# Las funciones avanzadas

Las funciones avanzadas (*advanced functions*) aportan funcionalidades nuevas muy interesantes de PowerShell v2. Estas últimas permiten un funcionamiento muy similar a los commandlets, y de manera rápida y sencilla.

En la versión 1.0 de PowerShell, la única forma para crear un commandlet era compilar su propio código desarrollado en C# o Visual Basic .NET y habilitarlo en Windows PowerShell a través de un snap-in. La versión 2 de PowerShell, simplifica esta forma de actuar para permitir al usuario crear sus propios «commandlets» directamente desde la consola. Ya no es necesario tener permisos de desarrollador para hacerlo.

Las funciones avanzadas requieren la palabra clave «CmdletBinding» para identificarlas como tales. La sintaxis de utilización es la siguiente:

```
function <nombre de la función> (<argumento>)
{
   [CmdletBinding()]
   param (<lista de parámetros>)
   # Bloque de instrucciones
}
```

*Ejemplo:*

*Función avanzada de nombre Map-Drive que permite conectar un lector de red.*

```
function Map-Drive
{
   [CmdletBinding()]
   Param([string]$Letra, [string]$Partición)
   $obj = New-Object -Com Wscript.Network
   $obj.MapNetworkDrive("$Letra:", $Partición)
}
```

Sin embargo, podemos observar que estas funciones avanzadas no se listan en un **`Get-Command -Commandtype cmdlet`**. En realidad, existen diferencias entre un commandlet «clásico» y una función avanzada. La principal es que un commandlet editado en la consola se considera como si no fuese realmente del mismo tipo. Este es el motivo por el cual el comando **`Get-Command -Commandtype cmdlet`** no devuelve las funciones avanzadas, sino sólo aquéllas compiladas en C# o VB .NET.

Para obtener las funciones avanzadas que hemos creado, tenemos que introducir la línea siguiente:

```
PS > Get-Command -Commandtype function
```

Cuando editamos una función avanzada, podemos definir los atributos que van a actuar sobre su comportamiento. He aquí la lista no exhaustiva:

| Atributos | Descripción |
|---|---|
| `SupportsShouldProcess` | Este atributo indica que la función avanzada permite las llamadas al método **`ShouldProcess`**. El método **`ShouldProcess`** informa al usuario acerca del resultado de la acción antes de que ésta modifique el sistema. Es decir que al especificar este atributo, el parámetro **`WhatIf`** está activado. |
| `DefaultParameterSet <parámetro>` | Este atributo especifica el nombre de o de los parámetros que la función debe utilizar cuando no sabe determinar cual de ellos debe adoptar. |
| `ConfirmImpact <Valor>` | Este atributo permite definir en qué momento la acción de la función |

| | |
|---|---|
| | deberá ser confirmada por una llamada al método **ShouldProcess**. Este último se llama únicamente cuando el valor asociado al parámetro **ConfirmImpact** (por defecto, se trata del valor medio) es superior o igual al valor de la variable **$ConfirmPreference**. Los valores posibles son: low, medium, high. |
| `Snapin <Nombre del Snap-in>` | Este atributo especifica el nombre del componente informático que se utiliza para hacer funcionar la función. |

*Ejemplo:*

```
Function Nombre-Verbo
{
  [CmdletBinding(SupportsShouldProcess=$true, ConfirmImpact="medium")]
  Param ([string]$Parametro)
  Begin
  {
     # Bloque de instrucciones
  }
  Process
  {
     # Bloque de instrucciones
  }
  End
  {
     # Bloque de instrucciones
  }
}
```

La principal ventaja que presenta una función avanzada con respecto a una función clásica, es que dispone de más control sobre sus parámetros, y eso es gracias a la utilización de atributos y argumentos (los argumentos permiten definir los atributos). Por ejemplo, para especificar que el valor pasado como atributo es de tipo string, y que procede de una tubería, basta con especificar el atributo `parameter` con argumento `ValueFromPipeline=$true`:

```
function Get-Result
{
   [CmdletBinding()]
   Param(
        [parameter(ValueFromPipeline=$true)]$valor
        )
   write-host  "el resultado de la tubería es: $valor"
}
```

O bien, si este valor es necesario para el funcionamiento de la función, entonces, especificando el argumento `Mandatory` a este mismo atributo `parameter`, éste será obligatorio para la ejecución del script.

```
function Get-Result
{
   [CmdletBinding()]
   Param(
       [parameter(Mandatory=$true,ValueFromPipeline=$true)]$valor
       )
   write-host  "el resultado de la tubería es: $valor"
}
```

El atributo mas utilizado se denomina `parameter` (ver arriba). Es él, mediante los argumentos que le son asignados, quien permite actuar sobre el comportamiento de un parámetro deseado. El conjunto de los argumentos utilizables por el atributo `parameter` son listados a continuación.

