Programmation du shell

M.Adam – B.Ernet – JM.Inglebert – P.Portejoie

Le nom de **shell** désigne l'interpréteur de commande du système Unix. Le shell est à la fois un langage de programmation et un langage de commande.
En tant que langage de commande, le shell réalise l'interface entre l'utilisateur et le système.
En tant que langage de programmation, le shell possède des variables de type chaîne de caractères et des structures de contrôle. Afin de simplifier l'utilisation, qui est orientée terminal, les chaînes de caractères ne sont pas délimitées. On peut donc utiliser la sortie de toute commande en tant que chaîne de caractères, ce qui permet la réalisation d'opérations arithmétiques.
Les commandes du shell sont comparables aux appels de fonctions du C. Il existe cependant 2 grandes différences quand à la notation :

- bien que les arguments soient des chaînes de caractères, ils n'ont pas besoin d'être délimités dans la plupart des cas
- on n'utilise ni parenthèse pour encadrer la liste des arguments, ni virgule pour les séparer.

Pour la lisibilité, une commande peut être découpée en plusieurs lignes, en terminant chaque ligne sauf la dernière par `\`. Il n'est pas nécessaire de terminer par `\` une ligne se terminant par un caractère spécial nécessitant une suite : `| && ||`

Evaluation

Elle se fait avec la commande interne `test` ou `[…]`. Le code retour de test peut-être utilisé par `&&` ou par `||`
Les opérateurs utilisés sont :

- pour les chaînes de caractère : `= != < > -n –z !`
  ex : `[ "$CH" ]` la variable est-elle initialisée ?
  `[ -n "$CH" ]`
  `[ ! -z "$CH" ]`
  `[ "$CH" = "azerty ]` comparaison
- pour les nombres : `–eq –ne –lt –le –gt –ge`
- les conjoncteurs : `-a -o`

Calculs

Ils peuvent être réalisés par la commande `expr` avec l'un des opérateurs suivants :
`+ - \* / % \( \) \< \>`

Type d'un fichier et accès

`[ -f fichier ]`	fichier ordinaire ?
`[ -d fichier ]`	répertoire ?
`[ -c fichier ]`	fichier spécial mode caractère ?
`[ -b fichier ]`	fichier spécial mode bloc ?
`[ -p fichier ]`	fichier spécial tube ?
`[ -r fichier ]`	droit en lecture ?
`[ -w fichier ]`	droit en écriture ?
`[ -x fichier ]`	droit en exécution ?
`[ -s fichier ]`	fichier non vide ?

Calculs

La commande `let` autorise l'utilisation des opérateurs suivants :

```
= + - * / % ^ ~ & | << >> ?:
+= -= *= /= %= ^= ~= &= |= <<= >>=
++ -- || &&
== != <= >=
```

Comparaison de chaînes

```
== !=
=
```

Comparaison de nombres

```
-eq -ne -gt -ge -lt -le
```

Opérateurs de conjonction et de disjonction

```
-a -o || &&
```

Structure conditionnelle

```
if condition
then
     commande_x
[else
     commande_y
]
fi
```

où `condition` s'écrit : `test expression` ou `[ expression ]`

Traitement de cas

```
case chaîne in
     chaîne_1)
          commande_x
          ;;
     chaîne_2)
          commande_y
          ;;
     …
     [*)
          commande_z
          ;;
     ]
esac
```

Itération sur liste

```
for variable [in liste]
do
     commande
done
```

Itération sur test

```
while condition
do
     commande_x
done

until condition
do
     commande_x
done
```

Omission d'itération

`continue`

on passe `n` itérations d'un bloc *do-done* par `continue n`

Sortie d'itération

`break`

on sort de *n* blocs *do-done* par `break n`

Gestion de menu

```
select choix in liste
do
     traitement_du_choix
done
```

L'instruction `break` permet de sortir du bloc *do-done*

Tableau

```
declare -a tableau[nb_elts]
tableau[indice]=valeur
echo ${#tableau[*]}
echo ${tableau[*]}
echo ${tableau[indice]}
```

Arguments

`set chaîne`

permet d'associer chaque mot de la chaîne à un paramètre positionnel

Groupement de commandes

On peut être amené à regrouper des commandes lorsque l'ensemble du périphérique standard d'entrée/sortie (stdin, stdout, stderr) doit être redirigé de la même façon.

Les commandes peuvent être regroupées par `()` ou par `{}`. La différence entre ces 2 formes de regroupement est que `()` posséde son propre environnement et donne donc lieu à la création d'un sous-shell.

