All about PowerShell ScriptBlock

Sivaprasad Padisetty (May 2014)

*ScriptBlock is a powerful concept in PowerShell, often misunderstood or not fully exploited. I thought of writing about a generic function “Wait”, since that depends on ScriptBlock, I decided to write about ScriptBlock. In the next blog, I will introduce you to “Wait”. A ScriptBlock is a collection of statements or expressions that can be used as a single unit. It can be saved in a variable, can be passed to function and can be executed remotely. This blog explores some of these aspects of a ScriptBlock.*

# Basics

ScriptBlock is a collection of statements between “{“ and “}”. It can be saved in a variable and executed using the call operator “&”. The same block can be executed multiple times.

$scriptblock = { "Hello World" }

& $scriptblock

& $scriptblock

Output:

Hello World

Hello World

# ScriptBlock with Parameters

ScriptBlock supports param block just like a function.

$scriptBlock = { param ($message) "Hello $message!" }

& $scriptBlock -Message World

Output:

Hello World!

# Variable Scoping

Unlike many languages ScriptBlock references variables dynamically as opposed to capturing current variables. Variables are not evaluated when the ScriptBlock is defined, instead it references the latest value of the variable.

$i = 2

$scriptblock = { "Value of i is $i" }

$i = 3

& $scriptblock # ScriptBlock refers to current values which is 3

$i = 4

& $scriptblock # ScriptBlock refers to current values which is 4

Output:

Value of i is 3

Value of i is 4

Behavior of updating a variable inside a ScriptBlock is consistent with rest of PowerShell. (e.g.) updating a variable in the function. Updates are not reflected in the parent scope with the “&” (call) operator, while the “.” (dot) updates the current scope.

$i = 2

$scriptblock = { $i = $i + 1 }

& $scriptblock

$i

. $scriptblock

$i

Output:

2

3

# Closure

From Wikipedia, “In programming languages, a closure is a function or reference to a function together with a referencing environment. A closure—unlike a plain function pointer—allows a function to access those non-local variables even when invoked outside its immediate lexical scope.”

PowerShell offers GetNewClosure that returns a ScriptBlock with captured variables. In that sense PowerShell offers best of both worlds. The following code fragment illustrates that.

$i = 2

$scriptblock = { "Value of i is $i" }.GetNewClosure()

$i = 3

& $scriptblock # To capture the vairables use GetNewClosure

The output of the above script is shown below. Note without closure, ScriptBlock would have referenced the updated value 3.

Value of i is 2

# ScriptBlock R-Value

If you used static code blocks in other languages like C/C#, you might be surprised to see the behavior of ScriptBlock definition (without assignment). In PowerShell, if the r-value is not assigned to a variable it is spit out to stdout. Let us look at an example “$i=1+2; $i”. In this case the first statement the r-value “1+2” is assigned to a variable $i. In the second statement “$i”, which is the r-value, since it is not assigned to a variable, the value of that is spit out to stdout.

ScriptBlock definition is really a r-value. Just defining a ScriptBlock means, it is a r-value, hence spit out to stdout without executing it. This might be confusing at first, until the concept is understood (certainly I was surprised!). To execute a ScriptBlock you can use the call operator “&”. Following code demonstrates this concept.

PS C:\> { $i = 1+2; $i }

$i = 1+2; $i

PS C:\> &{ $i = 1+2; $i }

3

PS C:\> { $i = 1+2; $i }.ToString()

$i = 1+2; $i

# Invoke-Command & Using

As seen before the ScriptBlock variables has dynamic scope. However the behavior with Invoke-Command is different. Since the execution happens in a different session, for efficiency reasons all the variables in the current scope is not copied to the remote session by default. If a variables has to be copied to the session, it has to be explicitly prefixed with “using:”.

#Create a new session

$s = New-PSSession -ComputerName host1 -Credential $cred

$t1 = 1

$sb = { "t1=$t1" }

#invoke the scriptblock locally

& $sb

#invoke the scriptblock remotely

Invoke-Command -Session $s -ScriptBlock $sb

#invoke the scriptblock remotely with 'using'

$sb = { "t1=$using:t1" }

Invoke-Command -Session $s -ScriptBlock $sb

Remove-PSSession $s

Output from the above script:

t1=1

t1=

t1=1

Explore & Enjoy!

/Siva