**Loops in Power Shell**

**Loop and it’s type**

For

Run a command block based on a conditional test.

Syntax

for (*init*; *condition*; *repeat*)

{*command\_block*}

Key

init Commands, separated by commas, to run before the loop begins.

Typically used to initialize a variable with a starting value.

condition If this evaluates to TRUE the for loop {*command\_block*} runs

when the loop has run once the condition is evaluated again

repeat Commands, separated by commas, to run each time the loop repeats.

A typical use of the for loop is to operate on a subset of the values in an array.  
In most cases, if you want to iterate **all** values in an array, consider using a foreach statement.

**Examples**

Count to 10:

PS> for($i=1; $i -le 10; $i++){Write-Host $i}

You can use carriage returns instead of semi-colons:

PS> for($i=1

$i -le 10

$i++){

Write-Host $i

}

# **ForEach (loop statement)**

Loop through a set of input objects and perform an operation (execute a block of statements) against each.

Syntax

ForEach (*item* **In** *collection*) {*ScriptBlock*}

key

*item* A variable to hold the current item

*collection* A collection of objects e.g. filenames, registry keys, servernames

*ScriptBlock* A block of script to run against each object.

The *collection* will be evaluated and stored as an array in memory before the *scriptblock* is executed.

The foreach statement does not use pipelining (unlike ForEach-Object ) If you use foreach in a command pipeline PowerShell will actually run the foreach alias that calls ForEach-Object.

Use the ForEach statement when the *collection* of objects is small enough that it can be loaded into memory.  
Use the ForEach-Object cmdlet when you want to pass only one object at a time through the pipeline, minimising memory usage. In most cases ForEach will run faster than ForEach-Object, there are exceptions, such as starting multiple background jobs. If in doubt test both options with Measure-Command.

In PowerShell 4.0 and later, the ForEach method provides even faster performance.

**Examples**

Loop through an array of strings:

 $trees = @("Alder","Ash","Birch","Cedar","Chestnut","Elm")  
  
 foreach ($tree in $trees) {  
   "$tree = " + $tree.length  
 }

Loop through a collection of the numbers, echo each number unless the number is 2:

 foreach ($num in 1,2,3,4,5) {  
  if ($num -eq 2) { continue } ; $num  
 }  
  
Loop through a collection of .txt files:

  foreach ($file in get-ChildItem \*.txt) {  
    Echo $file.name  
  }

# **While**

Run a command block based on the results of a conditional test.

Syntax

while (*condition*) {*command\_block*}

Key

condition If this evaluates to TRUE the loop {*command\_block*} runs.

when the loop has run once the condition is evaluated again

*command\_block* Commands, separated by commas, to run each time the loop repeats.

As long as the condition remains true, PowerShell reruns the {*command\_block*} section.

**Examples**

Count to 10:

PS> while($val -ne 10) { $val++ ; Write-Host $val }

You can use carriage returns instead of semi-colons:

PS> while($val -ne 10)

{

$val++

Write-Host $val

}

# **Do**

Run a command block based on the results of a conditional test.

Syntax

[:*Loop\_label*] **Do**

{

*command\_block*

} **while** (*condition*)

or

[:*Loop\_label*] **Do**

{

*command\_block*

} **until** (*condition*)

Key

condition If this evaluates to TRUE the loop {*command\_block*} runs.

when the loop has run once the condition is evaluated again

*command\_block* Commands, separated by commas, to run each time the loop repeats.

:*Loop\_label* An optional label than can be used to break or continue.

As long as the condition is met, PowerShell will continue to rerun the {*command\_block*} section unless interrupted by a break or continue statement.

**Examples**

Count to 10:

PS> Do { $val++ ; Write-Host $val } while($val -ne 10)

You can use carriage returns instead of semi-colons:

PS> Do { $val++

Write-Host $val

} while($val -ne 10)

# **Break**

Exit a program loop immediately.

Break can also be used to stop script execution when it is placed outside a loop or switch statement.

In a For, ForEach, While, Do loop or in a Switch statement you can add a break statement to exit each code block.

Choose the conditions under which you call the continue statement carefully, as it is easy to create an infinite loop. Most commonly an if statement can be used to define the break conditions.

**Examples**

Count to 10 but stop as soon as you reach number 5:

PS> $i = 0

PS> while ($i -lt 10) **{**

$i +=1

if ($i -eq 5) {break}

Write-Host $i

**}**

In a switch statement, break tells switch to stop looking further, in other words "*you have a match, so no need to perform the other tests.*"

$department = "sales"

switch ($department) **{**

"HR" {echo "HR found"; **break**}

"sales" {echo "found sales"; **break**}

"engineering" {echo "found engineering"; **break**}

**}**

# **Do...Until**

Run a command block based on the results of a conditional test.

Syntax

[:*Loop\_label*] **Do**

{

*command\_block*

} **while** (*condition*)

or

[:*Loop\_label*] **Do**

{

*command\_block*

} **until** (*condition*)

Key

condition If this evaluates to TRUE the loop {*command\_block*} runs.

when the loop has run once the condition is evaluated again

*command\_block* Commands, separated by commas, to run each time the loop repeats.

:*Loop\_label* An optional label than can be used to break or continue.

A Do..While loop continues as long as the condition is TRUE.  
A Do..Until loop continues as long as the condition is FALSE. Until is an antonym of While.

As long as the condition is met, PowerShell will continue to rerun the {*command\_block*} section unless interrupted by a break or continue statement.

**Examples**

Count to 10:

PS> Do { $val++ ; Write-Host $val } while($val -ne 10)

You can use carriage returns instead of semi-colons:

PS> Do { $val++

Write-Host $val

} while($val -ne 10)

# **Continue**

Return to top of a program loop, skip just this iteration of the loop.

In a For, ForEach or While loop you can add a continue statement to jump to the top of the innermost loop.

Choose the conditions under which you call the continue statement carefully, as it is easy to create an infinite loop.

**Example**

Count to 10 but miss out the number 5:

PS> $i =0

while ($i -lt 10)

{

$i +=1

if ($i -eq 5) {continue}

Write-Host $i

}

# **Return**

Exit the current scope, which can be a function, script, or script block.

Syntax

return

return *expression*

In PowerShell, the result of each statement is returned as output, even without an explicit Return keyword.

So   
 $price  
 return  
has exactly the same effect as:  
 return $price

**Example**

Add the number 5 and return the value:

function demoadd {

param ($value)

"Adding five"

$value += 5

return $value

}

C:\PS> $result = demoadd 2  
C:\PS>

C:\PS> $result

Adding five

7

The key point to note in the above example is that the "Adding 5" string is not displayed, but is instead assigned to the $result variable and returned along with the number.