PowerShell Operators [Complete Guide]

lazyadmin.nl/powershell/powershell-operators

August 21, 2024

When using PowerShell you can use a wide variety of operators in your script. They can be used in commands or expressions and are used to perform comparisons, define conditions, or assign and manipulate values.

If you are using PowerShell 7, then you can also use the new Ternary operators. These allow you to create online if-else statements, making your code more readable.

In this article, we are going to take a look at the different operators. I have divided them into groups, so you can easily reference them.

Assignment Operators

The most commonly used operators are the assignment operators. These operators are used to assign, manipulate, or append values to variables.

Operator	Example	Description
=	$a = 5	Assigns the value on the right to the left.
+=	$a += 3	Adds the right operand to the left and assigns it.
-=	$a -= 2	Subtracts the right operand from the left and assigns it.
*=	$a *= 4	Multiplies the left operand by the right and assigns it.
/=	$a /= 2	Divides the left operand by the right and assigns it.

PowerShell Assignment Operators

For example, we can assign the value 10 to the variable `$count` and add 5 to it, making the total 15:

```
$count = 10
$count += 5
Write-Output $count # Outputs 15
```

Comparison Operators

Comparison operators in PowerShell can be divided into 4 groups. We have operators to compare values, for example, to test if $a is greater than **$b** with `($a -gt $b)`, operators to find and even replace patterns, to test is a value appears in a set and to check if an object is a specific type.

The basic comparison operators in PowerShell are:

Operator	Example	Description
-eq	5 -eq $a	Returns `True` if both operands are equal.
-ne	5 -ne $a	Returns `True` if operands are not equal.
-gt	$a -gt 5	Returns `True` if the left operand is greater.
-lt	$a -lt 5	Returns `True` if the left operand is less.
-ge	5 -ge $a	Returns `True` if the left operand is greater or equal.
-le	5 -le $a	Returns `True` if the left operand is less or equal.

basic comparison operators

As you can see in the table above, the variable is on the right side when using the equality operators. The reason for this is when you use the `-eq` operator between an array and a scalar value (like a number), then PowerShell doesn't return **True** or **False**.

Instead, it will compare the scalar value with each element in the array. If the scalar value exists in the array, then it will return the array element instead of True.

```
$a = 0,1,2,3
$a -eq 0 # Returns 0
0 -eq $a # Returns False
```

Where can I send the Free PowerShell cheat sheet to?

We also have comparison operators that we can use to find or even replace patterns in strings. The `-match` and `-replace` operators use regex for the comparison whereas the `-like` operator uses a wildcard.

Operator	Example	Description
-match	$string -match “foo”	Returns `True` if the string matches the regex pattern.
-notmatch	$string -notmatch “foo”	Returns `True` if the string does not match the regex pattern.
-replace	$string -replace “foo”, “bar”	Replaces text that matches a regex pattern.
-like	$string -like “foo*”	Returns `True` if the string matches the wildcard pattern.
-notlike	$string -notlike “foo*”	Returns `True` if the string does not match the wildcard pattern.

String comparison operators

The containment comparison operators are used to check if a value is present or not in a reference set, like an array for example. Both operators, `-contains` and `-in`, can be used to check if an item is present in a collection.

The only difference between the two is the place of the value and collection. Which one you should use really depends on the context.

Operator	Example	Description
-contains	$array -contains 5	Returns `True` if the array contains the specified value.
-notcontains	$array -notcontains 5	Returns `True` if the array does not contain the value.
-in	5 -in $array	Returns `True` if the left operand is in the array.
-notin	5 -notin $array	Returns `True` if the left operand is not in the array.

containment comparison operators

```
$numbers = 2, 4, 6, 8, 10
if ($numbers -contains 4) {
Write-Output "The array contains 4."
}
```

The last comparison operators that we can use in PowerShell are the -is operators. This one isn't used a lot but allows you to check if an object is of the specified type or not.

Operator	Example	Description
-is	$object -is [Type]	Returns `True` if the object is of the specified type.
-isnot	$object -isnot [Type]	Returns `True` if the object is not of the specified type.

Logical Operators

Logical Operators are used to connect conditional statements in PowerShell to a single statement.

Operator	Example	Description
-and	(5 -gt 3) -and (6 -gt 2)	Returns `True` if both conditions are true.
-or	(5 -gt 7) -or (8 -gt 2)	Returns `True` if any of the conditions are true.
-not	-not (5 -eq 5)	Reverses the logic of its operand.
!	!(3 -eq 4)	Short form of `-not`.

Logical Operators

For example, we can use the -and operator to check if a value is less than 10 and greater than 2 with the command below:

```
$val = 5
if ($val -lt 10 -and $val -gt 2) {
Write-Output "Both conditions are met"
}
```

Redirection Operators

The redirection operators in PowerShell share some similarities with the ones from Command Prompt. But PowerShell also has some of its own.

You can use the redirection operators to send the output of a command to a text file. The different options not only allow you to send or append the normal (standard) output to a file. It's also possible to redirect only the error stream or warning stream. Which is great when you want to create an error log for your script.

