5. INTRODUCCIÓN A POWERSHELL

5.1 ¿Qué es PowerShell?

PowerShell es una solución de automatización de tareas multiplataforma formada por un shell de línea de comandos, un lenguaje de scripting y un marco de administración de configuración. PowerShell funciona en Windows 10, Linux y macOS.

A diferencia de la mayoría de los shells que solo aceptan y devuelven texto, PowerShell acepta y devuelve objetos .NET. El shell incluye las siguientes características:

- Un historial de línea de comandos sólido.
- Finalización con tabulación y predicción de comandos (vea about_PSReadLine).
- Admite alias de comando y parámetro.
- Canalización para encadenar comandos.
- Sistema de ayuda en la consola, similar a las páginas `man` de UNIX.

Nota:

.NET se trata de una **plataforma para el desarrollo de software** que fue lanzada por **Microsoft** con la finalidad de fusionar su amplio catálogo de productos, que va desde sus múltiples sistemas operativos hasta herramientas de desarrollo.

5.2 Enlaces a información sobre Powershell

¿Qué es PowerShell? - PowerShell | Microsoft Docs

Introducción - PowerShell | Microsoft Docs

Introducción a PowerShell - PowerShell | Microsoft Docs

Sistema de ayuda - PowerShell | Microsoft Docs

Detección de objetos, propiedades y métodos - PowerShell | Microsoft Docs

Comandos únicos de una línea y canalización - PowerShell | Microsoft Docs

Formato, alias, proveedores, comparación - PowerShell | Microsoft Docs

5.3 Cmdlets

Un cmdlet es un comando ligero que se usa en el entorno de PowerShell. Podemos crear nuestros propios CmdLets.

PowerShell usa un par de nombres verbo y sustantivo para dar nombre a los cmdlets. Por ejemplo, el cmdlet Get-Command incluido en PowerShell se usa para obtener todos los cmdlets que están registrados en el shell de comandos. El verbo identifica la acción que realiza el cmdlet y el nombre identifica el recurso en el que el cmdlet realiza la acción. Estos nombres se especifican cuando la clase .NET se declara como un cmdlet.

5.4 Cómo encontrar un comando concreto (Get-Command)

Get-Help Get-Command -Online

```
Get-Command
   [-Verb <String[]>]
   [-Noun <String[]>]
   [-Module <String[]>]
   [-FullyQualifiedModule <ModuleSpecification[]>]
   [-TotalCount <Int32>]
   [-Syntax]
   [-ShowCommandInfo]
   [[-ArgumentList] <Object[]>]
   [-All]
   [-ListImported]
   [-ParameterName <String[]>]
   [-ParameterType <PSTypeName[]>]
   [<CommonParameters>]
```

Práctica:

- Buscar comandos relacionados con “Servicios”, “Tareas Programadas”, “Credenciales” y “Dirección IP”.
  Pista 1: “*palabra_a_buscar*”
  Pista 2: Buscar en inglés

5.5 Obtener Ayuda sobre un comando concreto (Get-Help)

Get-Help Get-Help -Online

```
Get-Help -Name Get-Command -Full
Get-Help
   [[-Name] <String>]
   [-Path <String>]
   [-Category <String[]>]
   [-Component <String[]>]
   [-Functionality <String[]>]
   [-Role <String[]>]
   [-Full]
   [<CommonParameters>]
```

```
Get-Help -Name Get-Command -Detailed
Get-Help -Name Get-Command -Examples
Get-Help -Name Get-Command -Online
Get-Help -Name Get-Command -Parameter Noun
Get-Help -Name Get-Command -ShowWindow
```

Práctica:

- Probar el comando Get-Help con:
  - Get-Event
  - Get-ChildItem
  - Get-NetIPAddress
  - Get-ComputerInfo

  Nota: usar los parámetros Detailed, Examples, Online, Parameter y ShowWindow

5.6 Detección de objetos, propiedades y métodos (Get-Member)

Get-Help Get-Member -Online

```
Get-Member
    [-InputObject <PSObject>]
    [[-Name] <String[]>]
    [-MemberType <PSMemberTypes>]
    [-View <PSMemberViewTypes>]
    [-Static]
    [-Force]
    [<CommonParameters>]
```

Listar propiedades y métodos (usando Get-Member):

Get-Member obtiene las propiedades y los métodos de un objeto. Podemos pasarle un objeto a Get-Member de 2 formas:

CmdLet (si el resultado del CmdLet es un objeto):

- Forma 1: Get-Help | Get-Member
- Forma 2: Get-Member -InputObject Get-Help

Objeto:

- $items = Get-ChildItem
- Forma 1: $items | Get-Member
- Forma 2: Get-Member -InputObject $items

Listar sólo propiedades (usando Get-Member):

Quiero saber que propiedades tienes los objetos que devuelven los CmdLets Get-Service y Get-Process. Y sólo quiero esa información, no más.

