Systèmes d'exploitation

L2 Informatique

Scripts shell bash

Nicolas Garric - nicolas.garric@univ-jfc.fr

Scripts shell

- Scripts shell = langage de programmation interprété
- C'est un simple fichier texte **exécutable** (droit x) commençant par les caractères #!/bin/bash
- On peut choisir le langage shell que l'on souhaite : sh, tcsh, bash, ksh, csh, …
- Les syntaxes des différents shell sont légèrement différentes
- Bash = Bourne Again SHell

Pourquoi faire des scripts shell

- Permet de lancer des programmes évolués sans qu'il y ait d'interface graphique : serveurs distants, systèmes embarqués,...
- Permet d'automatiser des tâches répétitives d’administration des systèmes
- Comprendre (et améliorer ?) le fonctionnement de linux : de nombreux shell sont utilisés par le système

Inconvénients du shell

- Langage interprété donc lent. Peut être pénalisant pour certaines tâches
- Documentation peu accessible et assez complexe (cf mini tutoriel pour une synthèse en français)
- Mise au point des scripts laborieuse (surtout au début)

Principaux concepts

- **Variables** :
  - Manipulation
  - Évaluation
- **Structures de contrôle :**
  - Alternative : if then else
  - Itération : boucles for
  - Choix multiple : case
- **Fonctions**

Variables

- Les variables sont stockées comme des chaînes de caractères.

- On utilise les apostrophes pour les définir

```
var='sera la fin du monde'
```

- On utilise $ pour l'utiliser :

```
echo le 17 avril $var
```

Guillemets vs apostrophes

- On peut utiliser les guillemets pour définir les variables : les valeurs des variables qui sont contenues dans le texte seront évaluées

```
nom='toto'

var1="$nom est un etudiant"

var2='$nom est un etudiant'
```

- Avec `echo $var1` et `echo $var2` , on obtient :

```
toto est un etudiant

$nom est un etudiant
```

Expressions arithmétiques

- L'évaluation des expressions arithmétiques se fait avec `$(( ... ))`

```
n=12
```

`echo $((n+2))` donne `14`

`echo $n+2` donne `12+2`

Découpage des chemins

- Les commandes `dirname` et `basename` permettent d'extraire du nom complet le répertoire où se trouve un fichier et son nom

  `dirname /etc/bin/passwd` donne `/etc/bin`

  `basename /etc/bin/passwd` donne `passwd`

Évaluation des commandes

- Il est souvent utile de stocker dans une variable le résultat d'une commande
- Cela se fait avec `$( ... )`

```
fichiers=$(ls t*)
```

Découpage de chaînes

- Il existe beaucoup de fonctionnalités pour découper des chaînes
- `##` élimine la plus longue chaîne correspondant avec un motif

  `fic=/etc/bin/toto.txt`

  `echo ${fic##*.}` affiche l'extension
- `%` garde la plus courte chaîne correspondant avec un motif

  `echo ${fic%.*}` affiche le nom du fichier

Ligne de commande

- On peut récupérer les arguments d'une commande avec les variables `$0`, `$1`, ...
- `$#` représente le nombre d'arguments
- `$*` représente tous les arguments

si je lance le script `essai.sh philippe le bel`

`echo $1` affiche `le`

`echo $#` affiche `3`

Alternative (if then else)

```
if [ condition ]
then
    action1
else
    action2
fi
```

- Il existe tout un tas de conditions sur les fichiers les chaînes et les nombres

Alternative (if then else)

- Existence d'un fichier

```
if [ -e /etc/passwd ]
```

- Teste si une chaîne est vide

```
if [ -z "$var" ]
```

- Teste si un nombre est inférieur à une valeur

```
if [ $nombre -lt 17 ]
```

Boucles for

```
for x in un deux trois quatre
do
    echo x= $x
done

for arg in $*
do
    echo $arg
done
```

Boucles while

```
x=1
while [ $x -le 5 ]
do
  echo "x= $x"
  x=$(( $x + 1 ))
done
```

Choix multiple : case

```
case "$x" in
   go)
        echo "demarrage"
        ;;
   stop)
        echo "arret"
        ;;
    *)
        echo "valeur invalide de x ($x)''
esac
```

Fonctions

- Définition

```
mafonction() {
    echo "appel de mafonction..."
}
```

- appel

```
mafonction
mafonction
```