about_PSItem - PowerShell

sdwheeler ⋮ 12-15 minutes

about_PSItem

- Article
- 12/12/2022
-

In this article

1. Short description
2. Long description
3. ForEach-Object Process
4. Where-Object FilterScript
5. ForEach and Where methods
6. Delay-bind scriptblock parameters
7. Switch statement scriptblocks
8. Function process blocks
9. Filter definitions
10. The ValidateScript attribute scriptblock
11. The -replace operator's substitution scriptblock
12. See also

Short description

The automatic variable that contains the current object in the pipeline object.

Long description

PowerShell includes the `$PSItem` variable and its alias, `$_`, as automatic variables in scriptblocks that process the current object, such as in the pipeline. This article uses `$PSItem` in the examples, but `$PSItem` can be replaced with `$_` in every example.

You can use this variable in commands that perform an action on every object in a pipeline.

There are a few common use cases for `$PSItem`:

- in the scriptblock for the **Process** parameter of the `ForEach-Object`cmdlet
- in the scriptblock for the **FilterScript** parameter of the `Where-Object` cmdlet
- in the intrinsic methods **ForEach** and **Where**
- with delay-bind scriptblock parameters
- in a `switch` statement's conditional values and associated scriptblocks
- in the `process` block of a function

- in a `filter` definition
- in the scriptblock of the **ValidateScript** attribute
- in the substitution operand scriptblock of the `-replace` operator

The rest of this article includes examples of using `$PSItem` for these use cases.

ForEach-Object Process

The ForEach-Object cmdlet is designed to operate on objects in the pipeline, executing the **Process** parameter's scriptblock once for every object in the pipeline.

You can use `$PSItem` in the **Process** parameter's scriptblock but not in the **Begin** or **End** parameter scriptblocks. If you reference `$PSItem` in the **Begin** or **End** parameter scriptblocks, the value is `$null` because those scriptblocks don't operate on each object in the pipeline.

```
$parameters = @{
    Begin   = { Write-Host "PSItem in Begin is: $PSItem" }
    Process = {
        Write-Host "PSItem in Process is: $PSItem"
        $PSItem + 1
    }
    End     = { Write-Host "PSItem in End is: $PSItem" }
}

$result = 1, 2, 3 | ForEach-Object @parameters

Write-Host "Result is: $result"
```

```
PSItem in Begin is:
PSItem in Process is: 1
PSItem in Process is: 2
PSItem in Process is: 3
PSItem in End is:
Result is: 2 3 4
```

Where-Object FilterScript

The Where-Object cmdlet is designed to filter objects in the pipeline.

You can use `$PSItem` in the scriptblock of the **FilterScript** parameter, which executes once for each input object in the pipeline.

```
1, 2, 3 | Where-Object -FilterScript { ($PSItem % 2) -eq 0 }
```

```
2
```

In this example, the **FilterScript** checks to see if the current object is even, filtering out any odd values, and returns only 2 from the original list.

ForEach and Where methods

Both the ForEach and Where intrinsic methods for arrays take a scriptblock as an input parameter. You can use the `$PSItem` in those scriptblocks to access the current object.

```
@('a', 'b', 'c').ForEach({ $PSItem.ToUpper() }).Where({ $PSItem -ceq 'B' })
```

```
B
```

In this example, the scriptblock of the **ForEach** method uppercases the current object. Then the scriptblock of the **Where** method returns only `B`.

Delay-bind scriptblock parameters

Delay-bind scriptblocks let you use `$PSItem` to define parameters for a pipelined cmdlet before executing it.

```
dir config.log | Rename-Item -NewName { "old_$($_.Name)" }
```

Switch statement scriptblocks

In switch statements, you can use `$PSItem` in both action scriptblocks and statement condition scriptblocks.

```
$numbers = 1, 2, 3

switch ($numbers) {
    { ($PSItem % 2) -eq 0 } { "$PSItem is even" }
    default { "$PSItem is odd" }
}
```

```
1 is odd
2 is even
3 is odd
```

In this example, the statement condition scriptblock checks whether the current object is even. If it's even, the associated action scriptblock outputs a message indicating the current object is even.

The action scriptblock for the `default` condition outputs a message indicating the current object is odd.

Function process blocks

When you define a function, you can use `$PSItem` in the `process` block definition but not in the `begin` or `end` block definitions. If you reference `$PSItem` in the `begin` or `end` blocks, the value is `$null` because those blocks don't operate on each object in the pipeline.

When you use `$PSItem` in the `process` block definition, the value is the value is the current object if the function is called in the pipeline and otherwise `$null`.

```
function Add-One {
    process { $PSItem + 1 }
}

1, 2, 3 | Add-One
```

```
2
3
4
```

Tip

While you can use `$PSItem` in advanced functions, there's little reason to do so. If you intend to receive input from the pipeline, it's best to define parameters with one of the `ValueFromPipeline*` arguments for the Parameter attribute.

