

# Using Script Blocks in Windows PowerShell



**Jeff Hicks**

Author | Teacher

<https://jdhitsolutions.github.io>



# Script Block

A collection of statements or expressions that can be used as a single unit. A script block can use parameters and write to the pipeline.





## **A script block is core PowerShell element**

- Where-Object
- ForEach-Object
- Invoke-Command
- Ad-hoc commands

**It is a way of defining a block of commands that you want to execute**

**You'll see references to “scriptblocks” and “script blocks”**

**Complex script blocks are often used in PowerShell scripting**



```
PS C:\> $sb = { Get-Service | Where-Object {$_.status -eq 'running'} }
```

## Creating a Script Block

Place your code inside { }



```
PS C:\> $sb = { Get-Service | Where-Object {$_ .status -eq 'running'} }
```

## Creating a Script Block

Place your code inside { }

You can have as much code as you want

Where-Object is also using a script block



```
PS C:\> & $sb
```

## Running a Script Block

Use the Invoke operator &



```
PS C:\> & $sb
```

Status	Name	DisplayName
-----	----	-----
Running	AarSvc_94d3c	Agent Activation Runtime_94d3c
Running	Appinfo	Application Information
Running	AppMgmt	Application Management
Running	AppXSvc	AppX Deployment Service (AppXSVC)
Running	AudioEndpointBu...	Windows Audio Endpoint Builder
Running	Audiosrv	Windows Audio
...		

## Running a Script Block

Use the Invoke operator &



```
PS C:\> & $sb
```

Status	Name	DisplayName
-----	----	-----
Running	AarSvc_94d3c	Agent Activation Runtime_94d3c
Running	Appinfo	Application Information
Running	AppMgmt	Application Management
Running	AppXSvc	AppX Deployment Service (AppXSVC)
Running	AudioEndpointBu...	Windows Audio Endpoint Builder
Running	Audiosrv	Windows Audio
...		

## Running a Script Block

Use the Invoke operator &

The output is the same as if you had manually run the code in the script block

The script block makes it simple to re-use





```
PS C:\> Invoke-Command $sb
```

## Running a Script Block

Use the `Invoke-Command` cmdlet

Very useful when it comes to running commands over Windows PowerShell remoting



```
PS C:\> $sb = {Param([string]$log,[int]$count) Get-WinEvent -log $log -max $count}
```

## Using Parameters

Add a Param() block

Parameters are typically positional



```
PS C:\> $sb = {Param([string]$log,[int]$count) Get-WinEvent -log $log -max $count}  
PS C:\> &$sb system 2 | Format-List ProviderName,ID,LevelDisplayName,Message
```

## Using Parameters

**You should specify all parameter values**

**You can use the & operator**

**Positional parameters usually separated by spaces**



```
PS C:\> &$sb system 2 | Format-List ProviderName,ID,LevelDisplayName,Message
```

```
ProviderName      : Microsoft-Windows-Hyper-V-VmSwitch  
Id                : 233  
LevelDisplayName  : Information  
Message           : The operation 'Delete' succeeded on nic 612425AC-7915...
```

```
ProviderName      : Microsoft-Windows-Hyper-V-VmSwitch  
Id                : 234  
LevelDisplayName  : Information  
Message           : NIC 612425AC-7915-46D5-B24E-615F2D46AA2F successfully...
```

## Using Parameters

**You should specify all parameter values**

**You can use the & operator**

**Parameters separated by spaces**



```
PS C:\> &$sb -count 2 -log system | Format-List  
ProviderName,ID,LevelDisplayName,Message
```

```
ProviderName      : Microsoft-Windows-Hyper-V-VmSwitch  
Id                : 233  
LevelDisplayName  : Information  
Message           : The operation 'Delete' succeeded on nic 612425AC-7915...
```

```
ProviderName      : Microsoft-Windows-Hyper-V-VmSwitch  
Id                : 234  
LevelDisplayName  : Information  
Message           : NIC 612425AC-7915-46D5-B24E-615F2D46AA2F successfully...
```

