

# POWERSHELL

 [Descargar PDF](#)

## ▼ POWERSHELL

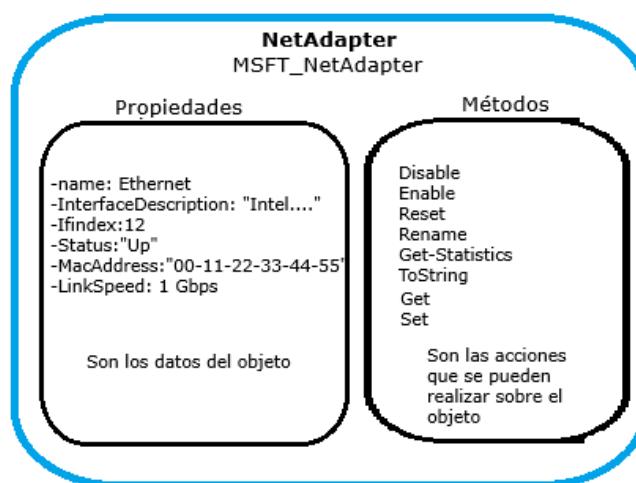
- CONSOLA
- ▼ CMDLETS
  - CONSULTA DE CMDLETS Y FUNCIONES
  - ▼ MODULOS
    - MOSTRAR MODULOS
    - ACTUALIZAR MODULOS
    - IMPORTAR MODULOS EN SESION
  - OBTENER AYUDA
- ▼ LEGIBILIDAD DEL CÓDIGO
  - BACKQUOTE
  - HERE-STRING
  - HASH-TABLE
  - ARRAYS
  - SCRIPT-BLOCKS
- TUBERIAS (|)
- OBJETOS
- FORMATEAR SALIDA
- ▼ CONVERTIR SALIDA A OTROS FORMATOS
  - ▼ CADENA DE TEXTO
    - FICHERO TXT
    - FICHERO CSV
    - FICHERO JSON
    - FICHERO XML
- ▼ SCRIPTS
  - ▼ POLITICAS DE EJECUCIÓN
    - TIPOS
    - AMBITOS
    - ADMINISTRACIÓN DE POLÍTICAS

# CONSOLA

La consola Powershell (`powershell.exe`) permite mediante el uso de objetos del sistema operativo consultar y/o modificar cualquier parámetro del sistema, incluidos evidentemente los de red. **Los objetos tienen propiedades que los definen y métodos u operaciones que se pueden realizar sobre los objetos.** También hay objetos que además incluyen a otros objetos o colecciones de ellos.

## CMDLETS

En **Powershell**, además del uso de órdenes sencillas existen los **cmdlet** y las **funciones**. Desde el punto de vista funcional un cmdlet y una función son similares, por lo que de ahora en adelante nos referiremos a ambos como cmdlet. **Los cmdlet son órdenes o métodos que permiten manipular los objetos y se suelen agrupar en módulos funcionales.** Utiliza el patrón de denominación **verbo-objeto**, donde verbo es el nombre del método a utilizar para manipular el objeto. Son comunes los verbos: Get, Set, New, Add, Remove, Rename, Disable, Enable, Start, Stop ... Los **parámetros** para la ejecución de un cmdlet **se indican mediante un guion -** y después un posible valor para dicho parámetro. Cuando ejecutamos un cmdlet la salida resultante es un objeto o una lista de objetos del sistema, aunque el sistema nos lo muestre como texto por la salida estándar en realidad se tratan de objetos.



### Objeto

Obtiene mediante el método Get el objeto cuya propiedad nombre es "Ethernet"

```
PS C:\> Get-NetAdapter -name "Ethernet"
Name      InterfaceDescription          ifIndex Status     MacAddress      LinkSpeed
----      ----
Ethernet  Realtek PCIe GbE Family Controller    15 Up       40-8D-5C-27-18-9B   100 Mbps
```

### Tip

Usa `TAB` para completar automáticamente nombres de cmdlets o parámetros.

```
PS C:\> Get-NetAdapter - <presiona TAB>
```

# CONSULTA DE CMDLETS Y FUNCIONES

Se puede obtener información de los comandos instalados en powershell con los cmdlet [Get-Command](#).

