Shell TD1

Cédric Enclos

TELECOM Nancy - Université de Lorraine

Octobre 2018

Le Shell

POSIX (Portable Operation System Interface)

Standards (1988) pour une interface commune à tous les systèmes UNIX

Interpréteur de commande :

- sh (Bourne Shell, 1971)
- csh (C Shell, 1978)
- ksh (Korn Shell, 1983)
- bash (Bourne Again Shell, 1988)
- zsh (Z Shell, 1990)

Syntaxe Bash
Shebang

Shebang (ou Sha-bang)

Sharp (#) Bang (!)
Entête d'un fichier texte pour définir un script

Pour un script bash

```
#! /bin/bash
```

Syntaxe Bash
Chaînes de caractères

' vs. "

' : Pas d'interprétation

" : Perméable à l'interprétation : les variables sont remplacées

Exemple

```
$ a=world
$ echo 'Hello $a'
Hello $a
$ echo "Hello $a"
Hello world
```

Syntaxe Bash
Backquotes

Backquotes ‘ ‘

Exécute la commande contenu entre les backquotes
Equivalent à $()

Exemple

```
$ a=date
$ echo $a
date
$ b=‘date‘ # Equivalent à $ b=$(date)
$ echo $b
mercredi 18 octobre 2018, 8:05:50 (UTC+0200)
```

Syntaxe Bash
if

Syntaxe sur une ligne

```
if [[ condition ]] ; then
  ...
elif command args ; then
  ...
else
  ...
fi
```

Syntaxe sur deux lignes

```
if [[ condition ]]
then
  ...
elif command args
then
  ...
else
  ...
fi
```

Syntaxe Bash
conditions

Chaînes de caractères

- Egalité : ==
- Différence : !=
- Infériorité stricte : <
- Supériorité stricte : >
- Chaîne nulle (vide) : -z
- Chaîne non nulle : -n

Entiers

- Egalité : -eq
- Différence : -ne
- Infériorité stricte : -lt
- Infériorité : -le
- Supériorité stricte : -gt
- Supériorité : -ge

On peut combiner les tests avec le "et" et le "ou" logique : && et ||

Syntaxe Bash

case

Case

```
case $a in
  ping)
    echo "pong!"
    ;;
  now)
    echo `date`
    ;;
  *)
    echo "Action par défaut"
esac
```

Syntaxe Bash
while

While

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

Boucle sur la lecture de la saisie clavier

```
while read line
do
  echo $line
done
```

Syntaxe Bash
for

For

```
for i in $list
do
  ...
done
```

Boucle sur les arguments

```
count=1
for i in "$@"
do
  echo "Arg $count : $i"
  count=$(expr $count + 1)
done
```

Syntaxe Bash
Gestion des arguments

Variables spéciales liées aux paramètres

- $* : Liste des paramètres séparés par un espace
- $@ : Pareil que $*. Comportement différent entre double quotes
- $0 : Nom du script
- $n : Paramètre numéro n, n compris entre 1 et 9
- $# : Nombre de paramètres

shift

Permet de décaler les paramètres
$1 prend l'ancienne valeur de $2, ...
L'ancienne valeur de $1 n'est plus accessible et $# est mis à jour

Syntaxe Bash
Fonctions

Fonction

```
function my_function() {# "function" optional
  ... # $1, $2 ... pour accéder aux paramètres
  return 0 # Optional
}
my_function param1 param2
```

Portée des variables

Les variables déclarées dans une fonction sont globales tant qu'elles n'ont pas été unset
Pour s'assurer que la variable n'est pas globale on peut utiliser le mot clé local (local a=3)

Redirections et Pipes

Entrée et Sortie d'un processus

stdin, stdout, stderr

Un processus possède :

- Une entrée (stdin) - Par défaut les saisies du clavier
- Une sortie (stdout) - Par défaut le terminal
- Une sortie d'erreur (stderr) - Par défaut le terminal

Redirections et Pipes

Redirections

Opérateurs de redirection de fichier

- < : redéfinit l'entrée
- > : redéfinit la sortie
- 2> : redéfinit la sortie d'erreur
- &> : redéfinit toutes les sorties
- >> : redéfinit la sortie vers la fin du fichier

/dev/null

Pour ignorer une sortie, il est possible de la redigirer vers le fichier spéciale /dev/null

Redirections et Pipes
Pipes

Les pipes | (ou tubes)

Permet de chaîner les commandes

commande1 | commande2 | commande3

On relie la sortie de commande1 vers l'entrée de commande2, et ainsi de suite

Exemple

```
ps -e | grep bash | wc -l
# Affiche le nombre de processus bash en cours
```