**Les boucles**

Nous allons découvrir dans ce chapitre un autre élément de base de tous les langages : les boucles. Ces structures permettent de répéter autant de fois que nécessaire une partie du code. En bash, on n'y échappe pas !

Les consignes sont les mêmes que pour le chapitre sur les conditions : il faut être vigilant sur la syntaxe. Une espace de trop ou de moins, l'oubli d'un caractère spécial et plus rien ne fonctionne. Soyez donc très rigoureux lorsque vous codez !

Si vous suivez cette simple règle, vous n'aurez pas de problèmes.

**while : boucler « tant que »**

Le type de boucle que l'on rencontre le plus couramment en bash estwhile.

Le principe est de faire un code qui ressemble à ceci :

TANT QUE test

FAIRE

------> effectuer\_une\_action

RECOMMENCER

En bash, on l'écrit comme ceci :

while [ test ]

do

        echo 'Action en boucle'

done

Il est aussi possible, comme pour leif, d'assembler les deux premières lignes en une, à condition de mettre un point-virgule :

while [ test ]; do

echo 'Action en boucle'

done

On va demander à l'utilisateur de dire « oui » et répéter cette action tant qu’il n'a pas fait ce que l'on voulait. Nous allons créer un scriptboucles.shpour l'occasion :

#!/bin/bash

while [ -z $reponse ] || [ $reponse != 'oui' ]

do

        read -p 'Dites oui : ' reponse

done

On fait deux tests.

1. Est-ce que$reponseest vide ?
2. Est-ce que$reponseest différent deoui ?

Comme il s'agit d'un OU (||), tant que l'un des deux tests est vrai, on recommence la boucle. Cette dernière pourrait se traduire par : « Tant que la réponse est vide ou que la réponse est différente deoui ».  
Nous sommes obligés de vérifier d'abord si la variable n'est pas vide, car si elle l'est, le second test plante (essayez, vous verrez).

Essayons ce script :

Dites oui : euh

Dites oui : non

Dites oui : bon

Dites oui : oui

Comme vous pouvez le voir, il ne s'arrête que lorsque l'on a tapéoui !

Il existe aussi le mot cléuntil, qui est l'exact inverse dewhile. Il signifie « Jusqu'à ce que ».  
Remplacez justewhileparuntildans le code précédent pour l'essayer.

**for : boucler sur une liste de valeurs**

Avertissement pour ceux qui ont déjà fait de la programmation : leforen bash ne se comporte pas de la même manière que leforauquel vous êtes habitués dans un autre langage, comme le C ou le PHP. Lisez donc attentivement.

**Parcourir une liste de valeurs**

La boucleforpermet de parcourir une liste de valeurs et de boucler autant de fois qu'il y a de valeurs.

Concrètement, la forme d'unforest la suivante :

POUR variable PRENANT valeur1 valeur2 valeur3

FAIRE

------> effectuer\_une\_action

VALEUR\_SUIVANTE

La variable va prendre successivement les valeursvaleur1,valeur2,valeur3. La boucle va donc être exécutée trois fois et la variable vaudra à chaque fois une nouvelle valeur de la liste.

En bash, la bouclefors'écrit comme ceci :

#!/bin/bash

for variable in 'valeur1' 'valeur2' 'valeur3'

do

        echo "La variable vaut $variable"

done

Ce qui donne, si on l'exécute :

La variable vaut valeur1

La variable vaut valeur2

La variable vaut valeur3

Vous pouvez donc vous servir duforpour faire une boucle sur une liste de valeurs que vous définissez :

#!/bin/bash

for animal in 'chien' 'souris' 'moineau'

do

        echo "Animal en cours d'analyse : $animal"

done

Animal en cours d'analyse : chien

Animal en cours d'analyse : souris

Animal en cours d'analyse : moineau

Toutefois, la liste de valeurs n'a pas besoin d'être définie directement dans le code. On peut utiliser une variable :

#!/bin/bash

liste\_fichiers=`ls`

for fichier in $liste\_fichiers

do

        echo "Fichier trouvé : $fichier"

done

Ce script liste tous les fichiers trouvés dans le répertoire actuel :

Fichier trouvé : boucles.sh

Fichier trouvé : conditions.sh

Fichier trouvé : variables.sh

On pourrait faire un code plus court sans passer par une variable$liste\_fichiersen écrivant :

#!/bin/bash

for fichier in `ls`

do

        echo "Fichier trouvé : $fichier"

done

Bien entendu, ici, on ne fait qu'afficher le nom du fichier, ce qui n'est ni très amusant ni très utile. On pourrait se servir de notre script pour renommer chacun des fichiers du répertoire actuel en leur ajoutant un suffixe-oldpar exemple :

#!/bin/bash

for fichier in `ls`

do

        mv $fichier $fichier-old

done

Essayons de voir si l'exécution du script renomme bien tous les fichiers :

$ ls

boucles.sh  conditions.sh  variables.sh

$ ./boucles.sh

$ ls

boucles.sh-old  conditions.sh-old  variables.sh-old

**À vous de jouer !** Essayez de créer un scriptmultirenommage.sh, reposant sur ce principe, qui va rajouter le suffixe-old… uniquement aux fichiers qui correspondent au paramètre envoyé par l'utilisateur !

./multirenommage.sh \*.txt

Si aucun paramètre n'est envoyé, vous demanderez à l'utilisateur de saisir le nom des fichiers à renommer avecread.

**Unforplus classique**

Pour les habitués d'autres langages de programmation, leforest une boucle qui permet de faire prendre à une variable une suite de nombres.

En bash, comme on l'a vu, leforpermet de parcourir une liste de valeurs. Toutefois, en trichant un peu à l'aide de la commandeseq, il est possible de simuler unforclassique :

#!/bin/bash

for i in `seq 1 10`;

do

        echo $i

done

Explication :seqgénère tous les nombres allant du premier paramètre au dernier paramètre, donc 1 2 3 4 5 6 7 8 9 10.

1

2

3

4

5

6

7

8

9

10

Si vous le voulez, vous pouvez changer le pas et avancer de deux en deux par exemple. Dans ce cas, il faut écrireseq 1 2 10pour aller de 1 à 10 en avançant de deux en deux ; cela va donc générer les nombres 1 3 5 7 9.

**En résumé**

* Pour exécuter une série de commandes plusieurs fois, on utilise des boucles.
* whilepermet de boucler tant qu'une condition est remplie. Le fonctionnement des conditions dans les boucles est le même que celui des blocsifdécouverts dans le chapitre précédent.
* forpermet de boucler sur une série de valeurs définies. À l'intérieur de la boucle, une variable prend successivement les valeurs indiquées.