Fonctions

définition de la fonction :

```
nom_fonction ()
{…
 [return valeur]
}
```

utilisation de la fonction :

```
nom_fonction
```

Une fonction peut être vue comme une commande externe au programme dans sa manière de gérer les arguments et de rendre un code retour. Les arguments passés à une fonction sont lus dans `$1` à `$9`, `$*` et `$@`. Ces variables sont locales à la fonction.

L'exécution d'une fonction peut s'interrompre par `return` qui rend un code retour. Si l'exécution n'est pas interrompue par `return`, le code retour est celui de la dernière commande exécutée.

Fonctionnalités supplémentaires du ksh

définition de variables :

`typeset –i`	`variable[=`*`valeur`*`]`	entière
`typeset –ib`	`variable[=`*`valeur`*`]`	entière en base b
`typeset –L`	`variable[=`*`chaîne`*`]`	chaîne cadrée à gauche
`typeset –L`*`n`*	`variable[=`*`chaîne`*`]`	chaîne de longueur n cadrée à gauche
`typeset –R`	`variable[=`*`chaîne`*`]`	chaîne cadrée à droite
`typeset –R`*`n`*	`variable[=`*`chaîne`*`]`	chaîne de longueur n cadrée à droite
`typeset –u`	`variable[=`*`chaîne`*`]`	chaîne en majuscules
`typeset –l`	`variable[=`*`chaîne`*`]`	chaîne en minuscules

affichage d'un message et lecture d'une entrée :

```
read var?"message"
```

Csh

Calculs

L'opérateur @ autorise l'utilisation des opérateurs suivants :

```
+ - * / % ! ~ < > << >> ^ & | == != =~ !~ <= >= && || ()
```

Opérateurs sur chaînes

```
== != =~ !~
```

Opérateurs sur nombres

```
< > <= >= ==
```

Opérateurs de conjonction et de disjonction

```
|| &&
```

Structure conditionnelle

```
if (condition) then
     commande_x
[else
     commande_y
]
endif
```

Traitement de cas

```
switch (chaîne)
case chaîne_1:
          commande_x
          breaksw
case chaîne_2:
          commande_y
          breaksw
default:
          commande_z
          breaksw
endsw
```

Itération sur liste

```
foreach variable (liste)
     commande
end
```

Itération sur test

```
while (commande_x)
     commande_y
end
```

Répétition

```
repeat n commande
```

Omission d'itération

`continue`

on passe `n` itérations d'un bloc *do-done* par `continue n`

Sortie d'itération

`break`

on sort de `n` blocs *do-done* par `break n`

Tableau

```
set tableau=(liste)
set tableau[indice]=valeur
echo $#tableau
echo $tableau
echo $tableau[indice]
```

Arguments

`set argv=(chaîne)`

permet d'associer chaque mot de la chaîne à un paramètre positionnel

Gestion du terminal

Elle se fait par la commande `tput` qui permet d'écrire des applications portables pour n'importe quel terminal. Toutes les possibilités de `tput` peuvent être retrouvées par `man 4 terminfo`.

- **Obtenir des informations sur le terminal**

  nombre de lignes de l'écran `tput lines`
  nombre de colonnes de l'écran `tput cols`
  code envoyé par la touche F1 `tput kf1`
  code envoyé par la touche home `tput khome`

- **Déplacer le curseur**

  en haut à gauche `tput home`
  en ligne L, colonne C `tput cup $L $C`
  N colonnes à gauche `tput cub $N`
  N colonnes à droite `tput cuf $N`
  N lignes en haut `tput cuu *N`
  N lignes en bas `tput cud $N`
  sur la ligne de statut `tput tsl`
  hors de la ligne de statut `tput fsl`

- **Gérer le curseur**

  effacer le curseur `tput civis`
  faire apparaître le curseur `tput cnorm`
  sauvegarder sa position `tput sc`
  restaurer sa position `tput rc`

- **Modifier les attributs d'écriture**

écrire en inverse	`tput rev`
écrire en clignotant	`tput blink`
écrire en surbrillant	`tput bold`
écrire en semi-brillant	`tput dim`
écrire en normal	`tput sgr0` `tput rmso`
écrire en souligné	`tput smul`
écrire en non souligné	`tput rmul`

- **Modifier le contenu de l'écran**

effacer tout l'écran	`tput clear`
effacer à partir du curseur	`tput ed`
effacer la ligne de statut	`tput dsl`
début mode insertion	`tput smir`
fin mode insertion	`tput rmir`
supprimer 1 caractère	`tput dch1`
supprimer 1 ligne	`tput dl1`

Calculs

Les calculs peuvent être réalisés à l'aide de `bc` qui est un interpréteur interactif. bc possède ses propres structures de contrôle ainsi qu'un mode opératoire immédiat.
Les utilisations les plus courantes de bc sont :

- les conversions de base
- les caulculs de grands nombres, entiers ou décimaux.

