TP 5 : Programmation shell

2015/16

Quelques informations/remarques liminaires
POSIX : une norme.
Les quatre premières lettres du mot forment l'acronyme de Portable Operating System Interface, et le X exprime l'héritage UNIX de l'interface de programmation.

Interpréteurs de commandes disponibles :
- dash : Debian Almquist SHell, conforme aux standards Posix (il manque toutefois certaines fonctionnalités).
- bash : Bourne Again SHell, qui offre plus de possibilités que le dash.
- sh : en fait, sauf exception (toujours possible), un lien symbolique vers le dash.
  On peut le vérifier en tapant ls -l /bin/*sh
  (ceci dit, selon les installations Unix et le serveur/machine sur lequel vous travaillez, sh peut aussi « pointer » vers un autre shell ou ... être le shell originel, c.a.d. le bourne shell, ou ... un autre shell qui a été renommé en sh ... oups ! Donc il peut être utile de contacter pour plus de précisions l'administrateur de la machine en question ou vous mêmes sur votre ordi personnel.

En mode interactif, c'est le bash qui est en principe le shell par défaut sous Linux.

Eléments de base

Affectation de variable

```
variable=...            # attention : pas d'espaces ni avant ni après le symbole '='
variable='commande'     # syntaxe "historique" ("vieux" bourne shell), tjrs d'actualité
variable=$( commande )  # syntaxe shell posix (ok en bash et en dash)

Exemples:
    i=0
    nbLigne='cat holique | wc -l'
    nbLigne=\$( cat holique | wc -l )
```

Calcul arithmétique

```
variable = 'expr expression opérateur expression'     # syntaxe "historique" ("vieux" bourne shell)
        Remarque: appel à la commande unix expr

variable =$(( expression opérateur expression ))      # syntaxe shell posix
```

```
expression : $variable ou constante

Exemples:
    i='expr $i + 1'
    i=$(( $i + 1 ))
```

Lecture et affichage (entrée/sortie standard)

```
echo $variable            # affichage de la valeur de la variable
read variable ...         # lecture de la valeur de la variable sur l'entrée standard

Exemples:
    (on suppose qu'on entre au clavier à chaque fois les mots
    "I love shell", suivis d'un saut de ligne)
    read a
        a vaut I love shell
    read a b
        a vaut I et b vaut love shell
    read a b c
        a vaut I , b vaut love et c vaut shell
```

Conditionnelles

```
if <condition> ; then <commandes>
elif <condition> ; then <commandes>
else <commandes>
fi

condition: test ou [ ] et opérateurs de comparaison
    chaines :     opérateurs avec un seul argument -z -n
                  opérateurs avec deux argument    =  !=
                  (vrai si chaine de longueur zéro, not zéro)
    numériques : -eq -ne -gt -ge -lt -le (equal, not equal, greater than,
                 greater or equal than, ...)

examples:
    if test -z "$nom"                          if [ -z "$nom" ]
    then echo "$nom est vide"                 then echo "$nom est vide"
    else ...                                   else ...
    fi                                         fi

    echo -n "entrez un nombre : "
    read nombre
    reste=$(( $nombre % 2 ))
    if [ $reste -eq 0 ] ; then
        echo $nombre est pair
    else
        echo $nombre est impair
    fi
```

```
Remarque: séparateur ';' nécessaire avant l'instruction then si elle est sur la même ligne que if
```

Itérations

```
for variable in liste
do
...
done

Exemple:
    mots="un deux trois"
    for unMot in $mots ; do echo $unMot ; done

Remarque: séparateur ';' nécessaire avant les 'instructions do et done si sur la même ligne que for

while condition ou commande
do
...
done


Exemple:
    NB=40
    echo "Ici Noe ! J'attends la fin du deluge"
    while test $NB -gt 0 ; do                    # while [ $NB -gt 0 ] ; do
        echo "Fin du deluge dans $NB jour(s)"
        sleep 1
        NB=`expr $NB - 1`                        # NB=$(( $NB - 1))
    done
    echo "Tiens, il s'est arrete de pleuvoir"
```

Exécution d'un script

- sh nomDuScript [paramètres]
- ./nomDuScript [paramètres]

déboguage :
options : -x, -v
sh -x nomDuScript

Paramètres :

- On obtient la valeur du 1er paramètre passé sur la ligne de commande avec \$1 ; 2$^{\text{ième paramètre}}$ avec \$2, ...
- Le nom du script est contenu dans \$0 et le nombre de paramètres dans \$#.
- La notation \$@ est équivalente à \$1 \$2 ... et peut être utilisée pour « balayer » la liste des paramètres (itération for).

Exercices

1. Ecrire un fichier shell qui prend en parametre 2 entiers et affiche la somme. S'il n'y a pas deux paramètres, il faut afficher un message d'erreur.
2. Ecrire un script shell qui affichera 5,4,3,2,1 en utilisant une boucle while.
3. Ecrire un script qui prend en paramètre trois entiers et qui affiche le plus grand. On affichera un message d'erreur s'il n'y pas pas trois arguments passés sur la ligne de commande.
4. Ecrire un script qui prend en paramètre un entier et affiche l'entier en ordre inverse (123 -> 321).
5. Ecrire un script qui prend en paramètre un entier et affiche la somme des digits qui le compose (123 -> 1+2+3 = 6).