Linux

Les scripts

Patrick FULCONIS

Les scripts shell

- **Le shell**
  - Le shell est un interpréteur de commandes et également un puissant langage de programmation.

- **Script shell**
  - Fichier texte constitué d'instructions (extension du fichier .sh)
  - Facilite l'enchaînement des commandes afin d'automatiser des tâches.

- **Constitution du fichier texte**
  - Ligne 1 : #! /bin/bash
    - Indique où se trouve l'exécutable bash. C'est lui qui va interpréter votre script.
  - Ligne 2 à n
    - Instructions diverses
  - Commentaires : commençent par # (sauf #!)

Les scripts shell

- **Shell : un langage interprété**
  - Les lignes d'instructions sont exécutées de manière séquentielles.
  - Si une ligne contient une erreur, le script continue (en général)
  - Les erreurs de syntaxe ne sont pas toujours bien expliquées, comparativement à un langage compilé (comme le C par exemple).

- **Exécution d'un script shell**

  3 manières :

  1 : bash mon_script

  2 : . mon_script (syntaxe aussi utilisée pour les fichiers de config .bash*)

  3 : ./mon_script (*mon_script* doit avoir le droit x (execution))

Les scripts shell

Exemples de scripts

- **Ex1 :**

```
#! /bin/bash
echo Bonjour
```

- **Ex2 :**

```
#! /bin/bash
ls /temp
```

- **Ex3 :**

```
#! /bin/bash
echo "Liste des stagiaires triés"
grep –i stagiaire liste.txt | sort
```

Les scripts shell

Passage de paramètres

Tout script peut être exécuté avec des paramètres (arguments)

Exemple : **./mon_script rouge 100**

Contenu du fichier

```
#! /bin/bash
echo "Le script a pour nom : $0"
echo "le premier paramètre est  : $1"
echo "le deuxième paramètre est  : $2"
echo "le nombre de paramètres est  : $#"
echo "les paramètres sont  : $*"                    # $*    $@
```

$0 (nom du script) prendra la valeur : mon_script
$1 (1er paramètre) prendra la valeur : rouge
$2 (2e paramètre) prendra la valeur : 100
$# (nb paramètres) prendra la valeur : **2**
$* (liste paramètres) prendra la valeur : rouge 100

Le contenu d'une variable est considérée par le shell comme une suite de caractères

Les scripts shell

- **Calcul d'expressions**
  - Comme tout est chaîne de caractère, l'écriture d'une expression comme `Total= 12 + 13` est impossible à évaluer
  - La commande **expr** sera utilisée pour évaluer un calcul (il existe d'autres possibilités, comme *bc*).
  - Syntaxe : **expr nb1 opérateur nb2** (nb1 et nb2 étant des nombres entiers)

- **Exemple :** **./mon_script 3 5**

mon_script

```
#! /bin/bash
echo la somme est de $(expr $1 + $2)
```

$(commande) ou `commande` : renvoie le résultat de commande

$((nb1 opérateur nb2)) **$(expr nb1 opérateur nb2)**

Attention à l'utilisation de l'opérateur *, car c'est aussi un **métacaractère**

Les scripts shell

- **Calcul d'expressions utilisant un métacaractère**

  Tout signe ayant une signification pour le shell **$; & () {} [] | > < ? * " ' \ ` !**
  doit être protégé par le caractère ****

- **Exemple**

```
#! /bin/bash
total=$(expr $1 \* $2)          # total est une variable prenant la valeur $1*$2
echo le total est de $total     # $total est le contenu de la variable total
```

Les caractères spéciaux

” ” protection de tous les métacaractères et des séparateurs (blancs, tabulation, retour-chariot) sauf $ \ `

’ ’ protection de tous les métacaractères et de tous les séparateurs.

\ protection de tout caractère spécial, y compris \

Exemples :

echo ”test (‘ \\ $SHELL * ” affiche : test (‘ \ /bin/bash *

echo ‘test (” \\ $SHELL * ’ affiche : test (” \\ $SHELL *

Les scripts shell : if - test

- if condition; then commande1; else commande2; fi
  if condition; then commande1; elif condition2; then commande2
  else commande3; fi

- ```
  if condition              # si la condition est vraie
  then commande1            # alors on execute commande1
  else commande2            # sinon on execute commande2
  fi
  ```

- Exemple : Test s'il y a 2 paramètres, et si oui, les additionne

```
if [ $# -ne 2 ]      # ou [ $# != 2 ]   # ou test $# != 2   # ou (( $# != 2 ))
then
        echo ''il faut 2 parametres''
else
        echo le total est de $(expr $1 + $2)
fi
```
```

Les scripts shell : test

Tests sur les fichiers

-d répertoire (directory)
-e existe
-f fichier
-s fichier de taille > 0
-r accessible en lecture
-w accessible en écriture
-x exécutable

Comparaison de chaînes

-z nulle
-n non nulle
chaine1 **==** chaine2
chaine1 **!=** chaine2

Comparaison de deux nombres

-eq (equal)	**==**
-ne (not equal)	**!=**
-lt (less than)	**<**
-le (less or equal)	**<=**
-gt (greater than)	**>**
-ge (greater or equal)	**>=**

Il est possible aussi d'utiliser la syntaxe :
if ((comparaison numérique **))** avec les opérateurs de comparaisons standards

Opérateurs Logiques

! Exp Négation de l'expression (NOT)
Exp1 **-a** Exp2 vrai si les deux expressions sont vraies (ET) AND (ou [[Exp1 **&&** Exp2]])
Exp1 **-o** Exp2 vrai si l'une des deux expressions est vraie (OU) OR
(ou [[Exp1 **||** Exp2]])

Les scripts shell : boucle for

for **var** in list; do things; done

