How to use PowerShell Where Object

lazyadmin.nl/powershell/where-object

August 22, 2023

When retrieving data with PowerShell you often only need a part of it. This is where the PowerShell Where Object cmdlet comes in. It allows you to select only the objects that you need from the results.

Good to know is that there is a big difference between using the `-Filter` parameter of a cmdlet and piping the `Where-Object` cmdlet behind it. Both can filter the results, but there is a big difference between them.

In this article, we will take a look at how to use the PowerShell Where-Object cmdlet and explain what the difference is with the Filter parameter.

PowerShell Where-Object

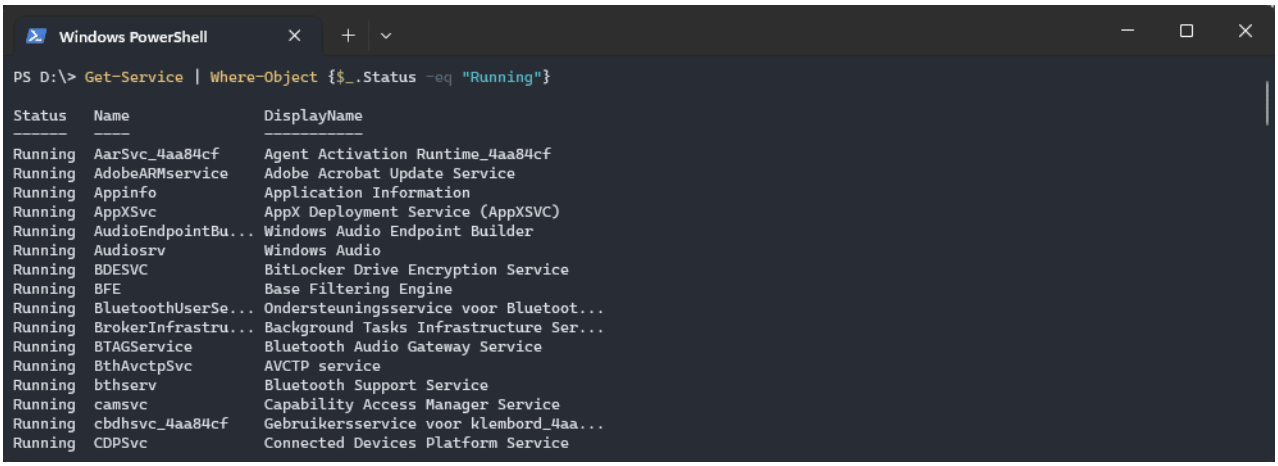
The Where-Object cmdlet can be piped behind any cmdlet in PowerShell to select (filter) only the objects that you need from the results. To select those objects we can use a script block with one or more conditions or a comparison statement.

The comparison statement is easier to read and write when you only want to filter the result on a single statement.

Get-Service | Where-Object -Property Status -eq "Running"
We can even minimize this further if you want, but I don't recommend it because it will make your scripts harder to read:

Get-Service | Where Status -eq "Running"
Or even:
Get-Service | ? Status -eq "Running"
The scripts block however allows you to use multiple conditions, but can also be used to filter the results on a single property:

Get-Service | Where-Object {$_.Status -eq "Running"}



To filter the results on multiple conditions, you will need to wrap each condition between paratheses `( )` and use an `-and` or `-or` operator between them. For example, if we want all services that are not running and where the starttype is set to automatic, we can do the following:

Get-Service | Where-Object {($_.Status -ne "Running") -and ($_.StartType -eq 'Automatic')}
You can find all possible comparison operators here in this article.

Where-Object vs Filter

Before we continue with the Where-Object cmdlet, it's good to understand the difference between actually filtering the results and selecting the results. If we for example want to get specific users from the Active Directory, we could do the following with the Where-Object cmdlet:

Get-ADUser -Filter * | Where-Object {($_.Name -like 'Zoe *') -and ($_.Title -eq "Fixer")}
Now the problem here is that the `Get-ADUser` cmdlet will first get all the users, and then pass the results to the `Where-Object` cmdlet to select only the users where the name starts with Joe. In small environments, you might not see any performance issue with this, but when you have 100.000 users in your Active Directory, you put an unnecessary strain on your server.

If the cmdlet supports it, always use the Filter or Name parameter. This is also known as the "Filter left" principle, which emphasizes on filtering as far left as possible:

Get-Aduser -filter {name -like 'Zoe *'} -properties title | Where-Object {$_.Title -eq "Fixer"}
This will get only the users where name starts with Zoe, includes the property title, and then select only the users that have the title "Fixer".

Where-Object Examples

We have taken a look at the basics of how to use the Where-Object cmdlet in PowerShell. I can explain each operator in detail, but it is easier to understand with the help of examples. So let's take a look at a couple of examples of how to use the where object

cmdlet:

Filtering by Value

Values are basically a property of an object. For example, to get all the files that are larger than 1MB, we can use the Get-ChildItem cmdlet and select only the objects where the property length greater is than 1MB:

```
Get-ChildItem -Path C:\Temp -Recurse | Where-Object {$_.Length -gt 1MB}
```

Filtering on multiple values

Sometimes you want to select the results based on multiple values. You could use multiple `-and` statements for this. But that will make your code harder to read and maintain. A better option is to use an array and check if the properties are in the array.

For this, we can use either the operator `-in` or `-contain` (or their counterparts). Take the following example, we have a list of fruits that we want and a list of fruits that are available. To select only the fruits that we want we can check if the fruit is in the fruits wanted array:

```
$fruitsWanted = @("Apple", "Grapes", "Kiwi", "Strawberry", "Watermelon")
$fruitsAvailable = @("Apple", "Banana", "Orange", "Grapes", "Mango", "Strawberry", "Pineapple", "Watermelon", "Kiwi", "Pear")
$fruitsAvailable | Where-Object {$_ -in $fruitsWanted}
```

Or the other way around with the contain operator:

```
$fruitsAvailable | Where-Object {$fruitsWanted -contains $_}
```

Now this is a simple example, but you can also use this method with nested properties of course.

Using Regular Expressions

The script block allows us to use a wide range of operators, but sometimes a wildcard isn't enough to get the results that you need. Good to know, is that you can also use regular expressions in your conditions. For example, if we want to get all services that start with the letter P followed by a digit we can do:

```
Get-service | Where-Object {$_.name -match "^p\d"}
```

Filtering by Dates

Another common method is to filter the results based on a date. For example when you want to get all the files that are created in the last 7 days. To do this we will use the greater than operator and a date object:

```
Get-ChildItem -path c:\temp | Where-Object {$_.LastWriteTime -gt (Get-Date).AddDays(-7)}
```

Wrapping Up

The Where-Object cmdlet in PowerShell allows you to easily filter the results, but keep the filter left principle in mind. Always use the filter or selection properties of the Get-cmdlet first, before using where object.

I hope you found this article helpful, if you have any questions, just drop a comment below.