## Uso de Get-Command

Obtener todos los cmdlet del objeto que contiene el patrón *-Net*.

```
PS C:\> Get-Command -Name *-Net*
 CommandType      Name          Version   Source
-----      -----
 Function        Get-Net6to4Configuration    1.0.0.0  NetworkTransition
 Function        Get-NetAdapter           2.0.0.0  NetAdapter
 Function        Get-NetAdapterAdvancedProperty 2.0.0.0  NetAdapter
 Function        Get-NetAdapterBinding       2.0.0.0  NetAdapter
 Function        Get-NetAdapterChecksumOffload 2.0.0.0  NetAdapter
 Function        Get-NetAdapterEncapsulatedPacketTaskOffload 2.0.0.0  NetAdapter
 Function        Get-NetAdapterHardwareInfo     2.0.0.0  NetAdapter
 Function        Get-NetAdapterIPsecOffload      2.0.0.0  NetAdapter
 Function        Get-NetAdapterLso            2.0.0.0  NetAdapter
 Function        Get-NetAdapterPacketDirect     2.0.0.0  NetAdapter
```

## Uso de Get-Command

Obtener todos los cmdlet que comienzan con el verbo *Format*.

```
PS C:\> Get-Command -Verb Format
 CommandType      Name          Version   Source
-----      -----
 Function        Format-Hex          3.1.0.0  Microsoft.PowerShell.Utility
 Function        Format-Volume        2.0.0.0  Storage
 Cmdlet         Format-Custom       3.1.0.0  Microsoft.PowerShell.Utility
 Cmdlet         Format-List          3.1.0.0  Microsoft.PowerShell.Utility
 Cmdlet         Format-SecureBootUEFI 2.0.0.0  SecureBoot
 Cmdlet         Format-Table         3.1.0.0  Microsoft.PowerShell.Utility
 Cmdlet         Format-Wide          3.1.0.0  Microsoft.PowerShell.Utility
```

## Uso de Get-Command

Obtener todos los cmdlet del módulo de nombre *NetTCPIP*

```
PS C:\> Get-Command -Module NetTCPIP
 CommandType      Name          Version   Source
-----      -----
 Function        Get-Net6to4Configuration    1.0.0.0  NetworkTransition
 Function        Get-NetAdapter           2.0.0.0  NetAdapter
 Function        Get-NetAdapterAdvancedProperty 2.0.0.0  NetAdapter
 Function        Get-NetAdapterBinding       2.0.0.0  NetAdapter
 Function        Get-NetAdapterChecksumOffload 2.0.0.0  NetAdapter
 Function        Get-NetAdapterEncapsulatedPacketTaskOffload 2.0.0.0  NetAdapter
 Function        Get-NetAdapterHardwareInfo     2.0.0.0  NetAdapter
 Function        Get-NetAdapterIPsecOffload      2.0.0.0  NetAdapter
 Function        Get-NetAdapterLso            2.0.0.0  NetAdapter
 Function        Get-NetAdapterPacketDirect     2.0.0.0  NetAdapter
```

## Uso de Get-Command

Obtener todos los cmdlet relacionados con el tratamiento de módulos.

```
PS C:\> Get-Command -Name "*Module*"
 CommandType Name          Version Source
----- ----          ----- -----
 Function    Find-Module    1.0.0.1 PowerShellGet
 Function    Get-InstalledModule 1.0.0.1 PowerShellGet
 Function    ImportSystemModules
 Function    InModuleScope    3.4.0   Pester
 Function    Install-Module   1.0.0.1 PowerShellGet
 Function    Publish-Module   1.0.0.1 PowerShellGet
 Function    Save-Module     1.0.0.1 PowerShellGet
 Function    Uninstall-Module 1.0.0.1 PowerShellGet
 Function    Update-Module    1.0.0.1 PowerShellGet
 Function    Update-ModuleManifest 1.0.0.1 PowerShellGet
 Cmdlet     Export-ModuleMember 3.0.0.0 Microsoft.PowerShell.Core
 Cmdlet     Get-Module       3.0.0.0 Microsoft.PowerShell.Core
 Cmdlet     Import-Module    3.0.0.0 Microsoft.PowerShell.Core
 Cmdlet     New-Module       3.0.0.0 Microsoft.PowerShell.Core
 Cmdlet     New-ModuleManifest 3.0.0.0 Microsoft.PowerShell.Core
 Cmdlet     Remove-Module    3.0.0.0 Microsoft.PowerShell.Core
 Cmdlet     Test-ModuleManifest 3.0.0.0 Microsoft.PowerShell.Core
```

# MODULOS

## MOSTRAR MODULOS

Para mostrar los módulos instalados usamos el cmdlet [Get-Module](#).

