



PowerShell

---

NOTES DE COURS

---

Contrôle de flux

---

Automne 2019

---

## Table des matières

1	Contrôle de flux .....	3
1.1	Alternative (if).....	3
1.2	Répétition (do/while) .....	5
1.3	Répétition avec do/until.....	6
1.4	Répétition avec while/until .....	7
1.5	Répétition avec for .....	8
1.6	Répétition avec foreach .....	9

# 1 Contrôle de flux

## 1.1 Alternative (if)

La structure if existe en PowerShell, sa syntaxe est calquée sur celle de C#. Les différences sont mineures :

```
if (<condition 1>)
{
    <liste d'instructions 1>
}
[elseif (<condition 2>)]
{
    <liste d'instructions 2>
}
[else]
{
    <liste d'instructions 3>
}
```

- Les conditions sont entre parenthèses
- Les blocs d'instructions sont entre accolades
- Le elseif est facultatif (et il peut y en avoir plusieurs)
- Le else est facultatif (mais il ne peut pas y en avoir plus d'un)
- La structure a un seul point de sortie - au plus un bloc d'instructions sera exécuté

Les opérateurs de comparaison commencent par un tiret suivi d'un mot (ou d'une abréviation) :

Opérateur	Description
-eq	égal ( <i>equal</i> )
-ne	non-égal ( <i>not equal</i> )
-lt	plus petit que ( <i>less than</i> )
-le	plus petit ou égal ( <i>less or equal</i> )
-gt	plus grand que ( <i>greater than</i> )
-ge	plus grand ou égal ( <i>greater or equal</i> )
-like	comme (permet de comparer avec le caractère générique "*")
-notlike	pas comme (l'inverse)
-contains	contient (permet de vérifier si une liste contient une valeur)
-notcontains	ne contient pas (l'inverse)

Les opérateurs logiques permettent d'unir plusieurs conditions pour obtenir un seul résultat global. Ils fonctionnent de façon très similaire à leurs équivalents C#:

Opérateur	Description
-and	et logique
-or	ou logique
-not	non (inverse le résultat du test)
!	non (une autre syntaxe)

### EXEMPLE

```
$a=8
$b=4
$c=4

if ($a -gt $b -and $a -gt $c)
{
    Write-Host "a est le plus grand"
}
elseif ($c -gt $a -and $c -gt $b)
{
    Write-Host "c est le plus grand"
}
else
{
    Write-Host "b est le plus grand"
}
```

## 1.2 Répétition (do/while)

Le forme do/while existe en PowerShell, sa syntaxe se rapproche de celle du C#:

```
do
{
    <liste d'instructions>
}
while (<condition>)
```

### EXEMPLE

```
$a=4

do
{
    Write-Host $a
    $a=$a-1
}
while ($a -gt 0)
```

### 1.3 Répétition avec do/until

Le forme do/until existe en PowerShell, sa syntaxe se rapproche de celle du C#:

```
do
{
    <liste d'instructions>
}
until (<condition>)
```

#### EXEMPLE

```
$a=4

do
{
    Write-Host $a
    $a=$a-1
}
until ($a -lt 0)
```

## 1.4 Répétition avec while/until

On peut commencer une boucle par while ou par do selon si on veut que la condition soit évaluée en entrant ou en sortant de la boucle.

Le until ne peut pas être placé au début - on devra se résoudre à inverser notre condition si c'est ce qu'on veut faire.

```
while (<condition>)  
{  
    <liste d'instructions>  
}
```

### EXEMPLE

```
$a=4  
  
while ($a -gt 0)  
{  
    Write-Host $a  
    $a=$a-1  
}
```

## 1.5 Répétition avec for

Les boucles for existent également :

```
for (<initialisation>; <condition>; <répétition>)  
{  
    <liste d'instructions>  
}
```

La syntaxe du for offre beaucoup de flexibilité.

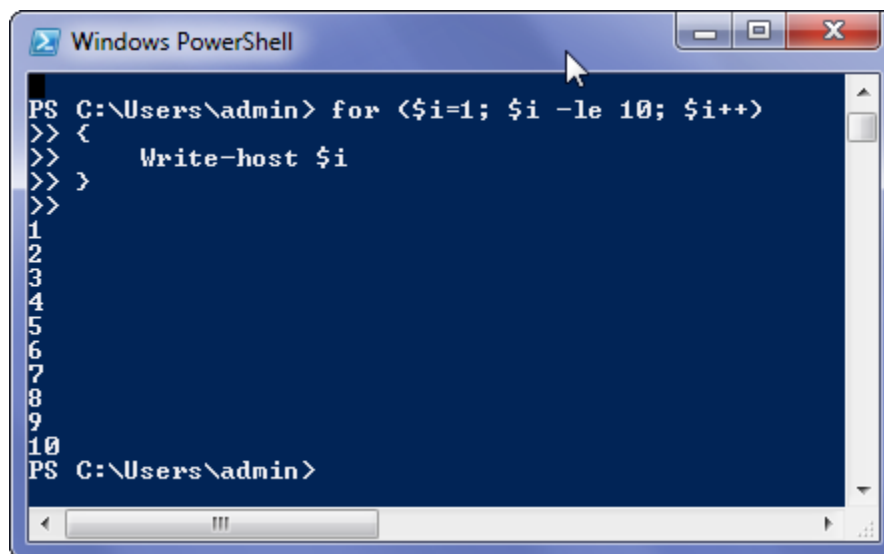
Un for est suivi de trois sections séparées par des points-virgules, le tout entre parenthèses.

La liste d'instructions à exécuter est entre accolades.

La partie initialisation contient les instructions à exécuter lorsqu'on entre dans la boucle la première fois.