```
# création de 10 fichiers .jpg et .gif à l'aide de la boucle for
for ((i=0;i<10;i++))          # syntaxe ((début; »fin »;incrément)) et  i++    i=i+1
do
     touch image$i.jpg image$i.gif
    # ou en forçant les nombres sur 2 chiffres :
    # touch $(printf "image%02d.jpg image%02d.gif" $i $i)
done
```

Autres syntaxes :

```
for i in 0 1 2 3 4 5 6 7 8 9
for i in {0..9}    (si bash récent)
for i in $(seq 0 9)
```

Autres listes possibles (fichiers) :

```
for i in *.txt              (boucle sur tous les fichiers *.txt du répertoire)
for i in $(cat fichier.txt)          (boucle sur le contenu de fichier.txt)
```

Les scripts shell : boucle while

Syntaxe

```
while condition
do
    commande1
    commande2
done
```

Tant que **condition** est vraie (code retour égal à 0) alors commande1 et commande2 sont exécutées.

Exemple

```
ville=a
while [ "$ville" != "grenoble" -a "$ville" != "paris"  ]
    do
    echo "Entrez la ville (grenoble ou paris) "
        read ville
    done
```

Les scripts shell : boucle until

Syntaxe

```
until condition
do
    commande1
    commande2
done
```

commande1 et commande2 sont exécutées jusqu'à que **condition** soit vraie (code retour égal à 0)

Exemple

```
ville=a
until [ "$ville" == "grenoble" -o "$ville" == "paris"  ]
    do
    echo "Entrez la ville (grenoble ou paris) "
        read ville
    done
```

Les scripts shell : case

Syntaxe

```
case variable in
    pattern1 ) commande1 ;;
    pattern2 ) commande2 ;;
esac
```

où pattern1 et pattern2 sont des modèles. Le Shell vérifie si la valeur de la variable "rentre" dans le modèle, et si oui, exécute la commande.
La fin de définition des pattern se fait par le signe ;;

Exemple

```
case $choix in
    1 )  echo "Vous avez choisi l'option 1"
         echo 'vous avez bien fait‘ ;;
    2 )  rm -Rf * ;;
    * )  echo "Les choix valides sont 1 ou 2" ;;
esac
```

Les scripts shell : fonctions

Syntaxe

```
nom_fonction() {
        <suite de commandes>
}
```

On peut appeler une fonction avec des arguments dont les valeurs sont 1,2...
Les variables du script sont visibles dans le corps de la fonction.

Exemple

```
accumule() {
    somme=‘expr $somme + $1‘
}
somme=0
for i in 1 2 3 4 5
do
    accumule $i
done
echo "somme = $somme"
```

Les variables d'environnements

- **Liste des variables système**
  - **HOME**
    contient la valeur du répertoire privé de l'utilisateur
  - **SHELL**
    contient le shell courant
  - **PATH**
    contient la liste des répertoires qui sont explorés par le shell lorsqu'une commande externe est lancée.
  - **USERNAME**
    contient le nom de l'utilisateur connecté
  - **...**

Les variables d'environnements

La commande **env** donne la liste des variables définies dans le shell courant.

- **Affichage de la valeur d'une variable**
  - Le caractère spécial **$** du shell permet de récupérer le contenu d'une variable.

```
echo $PATH
```

- **Modification de la valeur d'une variable**

  Syntaxe

  Variable=valeur (ou **export Variable=valeur**)

Les variables d'environnements

- **Modification de la variable PATH**

  PATH=$PATH:. (ajoute le répertoire courant au PATH)

- **Modification de la variable PS1**

  Cette variable contient la chaîne de caractères représentant le prompt.

  PS1= "**Saisir une commande => **"

  Ou

  PS1='**$PWD**'

- **« Durée de vie » des variables**

  Les variables définies dans un shell ne sont valables que pour ce shell. Il faut les mettre dans les fichiers .bashrc ou .bash_profile afin qu'elles soient définies pour tous les shells

Commandes diverses

- **mktemp**

  Création d'un fichier temporaire (dans /tmp par défaut)

  (*mktemp -d* : Création d'un répertoire temporaire)

- **basename**

  Récupération du nom de fichier sans son chemin, et, en option, sans son extension.

  basename ~/TP1/code.c retourne code.c
  basename ~/TP1/code.c .c retourne code