## Uso de Get-Module

Mostrar todos los módulos disponibles en la sesión actual.

```
PS C:\> Get-Module
ModuleType Version      Name                           ExportedCommands
----- ----      ----                           -----
Manifest   3.1.0.0      Microsoft.PowerShell.Management {Add-Computer, Add-Content, Checkpoint-Computer,...}
Manifest   3.1.0.0      Microsoft.PowerShell.Utility   {Add-Member, Add-Type, Clear-Variable,...}
Manifest   1.0.0.0      NetTCPIP
Script     2.0.0       PSReadLine                   {Get-PSReadLineKeyHandler,...}
```

## Uso de Get-Module

Mostrar todos los módulos disponibles en el sistema.

```
PS C:\> Get-Module -ListAvailable
```

No se ha mostrado la salida ya que es bastante extensa.

## ACTUALIZAR MODULOS

Se pueden actualizar los módulos del sistema a partir de repositorios como *Powershell Gallery*. Podemos Buscar, Instalar, Actualizar o Desinstalar módulos del sistema. Para ello se utilizan los cmdlets [Find-Module](#), [Install-Module](#), [Update-Module](#) y [Uninstall-Module](#) respectivamente.

## IMPORTAR MODULOS EN SESION

A partir de Powershell 3.0 no es necesario importar módulos instalados a la sesión actual, ya que el sistema lo hace automáticamente al utilizar cualquier cmdlet definido en un módulo. No obstante, se puede realizar manualmente con los cmdlets [Import-Module](#) y [Remove-Module](#) respectivamente.

## OBTENER AYUDA

Para obtener ayuda sobre el uso y las opciones que se pueden utilizar en un cmdlet se utiliza [Get-Help](#).

### Ayuda en línea

Obtener la ayuda sobre `Get-NetIPAddress`.

```
PS C:\> Get-Help -Name Get-NetIPAddress
NOMBRE
      Get-NetIPAddress
SINOPSIS
      Gets the IP address configuration.
SINTAXIS
      Get-NetIPAddress [[-IPAddress] <String[]>] [-AddressFamily <AddressFamily[]>] [-AddressState <AddressState[]>]
      [-AssociatedIPInterface <CimInstance>] [-CimSession <CimSession[]>] [-IncludeAllCompartments] [-InterfaceAlias <String[]>]
      [-InterfaceIndex <UInt32[]>] [-PolicyStore <String>] [-PreferredLifetime <TimeSpan>] [-PrefixLength <Byte[]>]
      [-PrefixOrigin <PrefixOrigin[]>] [-SkipAsSource <Boolean[]>] [-SuffixOrigin <SuffixOrigin[]>] [-ThrottleLimit <Int32>]
      [-Type <Type[]>] [<CommonParameters>]
```

### Tip

Pulsa `TAB` tras para completar el nombre de un cmdlet, una opción, valores de las opciones, etc ...

# LEGIBILIDAD DEL CÓDIGO

Para aumentar la legibilidad del código podemos utilizar varias técnicas.

- Backquote
- Here-strings
- Hash-Tables
- Arrays
- Script-Blocks

## BACKQUOTE

A veces, un cmdlet puede ser demasiado largo para escribirse en una sola línea. En estos casos, puedes dividirlo en varias líneas utilizando el carácter de escape denominado **backquote** ` al final de cada línea y pulsando la tecla `Enter`. Al hacerlo, aparecerá el símbolo `>>` en la línea siguiente, indicando que puedes continuar escribiendo.

