Programmation Shell

1. Introduction :

Le Shell est un "interpréteur de commande". Son rôle est d'analyser la commande tapée afin de faire réagir le système pour qu'il réponde aux besoins de l'utilisateur. C'est le premier langage de commandes développé sur Unix par Steve Bourne.
Il existe plusieurs Shell tels que :

- le Bourne Shell ("/bin/sh")
- le Korn Shell ("/bin/ksh")
- le c Shell ("/bin/csh") pour les utilisateurs préférant un langage apparenté au "C"
- le Job Shell ("/bin/jsh")
- le Shell réseau ("/bin/rsh")
- le Bourne Again Shell ("/bin/bash")
- le c Shell amélioré ("/bin/tcsh")
- le z Shell ("/bin/zsh")
- et d'autres encore à venir…

C'est un langage de commandes mais aussi un langage de programmation. Il permet donc :

- l'utilisation de variables
- la mise en séquence de commandes
- l'exécution conditionnelle de commandes
- la répétition de commandes

Un programme Shell appelé aussi "script" est un outil facile à utiliser pour construire des applications en regroupant des appels systèmes, outils, utilitaires.

Afin d'exécuter un script, il suffit de se placer dans le dossier où est le script, et de lancer :

```
sh nom_du_script
```

Si vous voulez l'exécuter avec « . », il faut le rendre exécutable avec `chmod`. Pour ceci, tapez dans le Shell la commande qui suit :

```
chmod +x nom_du_script ou chmod 0755 nom_du_script
```

Puis vous pouvez exécuter le script en faisant :

```
./ nom_du_script
```

2. Les variables spéciales

Complément Exercice 1 :

Les variables spéciales : En plus des arguments $1, $2, …, $9, le Shell prédéfinit des variables facilitant la programmation.

$1…$9 ➔ les 9 paramètres d'entrées du script
$0 ➔ contient le nom sous lequel le script est invoqué.
$# ➔ contient le nombre de paramètres passés en argument,
$* ➔ contient la liste des paramètres passés en argument,
$? ➔ contient le code de retour de la dernière commande exécutée,
$$ ➔contient le numéro de processus (PID) du Shell (décimal).

3. Les structures de contrôle

Complément Exercices 2 et 3 :

Le Shell possède des **structures de contrôle** telles qu'il en existe dans les langages de programmation d'usage général : instructions conditionnelles (if.. then.. else, test, case), itérations bornées et itérations non bornées.

3.1 Les instructions conditionnelles

Pour la programmation des actions conditionnelles, il y a trois outils :

- l'instruction if,
- la commande test qui la complète,
- l'instruction case.

➢ L'instruction if

Elle présente trois variantes qui correspondent aux structures sélectives à une, deux ou n alternatives.

La sélection à une alternative : if… then… fi

```
if  [ conditions  ]
  then
        commandes
fi
```

La sélection à deux alternatives : if… then… else… fi

```
if  [ conditions 1 ]
then
        commandes1
else
        commandes2
fi
```

La sélection à n alternatives : if… then…. elif… then… fi

```
if  [  conditions1  ]
then
        commandes1
elif  [ conditions2 ]
then
        commandes2
elif  [ conditions3 ]
then
        commandes3
…
else
        commandes0
fi
```

➢ La commande test

Elle constitue l'indispensable complément de l'instruction if. Elle permet très simplement :

- de reconnaître les caractéristiques des fichiers et des répertoires,
- de comparer des chaînes de caractères,
- de comparer algébriquement des nombres.

Cette commande existe sous deux syntaxes différentes :

```
        test condition
ou
        [ condition ]
```

La commande test répond à l'interrogation formulée dans « condition », par un code de retour nul en cas de réponse positive et différent de zéro sinon.