La condition représente la condition pour rester dans la boucle. Elle est évaluée à chaque itération.

### EXEMPLE



```
Windows PowerShell  
PS C:\Users\admin> for <$i=1; $i -le 10; $i++>  
>> {  
>>     Write-host $i  
>> }  
>>  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
PS C:\Users\admin>
```

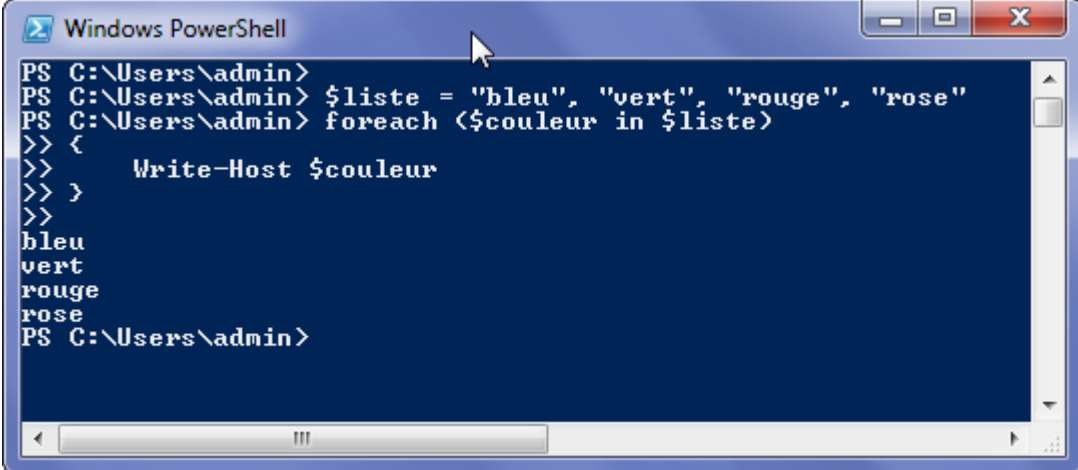


## 1.6 Répétition avec foreach

L'instruction foreach permet de parcourir une collection.

```
foreach ($<élément> in <collection>)  
{  
    <liste d'instructions>  
}
```

La première collection qui vient à l'esprit est une liste dans une variable :

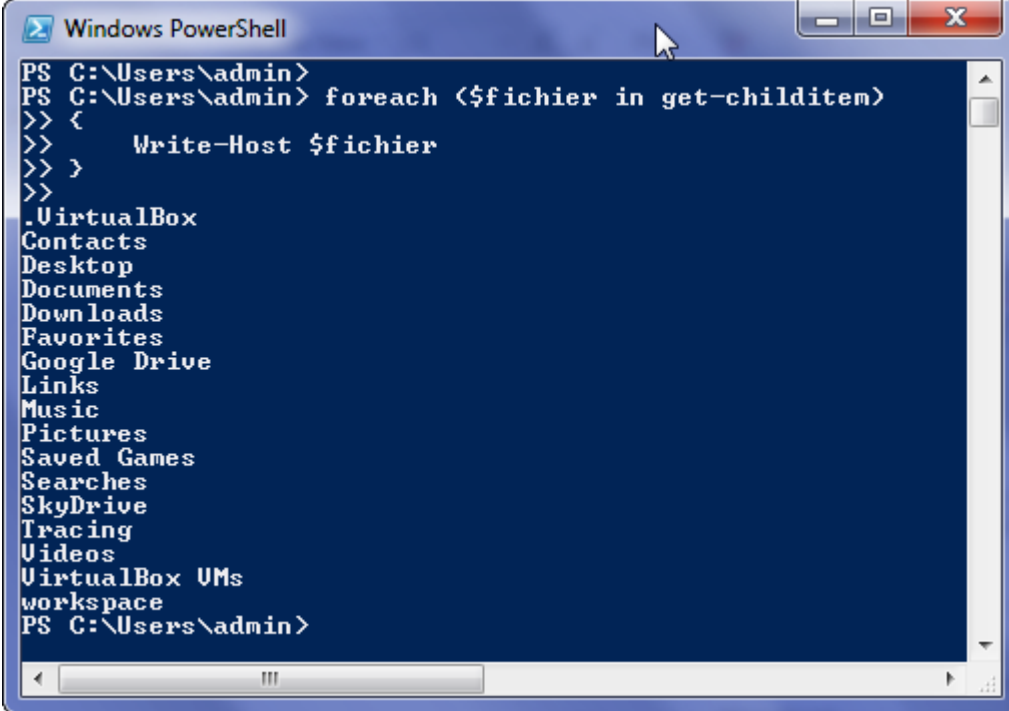


```
Windows PowerShell  
PS C:\Users\admin>  
PS C:\Users\admin> $liste = "bleu", "vert", "rouge", "rose"  
PS C:\Users\admin> foreach ($couleur in $liste)  
>> {  
>>     Write-Host $couleur  
>> }  
>>  
bleu  
vert  
rouge  
rose  
PS C:\Users\admin>
```

L'instruction foreach est pratique en PowerShell parce que beaucoup de fonctions retournent une collection d'objets, qu'on peut ensuite parcourir un à un.

## EXEMPLE

Get-ChildItem retourne une collection d'objets.



```
PS C:\Users\admin>
PS C:\Users\admin> foreach ($fichier in get-childitem)
>> {
>>     Write-Host $fichier
>> }
>>
.VirtualBox
Contacts
Desktop
Documents
Downloads
Favorites
Google Drive
Links
Music
Pictures
Saved Games
Searches
SkyDrive
Tracing
Videos
VirtualBox VMs
workspace
PS C:\Users\admin>
```