### Cmdlet en más de una línea

En el siguiente ejemplo después de escribir el carácter ` pulsamos la tecla `Enter` (no se escribe el literal `<Enter>`) y nos aparece el literal `>>` en la siguiente línea.

```
PS C:\> Set-DnsClientServerAddress -InterfaceAlias MiRed -ServerAddress `<Enter>
>> 1.1.1.1 -Validate:$false
```

## HERE-STRING

En PowerShell, utilizando una estructura denominada **here-string**, podemos escribir texto para pasarlo como parámetro a un cmdlet o incluso cmdlets complejos tratándolos como cadenas de caracteres. Algunas ventajas del uso de here-string son:

- Permite escribir texto en varias líneas sin necesidad de usar backquote (`).
- No hay necesidad de escapar comillas dentro del bloque.

```
#Estructura básica de un here-string
@'....'@  here-string literal
@"...."@  here-string con evaluación de variables
```

Veamos algunos ejemplos

### here-string

Crear una cadena literal

```
$nombre = "Ana"
$mensaje = @'
    Hola $nombre,
    texto literal sin evaluación de "variables".
'@
Write-Output $mensaje
```

Esta será la salida del comando

```
Hola $nombre,  
texto literal sin evaluación de "variables".
```

Crear una cadena con evaluación de variables

```
$nombre = "Ana"  
$mensaje = @  
    Hola $nombre,  
    texto literal con evaluación de "variables".  
"@  
Write-Output $mensaje
```

Esta será la salida del comando

```
Hola ana,  
texto literal con evaluación de "variables".
```

Podemos utilizar los here-string no sólo para crear cadenas que luego pasamos como parámetros a un cmdlet sino que incluso podemos escribir un cmdlet que luego será invocado mediante el uso de la función `Invoke-expression`.

## here-string e invoke-expression

```
#crear el cmdlet con here-string  
$cmdlet = @'  
New-NetIPAddress -InterfaceAlias "MiRed" -IPAddress 192.168.1.20 `  
-PrefixLength 24 -DefaultGateway 192.168.1.100 -Confirm:$false  
'@  
#ejecutar el cmdlet  
Invoke-Expression $cmdlet
```

## HASH-TABLE

En PowerShell, utilizando una estructura denominada **hash-table**, podemos definir los parámetros que queremos pasar a un cmdlet. Posteriormente para aplicar el hash-table al cmdlet se utilizará el operador **splatting** `@` que permite expandir la tabla definida a los parametros del cmdlet. Al igual que here-string se permiten varias lineas y las comillas dobles.

```
#Estructura básica de un hash-table  
$hash-table=@{  
    parametro-1=valor  
    parametro-2=valor  
    ....  
}  
cmdlet @hash-table
```



### Cuidado

En las hash-table hay que tratar de evitar definir parámetros de tipo switch como `WhatIf` ya que a veces no funcionan correctamente dependiendo de como se haya implementado el cmdlet.

## hash-table

```
#crear el hash-table
$ipParams = @{
    InterfaceAlias = "MiRed"
    IPAddress      = "192.168.1.20"
    PrefixLength   = 24
    DefaultGateway = "192.168.1.100"
    Confirm        = $false
}
#ejecutar el cmdlet con los parámetros
#Se observa que además del hash-table
#se pueden pasar también otros parámetros
New-NetIPAddress @ipParams -WhatIf
```

## ARRAYS

En PowerShell, utilizando una estructura denominada **array**, podemos definir de manera **posicional** los parámetros que queremos pasar a un cmdlet. Posteriormente, expandiremos el array definido mediante splatting (@) sobre el cmdlet



### Cuidado

El array es posicional y por tanto los parámetros deben coincidir en el mismo orden en el que los define el cmdlet. Es menos flexible que una hash-table.

## Array

```
#crear el array
$ipParams = @(
    "MiRed",           # InterfaceAlias
    "192.168.1.20",    # IPAddress
    24,                # PrefixLength
    "192.168.1.100",   # DefaultGateway
    $false              # Confirm
)
#ejecutar cmdlet
New-NetIPAddress @ipParams -WhatIf
```

## SCRIPT-BLOCKS

Un **script block** es un bloque de código delimitado por llaves **{ }** que PowerShell puede almacenar, ejecutar o pasar como argumento a un cmdlet o función. Un script-block, es en esencia, una unidad de código ejecutable que puede:

- Guardarse en una variable
- Pasarse como parámetro a otro comando
- Ejecutarse más tarde o repetidamente
- Modificarse dinámicamente.

