Structures de contrôle

Patrick MARCEL, Université d'Orléans

L2 Outils du développeur — S3

Boucle for

Répéter une commande en faisant parcourir un **ensemble de valeurs** (avec expansion) à une variable.

```
$ for i in America/New_York Europe/London
> Asia/Tokyo; do
> TZ=$i LC_ALL=en_GB.UTF-8 date;
> done
Thu Sep 27 03:33:31 EDT 2013
Thu Sep 27 08:33:31 BST 2013
Fri Sep 28 16:33:31 JST 2013
```

Séparateur et IFS

La **variable** `IFS` permet de redéfinir le séparateur de valeurs lors de l'**expansion**.

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin
$ (IFS=:; for p in $PATH; do echo $p; done)
/usr/local/bin
/usr/bin
/bin
```

Filtrage case

Exécution **conditionnelle** de la liste de commandes associée au premier **motif reconnaissant** un certain mot.

```
$ echo 'continuer (o/n) ?'; read a
case $a in
>    o|O) echo continuer;;
>    n|N) echo stop;;
>    *) echo "répondre o ou n";;
> esac
```

Valeur de retour

Tout **processus** s'arrête avec une **valeur de retour**, un entier compris entre 0 et 255.

```
$ true; echo $?
0
$ false; echo $?
1
```

Une valeur de retour **0** signifie **vrai**.
Une valeur de retour **non nulle** signifie **faux**.

Conditionnelle if

Exécution **conditionnelle** selon la **valeur de retour** d'une commande.

```
$ if ls conso.csv >/dev/null 2>&1; then
>  echo "fichier csv présent"
> else
>  gzip -c conso.gz > conso.csv
> fi
```

Boucle while

Exécution du **corps de la boucle** tant que la **valeur de retour** d'une commande est **0**.

```
$ t=true; while $t; do
>  echo "continuer (true/false) ?"
>  read t
> done
```

break et continue

La commande interne **break** sort immédiatement de la bouche **while** ou **for** englobante la plus proche.

La commande interne **continue** passe immédiatement à l'itération suivante de la boucle **while** ou **for** englobante la plus proche.

En ajoutant en argument un **niveau** $\texttt{n} \geqslant 1$ la commande considère la $\texttt{n}^{\text{e}}$ boucle englobante au lieu de la 1^{re}.

Conditions avec test et [

```
$ test -f /etc/passwd; echo $?
0
```

La commande interne **test** évalue une expression et termine avec la valeur de retour `0` si elle est vraie, `1` sinon. la notation `[ ... ]` est équivalente.

```
$ if [ 3 -gt 2 ]; then echo exact; fi
exact
```

Remarque Il est aussi possible d'utiliser le *bashisme* `[[` (supporte d'avantage d'opérateurs)

Tests sur les fichiers

`[ -e $X ]` vrai si le fichier existe

`[ -d $X ]` vrai si le répertoire existe

`[ -f $X ]` vrai si le fichier régulier existe

`[ -r $X ]` vrai si le fichier est accessible en lecture

`[ -w $X ]` vrai si le fichier est accessible en écriture

etc

Comparaison de chaînes

`[ $X ]` vrai si la chaine est non vide

`[ $X = $Y ]` vrai si les chaînes sont identiques

`[ $X != $Y ]` vrai si les chaînes sont distinctes

`[ $X < $Y ]` comparaison lexicographique ASCII

`[ $X > $Y ]` idem

Comparaison d'entiers

`[ $X -eq $Y ]` vrai si les entiers sont égaux

`[ $X -ne $Y ]` vrai si les entiers sont inégaux

`[ $X -lt $Y ]` vrai si $x < y$

`[ $X -le $Y ]` vrai si $x \leqslant y$

`[ $X -gt $Y ]` vrai si $x > y$

`[ $X -ge $Y ]` vrai si $x \geqslant y$

Variables

Les **variables shell** contiennent des chaînes de caractères.

```
$ workdir=/tmp
```

L'**affectation** de variable s'effectue avec l'opérateur `=`.

La commande interne `export` permet d'exporter une variable en tant que **variable d'environnement** pour les commandes suivantes.

La forme raccourcie `export workdir=/tmp` combine affectation et exportation.

Opérations arithmétiques

```
$ X=5
$ Y=2
$ echo $((5*(X+Y)+1))
36
```

L'expression arithmétique entourée de `$((...))` est évaluée et son résultat lui est substitué.

Remarque. Il est aussi possible d'utiliser la commande externe **expr** ou encore les *bashismes* **let** et `((...))`.

```
$ w=‘expr $u + $v‘
$ let ‘‘w=$u + $v’’
$ ((w=$u + $v))
$ (( w++ ))
```

Flottants avec bc

Le shell ne sait manipuler que des **chaînes de caractères** et des **entiers**.

La commande **bc** est une calculatrice qui sait travailler sur les flottants.

```
$ echo 'scale=2; 3/2' | bc
1.50
```

Substitution de variables

`${var}` : même chose que sans les accolades ;

`${#var}` : longueur du contenu de la variable var ;

`${var%motif}` : valeur de la variable var auquel au retire le plus petit suffixe motif ;

`${var#motif}` : valeur de la variable var auquel au retire le plus petit préfixe motif ;

`${var%%motif}` et `${var##motif}` : même chose que précédemment avec le plus grand.

Substitution de commande

```
$ kill $(cat /var/run/daemon.pid)
```

Lors de l'**expansion**, les commandes entourées par `$(...)` ou par `` `...` `` sont interprétées et leur résultat leur est substitué.

Les sauts de lignes sont interprétés comme des espaces.

Scripts élémentaires

Un **script shell** est un fichier contenant une séquence de **commandes**.

```
$ bash monscript
```

La commande **bash** accepte un nom de script en paramètre qui est alors exécuté au lieu de passer en mode interactif.

```
$ . monscript
```

La **commande interne** `.` exécute les commandes du script dans le shell courant.