La syntaxe de bc est très proche de celle du c.

- **Syntaxe de bc**

appel	`bc`
sortie	`quit` ou `q`

- **Opérateurs**

ce sont par ordre de précédence : `^ * / % + -`
L'ordre de précédence peut être modifié par l'utilisation de parenthèses `( )`

- **Changement de précision**

afficher la précision	`scale`
la modifier	`scale=`*`n`*

- **Fonctions**

racine carrée `sqrt()`
Les autres fonctions sont disponibles après chargement de la bibliothèque par `bc -l`
Ce sont :

`s()`	sinus
`c()`	cosinus
`a()`	arctangente
`e()`	exponentielle
`l()`	logaritgme népérien
`j(,)`	bessel

`s()/c()` tangente
`e(y*l(x))` x puissance y
`4*a(1)` constante PI
`e(1)` constante e
Les fonctions trigonométriques sont calculées en radians.

- **Base**
  Les bases d'entrée et de sortie peuvent être définies :
  définition de la base en entrée `ibase=`*`n`*
  définition de la base en sortie `obase=`*`n`*

- **Variables et tableaux**
  Les variables possèdent un nom composé d'un caractère alphabétique (26 variables)
  L'opérateur d'affectation est `=`
  Les opérations de post ou pré incrémentation/décrémentation sont possibles.
  Les variables prédéfinies sont :
  `scale=0`
  `ibase=10`
  `obase=10`
  L'index des tableaux va de 0 à 2047

- **Structures de contrôle**
  Itérations :
  `while` `(`*`condition`*`) instruction`
  `for` `(`*`initialisation`*`; `*`condition`*`; `*`incrémentation`*`) instruction`

  Traitement decondition :
  `if` `(`*`condition`*`)` **`then`** `instruction`
  L'alternative n'existe pas.

  `instruction` peut être encadrée par `{}` lorqu'il s'agit d'une instruction multiple.

- **Définition de fonction**
  Le nom des fonctions que l'on définit est composé d'un caractère alphabétique, mais il n'y a pas d'ambiguïté.
  `define` `nom (`*`arguments`*`) {`
  …
  `return` `(`*`valeur`*`)`
  `}`

  Des variables peuvent être déclarées localement à une une fonction. Cela se fait par
  `auto` `var1, var2, …`
  Cette instruction doit être unique et doit être la première de la fonction.

Exercice 1
Ecrire un script qui boucle tant qu'une réponse "oui" ou "non" n'est pas saisie.

Exercice 2
Ecrire un script qui calcule le factoriel d'un entier passé en paramètre

Exercice 3
Ecrire un script qui calcule le PGCD de 2 nombres passés en paramètres.

Exercice 4
Ecrire un script qui admet en paramètres au maximum 3 chaînes de caractères et qui les saisit si elles sont absentes. Puis il détermine si ces chaînes sont identiques et si oui lesquelles.

Exercice 5
Ecrire un script qui détermine le nombre de processus actifs par utilisateur connecté et place le résultat dans le fichier "userp.txt".

Exercice 6
- Ecrire une fonction qui détermine si une année passée en paramètre est bissextile ou non (retourne 1 si bissextile, 0 sinon)
- Ecrire un script qui saisit une date sous le format jjmmaaaa et qui détermine si elle est valide ou non.

Exercice 7
Ecrire un script qui liste un répertoire fourni en paramètre, le répertoire courant sinon, en faisant apparaître d'abords les fichiers et ensuite les répertoires.

Exercice 8
Ecrire un script qui liste un répertoire fourni en paramètre, le répertoire courant sinon, et qui calcule et affiche sa taille en octets.

Exercice 9
Ecrire un script qui liste les fichiers exécutables d'un répertoire fourni en paramètre, du répertoire courant sinon et produit un fichier contenant cette liste

Exercice 10
Ecrire un script qui admet en paramètre le nombre de disques des tours de Hanoï et propose la solution

Exercice 11
Ecrire un script qui affiche la taille du répertoire passé en paramètre, du répertoire courant sinon. La taille doit tenir compte de celles des sous-répertoires.

