

PowerShell – Check if variable is Null or Empty

 lazyadmin.nl/powershell/is-null-or-empty

April 9, 2024

One of the most common tasks when writing a script is to check if a variable is NULL or Empty in PowerShell. Null or empty values are always a bit challenging, because what is exactly null?

Checking if a value is null or not has become a lot easier in PowerShell 7 with the Null-coalescing operators. This operator eliminates the need for if-else statements to check if a value is Null or not.

In this article, I will explain how to check if a value is null, not null, or just empty. We look at the new Null-coalescing operator and why it's important to start with the `$null` variable in the comparison.

What is Null

When we are talking about NULL in a programming or scripting language, then we can assume it's a variable that doesn't have any value. In PowerShell however, Null can either be an empty value or an unknown value.

The NULL value is written as `$null` in PowerShell, and it's one of the automatic variables. It's an object with the value NULL. This allows you to use `$null` as a placeholder in collections or assign it to a variable.

A good way to demonstrate this is to create a collection and count the length of it. As you can see in the example below, the `$null` object is counted as one of the objects in the collection:

```
$fruits = "Apple", $null, "Pear"
$fruits.count
3
```

NULL vs Empty

Besides `$null` values, we can also have empty values in PowerShell. The difference between the two is that an empty value is a variable that is declared and assigned an actual empty value.

If we look at the example below, then the first variable is `$null`, even though we haven't assigned the NULL variable to it. The other variables are not null but are empty.

```
# Null variable
$variableIsNull
# Empty string variable
```

```
$emptyString = ""  
# Empty array variable  
$emptyArray = @()
```

Check if Null in PowerShell

There are a couple of ways to check if a variable or result is Null in PowerShell. The recommended way to check if a value is NULL is by comparing `$null` with the value:

```
if ($null -eq $value) {  
    Write-Host 'Variable is null';  
}
```

Important to note that the method above, returns to true on two occasions. When the value is NULL, as you would expect, but also when the variable is not defined.

When you want to check if a value is Not Null, then you can use the comparison operator `-ne` instead:

```
if ($null -ne $value) {  
    Write-Host 'Variable is not null';  
}
```

Another common way to check if a value is Null is to simply use an if statement without any comparisons. This also works, but it does more than checking if a value is null. The comparison below will actually do the following:

- not Null
- not empty
- not 0
- not an array,
- Length is not equal to 0
- not \$false

```
if ($value) {  
    Write-Host 'Variable is not null or empty';  
}
```

Null on the left-hand side

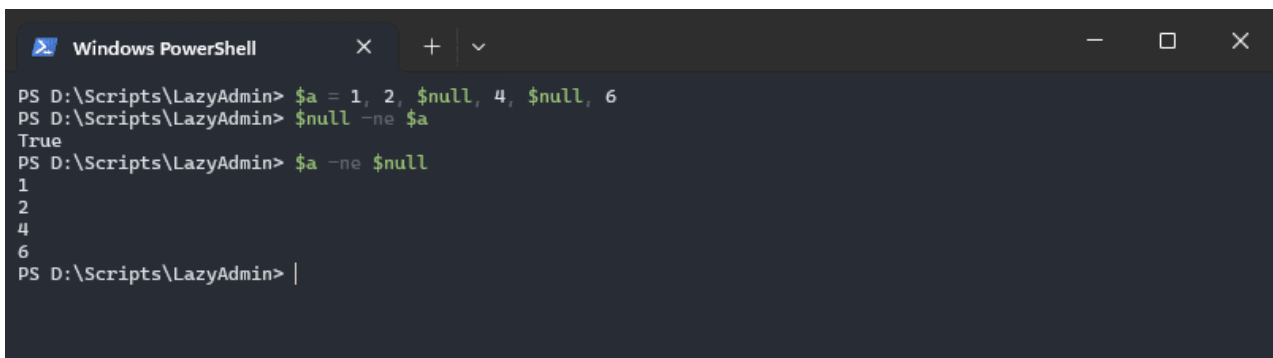
When you are using Visual Studio code or another IDE to write your PowerShell scripts, you will get a warning when you place the `$null` variable on the right-hand side in a comparison. This is for a good reason.

In some cases, placing them `$null` on the right-hand side can result in unexpected results. Take for example the array below. If we place `$null` on the left side, it will check if `$null` is not equal to our `$array`, which it isn't, so it returns true.

```
# Array
```

```
$a = 1, 2, $null, 4, $null, 6
# Check if the Array is not empty (null)
$null -ne $a # returns true
# Return everything from the array that is not Null
$a -ne $null
# Returns
1
2
4
6
```

But when you place `$null` it on the right-hand side, then it will return everything from the array, that does not equal `$null`.



```
Windows PowerShell
PS D:\Scripts\LazyAdmin> $a = 1, 2, $null, 4, $null, 6
PS D:\Scripts\LazyAdmin> $null -ne $a
True
PS D:\Scripts\LazyAdmin> $a -ne $null
1
2
4
6
PS D:\Scripts\LazyAdmin> |
```

FREE EMAIL SERIES!