Para ejecutar el código contenido en el script block utilizaremos `& $sb` o `Invoke-command -ScriptBlock $sb`.

```
#Estructura básica de un script-block
#defineir script-block
$sb={ codigo }
#ejecutar script-block
& Invoke-Command -ScriptBlock $sb
```

## script-block

```
#definicion del script block
$sb = {
    $parameters = @{
        InterfaceAlias = "Ethernet"
        IPAddress      = "192.168.1.20"
        PrefixLength   = 24
        DefaultGateway = "192.168.1.1"
        Confirm        = $false
    }
    New-NetIPAddress @parameters
}
#Ejecución del script block
& $sb
```

Se puede modificar dinámicamente la ejecución de un script-block mediante el paso de variables externas y la clausula `$using:variable`.

## script-block y using

```
#definicion del script block
$ip = "192.168.1.20"
$sb = {
    $parameters = @{
        InterfaceAlias = "Ethernet"
        IPAddress      = $using:ip
        PrefixLength   = 24
        DefaultGateway = "192.168.1.1"
        Confirm        = $false
    }
    New-NetIPAddress @parameters
}
#Ejecución del script block
& $sb
```

También podemos modificar dinámicamente la ejecución de un script-block si definimos parámetros para dicho script-block. Luego podemos sustituir los valores de los parámetros mediante un hash-table o un array.

## script-block y parámetros

```
#definir el script-block
$sb = {
    #definir los parametros del script-block
    param(
        [string]$InterfaceAlias,  # Posición 0
        [string]$IPAddress,       # Posición 1
        [int]$PrefixLength,       # Posición 2
```

```

    [string]$DefaultGateway, # Posición 3
    [switch]$Confirm # Posición 4
)
New-NetIPAddress -InterfaceAlias $InterfaceAlias ` 
    -IPAddress $IPAddress ` 
    -PrefixLength $PrefixLength ` 
    -DefaultGateway $DefaultGateway ` 
    -Confirm:$Confirm
}
# Definir valores de los parámetros mediante hash table
$parameters = @{
    InterfaceAlias = "Ethernet"
    IPAddress      = "192.168.1.20"
    PrefixLength   = 24
    DefaultGateway = "192.168.1.1"
    Confirm        = $false
}
#Ejecutar usando splatting
& $sb @ipParams

```

## TUBERIAS (|)

Se denomina tubería a la posibilidad de encadenar la salida de un cmdlet como entrada de otro. Para realizar tuberías se utiliza el operador pipe `|`. Es muy común utilizar tuberías para dar formato a la salida, por ejemplo, con `Format-Table`, `Format-List` o `Format-Wide`.

### Tuberías

Obtener los usuarios locales de nombre `admin` y `guest` en formato de lista y mostrar solamente las propiedades `Name` y `Enabled`.

```

PS C:\> Get-LocalUser -Name admin, guest | Format-List -Property Name, Enabled
Name      : admin
Enabled   : True

Name      : Guest
Enabled   : False

```

## OBJETOS

Como la salida de los cmdlets es también un objeto o una lista de objetos se pueden utilizar el pipe (tubería) y cmdlets adicionales para manipular los objetos. Algunos cmdlets para manipular objetos son: [ForEach-Object](#), [Where-Object](#), [Compare-Object](#), [Group-Object](#), [Select-Object](#), [Sort-Object](#) ...

### Objetos

Obtener el nombre e índice de las interfaces de red, ordenadas por número de índice.

```

PS C:\> Get-NetIPInterface | Select-Object -Property IfIndex, InterfaceAlias | Sort-Object ifIndex
ifIndex InterfaceAlias
----- -----
1 Loopback Pseudo-Interface 1
5 MiRed
14 Wi-Fi

```

También podemos asignar la salida de un cmdlet a una variable.

### Objetos y variables

Asignar a la variable `$i` el objeto obtenido por el cmdlet `Get-NetIPInterface`.

```
PS C:\> $i = Get-NetIPInterface -InterfaceAlias "MiRed" -AddressFamily IPv4
```

Para ver todas las propiedades y métodos de un objeto podemos usar el cmdlet `Get-Member`. Además podemos ver el valor de una propiedad, modificarlo o invocar un método con la sintaxis **objeto.propiedad** u **objeto.metodo()**.

