible en version libre, compatible avec bash.

Les variables d'environnements, les variables locales

- **Les variables d'environnement** permettant de paramétrer le fonctionnement du système (langue utilisé, chemins vers les fichiers, exécutables, chemin vers les librairies, etc)
- Lister les variables dites d'environnement : **env** sans aucun paramètre.
- **La variable locale** ne sera vue que par le shell où elle a été déclarée
- Déclarer une variable locale, utiliser simplement **MAVARIABLE=SAVALEUR**
- Effacer une variable, utiliser : **unset MAVARIABLE**
- Déclarer une variable qui sera vue par les shells fils , utilisez export. **export MAVARIABLE=SAVALEUR**
- **set** affichent les variables définies par le shell avec leurs valeurs. **set** affiche en plus les variables et les fonctions définies par l'utilisateur.

Variables d'environnement

- Utiliser la commande **env** pour afficher les variables globales
- **SHELL** : indique quel type de shell est en cours d'utilisation (sh, bash, ksh...)
- **PATH** : une liste des répertoires qui contiennent des exécutables que vous souhaitez pouvoir lancer sans indiquer leur répertoire.
- **EDITOR** : l'éditeur de texte par défaut qui s'ouvre lorsque cela est nécessaire.
- **HOME** : la position de votre dossier home.
- **PWD** : le dossier dans lequel vous vous trouvez.

Variables de substitution prédéfinies

- **$0** nom du script (pathname)
- **$1, ..., $9** arguments (du 1er au 9ème)
- **$#** nombre d'arguments
- **$*** liste d'arguments sous format **" $1 $2 $3 …."**
- **$@** liste d'arguments sous format **" $1" "$2 " "$3" …‘**
- **$?** code retourné par la dernière commande
- **$$** numéro de processus de ce shell
- **$!** numéro du dernier processus en arrière plan

Script Shell

- **Shell : Un Langage de programmation :**
  - La gestion des variables
  - Les tests
  - Les boucles
  - Des opérateurs
  - Des fonctions
- **Script Shell :**
  - Chaque ligne sera lue est exécutée.
  - Une ligne peut se composer de commandes **internes** ou **externes**, des commentaires ..
  - Exécutable : **+x**
  - La première ligne précise quel shell va exécuter le script. :
    - **# !/bin/bash**
    - **# !/bin/ksh**

Exemple

- **voir_a.sh**

```
#!/bin/bash
echo a=$a
a=bonjour
echo a=$a
```

- **$ a=salut**
- **$ ./voir_a.sh**

test

- Deux syntaxes :
- test expression ou bien [expression]
- Le résultat est récupérable par la variable $?
- **Il faut respecter un espace avant et après les crochets**

Tests sur les chaînes

- **Attention : placer les variables contenant du texte entre guillemets.**

Condition	Signification
-z *chaine*	vrai si la *chaine* est vide
-n *chaine*	vrai si la *chaine* est non vide
chaine1 = *chaine2*	vrai si les deux chaînes sont égales
chaine1 != *chaine2*	vrai si les deux chaînes sont différentes

Tests sur des nombres

Condition	Signification
num1 -eq *num2*	égalité
num1 -ne *num2*	inégalité
num1 -lt *num2*	inférieur (<)
num1 -le *num2*	inférieur ou égal (< =)
num1 -gt *num2*	supérieur (>)
num1 -ge *num2*	supérieur ou égal (> =)

Tests sur des fichiers

Condition	Signification
-e *filename*	vrai si *filename* existe
-d *filename*	vrai si *filename* est un répertoire
-f *filename*	vrai si *filename* est un fichier ordinaire
-L *filename*	vrai si *filename* est un lien symbolique
-r *filename*	vrai si *filename* est lisible (r)
-w *filename*	vrai si *filename* est modifiable (w)
-x *filename*	vrai si *filename* est exécutable (x)
file1 -nt *file2*	vrai si *file1* plus récent que *file2*
file1 -ot *file2*	vrai si *file1* plus ancien que *file2*

if

- Syntaxe

```
if suite_de_commandes1
        then
                suite_de_commandes2
        [ elif suite_de_commandes ; then suite_de_commandes ] ...
        [ else suite_de_commandes ]
fi
```

- **Exemple**

```
#!/bin/bash
if [ $1 = "Salah" ]
then echo "Salut Salah !"
else
echo "Je ne te connais pas !"
fi
```

Choix multiples case

- Permet de vérifier le contenu d'une variable.
- Syntaxe :

```
case Valeur in
        Modele1) Commandes ;;
        Modele2) Commandes ;;
        *) action_defaut ;;
esac
```

Boucle for

- Première forme :

```
for var
do
        suite_de_commandes
done
```

- Deuxième forme :

```
for var in liste_mots
do
        suites_de_commandes
done
```

Boucle for (suite)

- Exemple 1

```
#!/bin/bash
for i in `seq 1 10`;
do
        echo $i
done
```

- Exemple 2

```
#!/bin/bash
for variable in 'valeur1' 'valeur2' 'valeur3'
do
        echo "La variable vaut $variable"
done
```

Boucle while

- **Syntaxe :**

```
while suite_cmd1
do
        suite_cmd2
done
```

- **Exemple :**

```
while
        echo "tapez quelque chose : "
        read mot
        [ $mot != "fin" ];
do
        echo "vous avez tapé $mot"
        echo "tapez \"fin\" pour finir";
done
```

fonction

- Syntaxe

```
function nom_fct
{
        suite_de_commandes
}
```

- Exemple

```
$ function f0
> {
> echo Bonjour tout le monde !
> }
$
```

- Appel de la fonction f0

```
$ f0
Bonjour tout le monde !
```

Expressions arithmétiques : expr

- **expr** : Ancienne commande.
- **Opérateurs arithmétiques**

```
$ expr 2 + 3
$ expr 2 – 3        $ expr 2 + 3 \* 4
$ expr \( 2 + 3 \) \* 4
```

- **Opérateurs logiques :**

```
$ expr 2 = 2
1
$ echo $?
0
$ expr 3 \> 6
0
$ echo $?
1
```

Expressions arithmétiques : let, (())

- **Plus rapide**
- **Syntaxe mois contraignate** : Pas nécessaire de mettre les **espaces** entre chaque élément, ni de préfixer avec un **antislash** certains opérateurs.
- La manipulation de variables est **simplifiée** car il est possible d'affecter dans une expression, et les noms de variables ne sont pas préfixés par **$**
- Il existe un **plus grand nombre d'opérateurs** pour le calcul arithmétique
- **Opérateurs arithmétiques**

```
$let var=2+3
$echo $var
5
$((var=2**3))
$echo $var
8
$echo $((2*3))
6
```

Expressions arithmétiques : let, (())

- Opérateurs Logiques, incrémentation …

```
$ ((x=3))
$ while ((x>0))
> do
> echo $x
> ((x--))
> done
```