Using the **Parameter** attribute and cmdlet binding for advanced functions makes the implementation more explicit and predictable than processing the current object to get the required values.

One good use of `$PSItem` in advanced functions is to inspect the current object itself for debugging or logging when the function has multiple parameters that take input from the pipeline.

```
function Write-JsonLog {
    [CmdletBinding()]
    param(
        [parameter(ValueFromPipelineByPropertyName)]
        [string]$Message
    )
    begin {
        $entries = @()
    }
    process {
        $entries += [pscustomobject]@{
            Message   = $Message
            TimeStamp = [datetime]::Now
        }

        if ($PSItem) {
            $props  = $PSItem | ConvertTo-Json
            $number = $entries.Length
            Write-Verbose "Input object $number is:`n$props"
        }
    }
    end {
        ConvertTo-Json -InputObject $entries
    }
}
```

This example function outputs an array of JSON objects with a message and timestamp. When called in a pipeline, it uses the **Message** property of the current object for each entry. It also writes the JSON

representation of the current object itself to the verbose stream, so you can see the actual input compared to the output logs.

```
$Items = @(
    [pscustomobject]@{
        Name    = 'First Item'
        Message = 'A simple note'
    }
    [pscustomobject]@{
        Name    = 'Item with extra properties'
        Message = 'Missing message, has info instead'
        Info    = 'Some metadata'
        Source  = 'Where this came from'
    }
    [pscustomobject]@{
        Name    = 'Last Item'
        Message = 'This also gets logged'
    }
)

$Items | Write-JsonLog -Verbose
```

```
VERBOSE: Input object 1 is:
{
    "Name":  "First Item",
    "Message":  "A simple note"
}
VERBOSE: Input object 2 is:
{
    "Name":  "Item with extra properties",
    "Message":  "Missing message, has info instead",
    "Info":  "Some metadata",
    "Source":  "Where this came from"
}
VERBOSE: Input object 3 is:
{
    "Name":  "Last Item",
    "Message":  "This also gets logged"
}
[
    {
        "Message":  "A simple note",
        "TimeStamp":  "\/Date(1670344068257)\/"
    },
    {
        "Message":  "Missing message, has info instead",
        "TimeStamp":  "\/Date(1670344068259)\/"
    },
    {
        "Message":  "This also gets logged",
        "TimeStamp":  "\/Date(1670344068261)\/"
```

```
    }
]
```

Filter definitions

You can use `$PSItem` in the statement list of a filter's definition.

When you use `$PSItem` in a `filter` definition, the value is the current object if the filter is called in the pipeline and otherwise `$null`.

```
filter Test-IsEven { ($PSItem % 2) -eq 0 }

1, 2, 3 | Test-IsEven
```

```
False
True
False
```

In this example, the `Test-IsEven` filter outputs `$true` if the current object is an even number and `$false` if it isn't.

The ValidateScript attribute scriptblock

You can use `$PSItem` in the scriptblock of a ValidateScript attribute. When used with **ValidateScript**, `$PSItem` is the value of the current object being validated. When the variable or parameter value is an array, the scriptblock is called once for each object in the array with `$PSItem` as the current object.

```
function Add-EvenNumber {
    param(
        [ValidateScript({ 0 -eq ($PSItem % 2) })]
        [int[]]$Number
    )

    begin {
        [int]$total = 0
    }

    process {
        foreach ($n in $Number) {
            $total += $n
        }
    }

    end {
        $total
    }
}

Add-EvenNumber -Number 2, 4, 6
```

```
Add-EvenNumber -Number 1, 2
```

```
12

Add-EvenNumber:
Line |
  24 |  Add-EvenNumber -Number 1, 2
     |                          ~~~~
     | Cannot validate argument on parameter 'Number'. The
" 0 -eq ($PSItem % 2) " validation script for the argument
with value "1" did not return a result of True. Determine
why the validation script failed, and then try the command
again.
```

In this example the scriptblock for the **ValidateScript** attribute runs once for each value passed to the **Number** parameter, returning an error if any value isn't even.

The `Add-EvenNumber` function adds the valid input numbers and returns the total.

The -replace operator's substitution scriptblock

Starting in PowerShell 6, you can use `$PSItem` when calling the replace operator and defining a substitution scriptblock. When you do, the value of `$PSItem` is the value of the current match.

```
$datePattern = '\d{4}-\d{2}-\d{2}'
'Today is 1999-12-31' -replace $datePattern, { [datetime]$PSItem.Value }
```

```
Today is 12/31/1999 00:00:00
```

In this example, the substitution scriptblock replaces the original date string with the default format for the current culture by casting the value to **datetime**.

See also

- about_Arrays
- about_automatic_variables
- about_Comparison_Operators
- about_functions
- about_Script_Blocks
- about_Switch
- ForEach-Object
- Where-Object

In this article