### Objetos y variables

Mostrar las propiedades y métodos del objeto `$i` del ejemplo anterior.

```
PS C:\> $i | Get-Member
```

La ejecución del comando muestra una gran cantidad de propiedades y métodos.

### Objetos y variables

Mostrar la propiedad `ifIndex` del objeto `$i` del ejemplo anterior.

```
PS C:\> $i.ifIndex  
6
```

## FORMATEAR SALIDA

PowerShell tiene un conjunto de cmdlets que permite controlar cómo se muestra la salida de un cmdlet. Estos cmdlet comienzan con el verbo Format y además permiten seleccionar las propiedades que se quieren mostrar mediante la opción `-Property`.

Los cmdlets de formateo de la salida son:

Cmdlet	Descripción
Format-Wide	Muestra la salida en formato amplio
Format-List	Muestra la salida en formato lista
Format-Table	Muestra la salida en formato tabla

## Formateo de la salida

Mostrar el nombre de los cmdlets del Module `Win*` en Formato ancho a 3 columnas.

```
PS C:\>Get-Command -Module "Win*" | Format-Wide -Property name -Column 3
Get-WindowsUpdateLog      Get-WUAVersion          Get-WUIPendingReboot
Get-WULastInstallationDate Get-WULastScanSuccessDate Install-WUUpdates
Start-WUScan               Disable-WindowsErrorReporting Enable-WindowsErrorReporting
Get-WindowsErrorReporting
```

## Formateo de la salida

Mostrar la ayuda de `Format-Wide` en Formato Lista indicando solo nombre, el módulo y sinopsis.

```
PS C:\> Get-Help -Name "Format-Wide" | Format-List -Property ,Synopsis
ModuleName : Microsoft.PowerShell.Utility
Synopsis   : Formats objects as a wide table that displays only one property of each object.
```

## Formateo de la salida

Mostrar los usuarios del sistema en Formato *Tabla* indicando solo el nombre y si está o no habilitado.

```
PS C:\> Get-LocalUser | Format-Table -Property name, enabled -AutoSize
Name        Enabled
----        -----
Administrador True
DefaultAccount False
Invitado     False
Profesor     True
WDAGUtilityAccount False
```

# CONVERTIR SALIDA A OTROS FORMATOS

La salida de los cmdlet son objetos. A veces, es necesario obtener la salida en otros formatos.

## CADENA DE TEXTO

Para producir la salida como una cadena de texto se utilizarán el operador pipe y el cmdlet `Out-String`. Por ejemplo, un uso común es utilizar la salida de un cmdlet para asignar el valor a una variable (las variables se definen en powershell utilizando el símbolo `$`).

### Conversión a cadena de texto

Obtener el nombre de la interfaz *MiRed* y asignar a una variable.

```
PS C:\> $MiRedName = (Get-NetAdapter -Name MiRed | Format-Wide -Property ifIndex | Out-String).Trim()
```

El método `Trim()` elimina los espacios a izquierda y derecha de una cadena de texto.

## FICHERO TXT

Se utiliza el cmdlet [Out-File](#).

### Conversión a fichero de texto

Obtener todos los procesos del sistema y guardarlos en un fichero `.txt`.

```
PS C:\> Get-Process | Out-File -FilePath MisProcesos.txt
```

## FICHERO CSV

Se utiliza los cmdlets [ConvertTo-CSV](#) y [Out-File](#).

### Conversión a fichero CSV

Obtener todos los procesos del sistema y guardarlos en un fichero `csv`.

```
PS C:\> Get-Process | ConvertTo-Csv -NoTypeInformation | Out-File -FilePath MisProcesos.csv
```

## FICHERO JSON

Se utiliza los cmdlets [ConvertTo-JSON](#) y [Out-File](#).

### Conversión a fichero JSON

Obtener todos los procesos del sistema y guardarlos en un fichero `json`.

```
PS C:\> Get-Process | ConvertTo-Json | Out-File -FilePath MisProcesos.json
```

## FICHERO XML

Se utiliza los cmdlets [ConvertTo-XML](#) y [Out-File](#).

