Structure de contrôle for et if

Itération for

L'itération for possède plusieurs syntaxes dont les deux plus générales sont :

1. Première forme :

   for *var*

   do

   suite_de_commandes

   done

   Lorsque cette syntaxe est utilisée, la variable var prend successivement la valeur de chaque paramètre de position initialisé.

   Exemple : programme *for_arg*

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

for i
do
  echo $i
  echo "Passage a l'argument suivant ..."
done
- - - - - - - - - - - - - - - - - -
$ for_arg un deux => deux paramètres de position initialisés
un
Passage a l'argument suivant ...
deux
Passage a l'argument suivant ...
$
```

   La commande composée **for** traite les paramètres de position sans tenir compte de la manière dont ils ont été initialisés (lors de l'appel d'un programme shell ou bien par la commande interne **set**).

   Exemple : programme *for_set*

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

set $(date)

for i
do
  echo $i
done
- - - - - - - - - - - - - - - - - -
$ for_set
```

```
dimanche
17
décembre
2006,
11:22:10
(UTC+0100)
$
```

2. Deuxième forme :

   for *var* **in** *liste_mots*

   do

   suites_de_commandes

   done

   La variable *var* prend successivement la valeur de chaque mot de *liste_mots*.

   Exemple : programme *for_liste*

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

for a in toto tata
do
  echo $a
done
- - - - - - - - - - - - - - - - - - - -
$ for_liste
toto
tata
$
```

   Si *liste_mots* contient des substitutions, elles sont préalablement traitées par **bash**.

   Exemple : programme *affich.ls*

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

for i in tmp $(pwd)
do
  echo " --- $i ---"
  ls $i
done
- - - - - - - - - - - - - - - - - - - - - - - - -
$ affich.ls
--- tmp ---
gamma
--- /home/sanchis/Rep ---
affich.ls alpha beta tmp
$
```

   Exemple : Il est possible d'utiliser des intervalles :

```
#!/bin/bash
for i in {1..5}
do
   echo "Welcome $i times"
done
```

Il est également possible de spécifier un intervalle:

```
#!/bin/bash
echo "Bash version ${BASH_VERSION}..."
for i in {0..10..2}
  do
     echo "Welcome $i times"
 done
```

Choix if

La commande interne **if** implante le choix alternatif.

Syntaxe :

if *suite_de_commandes1*

then

suite_de_commandes2

[elif *suite_de_commandes* **; then** *suite_de_commandes* **] ...**

[else *suite_de_commandes* **]**

fi

Le fonctionnement est le suivant : *suite_de_commandes1* est exécutée ; si son code de retour est égal à **0**, alors *suite_de_commandes2* est exécutée sinon c'est la branche **elif** ou la branche **else** qui est exécutée, si elle existe.

Exemple : programme *rm1*

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

if rm $1 2>/dev/null
then echo $1 a ete supprime
else echo $1 n\'a pas ete supprime
fi
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - -
$ >toto    => création du fichier toto
$
$ rm1 toto
toto a ete supprime
```

```
$
$ rm1 toto
toto n'a pas ete supprime
$
```

Lorsque la commande *rm1 toto* est exécutée, si le fichier *toto* est effaçable, le fichier est effectivement supprimé, la commande unix **rm** renvoie un code de retour égal à **0** et c'est la suite de commandes qui suit le mot-clé **then** qui est exécutée ; le message *toto a ete supprime* est affiché sur la sortie standard.

Si *toto* n'est pas effaçable, l'exécution de la commande **rm** échoue ; celle-ci affiche un message sur la sortie standard pour les messages d'erreur que l'utilisateur ne voit pas car celle-ci a été redirigée vers le puits, puis renvoie un code de retour différent de **0**. C'est la suite de commandes qui suit **else** qui est exécutée : le message *toto n'a pas ete supprime* s'affiche sur la sortie standard.

Les mots **if, then, else, elif** et **fi** sont des mots-clé. Par conséquent, pour indenter une structure if suivant le « style langage C », on pourra l'écrire de la manière suivante :

if *suite_de_commandes1* **; then**

suite_de_commandes2

else

suite_de_commandes]

fi

La structure de contrôle doit comporter autant de mots-clés **fi** que de **if** (une branche **elif** ne doit pas se terminer par un **fi**).

