Les scripts en Korn Shell

Plan

- Exécution
- Variables
- Entrées/Sorties
- Récupération Des Paramètres
- Operations Sur Les Variables
- Chaînes De Caractères
- Redirections
- Enchainements
- Boite à outils de commandes
- Expressions Conditionnelles
- Structures de contrôle
- Fonctions
- Tableaux

Execution

- Par le shell courant : . fichier (symbole point)
- Par un nouveau shell :
  - ksh fichier
  - fichier (si fichier est executable)
    → Rendre le fichier executable (chmod)

- Pour être sûr que le script soit executé par un shell ksh, placer au début du script :

  #!/usr/bin/ksh

  (“#” est le symbole pour les commentaires)

Variables

- Variables : marquées par le symbole “$”
- Modification d’une variable :
  - PATH=/usr/bin
  - PATH=$PATH:$HOME/bin:.
- Liste des variables actuelles:
  - env ou set

Entrees/Sorties

- Saisie
  - read x?
  - read x?" message :”
  - Avec bash:
    echo –n “message:”
    read x (avec bash: sans “?”)
- Affichage
  - echo $x
  - print hello world

Récuperation Des Parametres(1)

- $0 : nom de la commande
- $# : nombre de paramètres
- $* : l'ensemble des paramètres
- $1 , $2 , ..., $9 : les 9 premiers paramètres

- exemple :
  - echo ce script $0 a $# arguments. Les voici: $*

Récupération Des Paramètres(2)

- Pour récupérer plus de 9 paramètres, utiliser shift
- shift n : décalage de n positions des numéros de paramètre (par defaut, décalage de 1)

- shift : par defaut, le plus utilisé

- Attention : on ne peut pas revenir en arrière

Operations Sur Les Variables

- Affectation :
  - v=aa
  - v=$HOME/toto
  - Attention : pas d'espace autour du signe =.

- Operations arithmetiques :
  - a=35
  - let a=a+1
  - let a=$a+1
  - a=$((a+1))
  - a=$(($a+1))
- ((a=a<<2+3)) equivaut à let a="a<<2+3" (on évite les ")

Chaines De Caracteres

- Simple quote : chaine brute, aucune substitution
  - echo ’mon repertoire est $HOME’
- Double quote : interprète les caractères $, \, et ‘
  - echo "mon repertoire est $HOME”
- Back quote : remplace la chaine par le résultat de son execution (la chaine est donc une commande)
  - echo "le repertoire courant est ‘pwd‘ "

Redirections (1)

- le shell dispose de 3 tuyaux de communication
  - l'entrée standard : stdin (0)
  - la sortie standard : stdout (1)
  - la sortie des erreurs : stderr (2)

- commande > fic1 (ls > tmp.txt)
- commande >> fic1 (concaténation)
- commande < fic2 (cat <tmp.txt)
- commande << LA FIN

  xyz
  abc
  LA FIN

Redirections (2)

- commande 2> fic1 (redirige la sortie 2)
  - Ou commande 2>> fic1
- commande1 | commande2
  - Connecter la sortie de la cmd1 vers l'entrée de la cmd 2
- d=$(date)
  - redirige la sortie de la commande (date) dans la variable d
  - Équivaut aux backquotes

Enchainements de commandes

- commande1 ; commande2
  - Enchainement inconditionnel
  - Equivaut à la touche entrée après chacune
  - (cd /etc ; ls) --> dans un autre shell
- commande &
  - Lancement en tache de fond
- commande1 && commande2
  - Si oui (2 si 1 est ok)
- commande1 || commande2
  - Si non (2 si 1 non ok)

- Comment cela est décidé?
  - Chaque commande unix renvoie une code d'erreur que l'on peut obtenir par $?

Boite à outils de commandes (1)

- head : debut du fichier
  - Les 10 premières lignes : head toto.txt
  - Les 20 premières lignes : head -20 toto.txt
- tail : fin du fichier
  - tail –f /var/log/syslog (reste connecté au fichier)
- wc : compte les lignes/mots/caractères
  - exemple : ls | wc –l
- Tr: translate, remplacer des caractères
  - echo "tickey touse" | tr "t" "m"
  - echo "mtickeyt mtouset" | tr –d "t"
  - echo "mickey <la souris> est tout petit" | tr -d "<*>"
  - df-h|tr –s ' ' (supprimer les répétitions)

Boite à outils de commandes(2)

- sort : tri
  - Tri alphabétique : sort toto.txt
  - Tri numérique : sort -n toto.txt

- cut : extrait un/plusieurs champ(s) des lignes d'entrée
  - Les mots 3 et 4 de chaque ligne :

    cut -f 3,4 -d ' ' toto.txt
- xargs
  - Convertie l'entrée standard en paramètre
  - ls *.txt | xargs cat

Boite à outils de commandes(3)

- grep :
  - grep bonjour toto.txt
  - grep "bonjour a tous" toto.txt
    - grep "^bonjour a tous" toto.txt (debut de ligne)
    - grep "bonjour a tous$" toto.txt (fin de ligne)
- Grep inversé
  - grep –v bonjour toto.txt