Operator	Example	Description
>	Get-Process > processes.txt	Redirects standard output to a file.
>>	Get-Process >> log.txt	Appends standard output to a file.
2>	Get-Command xyz 2> errors.txt	Redirects error output to a file.
2>>	Get-Command xyz 2>> errors.log	Appends error output to a file.
2>&1	Get-Process 2>&1 alloutput.txt	Redirects error stream to the succes stream
*>	Get-Process *> alloutput.txt	Redirects all streams to a file

Redirection Operators

For example, to write only the errors of a command to a file we can use the redirect stream operator:

```
# Write only the error to the file:
Get-ProcessCpu -name "explorer2" 2> C:\temp\process-error-log.txt
```

Split and Join Operators

The split and join operators are used to split or join strings.

Operator	Example	Description
-split	‘one,two,three’ -split ‘,’	Splits a string into an array based on a delimiter.
-join	(‘one’,’two’) -join ‘,’	Joins array elements into a single string.

Split and Join Operators

Type Operators

We have seen the type operator `-is` also in the comparison operator section. The operator is used to check if a variable is of a specific type. But besides the -is operator, we can also use the -as operator to convert a variable into a specific type.

Operator	Example	Description
-is	$var -is [int]	Checks if a variable is of a specified type.
-as	$var -as [string]	Tries to convert a variable to a specified type.
[Type]	[int]$var	Casts a variable to a specified type.

Type Operators

The -as operator is particularly handy when you need to convert a string into an integer for example

```
$number = "123"
if ($number -is [string]) {
$converted = $number -as [int]
Write-Output "Converted: $converted"
}
```

Unary Operators

Unary operators are used quite a lot, especially in loops, to either keep track of the index or for counting. Fun fact, the ++ operator is probably the most used, but most don't know that it's called a Unary operator.

Operator	Example	Description
–	-$var	Negates the value of the operand.
++	++$var	Increments the operand's value by 1.
—	–$var	Decrements the operand's value by 1.

Unary Operators

Special Operators

These operators don't really fit into the other groups, because each has there own special use case

Operator	Example	Description
&	& script.ps1	Executes a script or command.
.	. script.ps1	Executes a script in the current scope.
()	($a + $b) * $c	Ensure that expressions are evaluated in a specific order
$( )	$(Get-Date)	Subexpression operator; runs the command in a subexpression.
`	“This is a backtick `n”	Escape character; used to include special characters in strings.
..	$a = 1..10	Create an array with a range of integer from 1 to 10.

Special Operators

The grouping operators, better known as parentheses, are used to ensure that expressions or cmdlets are executed in a specific order. In the example below, we want the get the item path first, and then check if the length is greater than 100.

```
$path = "C:\MyScripts\script.ps1"
if ((Get-Item $path).Length -gt 100) {
Write-Host "Long path"
}
```

Ternary Operator Table

The ternary operator was released in PowerShell 7. It simplifies the if-else statements, making them easier to read. I do recommend only using the ternary operator for short if-else statements that fit on a single line.

Operator	Example	Description
? :	$result = ($x -gt 10) ? ‘High’ : ‘Low’	Returns High if $x is greater than 10, otherwise returns Low.

Ternary Operator

In the example below we check if the variable $x is greater than 10, if that is the case, we set the status to high, otherwise to low:

```
# Define a variable
$x = 15
# Use the ternary operator to assign a value based on a condition
$status = ($x -gt 10) ? 'High' : 'Low'
```

```
# Output the result
Write-Output "The status is $status"
```

Null-coalescing Operators

The null-coalescing operator `??` in PowerShell 7 allows you to quickly check if the given variable is null and set a default value instead.

If the value on the left side of the operator equals null, then the value on the right side (which can also be variable) is returned. You can also combine multiple null-coalescing operators to find the first non-null value.

Operator	Example	Description
??	$result = $value ?? DefaultValue	Returns `$value` if it is not null; otherwise, returns `DefaultValue`.
??=	$value ??= DefaultValue	Assigns `DefaultValue` to `$value` if `$value` is null.
?.	$result = $object?.Property	Accesses `Property` of `$object` only if `$object` is not null.
?[]	$element = $array?[index]	Accesses the element at `index` in `$array` only if `$array` is not null.

Null-coalescing Operators

One of the nice things about the Null-coalescing Operators is that it allows you to, for example, only access a property of an object if that object is not null:

```
# Define an object with a property
$person = [PSCustomObject]@{ Name = "Alice" }
# Access the 'Name' property only if $person is not null
$name = $person?.Name
```

Wrapping Up

There are a lot of operators in PowerShell that we can use. Most are quite common and are also used in other programming languages. But if you are relatively new to scripting or programming, then make sure that you pay extra attention to the null-coalescing and ternary operators for PowerShell 7.

Hope you found this list useful, make sure that you subscribe to the newsletter or follow me on Facebook so you don’t miss the latest articles.

I hate spam to, so you can unsubscribe at any time.