```
if ...
  then ...
  elif ...
    then ...
fi
```

```
if ...
  then ...
  else if ...
    then ...
  fi
fi
```

Dans une succession de **if** imbriqués où le nombre de **else** est inférieur au nombre de **then**, le mot-clé **fi** précise l'association entre les **else** et les **if**.

```
if ...
  then ...
    if ...
      then ...
    fi
  else ...
fi
```

Commande composée **[[** :

La commande interne composée **[[** est souvent utilisée avec la commande interne composée **if**. Elle permet l'évaluation d'expressions conditionnelles portant sur des objets aussi différents que les permissions sur une entrée, la valeur d'une chaîne de caractères ou encore l'état d'une option de la commande interne **set**.

Syntaxe : **[[** *expr_cond* **]]**

Les deux caractères **crochets** doivent être collés et un caractère séparateur doit être présent de part et d'autre de *expr_cond*. Les mots **[[** et **]]** sont des mots-clé.

Le fonctionnement de cette commande interne est le suivant : l'expression conditionnelle *expr_cond* est évaluée et si sa valeur est *Vrai*, alors le code de retour de la commande interne**[[** est égal à **0**. Si sa valeur est *Faux*, le code de retour est égal à **1**.
Si *expr_cond* est mal formée ou si les caractères **crochets** ne sont pas collés, une valeur différente est retournée.

La commande interne **[[** offre de nombreuses expressions conditionnelles ; c'est pourquoi, seules les principales formes de *expr_cond* seront présentées, regroupées par catégories.

- Permissions :
  - **-r** *entrée* vraie si *entrée* existe et est accessible en lecture par le processus courant.
  - **-w** *entrée* vraie si *entrée* existe et est accessible en écriture par le processus courant.
  - **-x** *entrée* vraie si le fichier *entrée* existe et est exécutable par le processus courant ou si le répertoire *entrée* existe et le processus courant possède la permission de passage.

```
$ echo coucou > toto
$ chmod 200 toto
$ ls -l toto
--w------- 1 sanchis sanchis 7 déc 17 17:21 toto
$
$ if [[ -r toto ]]
> then cat toto
> fi
```

```
$     => aucun affichage donc toto n'existe pas ou n'est pas
         accessible en lecture,
$     => dans le cas présent, il est non lisible
$
$ echo $?
0     => code de retour de la commande interne if
$
```

Mais,

```
$ [[ -r toto ]]
$
$ echo $?
1     => code de retour de la commande interne [[
```

- Types d'une entrée :
  - **-f** *entrée* vraie si *entrée* existe et est un fichier ordinaire.
  - **-d** *entrée* vraie si *entrée* existe et est un répertoire.

Exemple : programme *affic*

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

if [[ -f $1 ]]
then
              echo $1 : fichier ordinaire
              cat $1
elif [[ -d $1 ]]
              then
                echo $1 : repertoire
                ls $1
              else
                echo $1 : type non traite
fi
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - -
```

- Renseignements divers sur une entrée :
  - **-a** *entrée* vraie si *entrée* existe.
  - **-s** *entrée* vraie si *entrée* existe et sa taille est différente de zéro.
  - *entrée1* **-nt** *entrée2* vraie si *entrée1* existe et sa date de modification est plus récente que celle de *entrée2*.
  - *entrée1* **-ot** *entrée2* vraie si *entrée1* existe et est plus ancienne que celle de *entrée2*.
- Longueur d'une chaîne de caractères :
  - **-z** *ch* vraie si la longueur de la chaîne *ch* est égale à zéro.
  - **-z** *ch* vraie si la longueur de la chaîne *ch* est différente de zéro.
- Comparaisons de chaînes de caractères :
  - *ch1* **<** *ch2* vraie si *ch1* précède *ch2*.
  - *ch1* **>** *ch2* vraie si *ch1* suit *ch2*.

