

Accueil > Cours > Reprenez le contrôle à l'aide de Linux ! > Les boucles

# Reprenez le contrôle à l'aide de Linux !

🕒 30 heures 📶 Facile

Mis à jour le 29/06/2021



## Le contenu de ce cours n'est plus à jour

Nous avons archivé ce cours et n'actualiserons plus son contenu.

Accédez au contenu le plus récent en découvrant ce cours :



### SYSTÈMES & RÉSEAUX

#### Initiez-vous à Linux

📶 Easy 🕒 8 heures

Dans ce cours débutant, découvrez Linux : un système d'exploitation gratuit et fascinant qui vous donnera un contrôle sans précédent sur votre ordinateur ! Créé par des passionnés d'informatique, Linux est un vecteur important de la philosophie du libre et l'alternative parfaite à Windows ou macOS.

VOIR LE NOUVEAU COURS

# 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 est `while` .

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 le `if` , 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 script `boucles.sh` pour l'occasion :

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

```
while [ -z $reponse ] || [ $reponse != 'oui' ]  
do  
    read -p 'Dites oui : ' reponse  
done
```

On fait deux tests.

1. Est-ce que `$reponse` est vide ?
2. Est-ce que `$reponse` est différent de `oui` ?

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 de `oui` ».

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 tapé `oui` !



Il existe aussi le mot clé `until`, qui est l'exact inverse de `while`. Il signifie « Jusqu'à ce que ».

Remplacez juste `while` par `until` dans le code précédent pour l'essayer.

## for : boucler sur une liste de valeurs



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

### Parcourir une liste de valeurs

La boucle `for` permet de parcourir une liste de valeurs et de boucler autant de fois qu'il y a de

valeurs.

Concrètement, la forme d'un `for` est la suivante :

```
POUR variable PRENANT valeur1 valeur2 valeur3  
FAIRE  
-----> effectuer_une_action  
VALEUR_SUIVANTE
```

La variable va prendre successivement les valeurs `valeur1` , `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 boucle `for` s'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 du `for` pour 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_fichiers` en 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 `-old` par 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 script `multirenommage.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 avec `read`.

## Un `for` plus classique

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

En bash, comme on l'a vu, le `for` permet de parcourir une liste de valeurs. Toutefois, en trichant un peu à l'aide de la commande `seq`, il est possible de simuler un `for` classique :

```
#!/bin/bash
for i in `seq 1 10`;
do
    echo $i
done
```

Explication : `seq` génère 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 écrire `seq 1 2 10` pour 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.
- `while` permet de boucler tant qu'une condition est remplie. Le fonctionnement des conditions dans les boucles est le même que celui des blocs `if` découverts dans le chapitre précédent.
- `for` permet de boucler sur une série de valeurs définies. À l'intérieur de la boucle, une variable prend successivement les valeurs indiquées.

J'AI TERMINÉ CE CHAPITRE ET JE PASSE AU SUIVANT



LES CONDITIONS

LES FONCTIONS



## Le professeur



**Mathieu Nebra**

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

## Découvrez aussi ce cours en...



Livre



PDF

## OPENCLASSROOMS

Qui sommes-nous ?

Financements

Expérience de formation

Forum

Blog

Presse [🔗](#)

---

## OPPORTUNITÉS

Nous rejoindre [🔗](#)

Devenir mentor [🔗](#)

Devenir coach carrière [🔗](#)

---

## AIDE



FAQ

---

## POUR LES ENTREPRISES

Former et recruter

---

## EN PLUS

Boutique [🔗](#)

Mentions légales

Conditions générales d'utilisation

Politique de protection des données personnelles

Cookies

Accessibilité

---

 Français [▼](#)





