

How to use PowerShell If Else Statements

 lazyadmin.nl/powershell/if-else-statements

March 17, 2023

If-Else statements are the basis of any programming language, also in PowerShell. They are used to test if a condition is met, allowing you to perform different actions based on the result.

Multiple conditions, if-not statements, and else-if statements give you a variety of options to test conditions in each situation. A good understanding of the if statements and the different methods can really help you write better PowerShell scripts.

In this article, we are going to take a look at the PowerShell If Else statements, using the different methods like if not, if and, and how to use multiple conditions.

PowerShell If Statement

Let's start with the basic If statement in PowerShell. The If statement without any operators will check if a condition is true or not. When the condition is true, the script block will be executed. The condition is placed between parenthesis () and the script block is placed between curly brackets { }.

```
if (condition) {  
# code to execute if condition is true  
}
```

In the example below, we have set the variable `$isAdmin` to `True`. Next, we are going to check with an if statement if the variable is set to true and write an output:

```
$isAdmin = $true  
if ($isAdmin) {  
Write-Host "You have administrator privileges."  
}
```

In the example above we used a simple variable that is set to `$true` for the condition statement. But we can also use a comparison operator in the condition. Comparison operators return true or false depending on the result. Try the following inside your PowerShell window:

```
$num = 10  
$num -gt 5  
# Result  
True
```

A screenshot of a PowerShell console window. The prompt is 'rmens@LT3452 C:'. The user enters '\$num = 10'. The prompt is 'rmens@LT3452 C:'. The user enters '\$num -gt 5'. The output is 'True'. The prompt is 'rmens@LT3452 C:'. The user enters '|'. The output is '|'. The console window has a dark background and a title bar with 'C:\', '+', and '-' buttons.

PowerShell Comparison Operator

As you can see, the comparison operator returned true. So if we add this example inside the condition part of the If statement, we can test if the number `$num` is greater than 5, and output a string to the console:

```
$num = 10
if ($num -gt 5) {
# $num is greater then 5
Write-Host "$num is greater then 5"
}
```

Other comparison operators that you use in PowerShell are:

Operator	Counter-Part operator	Description
-eq	-ne	Equal or not equal
-gt	-lt	Greater or less than
-ge		Great than or equal to
-le		Less than or equal to
-Like	-NotLike	Match a string using * wildcard or not
-Match	-NotMatch	Matches or not the specified regular expression
- Contains	-NotContains	Collection contains a specified value or not
-In	-NotIn	Specified value in collection or not
-Replace		Replace specified value

PowerShell Operators

Using Cmdlets in If Conditions

Besides variables and comparison operators, we can also use cmdlets inside the if statement condition. For example, we can create a PowerShell session to Server01 only if the ping to the server was successful, using the test-connection cmdlet.

```

if (Test-Connection -TargetName Server01 -Quiet) {
New-PSSession -ComputerName Server01
}else{
Write-Host "Unable to connect to server01"
}

```

Tip

Want to learn more about writing PowerShell scripts, then make sure that you read [this getting started with PowerShell scripting guide](#)