Get-Service | Get-Member -MemberType Property

Get-Process | Get-Member -MemberType Property

Nota:

Si hacemos: “Get-Help | Get-Member”, veremos qué “-MemberType Property” es uno de sus posibles argumentos.

Listar sólo métodos (usando Get-Member):

Quiero saber qué métodos tiene el objeto que devuelve el CmdLet Get-Service para trabajar con servicios.

```
Get-Service | Get-Member -MemberType Method
```

Quiero saber qué métodos tiene el CmdLet Get-Process para trabajar con procesos.

```
Get-Process | Get-Member -MemberType Method
```

5.7 Filtrar y obtener propiedades (Select-Object)

Get-Help Select-Object -Online

```
Select-Object
      [-InputObject <PSObject>]
      [[-Property] <Object[]>]
      [-ExcludeProperty <String[]>]
      [-ExpandProperty <String>]
      [-Unique]
      [-Last <Int32>]
      [-First <Int32>]
      [-Skip <Int32>]
      [-Wait]
      [<CommonParameters>]
```

Select-Object me permite seleccionar qué propiedad/es quiero que me devuelva el CmdLet u objeto. Por ejemplo, quiero mostrar por pantalla la lista de procesos, pero sólo me interesa ver el nombre del proceso ,ID y % de CPU que están usando.

```
Get-Process | Select-Object -Property Name, Id, CPU
```

Nota:

¿Cuáles son todas las propiedades que puedo mostrar?

```
Get-Process | Get-Member -MemberType Properties
```

Práctica:

Mostrar el estatus del servicio “wuauserv” o Windows Update Service.

Get-Command -Noun “Service”

Get-Help Get-Service -Online

Get-Service | Get-Member

$service = Get-Service -Name “wuauserv”

$service.Status

5.8 Filtrar resultados (Where-Object):

Get-Help Where-Object -Online

```
Where-Object
     [-InputObject <PSObject>]
     [-Property] <String>
     [[-Value] <Object>]
     [-EQ]
     [<CommonParameters>]
```

Comparadores:

-EQ, -NE, -GT, -LT, -GE, -LE, -Match, -NotMatch, -Like, -NotLike, -Contains, -NotContains, -Is, -IsNot

Ejemplo 1:

Utilizando la sentencia del ejemplo anterior “Get-Process | Select-Object -Property Name, Id, CPU”, quiero:

Filtrar todos los procesos cuyo uso de CPU sea mayor a un 1%

Get-Process | Where-Object { $_.CPU -gt “1000”}

Quiero filtrar todos los procesos cuyo nombre sea “notepad” (abrimos la aplicación Notepad primero).

Get-Process | Where-Object { $_.Name -like “notepad”}

Ejemplo 2:

Quiero filtrar todos los eventos de seguridad del día de hoy.

```
Get-Command -Noun “*Event””

Get-EventLog *

Get-EventLog -LogName "Security"

Get-EventLog -LogName "Security" | Get-Member

$date = Get-Date “2021-11-03”

Get-EventLog -LogName "Security" | Where-Object { $_. TimeWritten -gt $date}
```

Ejemplo 3:

Quiero filtrar todos los eventos de seguridad cuyo evento es el 41 (apagado o reinicio inesperado)

```
Get-EventLog -LogName "Security" | Where-Object { $_. InstanceId -eq 41}
```

Ejemplo 4:

Obtener dirección IP de una subred determinada

```
Get-Command -Noun “*IPAddress*”

Get-NetIPAddress

Get-NetIPAddress | Select-Object IpAddress

Get-NetIPAddress | Where-Object { $_.IpAddress -like "192.168.152.*" }

$data = Get-NetIPAddress | Where-Object { $_.IpAddress -like "192.168.152.*" }

$data.IpAddress
```

5.9 ¿Conozco a fondo Powershell?

La respuesta es NO, pero lo que sí sé es:

- Encontrar el comando que necesito
- Buscar documentación sobre ese comando
- Ver qué datos devuelve ese comando
- Ver qué métodos tiene ese comando
- Filtrar la información que el comando me muestra por pantalla
- Filtrar los resultados del comando

¿Qué más necesito saber para empezar a programar scripts en Powershell?

Variables:

$mi_variable = “este_es_el_nombre_de_mi_variable”

Arrays:

```
> $data = @('Zero','One','Two','Three')
> $data.count
> 4
```

Operadores de comparación

- -eq
- -ne
- -gt
- -ge
- -lt
- -le
- -like
- -match
- -contains

Operadores lógicos y variables especiales

- -and, -or, -not, !
- $true, $false, $null

Fuente sobre todos los tipos de operadores:

acerca de los operadores - PowerShell | Microsoft Docs

Conocer la sintaxis de las estructuras de control:

- If
- Else
- Switch
- While
- Do – while
- For
- Foreach

Control de excepciones:

- Try – catch

Imprimir texto por pantalla:

- Write-Host “Hola Mundo”
- Write-Host $mi_variable
- Get-Help Write-Host -Online