| Argumento del atributo "parameter" | Descripción |
|---|---|
| `Mandatory` | El argumento `Mandatory` indica que el parámetro es obligatorio si el valor es igual a `$true`.<br>*Sintaxis:*<br>`Param([parameter(Mandatory=$true)]$valor)` |
| `Position` | El argumento `Position` especifica la posición del parámetro en la llamada a la función o al script.<br>*Sintaxis:*<br>`Param([parameter(Position=0)]$valor)` |
| `ParameterSetName` | El argumento `ParameterSetName` especifica el juego de parámetros al que un parámetro pertenece.<br>*Sintaxis:*<br>`Param([parameter(ParameterSetName='cifra')]$valor)` |
| `ValueFromPipeline` | El argumento `ValueFromPipeline` especifica que el parámetro acepta las entradas por tubería, si el valor es igual a `$true`.<br>*Sintaxis:*<br>`Param([parameter(ValueFromPipeline=$true)]$valor)` |
| `ValueFromPipelineByPropertyName` | El argumento `valueFromPipelineByPropertyName` especifica que el parámetro acepta la entrada procedente de una propiedad de un objeto de tubería. Esto significa, por ejemplo, que si la función incluye un parámetro denominado «valor» y que el objeto redirigido incluye una propiedad del mismo nombre («valor»), el parámetro en cuestión adquiere el contenido de la propiedad «valor» del objeto trasmitido.<br>*Sintaxis:*<br>`Param([parameter(ValueFromPipeline=$true)]$valor)` |
| `ValueFromRemainingArguments` | Al contrario del argumento anterior, `ValueFromRemainingArguments` especifica que el parámetro acepta los argumentos de la función.<br>*Sintaxis:*<br>`Param([parameter(ValueFromRemainingArguments =$true)]$valor)` |
| `HelpMessage` | El argumento `HelpMessage` permite indicar una descripción del contenido del parámetro.<br>*Sintaxis:*<br>`Param([parameter(HelpMessage="Una cifra entre 0 y 9999" )]$valor)` |

Evidentemente existen otros atributos aparte de `parameter`. Estos últimos, listados a continuación, no actúan sobre el comportamiento del parámetro sino sobre su contenido. Aquí tiene la lista:

| Argumento del atributo "parameter" | Descripción |
|---|---|

| | |
|---|---|
| `Alias` | Permite indicar un alias en el parámetro.<br>*Sintaxis:*<br>`Param([alias("CN")]$valor)` |
| `AllowNull` | Permite indicar que se autoriza un valor nulo como valor de parámetro.<br>*Sintaxis:*<br>`Param([AllowNull()]$valor)` |
| `AllowEmptyString` | Permite indicar que se autoriza una cadena vacía como valor de parámetro.<br>*Sintaxis:*<br>`Param([AllowEmptyString()]$valor)` |
| `AllowEmptyCollection` | Permite indicar que se autoriza una colección vacía como valor de parámetro.<br>*Sintaxis:*<br>`Param([AllowEmptyCollection()]$valor)` |
| `ValidateCount` | Permite indicar el número mínimo y el número máximo de argumentos que se pueden proporcionar al parámetro en cuestión.<br>*Sintaxis:*<br>`Param([ValidateCount(1,3)]$valor)` |
| `ValidateLength` | Permite definir la longitud mínima y la longitud máxima del valor pasado por parámetro (número de caracteres, por ejemplo).<br>*Sintaxis:*<br>`Param([ValidateLength(1,5)]$valor)` |
| `ValidatePattern` | Permite definir el valor pasado por parámetro según un modelo establecido con las expresiones regulares.<br>*Sintaxis:*<br>`Param([ValidatePattern("[A*]")]$cadena)` |
| `ValidateRange` | Permite definir una rango de valor (valor min y valor max).<br>*Sintaxis:*<br>`Param([ValidateRange(0,20)]$valor)` |
| `ValidateScript` | Permite especificar que un bloque de script se utilice para validar el valor proporcionado por parámetro. Para que el valor sea aceptado, el bloque de script deberá devolver el valor `$true`.<br>*Sintaxis:*<br>`Param([ValidateScript({$_ -le 99 })]$valor)` |
| `ValidateSet` | Permite especificar uno o varios valores a los cuales el valor del parámetro debe corresponder.<br>*Sintaxis:* |

| | |
|---|---|
| | `Param([ValidateSet("Rojo", "Azul", "Verde")]$color)` |
| `ValidateNotNull` | Permite especificar que el valor pasado como argumento no sea `null`.<br>*Sintaxis:*<br>`Param([ValidateNotNull()]$valor)` |
| `ValidateNotNullOrEmpty` | Permite especificar que el valor pasado como argumento no sea `null` o vacío.<br>*Sintaxis:*<br>`Param([ValidateNotNullOrEmpty)]$valor)` |