If-Else Statement

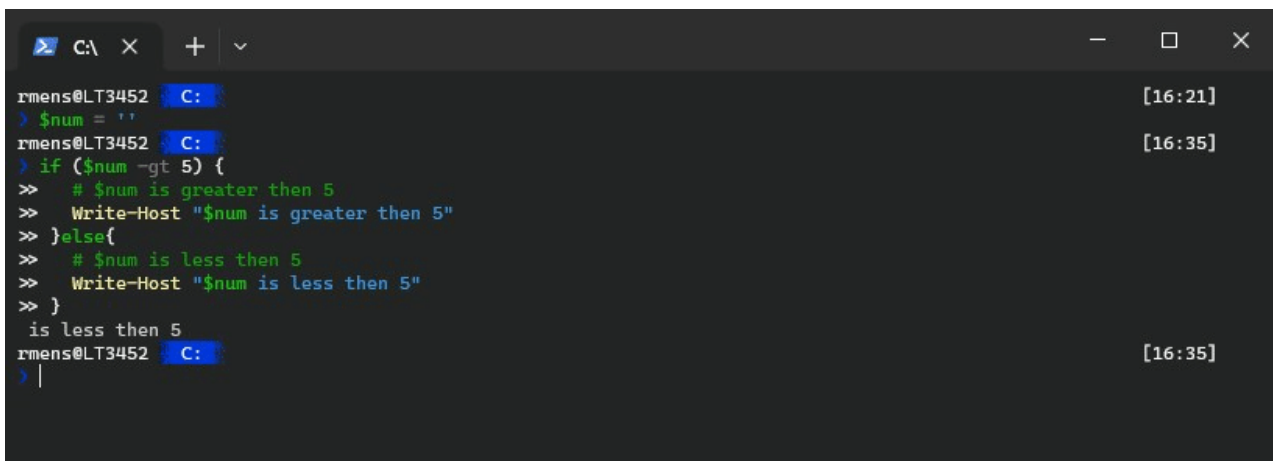
An if statement is often accompanied by an else statement in PowerShell. The else part of the statement is executed when the condition is false or null. Taking our last example, we can add an else statement that will tell us if the `$num` is less than 5:

```

$num = 10
if ($num -gt 5) {
# $num is greater then 5
Write-Host "$num is greater then 5"
}else{
# $num is less then 5
Write-Host "$num is less then 5"
}

```

The else part of the if-else statement will also trigger when the variable isn't set. For example, if we set the variable `$num` to an empty string, the else condition will trigger:



The screenshot shows a PowerShell terminal window with the following commands and output:

```

rmens@LT3452 C:
> $num = ''
rmens@LT3452 C:
> if ($num -gt 5) {
>> # $num is greater then 5
>> Write-Host "$num is greater then 5"
>> }else{
>> # $num is less then 5
>> Write-Host "$num is less then 5"
>> }
is less then 5
rmens@LT3452 C:
> |

```

The output shows that the 'else' branch was executed because the condition '\$num -gt 5' was false (since \$num is an empty string).

PowerShell if else

If Elseif Else Statement

We can expand our If Else statement even further with an `elseif` statement. Elseif allows you to test for an additional condition. Good to know is that you are not limited to one elseif condition, you can add as many as you want between your if and else statements.

But keep in mind that when the if statement or one of the elseif statements is met, then others won't be tested anymore.

```
$num = 10
if ($num -gt 10) {
Write-Host "$num is greater than 10"
}elseif ($num -gt 5) {
Write-Host "$num is greater than 5"
}else{
Write-Host "$num is less than 5"
}
# Result
10 is greater than 5
```

Another option for multiple `elseif` statements is to use a switch statement or use a collection and the `-in` comparison operator. The `-in` operator allows you to check if the collection contains the specified value or not. For example, we can check if the fruit is listed in the collection:

```
$collection = @(
'apple',
'banana',
'kiwi',
'raspberry'
)
$fruit = 'kiwi'
if ($fruit -in $collection) {
write-host "$fruit is a approved fruit"
}
# Result
kiwi is a approved fruit
```

PowerShell If And Statement

If And statements in PowerShell allow you to test for multiple conditions inside a single if. This sometimes eliminates the need for nested If statements or multiple if-else statements. If and statements check if two or more conditions are true. When all conditions are true, then the script block is executed:

```
# Using the -and operator
if ((condition1) -and (condition2)) {
# code to execute when condition 1 and condition 2
# are both true
}
```

I prefer to place each condition between parenthesis (), which makes the code more readable. But they are not always necessary, in the example above you could also leave them out, like this:

```
if ($num -ge 10 -and $num -lt 20) { #code }
```

So let's start with a simple example, we are going to check if the number is between 10 and 20. We first check if the variable `$num` is greater than 10 and in the second condition if it's less than 20.

```
$num = 12
if (($num -gt 10) -and ($num -lt 20)) {
Write-Host "$num is between 10 and 20"
}
```

Tip: you could also use the `-in` operator to check if the number is within a range:

```
if ($num -in 11..19) {
Write-Host "$num is between 10 and 20"
}
```

Note that we are testing if the number is between 10 and 20. If you want to include 10 and 20 then use the `-ge` (greater than or equal to) and `-le` (less than or equal to).

