Programmation Shell

•

January 17, 2017

Table des matières

Introduction

Introduction

Nous avons vu l'utilisation interactive du shell :

- Séquence de caractères terminée par "Return"
- Analyse de la séquence. Si correcte, exécution, si incorrecte, message d'erreur
- Attente d'une nouvelle séquence, etc.

Script shell : série de commandes dans un fichier.

Introduction

Pourquoi ?

- Utile pour créer des commandes personnalisées, combinant des commandes existantes
- Automatisation de tâches

Shell scripting

Shell scripting

Comment ?

- Créer un fichier (exécutable)
- Y ajouter les commandes
- L'éxécuter

Memes notions habituelles :

- Composition séquentielle (éxecution de c1 puis c2)
- Composition conditionnelle (éxécution de c1 si condition vraie, c2 sinon)
- Itération (tant que condition vraie, éxecuter...)

Mais certaines différences :

- Les instructions sont ici des commandes UNIX
- La syntaxe est différente

Shell scripting

```
#!/bin/bash
mkdir TP1
cd TP1
cp /home/depot/CSH/TP1/* .
echo le répertoire courant est
pwd
echo il contient
ls -l
```

- première ligne : ne correspond pas à une commande Unix, mais indique à l'interpréteur de commande le nom du programme à utiliser pour interpréter le fichier. Un # seul : commentaire.
- cd $HOME/CSH : répertoire.

Shell scripting

Exercice : Ecrire un fichier de commande de nom "rangeTP1.sh" qui déplace tous les fichiers terminés par ".c" du répertoire TP1 dans le répertoire TP1/Sources (à créer) et tous les autres dans le répertoire TP1/Divers (à créer également) et affiche le contenu de ces deux répertoires. On suppose que rangeTP1.sh se trouve dans le même répertoire que les fichiers ".c" et les autres fichiers.

Shell scripting

Une solution :

```
mkdir Sources
mv *.c Sources
mkdir Divers
mv * Divers #Warning !
mv Divers/Sources .
echo "Le répertoire TP1/Sources contient :"
ls Sources
echo "Le répertoire TP1/Divers contient :"
ls Divers
```

Shell scripting

Passage d'arguments : les arguments sont désignés en foction de leur position sur la ligne de commande :

- $0 : le nom du script
- $1 : le premier argument
- $2 : le deuxième argument..
- $# : le nombre de "vrais" arguments (sans le nom du script)
- $* : tous les paramètres

Shell scripting - Les variables

Deux types de variables:

- **Variables système**, en majuscules
- **Variables définies par l'utilisateur**

Variables systèmes les plus courantes :

- BASH : chemin du shell (/bin/bash par exemple)
- HOME : chemin du répertoire home
- PATH : localisation des fichiers exécutables (usr/bin:/sbin:/bin:/usr/sbin par exemple)
- etc.

Syntaxe pour définir une variable utilisateur : name=value. Syntaxe pour récupérer la valeur d'une variable : $name.

Shell scripting - Arithmétique

Syntaxe : expr op1 opérateur op2 Exemples :

- expr 1 + 3
- expr 2 - 1
- expr 10
  3
- echo ‘expr 6 + 3‘

Shell scripting - A propos des quotes

3 types de quotes :

- Double quotes " " : perméables à l'interprétation
- Simple quotes ' ' : bloquent le mécanisme de substitution
- Backquotes ` : éxecution du code contenu.

Shell scripting - Exemple

```
$ a=46    écrire le = SANS ESPACE de part et d'autre
$ echo $a
    46
$ b=livre
$ echo $b
    livre
$ c='date'
$ echo $c
    date
$ d="le chien"
$ echo $d
    le chien
$ d='le chien'
$ echo $d
    le chien
$ echo "$a"
$   46
$ echo '$a'
    $a
$ c=`date`
$ echo $c
   Fri Feb 19 14:59:01 CET 2010
$ echo `$a`
   46: command not found
$ echo '$c'
   $c
$ c=le chien
    sh: chien: not found   (message d'erreur)
```

Shell scripting - Read

Syntaxe : read variable1, variable2,...,variableN.
Utilisé pour récupérer l'entrée clavier, et la stocker dans une variable.
Exemple :

```
echo "Your first name please:"
read fname
echo "Hello $fname."
```

