**Removing Objects from the Pipeline (Where-Object)**

* Article
* 10/08/2021
* 4 minutes to read
* 2 contributors

In PowerShell, you often generate and pass along more objects to a pipeline than you want. You can specify the properties of particular objects to display by using the Format-\* cmdlets, but this does not help with the problem of removing entire objects from the display. You may want to filter objects before the end of a pipeline, so you can perform actions on only a subset of the initially-generated objects.

PowerShell includes a Where-Object cmdlet that allows you to test each object in the pipeline and only pass it along the pipeline if it meets a particular test condition. Objects that do not pass the test are removed from the pipeline. You supply the test condition as the value of the **FilterScript** parameter.

**Performing Simple Tests with Where-Object**

The value of **FilterScript** is a *script block* - one or more PowerShell commands surrounded by braces ({}) - that evaluates to true or false. These script blocks can be very simple, but creating them requires knowing about another PowerShell concept, comparison operators. A comparison operator compares the items that appear on each side of it. Comparison operators begin with a hyphen character (-) and are followed by a name. Basic comparison operators work on almost any kind of object. The more advanced comparison operators might only work on text or arrays.

**Note**

By default, when working with text, PowerShell comparison operators are case-insensitive.

Due to parsing considerations, symbols such as <,>, and = are not used as comparison operators. Instead, comparison operators are comprised of letters. The basic comparison operators are listed in the following table.

| **Comparison Operator** | **Meaning** | **Example (returns true)** |
| --- | --- | --- |
| -eq | is equal to | 1 -eq 1 |
| -ne | Is not equal to | 1 -ne 2 |
| -lt | Is less than | 1 -lt 2 |
| -le | Is less than or equal to | 1 -le 2 |
| -gt | Is greater than | 2 -gt 1 |
| -ge | Is greater than or equal to | 2 -ge 1 |
| -like | Is like (wildcard comparison for text) | "file.doc" -like "f\*.do?" |
| -notlike | Is not like (wildcard comparison for text) | "file.doc" -notlike "p\*.doc" |
| -contains | Contains | 1,2,3 -contains 1 |
| -notcontains | Does not contain | 1,2,3 -notcontains 4 |

Where-Object script blocks use the special variable $\_ to refer to the current object in the pipeline. Here is an example of how it works. If you have a list of numbers, and only want to return the ones that are less than 3, you can use Where-Object to filter the numbers by typing:

Copy

1,2,3,4 | Where-Object {$\_ -lt 3}

1

2

**Filtering Based on Object Properties**

Since $\_ refers to the current pipeline object, we can access its properties for our tests.

As an example, we can look at the **Win32\_SystemDriver** class in WMI. There might be hundreds of system drivers on a particular system, but you might only be interested in a particular set of the system drivers, such as those which are currently running. For the **Win32\_SystemDriver** class the relevant property is **State**. You can filter the system drivers, selecting only the running ones by typing:

PowerShellCopy

Get-CimInstance -Class Win32\_SystemDriver |

Where-Object {$\_.State -eq 'Running'}

This still produces a long list. You may want to filter to only select the drivers set to start automatically by testing the **StartMode** value as well:

PowerShellCopy

Get-CimInstance -Class Win32\_SystemDriver |

Where-Object {$\_.State -eq "Running"} |

Where-Object {$\_.StartMode -eq "Auto"}

OutputCopy

DisplayName : RAS Asynchronous Media Driver

Name : AsyncMac

State : Running

Status : OK

Started : True

DisplayName : Audio Stub Driver

Name : audstub

State : Running

Status : OK

Started : True

...

This gives us a lot of information we no longer need because we know that the drivers are running. In fact, the only information we probably need at this point are the name and the display name. The following command includes only those two properties, resulting in much simpler output:

PowerShellCopy

Get-CimInstance -Class Win32\_SystemDriver |

Where-Object {$\_.State -eq "Running"} |

Where-Object {$\_.StartMode -eq "Manual"} |

Format-Table -Property Name,DisplayName

OutputCopy

Name DisplayName

---- -----------

AsyncMac RAS Asynchronous Media Driver

bindflt Windows Bind Filter Driver

bowser Browser

CompositeBus Composite Bus Enumerator Driver

condrv Console Driver

HdAudAddService Microsoft 1.1 UAA Function Driver for High Definition Audio Service

HDAudBus Microsoft UAA Bus Driver for High Definition Audio

HidUsb Microsoft HID Class Driver

HTTP HTTP Service

igfx igfx

IntcDAud Intel(R) Display Audio

intelppm Intel Processor Driver

...

There are two Where-Object elements in the above command, but they can be expressed in a single Where-Object element by using the -and logical operator, like this:

PowerShellCopy

Get-CimInstance -Class Win32\_SystemDriver |

Where-Object {($\_.State -eq 'Running') -and ($\_.StartMode -eq 'Manual')} |

Format-Table -Property Name,DisplayName

The standard logical operators are listed in the following table.

| **Logical Operator** | **Meaning** | **Example (returns true)** |
| --- | --- | --- |
| -and | Logical and; true if both sides are true | (1 -eq 1) -and (2 -eq 2) |
| -or | Logical or; true if either side is true | (1 -eq 1) -or (1 -eq 2) |
| -not | Logical not; reverses true and false | -not (1 -eq 2) |
| ! | Logical not; reverses true and false | !(1 -eq 2) |