Cours UNIX
Programmation Bash

Jean-Claude Bajard

IUT - université Montpellier 2

Les shells disponibles

Shell	Program	Description
Bourne shell	sh	disponible sur toute plateforme Unix
C shell	csh	shell développé pour BSD
Korn shell	ksh	Bourne shell amélioré par ATT
Bourne again shell	bash	Shell linux ; version améliorée de sh

Caractères spéciaux du shell : métacaractères

Caractères	Signification
tabulation, espace	délimiteurs des mots ; un au minimum
retour charriot	fin de la commande à exécuter
&	lance une commande en tâche de fond
' " \	changent l'interprétation des caractères spéciaux
<> << >> ` |	caractères de redirection d'entrées/sorties
* ? [] [^]	caractères de substitution de noms de fichiers
$	valeur d une variable
;	séparateur de commandes sur une seule ligne

Scripts BASH

```
#!/bin/bash                          indique le shell d'exécution
fernand="Hello les enfants"          initialise fernand
toto=22                              texte pas de numérique
titi=`expr $toto + 1`                donne à titi le résultat de la somme
echo $fernand                        affichage
echo $toto
exit $titi
```

#! en début de fichier indique le shell utilisé: Sha-Bang

début de commentaires

exit sortie du script, valeur mise dans $PIPESTATUS

Exécution d'un Script BASH

1. ajouter le droit d'exécution avec la commande *chmod*

2. lancer simplement le fichier en indiquant son nom

Exemple:

```
% vi toto
% chmod u+x toto
% toto\\
Hello les enfants
22
% echo $PIPESTATUS
23
```

Les Variables

- **déclaration:** *nomvar=chaînedecaractères*

  bonjour="hello les petits"

- **Substitution :** *$nomvar* ou encore *${nomvar}*

  echo $bonjour

- **remarques :**

  " *$nomvar*" interprétation de la variable

  ' *$nomvar*' non interprétation de la variable

Commandes utiles pour les variables

- **pour la lecture au clavier :** *read*

  read toto

- **pour évaluer une expression :** *expr*

  toto=22

  titi=‘expr $toto + 1‘

- **pour assigner le résultat d’une opération arithmétique :** *let*

  let riri=titi+1

  let “riri = titi + 1”

Les arguments de la ligne de commande

- **$#** nombre d'arguments
- **$0** nom du programme
- **$1 ... $n** nom des arguments
- **$*** la liste des arguments

les opérateurs arithmétiques

- + addition, - soustraction, * multiplication
- / quotient, % reste
- comme en language C : +=, -=, *=, /=, %=

  let " riri *= 4" # multiplie par 4 le contenu de riri

opérations binaires

- << left shift (multiplie par 2), >> right shift (divise) de même on a <<= et >>=

  let "var <<= 2" # results in var left-shifted 2 bits (multiplied by 4)

- **&** and, | or, ~ négation, **!** not, ^ xor de même **&=**, |=, ^=,

les relations de tests

- $<$ inférieur, $<=$ inférieur ou égal, $>$ supérieur, $>=$ supérieur ou égal
- $==$ égalité
- $!=$ non égal, **&&** et logique, $||$ ou logique

Conditionnel

```
if [ condition-true ]
                then
                        command 1
                        command 2
                        ...
                else
                        # Option si test faux
                        command 3
                        command 4
                        ...
                fi
```

ou encore

```
if [ -x filename ]; then    # sur une ligne
```

Tests sur le type de fichier

	Returns true if...
-e	file exists
-f	file is a regular file
-s	file is not zero size
-d	file is a directory
-b	file is a block device (floppy, cdrom, etc.)
-c	file is a character device (keyboard, modem, sound card, etc.)
-L	file is a symbolic link

Tests sur les permissions du fichier

-r file is readable (has read permission)
-w file has write permission
-x file has execute permission

-g group-id flag set on file
-u user-id flag set on file
-k "sticky bit" set
-O you are owner of file
-G group-id of file same as yours

Tests entre fichiers

f1 -nt f2	file f1 is newer than f2
f1 -ot f2	file f1 is older than f2
f1 -ef f2	files f1 and f2 are links to the same file

les tests arithmétiques

-eq is equal to ($a -eq $b)
-ne is not equal to ($a -ne $b)
-gt is greater than ($a -gt $b)
-ge is greater than or equal to ($a -ge $b)
-lt is less than ($a -lt $b)
-le is less than or equal to ($a -le $b)

les tests sur les chaînes

=	is equal to ($a = $b)
!=	is not equal to ($a != $b)
\<	is less than, in ASCII alphabetical order ($a \< $b)
\>	is greater than, in ASCII alphabetical order ($a \> $b)
-z	string is "null", that is, has zero length
-n	string is not "null".

exemple

```
#!/bin/bash
titi=22
let riri=titi+1

if [ $riri -gt $titi ]
then echo ok
else
echo non
fi
```

les boucles for

```
for     [arg] in [list]
do
        command...
done
```

remarque: sur une ligne,

for [arg] in [list] ; do

les boucles for : exemple

```
#!/bin/bash
for planet in    Mercury Venus Earth Mars Jupiter Saturn Uranus
do
                 echo $planet
done

echo

                 # Entire 'list' enclosed in quotesis a single variable.
for planet in    "Mercury Venus Earth Mars Jupiter Saturn Uranus"
do
                 echo $planet
done
exit 0
```

les boucles while

```
while   [condition]
do
        command...
done
```

remarque: sur une ligne

while [condition**] ; do**

les boucles while : exemple

```
#!/bin/bash
var0=0

while           [ "$var0" -lt 10 ]
do
                echo -n "$var0 "
                # -n suppresses newline.
                var0=`expr $var0 + 1`
                # var0=$(($var0+1)) also works.
done

echo
exit 0
```

les boucles until

```
until   [condition-is-true]
do
        command...
done
```

remarque: sur une ligne

```
until [condition-is-true] ; do
```

les boucles until : exemple

```
#!/bin/bash

until                                   [ "$var1" = end ]
# Tests condition at top of loop.
do
                                        echo "Input variable #1 "
                                        echo "(end to exit)"
                                        read var1
                                        echo "variable #1 = $var1"
done

exit 0
```

cas conditionnel

```
case "$variable" in

"$condition1" )
        command...
        ;;

"$condition2" )
        command...
        ;;

esac
```

cas conditionnel

```
#!/bin/bash
echo
echo                    "Hit a key, then hit return."
read                    Keypress

case "$Keypress"        in
[a - z] )               echo "Lowercase letter";;
[A - Z] )               echo "Uppercase letter";;
[0 - 9] )               echo "Digit";;
* )                    echo "Punctuation, whitespace, or other";;
esac
# Allows ranges         of characters in [square brackets].
exit 0
```