Boite à outils de commandes(4)

- Uniq: supprime les doublons
  - Exemple: cat /etc/passwd|cut -d ':' -f 7|uniq
- Seq: génère une série de nombres
  - seq fin
  - seq debut fin
  - seq debut increment fin
  - À utiliser avec la boucle for
- basename / dirname
  - Permet d'extraire une partie d'un chemin de fichier
  - basename /etc/init.d/x11-common →x11-common
  - dirname /etc/init.d/x11-common → /etc/init.d

Expressions Conditionnelles

- Expressions conditionnelles :
  - [expression] (Attention : les espaces sont obligatoires)
  - Applicable au if, while, etc (“if [expression]”)
  - *faire un “man test” pour les possibilités offertes*
  - *Exemple: if* ["$var" != “mickey"]
- Expressions sur les variables :
  - Comparaisons alphabetiques : =, !=
  - Comparaisons numeriques : -eq , -ne , -lt , -le , -gt , -ge
  - Connecteurs logiques : ! , && , ||
    - Exemple : ["$1" != "$HOME"] && [$2 -eq 34]
- Expressions sur les fichiers (extrait):
  - -a : existence
  - -d : répertoire
  - -s : fichier de taille non nulle
- Expressions sur les chaines de caractères (extrait):
  - -z *chaine* vrai si la *chaine* est vide [-z "$VAR"]
  - -n *chaine* vrai si la *chaine* est non vide [-n "$VAR"]
  - Noter l’usage des quotes: permet de s’assurer que le test portera bien sur une chaine

If... Then... Else... Fi

```
if [ -s $1 ]
then echo "$1 existe"
else
    echo "$1 n’existe pas ou est vide"
fi

if [ $x -gt 10 ]
then
    let x=$x-10
fi
```

While... Do... Done

```
while [ a -gt 0 ]      ( ou while true )
do
    echo "a=$a"
    let a=$a-1
done

while [ $# -gt 0 ]
do
    echo $1
    shift
done
```

For... Do... Done

```
for x in 1 2 3
do
    echo $x
done

for x in `ls *.c`
do
    echo "Compilation de $x"
    cc -o `basename $x.c` $x
done
```

→ for a est equivalent à for a in $*

Case... Esac

```
case $1 in
    *.c)    echo "$1 est un fichier C";;
    *.txt)  echo "$1 est un fichier texte";;
    *)
        echo "$1 n'est pas d'un format connu"
        echo "Completez le programme!";;
esac
```

";;" pour finir le cas

Fonctions (1)

- A considerer comme des scripts independants, lancés depuis le script principal

- A l'intérieur d'une fonction, $1 , $2 , shift , etc... manipulent les parametres de la fonction, pas ceux du script principal.

- Definition :

```
function toto {
..
}
```

- Appel de la fonction (avec des parametres) :

  → toto a b

Fonctions(2)

- Exemple :

```
function add {
    tmp=0
    while [ $# -ne 0 ]
    do
        let tmp=$tmp+$1
        shift
    done
    echo $tmp
//    return $tmp
}

x=‘add 1 2 3‘
echo "x=$x"
```

A noter : le passage de parametre de retour sur la sortie standard

- **Voir la variante avec $?**

```
add 1 2 3
P=$?
```

Tableaux

- seuls les tableaux à 1 dimensions sont gérés
- Création d'éléments :
  - unique: nom[3]=toto
  - En groupe: set -A couleurs rouge vert bleu
- Accès aux éléments:
  - Echo Le tableau couleurs a ${#couleurs[*]} elements.
  - Echo le contenu est ${couleurs[*]}

Exercices

1. Ecrire un script qui Lis 2 nombres au clavier et affiche leur somme (read, echo, let)

2. Ecrire un script qui en fonction d'un parametre indiquant le user ID, affiche le nom du user et le répertoire home à partir du fichier /etc/passwd (cat, grep, cut)

3. Ecrire un script qui en fonction d'un nom (paramètre) affiche son contenu (ls) si c'est un répertoire et fait un more sinon. (test)

4. Ecrire un script qui en fonction d'un nom de fichier crée une archive tar portant le nom de fichier archive"+"yymmddhhmmss"+".tar" où la date est calculée dynamiquement (date, tar)

5. Ecrire un script qui renomme tous les fichiers d'un répertoire donné avec l'extension ".old" (on pourra faire plus sympa en remplacant old par la date courante dans l'extension)

Ex 1

```
#!/bin/ksh

read nb1?"entrez un nombre:"

read nb2?"entrez un second nombre:"

let solution=$nb1+$nb2
print "$nb1 + $nb2 = $solution"
```

Ex 2

```
#!/bin/ksh
#Script compteur de loggins

times=$(who | grep $1 | wc -l)
print "$1 est logge $times fois simultanement."
```

Ex 3

```
#!/bin/ksh

for file in $*
do
  if [ -d $file ]
  then
     print "utilisation de ls"
     ls $file
  else
     more $file
  fi
done
```