L'ordre des chaînes *ch1* et *ch2* est commandé par la valeur des paramètres régionaux.

- *ch* **==** *mod* vraie si la chaîne *ch* correspond au modèle *mod*.
- *ch* **!=** *mod* vraie si la chaîne *ch* ne correspond pas au modèle *mod*.

mod est un modèle de chaînes pouvant contenir caractères et expressions génériques.

```
$ a="au revoir"
$
$ [[ $a == 123 ]]     => faux
$
$ echo $?
1
$
$ [[ $a == a* ]]      => vrai, la valeur de a commence par le
caractère a
$
$ echo $?
0
$
```

Si par mégarde *ch* ou *mod* ne sont pas définies, la commande interne **[[** ne provoque pas d'erreur de syntaxe.

Exemple : programme *testval*

```
#!/bin/bash

if [[ $1 == $a ]]
  then echo OUI
  else echo >&2 NON
fi
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
$ testval coucou
NON    => dans testval, $1 est remplacé par coucou, la variable
a n'est pas définie
$
$ testval
OUI    => aucun des deux membres de l'égalité n'est défini
$
```

- Etat d'une option :
  - **-o** *opt* vraie si l'état de l'option *opt* de la commande interne set est à *on*.

```
$ set -o | grep noglob
noglob off
$
$ if [[ -o noglob ]]; then echo ON
> else echo OFF
> fi
OFF
$
```

- Composition d'expressions conditionnelles :
  - **(** *expr_cond* **)** vraie si *expr_cond* est vraie. Permet le regroupement d'expressions conditionnelles.
  - **!** *expr_cond* vraie si *expr_cond* est fausse.
  - *expr_cond1* **&&** *expr_cond2* vraie si les deux *expr_cond* sont vraies. L'expression *expr_cond2* n'est pas évaluée si *expr_cond1* est fausse.

- *expr_cond1* **||** *expr_cond2* vraie si une des deux *expr_cond* est vraie. L'expression *expr_cond2* n'est pas évaluée si *expr_cond1* est vraie.

Les quatre opérateurs ci-dessus ont été listés par ordre de priorité décroissante.

```
$ ls -l /etc/at.deny
-rw-r----- 1 root daemon 144 jan 3 2006 /etc/at.deny
$
$ if [[ ! ( -w /etc/at.deny || -r /etc/at.deny ) ]]
> then
>   echo OUI
> else
>   echo NON
> fi
OUI
$
```

Le fichier **/etc/at.deny** n'est accessible ni en lecture ni en écriture pour l'utilisateur *sanchis* ; le code de retour de la commande interne **[[** sera égal à zéro car l'expression conditionnelle est vraie.

Attention : on prendra soin de séparer les différents opérateurs et symboles par des **espaces**

```
$ if [[ !( -w /etc/at.deny || -r /etc/at.deny ) ]]
-bash: !: event : not found
$
```

Dans l'exemple ci-dessus, il n'y a aucun **blanc** entre **!** et **(**, ce qui provoque une erreur.

En combinant commande interne **[[**, opérateurs sur les codes de retour et regroupements de commandes, l'utilisation d'une structure **if** devient inutile.

```
$ [[ -r toto ]] || {
>   echo >&2 "Probleme de lecture sur toto"
> }
Probleme de lecture sur toto
$
```

Remarque : par souci de portabilité, **bash** intègre également l'ancienne commande interne **[**. Celle-ci possède des fonctionnalités similaires à celles de **[[** mais est plus délicate à utiliser.

```
$ a="au revoir"
$
$ [ $a = coucou ]    => l'opérateur égalité de [ est le symbole
=
-bash: [: too many arguments
$
```

Le caractère **espace** présent dans la valeur de la variable a provoque une erreur de syntaxe. Il est nécessaire de prendre davantage de précaution quand on utilise cette commande interne.

```
$ [ "$a" = coucou ]
$ echo $?
1
```