Programmation en bash

Formation Linux

Plan

Objectifs

– Comprendre le système des scripts de démarrage
– Ecrire des scripts simples

Les variables en bash

- Variables scalaires
  - Pas d’espaces autour du =
  - Pas de $ pour la déclaration
  - Utilisation du $ pour utiliser la variable
- Devant le nom
- Le nom de la variable peut être mis entre { }
  - ${comme_ca}

```
[user@linux ~]$ os="GNU/Linux"
[user@linux ~]$ echo “This OS is $os"
This OS is GNU/Linux
```

Les variables en bash

- Les tableaux
  - Variable scalaire
  - Utilisation des [ ]
- Pour écrire dedans
- Pour accéder aux valeurs
  - Les tableaux associatifs dans Bash4
- Indexés par des chaînes de caractère

```
[user@linux ~]$ array[0]="dummy”
[user@linux ~]$ array[1]="foo”
[user@linux ~]$ array[2]="bar”
[user@linux ~]$ echo ${array[1]}
foo
```

Exécuter des commandes

  – Back quote `
  – Parenthèse
- Les quotes : « " » « ' » « ` »
- Commentaires : #

```
[user@linux ~]$ var=`uname -sr`
[user@linux ~]$ var2=$(uname -sr)
[user@linux ~]$ echo $var $var2
Linux 2.6.25-2-686 Linux 2.6.25-2-686

# this is a comment
var="ls"
echo "var = $var" # print : var = ls
echo 'var = $var' # print : var = $var
echo ‘”$var”’ # print : ls result
# this is a comment
var="ls"
echo "var = $var" # print : var = ls
echo 'var = $var' # print : var = $var
echo ‘”$var”’ # print : ls result
```

Ecrire sur l’écran,

- echo -e CHAINE
  - -e : active les séquences
  - d’échappements \
- Read [-p PHRASE] [variable]

```
[user@linux ~]$ read -p “What is the OS Name ? " os
What is the OS Name ? OpenBSD
[user@linux ~]$ echo “This Os is $os"
This Os is OpenBSD

read # store the input into REPLY
read VAR # store the input into VAR
```

Ecrire un script

– Rappelez vous le #!shebang
  - Première ligne du fichier
  - Permet de choisir le bon interpréteur

– Bash par défaut si pas de shebang

```
#!/bin/bash

#!/usr/bin/perl

#!/usr/bin/python
```

Ecrire un script

– Rendre un script exécutable
– Lancer le script

```
[user@linux ~]$ chmod +x script.sh

./script.sh

source script.sh

. script.sh
```

It starts the program specified in the shebang to execute the script. Only exported variables are available from within the script.

The script is executed form within the running shell. The script has read/write access on all the variables defined by the shell.

Les arguments en ligne de commande

- Accès aux arguments
- Facilités
  - $* :
- Tous les arguments en tant que chaine
  - $@ : sous forme de tableau

```
#!/bin/bash
# Argv script example
#
echo $0 $1 $2

[user@linux ~]$./script labo linux
./script labo linux
```

Les codes de sorties

- Instruction exit
  - Convention sur les Unix
- 0 : Tout va bien
- != 0 : Echec
- L’instruction trap permet de récupérer les
- signaux
- echo $?

```
trap “echo You’ve pressed Ctrl-C” SIGINT trap “echo You’ve pressed Ctrl-C” SIGINT
```

Les tests

- Permet de tester des booléens
  - [ ] : pour une condition
  - [[ ]] : pour plusieurs
  - Retourne 0 si vrai
  - Manuel : man test

```
[user@linux ~]$ test 1 -gt 0; echo $?
0
[user@linux ~]$ [ 1 -lt 0 ]; echo $?
1
[user@linux ~]$ [[ 1 -lt 0 && 3 -gt 2 ]]; echo $?
1
[user@linux ~]$ test 1 -gt 0; echo $?
0
[user@linux ~]$ [ 1 -lt 0 ]; echo $?
1
[user@linux ~]$ [[ 1 -lt 0 && 3 -gt 2 ]]; echo $?
1
```

