How to use PowerShell Write Output

lazyadmin.nl/powershell/write-output

February 29, 2024

The PowerShell Write-Output cmdlet is used to send an object or variable to the PowerShell pipeline. When it's the last command in a pipeline, then the output is displayed in the console.

Write-Output is basically the PowerShell equivalent of `echo` or `print` in other programming languages.

In this article, we are going to take a look at how to use the Write-Output cmdlet and what the difference is with the Write-Host cmdlet and Return statement.

PowerShell Write-Output Cmdlet

The write-output cmdlet will send an object, string, or the value of a variable to the success stream of PowerShell. When the cmdlet is the last in the pipeline, then the results will be displayed in the console.

You can see the Write-Output cmdlet as an equivalent of the Echo or Print command in other programming languages. The command Echo is even an alias in PowerShell for the Write-Output cmdlet.

Often you don't need to use the `Write-Output` cmdlet, because the default behavior of most cmdlets is to show the output by default if nothing is piped behind it. For example, to view all running processes on your computer you can use the `Get-Process` cmdlet:

```
# Both command will return the same result:
Get-Process
Get-Process | Write-Output
```

Or when you want to view the value of a variable, you could pipe `Write-Output` behind it, but that is not necessary:

```
$var = "Hello world"
# These will all produce the same result in console
$var
$var | Write-Host
Write-Host $var
$var | Write-Output
```

Write-Output vs Write-Host

When we are talking about the `Write-Output` cmdlet, the question often is, what is the difference with Write-Host, and when to use what? Even though both cmdlets often show the result in the console, there is an important difference between the two.

With the `Write-Host` cmdlet, you print a string directly to the console. With `Write-Output` however, you will send the string to the success stream (stdout) of PowerShell. And this difference is really important to understand.

Take a look at the example below, the function gets the users from the CSV file, and if the users exist in Azure AD, it will add the user to a specific group. Now to keep track of what is happening, we have used the `Write-Host` cmdlet:

```
Function Add-UsersToGroup {
# Import the CSV File
$users = (Import-Csv -Path $path -Delimiter $delimiter -header "name").name
# Find the users in the Active Directory
$users | ForEach-Object {
$user = Get-AzureADUser -filter "$filter eq '$_'" | Select-Object ObjectID
if ($user) {
Add-AzureADGroupMember -ObjectId $group -RefObjectId $user
Write-Host "$_ added to the group"
}else {
Write-Warning "$_ not found in Azure AD"
}
}
}
```

This works great when you run the script locally in your console. But when you are using this function in an Azure Runbook, or want to write the results into a log file? The output of `Write-Host` will be lost because it was written to the console.

If we would use `Write-Output` in this case, then the success message that the users were added to the group, would be sent to the success stream of PowerShell. This allows us to view the results in the Azure Runbook log or redirect the results of this script to a log file.

So in short, you should use Write-Host for interactive scripts, that are run directly in the console only. But for scripts that run as background jobs, or in Azure Runbooks, you should really use Write-Output.

Return vs Write-Output

Also, a question commonly asked is what is the difference between `return` and `Write-Output` in PowerShell? The biggest difference between the two is that the return statement will exit a function or script block. Whereas the Write-Output cmdlet only returns the value, but won't exit the function.

We can see the difference between the two with the following examples:

```
function calculation {
param ($value)
Write-Output "Please wait. Working on calculation..."
Write-Output $value += 42
}
calculation -value 8
# Result
Please wait. Working on calculation...
8
+=
42
```

In the example above we have only used the `Write-Output` cmdlet. As you can see in the results, the calculation isn't executed, instead each object is returned individually.

```
function calculation {
param ($value)
return "Please wait. Working on calculation..."
return $value += 42
}
calculation -value 8
# Result
Please wait. Working on calculation...
```

If we would use the `return` statement instead, then only the string is returned, because the `return` statement will exit the function.

So to show the message and return the calculation, we will need to use both:

```
function calculation {
param ($value)
Write-Output "Please wait. Working on calculation..."
return $value += 42
}
calculation -value 8
# Result
Please wait. Working on calculation...
50
```

Using Variable inside Write Output

Good to know is that we can use variables inside the output of the write-output cmdlet. A simple string variable can be included in the string of write-output when using the double-quotes `" "`. Keep in mind that you can't use variables inside strings wrapped with single-quotes `' '`. The difference here is that PowerShell only processes strings that are wrapped in double quotes.

```
$user = "megan@lazyadmin.nl"
Write-Output "Getting mailbox of $user"
# Result
Getting mailbox of megan@lazyadmin.nl
```

But when working with objects, you might have noticed that you can’t output the items of an object. For example, this **won’t** work:

```
$user = [PSCustomObject]@{
DisplayName = "Megan"
Email = "megan@lazyadmin.nl"
}
Write-Output "Getting mailbox of $user.displayname"
# Result
Getting mailbox of @{DisplayName=Megan; Email=megan@lazyadmin.nl}.displayname
```

To show the display name of the user object, we will need to wrap it in parentheses, like this:

```
$user = [PSCustomObject]@{
DisplayName = "Megan"
Email = "megan@lazyadmin.nl"
}
Write-Output "Getting mailbox of $($user.DisplayName)"
# Result
Getting mailbox of Megan
```

Wrapping Up

Keep in mind that the default behavior of PowerShell is to send the results of a cmdlet to the success stream, which in return is displayed in the console. So often you don’t need to use the Write-Output cmdlet.

However, when you want to show information or progress of a function or script that runs as a background job or in an Azure Runbook for example, then make sure that you use the Write-Output cmdlet.

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