## Using Parameters

**You should specify all parameter values**

**You can use the & operator**

**Parameters separated by spaces**

**Although you can use parameter names with the & operator**



```
PS C:\> $p = @{count=5;log='system'}  
PS C:\> &$sb @p
```

## Using Parameters

Splatting is also an option with the Invoke operator



```
PS C:\> $sb = {Param([string]$log,[int]$count) Get-WinEvent -log $log -max $count}  
PS C:\> Invoke-Command -ScriptBlock $sb -ArgumentList System,2
```

## Using Parameters

Or use Invoke-Command  
Parameters separated by commas



```
PS C:\> Get-Process | Where-Object {$_.ws -ge 100MB} | Select-Object ID,Name,WS
```

## Using Script Blocks

Script blocks are used everywhere in PowerShell





```
PS C:\> Get-Process | Where-Object {$_.ws -ge 100MB} | Select-Object ID,Name,WS
```

Id	Name	WS
--	----	--
11696	slack	138956800
22480	Dropbox	209121280
9364	explorer	200912896
1696	firefox	409370624
18724	googledrivesync	178024448
...		

## Using Script Blocks

Script blocks are used everywhere in PowerShell  
**Where-Object** uses a filtering script block



```
$new = @{  
    Path      = "C:\Work"  
    ItemType = "Directory"  
    Force     = $True  
}
```

## Using Script Blocks

This is a hash table



```
$new = @{  
    Path      = "C:\Work"  
    ItemType = "Directory"  
    Force     = $True  
}  
1..10 | ForEach-Object {  
    New-Item -Name "Data-$_" @new  
}
```

## Using Script Blocks

This is a hash table

ForEach-Object invokes a script block for each processed item

The hash table is splatted to New-Item



```
$new = @{  
    Path      = "C:\Work"  
    ItemType = "Directory"  
    Force     = $True  
}  
1..10 | ForEach-Object {  
    New-Item -Name "Data-$_" @new  
}
```

## Using Script Blocks

This is a hash table

ForEach-Object invokes a script block for each processed item

The hash table is splatted to New-Item

Create folders Data-1 to Data-10 in C:\Work



# Using a Job Script Block

```
Start-Job {  
    param([string]$log,[int]$count)  
    Get-WinEvent -FilterHashtable @{  
        LogName = $log  
        SuppressHashFilter = @{Level=4}  
    } -MaxEvents $count |  
    Group-Object ProviderName -NoElement |  
    Sort-Object Count -Descending  
} -ArgumentList System,1000 -Name LogInfo
```



# Getting Job Results

```
Receive-Job loginfo -Keep | Format-Table -AutoSize
```

Count	Name
273	Microsoft-Windows-Hyper-V-VmSwitch
188	Microsoft-Windows-DistributedCOM
103	Service Control Manager
71	Microsoft-Windows-Kernel-General
59	Microsoft-Windows-Kernel-Processor-Power
40	Microsoft-Windows-FilterManager
32	Netwtw10
21	Microsoft-Windows-Time-Service
18	Microsoft-Windows-DHCPv6-Client
...	



# Building a Function

```
Function Get-LogInfo {  
    Param(  
        [string]$Log = "System",  
        [int]$Count = 100  
    )  
    Get-WinEvent -FilterHashtable @{  
        LogName = $log  
        Level    = 2,3  
    } -MaxEvents $count | Group-Object ProviderName -NoElement |  
    Sort-Object Count -Descending  
} #end function
```

**This can be developed further into a rich PowerShell command**





## Key Take-Aways

Script Blocks are used often in Windows PowerShell

They are treated as units of code

They can use parameters

They can write objects to the pipeline

You can create your own for ad-hoc work from the console

Don't focus on script blocks alone – recognize them when you see them

Help about\_script\_blocks

