I106B
19. Boucles, listes et tableaux

HTTPS://WEB.ARCHIVE.ORG/WEB/20161119052258/HTTP://RYANSTUTORIALS.NET:80/BASH-SCRIPTING-TUTORIAL/BASH-LOOPS.PHP

Tableau

- **Toutes les variables sont des tableaux !**
- Les indices commencent à 0
- Accès à un indice particulier :
  - `t[i]="valeur"` → assignation à l'indice `i` de `t`
  - `echo ${t[i]}` → accès à la valeur de `t[i]`
  - `$` optionnel pour accéder à la valeur de l'indice i

    `${t[$i]}` ↔ `${t[i]}`
- Toute variable est un tableau ⇒ accès habituel à la variable = accès à l'indice 0 de ce tableau :
  - `t="toto"` ⇔ `t[0]="toto"`
  - `echo $t` ⇔ `echo ${t[0]}`

Tableau

- Pas de contrôle de borne, tous les indices sont disponibles
- Trous possibles
- Nombre d'éléments dans le tableau t :

```
${#t[*]}
```

  - Il s'agit de la taille logique du tableau
  - Attention : s'il y a des trous dans le tableau, le nombre d'éléments ne donnera pas l'indice du dernier élément + 1

Liste

- Une variable peut aussi être considérée comme une liste
  - Les éléments sont séparés par des *whitespaces* (i.e. espace, passage à la ligne, tabulation)

```
liste="el1 el2 el3 el4"

fichiers=$(ls)
```

- **$*** → *liste* des arguments du script

(rappel: $# = nombre d'arguments)

Conversions liste ↔ tableau

En considérant que `t` est un tableau et `l` une liste:

- conversion liste → tableau

```
t=($l)
```

- conversion tableau → liste

```
l=${t[*]}
```

while


```
while cmd1
do
   cmd2
done
```

→ Tant que `cmd1` réussit (code retour=0):

→ exécute `cmd2`

Attention: les espaces et les retours à la ligne sont **important**s.

Exemple


```
#!/bin/bash
cnt=0
args=($*)
while [ $cnt -lt ${#args[*]} ]
do
    ((cnt++))
    echo argument $cnt: ${args[cnt-1]}
done
```

while (suite)

Les boucles infinies permettent de garder le shell à l'écoute. Dans ce cas, il faut utiliser la commande `exit` ou `break` pour sortir de la boucle.

```
while (true)   # ou while true ou while ((1))
do
  read -p "Enter a line (quit to stop): " rep
  echo $rep
  if [ "$rep" = "quit" ]; then
    exit 0
  fi
done
```

until

Comme `while`, mais la condition est inversée.

```
#!/bin/bash
cnt=0
args=($*)
until [ $cnt -ge ${#args[*]} ]
do
    echo ${args[$cnt]}
    ((cnt++))
done
```

for


```
for var in $list
do
   cmd2
done
```

Assigne `var` successivement à tous les éléments de la liste `list`:

exécute `cmd2`

Le `for` du bash est donc une boucle « for each ».

Attention: les espaces et les retours à la ligne sont **important**s.

Exemple


```
for i in 1 2 3 4 5 ; do
    echo $i
done

liste="a b c d"
for el in $liste ; do
    echo $el
done

for arg in $* ; do
    echo argument: $arg
done
```

for range


```
#!/bin/bash
for value in {1..5}    # valeurs: 1 2 3 4 5
do
   echo $value
done

for value in {10..0..2}    # valeurs: 10 8 6 4 2 0
do
   echo $value
done
```

Notez que les bornes d'un *range* ne peuvent pas être des variables (dans ce cas, mieux vaut utiliser une boucle *while* avec compteur).

break & continue

- **`break`** : quitte instantanément une boucle
- **`continue`** : passe à l'itération suivante

Parcours d'une liste de fichiers (1)

- Imaginons qu'on crée deux fichiers :

```
touch f1 ; touch "f2 f3"
```

- Itération par globbing :

```
for f in * ; do echo $f ; done
# f1
# f2 f3
```

→ Le globbing génère une liste correcte, même si les fichiers contiennent un espace.

Parcours d'une liste de fichiers (2)

- Imaginons qu'on crée deux fichiers :

```
touch f1 ; touch "f2 f3"
```

- Itération par substitution de commande :

```
for f in $(ls) ; do echo $f ; done
# f1
# f2
# f3
```

→ La substitution de commande renvoie une liste pour laquelle les retours à la lignes **et** les espaces sont des séparateurs, y compris dans les noms de fichiers!

Parcours d'une liste de fichiers (3)

- Imaginons qu'on crée deux fichiers :

```
touch f1 ; touch "f2 f3"
```

- Itération avec while et read :

```
ls | while read f ; do echo $f ; done
# f1
# f2 f3
```

→ La sortie de `ls` est redirigée vers `read` qui sépare les éléments sur base des retours à la ligne uniquement.

Exemples de scripts

- Quelques scripts utilisant des alternatives et boucles sont disponibles dans le repertoire "**ex_scripts**".