Shell scripting - Conditions

Syntaxe :

```
if condition
then
commande1 si la condition est vraie ou si l'exit status
 de la condition est 0
...
fi
```

Syntaxe des tests : test expression OR [expression]

Shell scripting - Conditions

Table 1: Opérateurs dans un script shell

Shell	Représentation habituelles
-eq	$==$
-ne	$!=$
-lt	$<$
-le	$\leq$
-gt	$>$
-ge	$\geq$
-a	AND
-o	OR

Shell scripting - Boucles

Syntaxe de la boucle for:

```
for { nom de variable } in { liste }
            do
                    commandes à exécuter
            done

for (( expr1; expr2; expr3 ))
         do
   commandes à exécuter
         Done
```

Shell scripting - Boucles

Exemples :

```
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done

for ((  i = 0 ;  i <= 5;  i++  ))do
echo "Welcome $i times"
done
```

Shell scripting - Boucles

Syntaxe:

```
while [ condition ]
           do
                 commande1
                 commande2
                 ....
            done
```

Exemple :

```
n=$1
i=1
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n`"
  i=`expr $i + 1`
done
```

Exercices

Ecrire une commande de lecture permettant de boucler sur une lecture au clavier jusqu'à obtention de la chaîne de caractères
"oui" ou "non".

Exercices

```
#! /bin/sh
# lecture oui non : commande lecture

q="Repondez par oui ou non: "
echo $q
# dans echo -n $q  l'option  -n permet de ne pas passer à la ligne suivante

read reponse
echo $reponse

while [ "$reponse" != oui  -a $reponse != "non" ]
# des espaces OBLIGATOIRES à cote de !=
do
  echo $q
  read reponse
done
```

Exercices

Ecrire une commande enlever nom liste_de_noms qui affiche sur la sortie standard la liste de noms privée de toutes les occurences de nom.

Exercices

```
#! /bin/sh
# commande:  enlever nom liste_de_noms

if [ $# -le 1 ]
then
   echo "usage : $0 nom liste_de_noms"
   exit 1
fi
# $# est le nombre de vrais arguments (sans le nom de la cde)

nom=$1
shift

# $* désigne tous les arguments de la ligne de commande
for i in $*
do
  if [ $i != $nom ]
  then echo -n "$i "
  fi
done

# pour passer à la ligne avant de terminer
echo
exit 0
```

Exercices

Ecrire une commande **islog** *permettant de tester si un utilisateur est connecté sur la machine hôte, en utilisant la commande* **who**.

Par exemple :
islog dupont affichera dupont present si dupont est connecté, ou dupont non connecte dans le cas contraire.

(On supposera que) La commande who affiche les informations suivantes : (cet exercice ne peut pas être vraiment testé maintenant de cette façon).

```
collin   tty7         2010-03-11 08:31
collin   pts/0        2010-03-11 08:32
dupont   tty3         2010-03-11 08:25
martin   tty5         2010-03-11 08:40
```

Exercices

```
#! /bin/sh
# commande islog

if [ $# != 1 ]
then
  echo usage : $0 "nom utilisateur"
  exit 0
fi


# who | grep -q $1
# ou bien la commande grep suivante plus affinée:

who | grep -q "^$1\>"

if [ $? -eq  0  ]
  then echo $1 present

  else echo $1 "non connecte"
fi

exit 0
```

Exercices

Ecrire un script-shell **substitution** *effectuant la substitution dans le fichier passé en premier argument, de toutes les occurences de la chaîne de caractères passée en deuxième argument par la chaîne passée en troisième argument. Le résultat sera rangé dans le fichier désigné par le premier argument.*
Restriction : les arguments ne comportent pas d'espace.

Exercices

```
#! /bin/sh
# commande substitution

if [ $# != 3 ]
then
  echo usage : $0 fichier chaine1 chaine2
  exit 0
fi


sed 's/'$2'/'$3'/g' $1 > /tmp/$$
mv -f /tmp/$$ $1

# On teste par :  ./substitution fichierSub "chaine" "essai"
# ou encore par : ./substitution fichierSub chaine essai
# puisque les chaînes ne contiennent pas d'espace.
# Script correct si on ne veut pas d'espace dans les chaines à substituer,
```

Questions?