➢ L'instruction case

L'instruction case est une instruction très puissante et très commode pour effectuer un choix multiple dans un fichier de commandes.

```
case chaine in
motif1) commandes 1 ;;
motif2) commandes 2 ;;
….
….
motifn) commandes n ;;
esac
```

La « chaine » dans « case » peut prendre diverses formes :

- un chiffre,
- une lettre ou un mot,
- des caractères spéciaux du Shell,
- une combinaison de ces éléments.

Les opérateurs de chaînes :

```
-d nom vrai si le répertoire nom existe,
-f nom vrai si le fichier nom existe,
-s nom vrai si le fichier nom existe et est non vide,
-r nom vrai si le fichier nom existe et est accessible en lecture,
-w nom vrai si le fichier nom existe et est accessible en écriture,
-x nom vrai si le fichier nom existe et est exécutable,
-z chaîne vrai si la chaîne de caractères chaîne est vide,
-n chaîne vrai si la chaîne de caractères chaîne est non vide,
c1 = c2 vrai si les chaînes de caractères c1 et c2 sont identiques,
c1 != c2 vrai si les chaînes de caractères c1 et c2 sont différentes,
```

Les opérateurs relationnels :

```
n1 -eq n2 vrai si les entiers n1 et n2 sont égaux.
n1 -ne n2 vrai si les entiers n1 et n2 sont différents.
n1 -lt n2 vrai si les entiers n1 et n2 sont égaux.
n1 -le n2 vrai si les entiers n1 et n2 sont égaux.
n1 -gt n2 vrai si les entiers n1 et n2 sont égaux.
n1 -ge n2 vrai si les entiers n1 et n2 sont égaux.
```

Les opérateurs logiques :

Les expressions peuvent être niées par l'opérateur logique de négation ! et combinées par les opérateurs « ou logique » o et « et logique » a.

3.2 Les itérations

La présence des instructions itératives dans le Shell en fait un langage de programmation complet et puissant. Le Shell dispose de trois structures itératives : for, while et until.

➢ **Itération bornée : La boucle for**

Trois formes de syntaxe sont possibles :

1) Forme 1

```
for variable in chaine1 chaine2… chaineN
do
        commandes
done
```

2) Forme 2

```
for variable
do
        commandes
done
```

3) Forme 3

```
for variable in *
do
        commandes
done
```

Pour chacune des trois formes, les commandes placées entre « do » et « done » sont exécutées pour chaque valeur prise par la variable du Shell variable. Ce qui change c'est l'endroit où variable prend ses valeurs. Pour la forme 1, les valeurs de variable sont les chaînes de chaine1 à chaineN. Pour la forme 2, variable prend ses valeurs dans la liste des paramètres du script. Pour la forme 3, la liste des fichiers du répertoire constitue les valeurs prises par variable.

➢ **Itérations non bornées : while et until**

```
while conditions1
do
        commandes1
done
```

```
until conditions2
do
        commandes2
done
```

Les « commandes1 » sont exécutées tant que (while) « conditions1 » est vrai.
Les « commandes2 » sont exécutées jusqu'à (until) « conditions2 » sera non valide.

Complément Exercice 4 :

Les commandes let et (( )) sont équivalentes, elles servent à :

- affecter des variables numériques : $ let x=1 ou $ ((x=4))
- faire des calculs : $ let x=x+1 ou $ ((x +=1))
- évaluer les tests numériques : $ let (x=x-1) < 0 ou $ (((x=x-1) ‹0))

La commande « expr » permet d'évaluer des expressions arithmétiques ou logiques.

Complément Exercice 5 :

La commande « tail » permet d'afficher la dernière partie d'un fichier. Par défaut, les 10 derniers lignes du fichier.
La commande « head » permet d'afficher le début d'un fichier. Par défaut, les 10 premiers lignes du fichier.
La commande « shift » permet de décaler la liste des arguments d'une ou plusieurs positions vers la gauche. Cette commande est utilisée dans le cas où le premier argument n'a pas à subir le même traitement que les suivants. Dans ce cas, le premier argument doit être sauvegardé dans une variable et l'exécution de la commande shift entraine le remplacement du premier argument par le second et ainsi de suite.