### Conversión a fichero XML

Obtener todos los procesos del sistema y guardarlos en un fichero `xml`.

```
PS C:\> Get-Process | ConvertTo-XML -NoTypeInformation -As String | Out-File -FilePath MisProcesos.xml
```

# SCRIPTS

Los scripts de Powershell son ficheros de texto que contienen una serie de comandos y/o funciones que se ejecutan en el orden en que aparecen. Los scripts pueden contener comentarios, variables, estructuras de control, funciones y cmdlets. Los ficheros de scripts suelen tener las siguientes extensiones.

Extensión	Descripción
.ps1	Script
.psm1	Script de módulo
.ps1xml	Script de configuración

## POLITICAS DE EJECUCIÓN

La ejecución de scripts en Powershell está restringida por las [políticas de ejecución](#). Estas políticas de ejecución son una característica de seguridad que controla las condiciones en que Powershell carga ficheros de configuración y ejecuta los scripts.

### TIPOS

- **Restricted**
  - No se puede ejecutar ningún script, solamente se permiten órdenes o cmdlets individuales por línea de comandos.
  - Es la política predeterminada para los sistemas Windows de tipo desktop o cliente.
  - Seguridad muy alta.
- **AllSigned**
  - Se pueden ejecutar solamente los scripts y ficheros de configuración locales firmados por un editor de confianza.
  - Seguridad alta.
- **RemoteSigned**
  - Se pueden ejecutar los scripts y ficheros de configuración bajados de internet y firmados por un editor de confianza.
  - Se pueden ejecutar todos los scripts y ficheros de configuración locales, aunque no estén firmados.
  - Seguridad media.
- **Unrestricted**
  - Se puede ejecutar cualquier script y fichero de configuración.
  - Es la política predeterminada para los sistemas no Windows que incluyen Powershell y hoy en día no se puede cambiar.
  - Avisa al usuario antes de ejecutar scripts y ficheros de configuración que no son de la zona de la intranet local.
  - Seguridad baja.
- **Bypass**
  - Se pueden ejecutar cualquier script y fichero de configuración.
  - No hay ningún aviso, petición o permiso para la ejecución.
  - Seguridad muy baja.
- **Default**
  - Establece la política de ejecución predeterminada.

- `Restricted` para los sistemas Windows de tipo desktop o cliente.
- `RemoteSigned` para los sistemas Windows de tipo servidor.
- `Unrestricted` para los sistemas no Windows y no se puede cambiar.

- **Undefined**

- El administrador no ha configurado ninguna política de seguridad.
- En este caso se aplican las políticas de tipo `Default`.

## AMBITOS

Las políticas de ejecución se pueden configurar y ejecutar en diferentes **ámbitos** (`scopes`). Los ámbitos por orden de preferencia son: *MachinePolicy*, *UserPolicy*, *Process*, *CurrentUser* y *LocalMachine*

## ADMINISTRACIÓN DE POLÍTICAS

Las políticas de ejecución se pueden administrar a través del objeto `ExecutionPolicy` y los cmdlets [Get-ExecutionPolicy](#) y [Set-ExecutionPolicy](#).

### Políticas de ejecución

Obtener la configuración actual de las políticas de ejecución.

```
PS C:\> Get-ExecutionPolicy -List
Scope      ExecutionPolicy
-----      -----
MachinePolicy      Undefined
UserPolicy        Undefined
Process          Undefined
CurrentUser       Undefined
LocalMachine     RemoteSigned
```

Obtener configuración actual de políticas de ejecución para el ámbito `CurrentUser`.

```
PS C:\> Get-ExecutionPolicy -Scope CurrentUser
```

Configurar `Bypass` para el ámbito `LocalMachine`.

```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope LocalMachine
```

También es válido el siguiente cmdlet:

```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy Bypass
```

Configurar `Bypass` para el ámbito `CurrentUser`.

```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope CurrentUser
```

Borrar la configuración del ámbito `CurrentUser`.

```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy Undefined -Scope CurrentUser
```



### Tip

Recordad, que si la configuración es `Undefined` entonces se aplica la configuración por defecto para el ámbito en función de que el sistema sea un cliente, un servidor o un sistema no Windows.