

10. Escribiendo funciones

Wednesday, September 9, 2020 8:44 PM

Recuerda que para desplegar en pantalla todos los comandos de PowerShell que son funciones, puedes ejecutar el siguiente comando:

```
PS> Get-Command - CommandType Function
```

Sintaxis

1	function Nombre-Funcion
2	{
3	# Código de la función
4	}

Ejemplo:

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> function Show-Message {
>> >> Write-Host "Soy una función"
>> }
PS C:\WINDOWS\system32> Show-Message
Soy una función
PS C:\WINDOWS\system32>
```

Se recomienda seguir la regla para el nombre de las funciones: Verbo-Sustantivo (en inglés). Para ver una lista de verbos para utilizar, puedes usar el comando `Get-Verb`.

Cada que quieras cambiar el comportamiento de una función, la puedes redefinir, es decir, volver a escribirla.

Las funciones pueden ser definidas directamente en consola o en un script. Para funciones pequeñas no hay problema con usar la consola, pero se recomienda escribir las funciones grandes en un script o en un módulo, al cual podremos hacer la llamada después para que cargue la función en la memoria.

Agregando parámetros a las funciones

Las funciones en PowerShell pueden tener el número de parámetros que nosotros decidamos. Recordemos que el uso de parámetros permite a las funciones tener variabilidad en su comportamiento.

```
Ejemplo10.ps1 x
1 function show-Message {
2     [CmdletBinding()] param([Parameter()] [string] $msg)
3     Write-Host "Mensaje:" $msg
4 }
```

El atributo `Parameter[]`

El paso de parámetros a las funciones es opcional, pero con `Parameter[]` podemos cambiar eso.

```
PS C:\WINDOWS\system32> Show-Message -msg "Este es mi mensaje de prueba"
Mensaje: Este es mi mensaje de prueba
```

```
PS C:\WINDOWS\system32> Show-Message
Mensaje:
```

Si queremos que cada que se ejecute la función sea obligatorio enviar el parámetro `$msg`, debemos hacer lo siguiente:

```
1 function Show-Message {
2     [CmdletBinding()] param([Parameter(Mandatory)] [string] $msg)
3     Write-Host "Mensaje:" $msg
4 }
```

```
PS C:\WINDOWS\system32> Show-Message
cmdlet Show-Message at command pipeline position 1
Supply values for the following parameters:
msg: Mensaje que me obligaron a escribir
Mensaje: Mensaje que me obligaron a escribir
```

Parámetros con valores por omisión

```
1 function Show-Message {
2     [CmdletBinding()] param([Parameter()] [string] $msg = "Mensaje defa
3     Write-Host "Mensaje:" $msg
4 }
```

```
PS C:\WINDOWS\system32> Show-Message
Mensaje: Mensaje default
```

Parámetros con lista de valores permitidos

```
1 function Show-Message {
2     param([Parameter(Mandatory)] [ValidateSet('Hola','Adios')] [string]$msg)
3     Write-Host "Mensaje:" $msg
4 }
```

```
PS C:\WINDOWS\system32> c:\users\marle\Downloads\PC64\Ejemplo12.ps1
PS C:\WINDOWS\system32> Show-Message
cmdlet Show-Message at command pipeline position 1
Supply values for the following parameters:
msg: Algo
Show-Message : Cannot validate argument on parameter 'msg'. The argument "Algo" does not belong
set "Hola,Adios" specified by the ValidateSet attribute. Supply an argument that is in the set a
then try the command again.
At line:1 char:1
+ Show-Message
+ ~~~~~
+ CategoryInfo          : InvalidData: () [Show-Message], ParameterBindingValidationExcepti
+ FullyQualifiedErrorId : ParameterArgumentValidationError, Show-Message

PS C:\WINDOWS\system32> Show-Message -msg Adios
Mensaje: Adios
```

Agregando más parámetros

```
1 function Show-Message {
2     param([Parameter(Mandatory)] [string] $msg,
3           [Parameter(Mandatory)] [ValidateSet(62,64)] [int]$grupo)
4     Write-Host "Mensaje:" $msg "`nAtte.- Grupo" $grupo
5 }
```

```
PS C:\WINDOWS\system32> Show-Message
cmdlet Show-Message at command pipeline position 1
Supply values for the following parameters:
msg: Hola, qué tengas bonito día
grupo: 62
Mensaje: Hola, qué tengas bonito día
Atte.- Grupo 62
```

Parámetros de entrada con el Pipeline

```
1 function Show-Message {
2     param([Parameter(Mandatory,valueFromPipeline)] [string] $msg,
3           [Parameter(Mandatory)] [ValidateSet(62,64)] [int]$grupo)
4     Write-Host "Mensaje:" $msg "`nAtte.- Grupo" $grupo
5 }
```

```
PS C:\WINDOWS\system32> $mensajes = @("Hola","Bonito día","Adios!")
PS C:\WINDOWS\system32> $mensajes | Show-Message -grupo 62
Mensaje: Adios!
Atte.- Grupo 62
```

Para corregir lo anterior:

```
1 function Show-Message {
2     param([Parameter(Mandatory,valueFromPipeline)] [string] $msg,
3           [Parameter(Mandatory)] [ValidateSet(62,64)] [int]$grupo)
4     process {
5         Write-Host "Mensaje:" $msg "`nAtte.- Grupo" $grupo }
6 }
```

```
PS C:\WINDOWS\system32> $mensajes = @("Hola","Bonito día","Adios!")
PS C:\WINDOWS\system32> $mensajes | Show-Message -grupo 62
Mensaje: Hola
Atte.- Grupo 62
Mensaje: Bonito día
Atte.- Grupo 62
Mensaje: Adios!
Atte.- Grupo 62
```

También se pueden incluir bloques para procesos al inicio y/o al final de la función:

```
1 function Show-Message {
2     param([Parameter(Mandatory,valueFromPipeline)] [string] $msg,
3           [Parameter(Mandatory)] [ValidateSet(62,64)] [int]$grupo)
4     begin{
5         Write-Host "Bienvenido a mi función" }
6     process {
7         Write-Host "Mensaje:" $msg "`nAtte.- Grupo" $grupo }
8     end{
9         Write-Host "La función ha concluido con éxito" }
10 }
```

```
PS C:\WINDOWS\system32> $mensajes = @("Hola","Bonito día","Adios!")
PS C:\WINDOWS\system32> $mensajes | Show-Message -grupo 62
Bienvenido a mi función
Mensaje: Hola
Atte.- Grupo 62
Mensaje: Bonito día
Atte.- Grupo 62
Mensaje: Adios!
Atte.- Grupo 62
La función ha concluido con éxito
```

Referencias:

- Bertram, Adam (2020) . *PowerShell for SysAdmins*. "no starch press"
<https://nostarch.com/powershellsysadmins>

