La programmation Shell

A. Lasfar

(ali.lasfar@gmail.com)

Introduction

Le Shell est l'interface du système Unix, il fonctionne en deux mode :

- Mode interactif
- Mode programmé (scripts)

Fonctionnement

Lors du lancement d'une commande, le Shell effectue les opérations suivantes :

- Analyse de la ligne de commande
- Recherche du fichier contenant le programme de la commande
- Création du processus fils qui va exécuter ce programme
- Passage d'arguments et exécution

Types de Shell

Bourne Shell (sh)
C-Shell (csh)
Tc – Shell (tcsh)
Korn Shell (ksh)

La ligne de commande

·La ligne de commande va du prompt du Shell jusqu'au CR (Entrée)
·Le premier mot est le nom du fichier exécutable ou d'une commande Shell
·La commande peut être suivie par des options et des arguments séparés par des espaces
·La ligne de commande n'est pas exécutée avant le CR (Entrée)

La ligne de commande

Le Shell connaît 4 types de lignes de commandes
- simples : mode forground > cmd
  mode background > cmd &
- séquentielles : > cmd1 ; cmd2; cmd3; …
- pipeline : cmd1 | cmd2 | cmd3 | …
- groupées : (cmd1 ; cmd2; …) > fichier

Exemple (ligne de commandes groupées)
> (who ; echo "sont connectés") > Toto

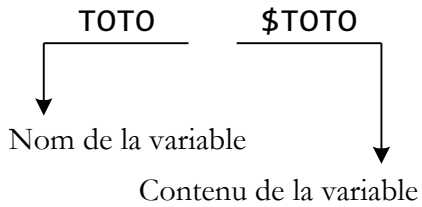
Exemple (ligne de commandes séquentielles)
> Date ; who ; ls

Les variables Shell

Les variables Shell

Une variable est identifiée par son nom.
Le contenu de la variable est identifié par le symbole $ placé devant le nom.

Par exemple:



Les variables Shell

Pour afficher le contenu d'une variable, on utilise la commande echo suivant la syntaxe : echo $nom_variable

Exemple :

nom = Unix

> echo nom affiche nom

> echo $nom affiche Unix

Les variables Shell prédéfinies

Ces variables sont appelées variables d'environnement. La commande printenv permet de les afficher.

Exemple :

> printenv

HOME=/usr/yassine

PATH=/bin:/usr/bin

LOGNAME=yassine

SHELL=/bin/csh

...

Opération sur les variables d'environnement

- Visualiser le contenu d'une variable d'environnement : > echo $varaible
- Visualiser la liste des variables d'environnement : > printenv
- Modifier le contenu d'une variable : > set ou setenv variable valeur
- Suppression : > unset variable

Les variables Shell locales et globales

Une variable ordinaire est une variable locale, elle n'est connue que dans le Shell où elle a été créée.

Une variable de même nom créée dans un sous-Shell sera une variable distincte

Exemple

```
> Cours = Unix ; echo $Cours
> xterm #création d'un sous-Shell
> echo $Cours #affiche ligne vide
```

Les variables Shell locales et globales

La commande **export** permet de rendre globale une variable locale, c.à.d tous les sous-Shell possèdent une copie de cette variable.

Exemple

```
> Cours = Unix ; export Cours
> echo $Cours #affiche Unix
> xterm #création d'un sous-Shell
> echo $Cours #affiche Unix
```


Les scripts Shell

Un script Shell est un programme basé sur l'utilisation des commandes Shell.

Exemple

```
> echo "quel est ton nom"
> read nom
> echo "ravi de te connaître", $nom
```

Les arguments

Dans les scripts les arguments sont référencées par leur position dans la liste des arguments :

- $1 : premier argument
- $2 : deuxième argument
- $3 : troisième argument

Par convention :

- $0 : nom du fichier script
- $* : liste des arguments
- $# : nombre d'arguments

Les arguments

Exemple (script Arguments)

```
echo je suis le script $0
echo le nombre d'arguments est $#
echo mes arguments sont $*
echo le premier argument est $1
```

```
> Arguments un deux trois
je suis le script Arguments
le nombre d'arguments est 3
mes arguments sont un deux trois
le premier argument est un
```

Rendre exécutable un script : pour rendre un script exécutable directement : on utilise la commande chmod : > chmod u+x nom_script

Substitution des commandes : lorsqu'une chaîne de caractères est entourée par ` `, le Shell l'interprète en tant que commande et la remplace par son résultat

> echo nous sommes le `date` # affiche nous sommes le Lun Mars 8 04 20:40:34

Exemple 1 de script :

```
echo "Bonjour tout le monde"
echo "Quel est votre nom?"
read NOM
echo "Bonjour $NOM"
```

Exemple 2 de script :

```
# Fichier premier.sh.
# Un premier script shell.

echo moi, $USER, faire sur $HOSTNAME mon premier script
shell
echo -n "aujoud'hui, le "
date
echo
```