Using If Or statements

Similar to the `if` and to check if multiple conditions are met, can we also test if one of the conditions is met with the conditional operator `-or`. When using the `-or` operator only one of the conditions must be true for the script block to be executed.

```
if ((condition1) -or (condition2)) {
# code to execute when condition 1 or condition 2 is true
}
```

For example, we can check whether the user role is admin or developer before we continue:

```
$role = "developer"
if (($role -eq 'admin') -or ($role -eq 'developer')) {
# code to execute
write-host 'The user is allowed to continue'
}
```

Powershell If Not

The PowerShell `If` statement checks if a particular condition is true, but sometimes you only want to check if a condition is false. For this, we can use the `If Not` statement in PowerShell. This will check if a condition is not met. An `if not` statement is written by placing the condition between parenthesis () and place an `!` or `-not` in front of it:

```
if (!(condition1)) {
# code to execute when condition 1 is false
}
# or using -not
```

```
if (-not(condition1)) {  
# code to execute when condition 1 is false  
}
```

A good example for this is the [Test-Path cmdlet](#), which checks if a path or file exists. Before we copy files to a directory we need to check if the folder exists, if not, we need to create it.

```
If (!(Test-Path -Path $pathToFolder -PathType Container)) {  
New-Item -ItemType Directory -Path $pathToFolder | Out-Null  
}
```

PowerShell Ternary Operator

One line if statements are common practice in many programming languages these days. A one-line if statement is actually called a ternary operator, and we can use it in PowerShell 7. The ternary operator is great when you have simple short if statements that you can write on a single line. When an if statement triggers a multi-line script block, then the ternary operator isn't the best choice.

The ternary operator uses the following structure for the if-else statement:

```
<condition> ? <script-when-true> : <script-when-false>;
```

Let's take our example from the beginning of the article. We are going to check if the number is larger than 5 or not. Instead of writing the full if-else statement which is 5 lines long at least, we can write it on a single line using the ternary operator:

```
$value = 10  
$value -gt 5 ? (Write-Host 'Greater than 5') : (Write-Host 'Less than or equal to 5')
```

PowerShell If Null

Another common practice is to check if a variable or result from a cmdlet is null or not. A null variable is a variable that has not been assigned a value or has been explicitly assigned a null value. Simply set, a null variable doesn't have any data stored in it.

To check if a variable or result is null or empty we compare it with the \$null variable. The \$null variable must be on the left side, to make sure that PowerShell compares the value correctly.

Let's say we want to check if a variable is empty or not. If we would place the variable with the empty string on the left side, the result would be that the variable is not empty

```
$var = ""  
if ($var -eq $null) {  
Write-Host "The variable is null."  
}else{  
Write-Host "The variable is not empty."
```

```
}
```

```
# Result
```

The variable is not empty.

However, if we place the \$null variable first, then the result would be that the variable is empty:

```
$var = ""
```

```
if ($null -eq $var) {
```

```
Write-Host "The variable is null."
```

```
}else{
```

```
Write-Host "The variable is not empty."
```

```
}
```

```
# Result
```

The variable is null.

The different results happen because of two reasons in PowerShell:

- **\$null is a scalar value.** Comparison operators return a Boolean value when the value on the left is a scalar, or matching values/empty array if it's a collection.
- **PowerShell performs type casting left to right.** Which results in incorrect comparisons when \$null is cast to other scalar types.

So if you want to check if a value is null, then you must place the \$null on the left side.

Checking if a value is null in PowerShell isn't only used with variables. You can also use it to check if a cmdlet returns any data or not. For example, we can check if a PowerShell module is installed with the following code:

```
if ($null -eq (Get-Module -ListAvailable -Name Microsoft.Graph)) {
```

```
Write-host "Microsoft Graph module is not installed"
```

```
}
```

Wrapping Up

PowerShell If Else statements are the basis of writing scripts. They allow you to compare values and make decisions in your script based on the results. When using multiple elseif conditions you might want to consider using the [PowerShell Switch statements](#). This allows you to check multiple conditions in a cleaner and simpler way.

I hope you found this article useful, 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.