How-to: PowerShell Operators `$( ) @( ) :: &`

`( )` Grouping Expression operator.

Parenthesis/Brackets work just as they do in mathematics, each pair will determine the order of evaluation and return the result of the expression within.

```
PS C:\> (2 + 3) * 5
```

A shortcut syntax is available `(…).property` that returns a single property from an item or a collection (PowerShell V3.0):

```
PS C:\> (dir).FullName
```

To return multiple properties, pipe to ForEach-Object (%) or Select-Object

Simple Parenthesis will also evaluate variable assignments - displaying the value(s).
for example:

```
$msg = "Hello World"
"$msg"
```

can be rewritten as:

```
($msg = "Hello World")
```

`$( )` Subexpression operator.

While a simple ( ) grouping expression means '*execute this part first*', a subexpression $( ) means '*execute this first and then treat the result like a variable*'.

Use a SubExpression to return specific properties of an object.

Unlike simple ( ) grouping expressions, a subexpression can contain multiple ; semicolon ; separated ; statements.
The output of each statement contributes to the output of the subexpression. For a single result, it will return a scalar. For multiple results, it will return an array.
Subexpressions allow you to evaluate and act on the results of an expression in a single line; with no need for an intermediate variable:

```
if($(code that returns a value/object) -eq
"somevalue") { do_something }
```

```
PS C:\> $city="Copenhagen"
PS C:\> $strLength = "$($city.length)" #n.b.
not "$city.length" that would return
"Copenhagen.Length"
10
```

```
PS C:\> "The result of 2 + 3 = $(2+3)"
PS C:\> $(Get-WMIObject win32_Directory)
```

@( ) Array Subexpression operator.

An array subexpression behaves just like a subexpression except that it guarantees that the output will be an array. This works even if there is no output at all (gives an empy array.)
If the result is a scalar value then the result will be a single element array containg the scalar value.
(If the output is already an array then the use of an array subexpression will have no effect, it won't wrap one array inside of another array.)

```
PS C:\> @(Get-WMIObject win32_logicalDisk)
```

Using either `$( )` or `@( )` will cause the powershell parser to re-evaluate the parsing mode based on the first non-whitespace character inside the parentheses. A neat effect of this is that object properties will be evaluated instead of being treated as literal strings:

`"$user.department"` ==> JDOE.`department`
`"$($user.department)"` ==> "Human Resources"

`::` Static member operator

Call the static properties operator and methods of a .NET Framework class.
To find the static properties and methods of an object, use the `-Static` parameter of Get-Member:

```
[string] | gm -static
[datetime] | gm -static
```

Run the static operators prefixing them with ::

```
[datetime]::now
[datetime]::Utcnow
[string]::join('~','aaa','bbb','ccc')
aaa~bbb~ccc
```

`,` Comma operator

As a binary operator, the comma creates an array.
As a unary operator, the comma creates an array with one member. Place the comma before the member.

`&` Call operator

Run a command, script, or script block. The call operator, also known as the "invocation operator," lets you run

commands that are stored in variables and represented by strings. Because the call operator does not parse the command, it cannot interpret command parameters.

```
C:\PS> $c = "get-executionpolicy"
C:\PS> $c
get-executionpolicy
C:\PS> & $c
AllSigned
```

. Dot sourcing operator

Run a script in the current scope so that any functions, aliases, and variables that the script creates are added to the current scope. (without dot sourcing, the variables created within a script will all disappear when the script finishes.)

```
. C:\sample1.ps1
. .\sample2.ps1
```

Note: The dot sourcing operator is followed by a space. Use the space to distinguish the dot from the dot (.) symbol that represents the current directory.

-f Format operator

Format a string expression.

Place `{0} {1}` etc. into the string as placemarkers where you want the variables to appear, immediately follow the string with the `-f` operator and then lastly, the list of comma separated variables which will be used to populate the placemarkers.

```
Get-ChildItem c:\ | ForEach-Object {'File
{0} Created {1}' -
f$_.fullname,$_.creationtime}
```

Optional format string(s) can be included to add padding/alignment and display dates/times/percentages/hex etc correctly, see the -f format page for full details.

. . Range operator

Produce a sequence of numbers:

```
10..20
5..25
```

This can also be used with a fractional prefix (.)

```
-5...9  # all integer values between -5 and
0.9
-5..0.9 # a more readable version of the
same thing.
```

“No need to ask. He's a smooth operator, smooth operator, smooth operator..” ~ Sade

Related PowerShell Cmdlets:

Comparison Operators -eq -neq -contains etc.
Scriptblock - A collection of statements: { … }
Variables - PowerShell Variables and basic Mathematical operators (+ - = /)
Pipelines - Pass objects down the pipeline.