Exemple 3 de script :

```
echo "Je vais calculer le produit de X par Y"
echo "Entrez X"
read X
echo "Entrez Y"
read Y
echo "X*Y = $X*$Y = $[X*Y]"
```

Les caractères spéciaux : se sont

- Les expressions régulières : ? , * , []
- Les séparateurs : ' , " , { } , () …
- Les Métacarctères : \ , $, !
- Les opérateurs particuliers : | , &

Mécanisme d'échappement :

\ : négation du caractère précédent

' ' : négation de tous les caractères spéciaux

Les commentaires : les commentaires doivent commencer par le caractère #

Exemple : # ceci est un commentaire

Les variables spéciales :

- $# : nombre d'arguments
- $? : état de sortie de la dernière commande
- $$: contient le PID du processus courant
- $! : le PID du dernier processus en bg
- $0 : contient le nom de la commande
- $* : la liste des arguments

Exemple 1 : > date >> datelog
> echo $? # affiche 0

Exemple 2 : > date >> datelog &
1234
> echo $! # 1234

L'instruction d'écriture : echo
L'instruction de lecture : read

Exemple
> echo "donner le nom du fichier"
> Read fichier
> Echo $fichier

Les structures de contrôle

Les instructions conditionnelles

A. La sélection à une alternative : if … then … fi

```
if   commande
then commandes
fi
```

Les commandes sont exécutées si la commande condition renvoie un code de retour nul ($?=0)

Exemple

```
if grep –i "yassine" /etc/passwd
then echo l\'utilisateur yassine est connu du système
fi
```

Les instructions conditionnelles

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

```
if   commande
then commandes 1
else commandes 2
fi
```

Les commandes 1 sont exécutées si la commande condition renvoie un code de retour nul ($?=0), sinon les commandes 2 sont exécutées.

Exemple

```
if grep –i "yassine" /etc/passwd
then echo l\'utilisateur yassine est connu du système
else echo l\'utilisateur yassine n\'est pas connu
fi
```


Les instructions conditionnelles

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

```
if   commande 1
then commandes 1
elif commande 2
then commandes 2
elif commande 3
…
fi
```

La commande test

La commande test : constitue l'indispensable complément de l'instruction if. Elle permet :

- De reconnaître les caractéristiques des fichiers et des répertoire
- De comparer des chaînes de caractères
- De comparer algébriquement des nombres.

La commande test

Quelque conditions de test :

 # Fichiers (fic est un fichier)

- test –f fic : fic est un fichier normal.
- test –s fic : fic est un fichier non vide.
- test –d fic : fic est un repertoire.
- test –r fic : fic est accessible en lecture.
- test –w fic : fic est accessible en ecriture.

La commande test

Quelque conditions de test :

 # Comparaison d'une variable a un nombre...

-eq : egal.
-ne : different.
-lt : <.
-gt : >.
-le : <=.
-ge : >=.

La commande test

Quelque conditions de test :

Test Signification

-a fich existe
-b fich fichier spécial bloc
-c fich fichier spécial caractère du processus
-d fich répertoire
-p fich tube nommé
-e fich existe
-r fich lisible
-f fich fichier ordinaire
-S fich fichier socket
-G fich appartient au groupe
-s fich de taille non nulle du processus
-u fich set-uid positionné
-g fich set-gid positionné -w fiich modifiable
-k fich sticky-bit positionné -x fiich exécutable

Opérateurs booléens

! (négation)
&& ou -a (conjonction)
|| ou -o (disjonction)

Utilisation de if : exemple 1

```
# S'il n'y a pas de paramètres fournis par l'utilisateur...

if [ $# -eq 0 ]
then
        echo Aucun argument reçu !
        echo "$0 risque de ne pas bien marcher..."
        echo echo "Normalement il faut fournir le nom d'un      fichier"
        echo "Conseil : lancer "$0 param1"
        echo
fi
```

Exercice 1

Ecrire le script « copy1 » qui copier le premier paramètre dans le deuxième

```
if [ $# -eq 2 ]
then
    if [ -f $2 ]
    then echo fichier $2 existe déjà ; exit
    elif [ -f $1 ]
    then cp $1 $2
    else echo fichier $1 inexistant ; exit
    fi
else
    echo nombre de paramètres incorrect
fi
```


Exercice 2

Vérifier si le fichier passé en argument existe dans le répertoire courant, si oui on affiche son contenu à l'aide de la commande less, sinon on affiche le message suivant « fichier absent »

Exercice 2

```
if test -f "$1"
then
        echo Le fichier $1 est bien présent.
        echo "Nous allons maintenant lire ce fichier grace      a       la
commande less"
        echo 'Pour quitter less, taper "q"'
        echo "Voulez vous lire le fichier $1 (o pour    l'editer) ?"
        read reponse
        if [ $reponse = "o" ]
                then
                        less "$1"
                else echo "Ok, pas d'edition du fichier..."     fi
else
echo "Fichier $1 absent"
fi
```