Correction

Exercice 1

```
#!/bin/sh
#-----------------------------------------------
# Boucle tant que la reponse n'est ni oui ni non
#-----------------------------------------------

while true
do echo -n "Entrez oui ou non: "
   read x
   x=`echo $x|sed "s/[Oo][Uu][Ii]/OUI/"`
   x=`echo $x|sed "s/[Nn][Oo][Nn]/NON/"`
   [ "$x" = "OUI" -o "$x" = "NON" ] && break
done
```

Exercice 2

```
#!/bin/ksh
#----------
# Factoriel
#----------

function fact
{ [ `expr $1` -eq 0 ] && return 1
  fact `expr $1 - 1`
  return `expr $1 \* $?`
}

[ $# -ne 1 ] && exit
fact $1
echo "Factoriel $1=$?"
```

Exercice 3

```
#!/bin/ksh
#----------
# PGCD
#----------

function pgcd
{ [ $1 -eq $2 ] && return $1
  if [ $1 -lt $2 ]
    then pgcd $1 `expr $2 - $1`
    else pgcd `expr $1 - $2` $2
  fi
}

[ $# -ne 2 ] && exit
pgcd $1 $2
echo "PGCD($1,$2)=$?"
```

Exercice 4

```
#!/bin/sh
#-----------------------------------------------
# Admet de 0 a 3 chaines en parametre
# Boucle tant que 3 chaines ne sont pas fournies
# Test de l'egalite entre 3 chaines
#-----------------------------------------------
n=$#
ch=$*
#ch=$@
while [ $n -lt 3 ]
 do echo -n "Entrez la chaine `expr $n + 1`: "
    read x
    if [ "$x" ]
      then ch="$ch $x"
           n=`expr $n + 1`
    fi
 done
set $ch
if [ "$1" = "$2" -a "$2" = "$3" ]
  then echo "Les 3 chaines sont identiques"
  else if [ "$1" = "$2" ]
         then a=1;b=2
         else if [ "$1" = "$3" ]
                then a=1;b=3
                else if [ "$2" = "$3" ]
                       then a=2;b=3
                       else a=0
                     fi
              fi
       fi
       if [ ! $a -eq 0 ]
         then echo "Les chaines $a et $b sont identiques"
         else echo "Les 3 chaines sont differentes"
       fi
fi
```

Exercice 5

```
#!/bin/sh
#-----------------------------------------------------------------
# Determine le nombre de processus actifs par utilisateur connecte
# et place le resultat dans le fichier userp.txt
#-----------------------------------------------------------------
umask 177
me=`whoami`
set `who|cut -c1-9|sort`
for i in $*
do echo -n "$i: "
   n=`ps -u $i|wc -l`
   if [ $i = $me ]
     then echo `expr $n - 4`
     else echo $n
   fi
done>userp.txt
```

Exercice 6

```
#!/bin/sh
#---------------------------------------
# Lit une date sous le format: jj/mm/aaaa
#                     jj-mm-aaaa
#                     jjmmaaaa
# en extrait le jour, le mois et l'annee
# et controle sa validite
#---------------------------------------

function bissextile ()
{ if [ `expr $1 % 400` -eq 0 ] ||
     [ `expr $1 % 100` -ne 0 -a `expr $1 % 4` -eq 0 ]
    then return 1
    else return 0
  fi
}

while true
do echo -n "Entrez une date: "
   read date
   [ ! "$date" ] && break
   date=`echo $date|sed "s/[/-]//g"`
   set `echo -n "$date"|wc -c`
   if [ `expr $1` -eq 8 ]
    then if [ `expr "$date" : '[0-9]*'` -eq 8 ]
           then jj=`echo "$date"|cut -c1-2|sed "s/^0//"`
                mm=`echo "$date"|cut -c3-4|sed "s/^0//"`
                aa=`echo "$date"|cut -c5-8`
                case "$mm" in
                 1|3|4|5|7|8|10|12) set 0 31;;
                 2)                 bissextile $aa
                                    if [ $? -eq 1 ]
                                      then set 0 29
                                      else set 0 28
                                    fi;;
                 4|6|9|11)          set 0 30;;
                 *)                 set 1;;
                esac
                if [ $1 -eq 0 ]
                  then if [ $jj -eq 0 -o $jj -gt $2 ]
                         then echo "ERREUR: jour errone"
                         else break
                       fi
                  else echo "ERREUR: mois errone"
                fi
          else echo "ERREUR: Caracteres non numeriques"
         fi
    else echo "ERREUR: Format de date incorrect"
   fi
done
if [ "$date" ]
  then echo "$jj $mm $aa"
fi
```