Les expressions mathématiques :

- expr
  - Ecrit le résultat sur STDOUT
  - Division entière (euclidienne)
- let
  - Dans bash
  - Pas de STDOUT
  - Fonctionne avec des variables
- Expansion arirhmetique
  - $((3+2))

```
[user@linux ~]$ expr 4 + 3
7
[user@linux ~]$ i=0; let i++; echo $i
1
[user@linux ~]$ echo “3+2=$((3+2))”
3+2=5
```

Les conditions

– If, then

– If, then, else

```
if [ ‘a’ = ‘a’ ]; then echo "true"; fi
if [ ‘a’ = ‘a’ ]; then
echo "true";
fi
if [ ‘a’ = ‘a’ ]; then echo "true"; fi
if [ ‘a’ = ‘a’ ]; then
echo "true";
fi
```

```
if [ ‘a’ = ‘a’ ]; then echo "true";
else echo "false"; fi
if [ ‘a’ = ‘a’ ]; then
echo "true";
else
echo "false";
fi
if [ ‘a’ = ‘a’ ]; then echo "true";
else echo "false"; fi
if [ ‘a’ = ‘a’ ]; then
echo "true";
else
echo "false";
fi
```

Les conditions

– If, then, elif

– Case

```
case "$OS" in
Unix)
echo ’Your operating system is Unix’
;;
[Bb][Ss][Dd])
echo ‘your operating system is a BSD’
;;
Lin*)
echo ‘your operating system is GNU/Linux’
;;
*)
echo ‘your operating system is unknown’
;;
esac
```

```
if [ ‘a’ = ‘a’ ]; then echo "true";
elif [ ‘b’ = ‘b’ ]; echo "false"; fi
if [ ‘a’ = ‘a’ ]; then
echo "true";
elif [ ‘b’ = ‘b’ ];
echo "false";
fi
if [ ‘a’ = ‘a’ ]; then echo "true";
elif [ ‘b’ = ‘b’ ]; echo "false"; fi
if [ ‘a’ = ‘a’ ]; then
echo "true";
elif [ ‘b’ = ‘b’ ];
echo "false";
fi
```

Les boucles

– Tant que : while condition ; do lot of things ; done
  ● Entre dans la boucle si la condition est vraie
– A moins que : until condition do lot of things ; done
  ● Contraire du while, entre si la condition est fausse

```
i=0
while [ $i -lt 3 ]; do
echo $i
let i+=1
done
```

```
i=0
until [ $i -eq 3 ]; do
echo $i
let i+=1
done
```

Les boucles

- For each :
  - Boucle sur une liste de valeurs
- Séparées par la variable $IFS (saut de ligne par défaut)
- Chaînes et nombres
  - {m..n} : génère une liste de nombres de m à n séparés par des
- espaces

```
for i in {1..5}; do
echo $i
done
```

```
for chain in INPUT OUTPUT FORWARD; do
echo $chain
done
```

Les boucles

- For each :
  - Contrôle du flot de la boucle
- break : interrompre la boucle
- continue : Reprend au début de la boucle

Les fonctions

- Portés des variables :
  - Par défaut : toutes les variables sont globales
- Visible et modifiable depuis l'extérieur
- Le mot clé local
  - Protège la variable
- Arguments :
  - On peut en déclarer
  - Ou utiliser $1, $2...
- Code de retour :
  - Toujours numérique
- Retourner une chaîne : Ecrire sur STDOUT

Les fonctions

- Syntaxe générale
- Appel de fonction

```
function my_function() {
local var1
local var2="value"
command1
command2
return val;
}

my_function #returns int
a=$?
str=$(my_string_function) #writes to STDOUT (i.e: echo)
```