Level Up with PowerShell

5 Emails, Endless Skills

Null-coalescing Operator

If you are using PowerShell 7, and you actually should, then you can use one of the new null conditional operators to quickly check if a variable or property is null or not. This way you don't need to write If statements every time you only want to check if a variable is null or not.

The Null coalescing operator `??` returns the value on the left side if it's not null. If the value on the left is null, then it returns the results on the right side.

```
$nullableVariable = $null
# If $nullableVariable equal NULL, return "Default Value"
```

```
$defaultValue = $nullableVariable ?? "Default Value"
```

```
# Result of $defaultValue
```

```
The value is: Default Value
```

We can also use this method to check the result of a function and call another function or cmdlet if the results are null:

```
function Get-LastUsedDate {
```

```
    return $null
```

```
}
```

```
# Use null-coalescing operator to handle null return value
```

```
$result = (Get-LastUsedDate) ?? (Get-Date).ToShortDateString()
```

It's also possible to chain multiple Null Coalescing operators together to find or return the first variable that is not null.

```
# Define a chain of variables where one may be null
```

```
$firstVariable = $null
```

```
$secondVariable = "Value 2"
```

```
# Use null-coalescing operator to choose the first non-null value
```

```
$chosenValue = $firstVariable ?? $secondVariable ?? "No value found"
```

```
# Output the chosen value
```

```
Write-Host "Chosen value: $chosenValue"
```

Null-coalescing Assignment Operator

The Null-coalescing assignment operator `??=` makes it easier to check if a variable on the left-hand side of the operator is null or not, and assign the value of the right-hand side to it if it's null.

In the example below, we check if the variable `$existingValue` is null or not. If it's null, then we set the variable to the default value:

```
# Variable that may or may not have a value
```

```
$existingValue = "Existing Value"
```

```
# Assign a new value using the Null-coalescing assignment operator
```

```
$existingValue ??= "Default Value"
```

```
# Result of $existingValue
```

```
Existing Value
```

Null-conditional operators `?.` and `?[]`

New in PowerShell 7.1 are the Null-conditional operators `?.` and `?[]`. These operators allow you to safely access properties and elements of an object that might be null.

If you access a property or element without the null-conditional operator, you normally won't get an error. Only if you are using Strict mode you do get an error. So in these cases, it's important to check if you are not trying to access a member of a null-valued object.

```
# Empty object
$cities = $null
# Set strictmode for test
Set-StrictMode -Version 2
# Will throw an error
$cities.language
# No error with null-conditional operator
${cities}?.language
```

Check if Empty in PowerShell

Besides checking if a value is Null another common practice is to check if a value is empty. You can only check if a string or array is empty in PowerShell.

To check if a string is empty, simply compare it with ""

```
$string = ""
if ($string -eq "") {
write-host "Empty string found"
}
```

If you want to check if an array is empty, the best option is to check the length of the array:

```
$array = @()
if ($array.Count -gt 0) {
write-host "Array is not empty"
}
```

IsNullOrEmpty

When working with strings, you can also use the static string function to check if the value is either null or empty. If you want to know if the value is not null or empty, then you only need to place **-not** in front of it

```
$string = $null
if ([String]::IsNullOrEmpty($string)) {
write-host "Empty string"
}
# Check if not empty or null
if (-not [String]::IsNullOrEmpty($string)) {
write-host "String is Not empty"
}
```

Check for Null in a Function

When you are creating your own functions in PowerShell, you can ensure that a parameter is not null or empty by using the validator `ValidateNotNullOrEmpty`. This way you don't have to do additional checks in the function.

```
function Test-Input {  
    param(  
        [Parameter(Mandatory)]  
        [ValidateNotNullOrEmpty()]  
        [string]$input  
    )  
    Write-Host "Input received: $input"  
}  
# Test the function  
Test-Input -input "Hello"
```

Wrapping Up

Checking for Null values makes your script more robust and will result in fewer errors in your script. Make sure that you try out the new PowerShell null-coalescing operators because they are quite handy to work with.

Also, make sure that you always keep the `$null` values on the left-hand side of the comparison.

I hope you liked this article, if you have any questions, just drop a comment below!

Did you **Liked** this **Article**?

Get the latest articles like this **in your mailbox**
or share this article

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