L'instruction case

```
case chaîne in
motif1) commandes 1;;
motif2) commandes 2;;
…
Motifn) commandes n;;
esac
```

Exemple :

```
Case $# in
0) echo $0 sans argument;;
1) echo $0 possède un argument;;
2) echo $0 possède deux arguments ;;
*) echo $0 a plus de deux arguments;;
esac
```

L'instruction case

Exemple 2 :

```
      # case sur un lecture clavier.

echo "Voulez vous continuer le programme ?"
read reponse
case $reponse in
   [yYoO]*) echo "Ok, on continue";;
   [nN]*) echo "$0 arrête suite a la mauvaise volonté
   de l'utilisateur :-)" exit ;;
   *) echo "ERREUR de saisie"
exit ;;
esac
```


Exercice

On suppose que la date est donnée sous format américain (Sat Jan 1 17:08:23 MET 2006), écrire un script qui affiche la date sous format français (Le samedi 1 janvier 2006 à 17:08:23).

La commande set

set `date` : m.a.j des variables de position avec le résultat de date

Exemple :

Wed Mar 16 8:50:26 GMT 2006

Nous aurons les correspondances suivantes :

Résultat de date: Wed Mar 16 8:50:26 GMT 2006

Variables : $1 $2 $3 $4 $5 $6

Correction de l'exercice

```
set `date`
case $1 in
    Mon) J=Lundi;;
    Tue) J=Mardi;;
    Wed) J=Mercredi;;
    Thu) J=Jeudi;;
    Fri) J=Vendredi;;
    Sat) J=Samedi;;
    Sun) J=Dimanche;;
esac
```

```
case $2 in
    Jan) M=Janvier;;
    Feb) M=Février;;
    Mar) M=Mars;;
    Apr) M=Avril;;
    May) M=Mai;;
    Jun) M=Juin;;
    Jul) M=Juillet;;
    Aug) M=Aout;;
    Set) M=Septembre;;
    Oct) M=Octore;;
    Nov) M=Novembre;;
    Dec) M=Décembre;;
esac
```

```
echo $J $3 $M $6
echo il est : $4
```

Exercice 1

Écrivez un script qui dit si le paramètre passé est :

- un fichier
- un répertoire
- n'existe pas

Exercice 2

En utilisant la structure case, écrire un script qui :

- affiche un menu (GI, ARI, TC, TM)
- demande à l'utilisateur de saisir une option du menu
- affiche à l'utilisateur l'option qu'il a choisie

La boucle for, forme 1

```
for  variable in  valeur1 valeur2 … valeurn
do
   Commandes
done
```

Les valeurs de variable sont valeur1 valeur2 … valeurn

Exemple : cat for1.sh

```
for i in Unix Dos VMS
do
echo j\'ai suivi le cours de $i
done
```


La boucle for, forme 2

```
for  variable
do
   Commandes
done
```

variable prend ses valeurs dans la liste des paramètres du script.

Exemple : cat for2.sh

```
for i
do
echo $i
done
```

La boucle for, forme 3

```
for  variable in *
do
   Commandes
done
```

La liste des fichiers (noms des fichiers et noms des dossiers) du répertoire constitue les valeurs prise variable

Exemple :

```
for i in *
do
echo $i
done
```


La boucle for

Exercice : écrire un script Shell qui affiche uniquement les répertoires du répertoire courant. :

```
for i in *
do
      if [ -d $i ]
      then echo $i ": répertoire "


      fi

done
```

Exercice

Compter le nombre de fichiers et le nombre de répertoire du répertoire courant.

```
NF=0
NR=0
for i in *
do
        if [ -f $i ]
        then NF=$[NF+1]
        elif [ -d $i ]
        then NR=$[NR+1]
        fi
done
echo le nombre de fichiers est $NF
echo le nombre de répertoires est $NR
```


Les boucles while et until

```
while commande1
do commandes
done
```

```
until commande1
do commandes
done
```

Les commandes sont exécutées tant que (while) ou jusqu'à ce que (until) la commande1 retourne un code nul (la commande est vraie)

Les boucles while et until

Exemple while :

```
while [$1 != fin];do
echo $1; shift
done
```

> While 1 2 3 4 fin 5 6

Exemple until :

```
until [$1 = fin];do
echo $1; shift
done
```

> until 1 2 3 4 fin 5 6 # affiche 1 2 3 4

Exercice

En utilisant la structure until...do...done, écrire un script qui :

- demande à un utilisateur de saisir une commande
- exécute la commande ou affiche un message d'erreur si la commande ne s'est pas déroulée.
- répète cette opération tant que l'utilisateur le désire.