Exercice 7

```
#!/bin/sh
#-----------------------------------------------------------------
# Affiche le nombre blocs occupes par les fichiers et repertoires
#      dans le repertoire passe en parametre,
#      repertoire courant sinon
#-----------------------------------------------------------------

if [ $# -eq 0 ]
  then set .
  else if [ $# -gt 1 ]
         then echo "Nombre de parametres incoherent"; exit
         else if [ ! -d $1 ]
                then echo "Repertoire inexistant";exit
              fi
       fi
fi

cd $1
rep=`pwd`
echo -e "La liste des"`ls|wc -l`" fichiers et repertoires de $rep
est:\n"
echo "Fichiers :"
echo "==========="
for i in `ls`
 do
    [ -f $i ] && echo -e "\t$i"
 done
echo "Repertoires"
echo "============"
for i in `ls`
 do
    [ -d $i ] && echo -e "\t$i"
 done
echo -e "\nIls utilisent au total "`ls -l|cut -f2 -d" "` blocs"
```

Exercice 8

```
#!/bin/sh
#-------------------------------------------------
# Affiche le nombre d'octets occupes par les fichiers
#      dans le repertoire passe en parametre,
#      repertoire courant sinon
#-------------------------------------------------

if [ $# -eq 0 ]
  then set .
  else if [ $# -gt 1 ]
         then echo "Nombre de parametres incoherent"; exit
         else if [ ! -d $1 ]
                then echo "Repertoire inexistant";exit
              fi
       fi
fi

cd $1
```

```
rep=`pwd`
tailles=`ls -l|grep '^-'|sed "s/[ ]\{1,\}/#/g"|cut -f5 -d#`
fichiers=`ls -l|grep '^-'|sed "s/[ ]\{1,\}/#/g"|cut -f9 -d#`
if [ "$fichiers" ]
  then set $fichiers
       echo -e "La liste des $# fichiers de $rep est:\n"
       echo $*
       total=0
       set $tailles
       for i
          do total=`expr $total + $i`
          done
       echo -e "\nIls utilisent au total $total octets"
  else echo "Aucun fichier"
fi
```

Exercice 9

```
#!/bin/sh
#-----------------------------------------------------
# Produit un fichier listex.txt contenant la liste de
#         tous les fichiers executables
#      du repertoire passe en parametre,
#      repertoire courant sinon
#-----------------------------------------------------

if [ $# -eq 0 ]
  then set .
  else if [ $# -gt 1 ]
         then echo "Nombre de parametres incoherent"; exit
         else if [ ! -d $1 ]
                then echo "Repertoire inexistant";exit
              fi
       fi
fi

cd $1
( echo "Liste des fichiers executables du repertoire :"
  pwd
  echo
  for i in `ls`
     do [ -x $i ] && echo $i
     done
) > listex.txt
```

Exercice 10

```
#!/bin/ksh
#---------------------------
# Tours de Hanoi
#  admet 3 parametres :
#   - le nombre de tours
#   - la tour d'origine
#   - la tour de destination
#---------------------------

function hanoi
{ if [ `expr $1` -eq 1 ]
    then echo "Deplacer $2 vers $3"
    else typeset -i n=`expr $1 - 1`
         typeset -i o=$2
         typeset -i d=$3
         typeset -i i=`expr 6 - $o - $d`
         hanoi $n $o $i
         echo "Deplacer $o vers $d"
         hanoi $n $i $d
fi
}
[ $# -lt 3 ] && exit
hanoi $1 $2 $3
```

Exercice 11

```
#!/bin/ksh
#-----------------------------------------------------
# Affiche le nombre d'octets occupes par les fichiers
#      dans le repertoire passe en parametre,
#      repertoire courant sinon
#-----------------------------------------------------

function taille
{ # Variables locales
  typeset -i som=0
  typeset i
  typeset rep=`pwd`
  #-----------------

  cd $1
  for i in `ls`
   do
    if [ -f $i ]
      then
       taille=`ls -l $i|grep '^-'|sed "s/[ ]\{1,\}/#/g"|cut -f5 -d#`
       som=`expr $som + $taille`
      else if [ -d $i ]
             then taille $i
                  som=`expr $som + $?`
           fi
    fi
   done
  cd $rep
  return $som
```

```
}

if [ $# -eq 0 ]
  then set .
  else if [ $# -gt 1 ]
         then echo "Nombre de parametres incoherent"; exit
         else if [ ! -d $1 ]
                then echo "Repertoire inexistant";exit
              fi
       fi
fi

taille $1
total=$?
echo "Le repertoire `cd $1;pwd` contient $total octets"
```