How to use the PowerShell Switch Statement

lazyadmin.nl/powershell/powershell-switch-statement

May 9, 2023

The PowerShell Switch statement allows you to test against multiple conditions. You can compare the switch statements with multiple if statements, only in a simpler and cleaner way.

What most people don't know is that a switch statement isn't limited to a single condition. You can actually execute multiple conditions and event test multiple values at the same time. Using the switch statement is preferred over using multiple `If` statements, because it makes your code easier to read and maintain.

In this article, we are going to look at how to use the Switch statement in PowerShell and how to get the most out of it.

Using the PowerShell Switch Statement

Let's first take a look at the basic usage of the PowerShell Switch Statement. The switch statements compared a value against multiple conditions. If a condition is met, then the code in the script block is executed.

In the example below, we have a variable `$car`. The switch statement will check if the variable car matches one of the conditions. When the condition is met, the script block will be executed, and in this case, it will write a string to the console. If no match is found, then the default condition is executed.

```
$car = "Toyota"
switch ($car) {
"Toyota" {
Write-Host "The car is a Toyota."
}
"Honda" {
Write-Host "The car is a Honda."
}
"Ford" {
Write-Host "The car is a Ford."
}
default {
Write-Host "The car is not recognized."
}
}
# Result
The car is a Toyota.
```

The conditions are not limited to strings or numbers, but we can also use comparisons in the condition. For example, if we want to check if a number is lower than a specific value:

```
$value = 5
switch ($value) {
{$_ -le 10} {
Write-Host "The input is lower then 10"
}
{$_ -le 20} {
Write-Host "The input is lower then 20"
}
}
# Result
The input is lower then 10
The input is lower then 20
```

Now if you run the code above, you will notice that both conditions are executed. And that is correct, because the value, 5, is not only lower than 10 (the first condition) but also lower than 20. To prevent both conditions from being executed we can use the `break` statement. This way the switch statement will stop testing the condition if a match is found:

```
$value = 5
switch ($value) {
{$_ -le 10} {
Write-Host "The input is lower then 10";
Break
}
{$_ -le 20} {
Write-Host "The input is lower then 20";
Break
}
}
# Result
The input is lower then 10
```

Switch Multiple Values

With the switch statement in PowerShell, we can also test multiple values instead. To do this you will need to comma-separate the values. Each value will then be tested against the conditions inside the switch statement.

For example, we have a list of fruits, and we want to know which fruits we have:

```
$fruit = "grape","kiwi","pear"
switch ($fruit) {
"apple" { Write-Host "We have an apple." }
"orange" { Write-Host "We have an orange." }
"banana" { Write-Host "We have a banana." }
"pear" { Write-Host "We have a pear." }
"strawberry" { Write-Host "We have a strawberry." }
"grape" { Write-Host "We have a grape." }
"mango" { Write-Host "We have a mango." }
"kiwi" { Write-Host "We have a kiwi." }
"pineapple" { Write-Host "We have a pineapple." }
"watermelon" { Write-Host "We have a watermelon." }
default { Write-Host "The fruit is not recognized." }
}
# Result
We have a grape.
We have a kiwi.
We have a pear.
```

PowerShell Switch Parameters

The PowerShell Switch Statements come with a couple of parameters that we can use to determine how the value is tested:

- **Wildcard** – You can use wildcard characters, like ***** and **?** in your conditions.
- **CaseSensative** – Indicates that the conditions are case sensitive. By default, the conditions are case insensitive
- **Exact** – Enable by default. The switch statement will always perform an exact comparison, only not if you use the wildcard parameter
- **Regex** – Indicates that each condition is treated as a regular expression.
- **`File`** – Allows you to use a file for the value. This way you can check if the contents of a file match one of the conditions.

Wildcard

The `Wildcard` parameter allows you to use wildcards in your conditions. This way only a part of the string has to match the value, instead of an exact match. You can use the `*` to check if a string start or ends with the specified characters. And `?` is used to replace a single character:

```
$name = "John"
switch -Wildcard ($name) {
"Jo*" {
Write-Host "The name starts with 'Jo'."
}
"J?hn" {
Write-Host "The name looks like John."
}
"J*o*" {
```

```
Write-Host "The name contains 'J' and 'o'."
}
default {
Write-Host "The name does not match any pattern."
}
}
```

CaseSensative

The PowerShell Switch statement is case-insensitive by default. But when you add the `casesensative` parameter, then the values should be matched in a case-sensitive manner:

```
$language = "PowerShell"
switch -CaseSensitive ($language) {
"powershell" {
Write-Host "The language is PowerShell in lower case."
}
"PowerShell" {
Write-Host "The language is PowerShell in mixed case."
}
"PYTHON" {
Write-Host "The language is Python in upper case."
}
default {
Write-Host "The language is not recognized."
}
}
```

Regex

Regexes are a great way to check if strings match a particular pattern. Creating regexes can sometimes be a bit challenging, but they are pretty powerful nevertheless. When you specify the regex parameter in the switch statement, then each condition is treated as a regex string.

In the example below, we check if the input is either a valid email address, domain name, or phone number:

```
$input = "example@lazyadmin.nl"
switch -Regex ($input) {
"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$" {
Write-Host "The input is a valid email address."
}
"^[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$" {
Write-Host "The input is a valid domain name."
}
"^\+?[0-9]{1,3}\s?[0-9]{7,14}$" {
Write-Host "The input is a valid phone number."
}
default {
Write-Host "The input is not recognized."
}
}
```

File

A less know, but still very powerful option is the file parameter in the Switch statement. The file parameter indicates that the value is a file. This way we can for example check if the contents of a file match one of the conditions:

```
$file = "C:\Documents\example.txt"
switch -File $file {
{$_ -match "apple"} {
Write-Host "The file contains the word 'apple'."
}
{$_ -match "banana"} {
Write-Host "The file contains the word 'banana'."
}
{$_ -match "orange"} {
Write-Host "The file contains the word 'orange'."
}
default {
Write-Host "The file does not contain any recognized words."
}
}
```

Another option is, for example, to loop through a directory with files. We can then check each file if they match one of the file types specified in the conditions:

```
$path = "C:\Documents"
Get-ChildItem $path | ForEach-Object {
switch -File ($_.FullName) {
"*.txt" {
Write-Host "$($_.Name) is a text file."
}
"*.docx" {
Write-Host "$($_.Name) is a Word document."
}
"*.xlsx" {
Write-Host "$($_.Name) is an Excel workbook."
}
default {
```

```
Write-Host "$($_.Name) is not a recognized file type."
}
}
}
```

Wrapping Up

When you need to use multiple if statements in your script, then it’s often better to use the PowerShell Switch Statement. It allows you to easily compare single or multiple values against multiple conditions.

I hope this article gave you a better understanding of the switch statement. If you have any questions, just drop a comment below.