PowerShell

Contenido

PowerShell es una interfaz de consola con posibilidad de escritura y unión de comandos por medio de instrucciones. Es compatible con muchas plataformas, incluyendo Windows, macOS y Linux. PowerShell se distribuye bajo la **licencia MIT**, que es una licencia de software libre y de código abierto

Los **cmdlets** (comandos) de PowerShell son comandos simples que se utilizan para realizar tareas específicas dentro de PowerShell, trabajan directamente con objetos .NET y sus métodos, podemos ver todos los los cmdlets, funciones u scripts con **Get-Command**

Un objeto es una instancia de una clase, los objetos tienen propiedades (atributos o campos) y métodos, por ejemplo :

```powershell
# Obtener la fecha y hora actuales
$date = Get-Date

# Acceder a una propiedad
$day = $date.Day             # Obtiene el día del mes
$year = $date.Year           # Obtiene el año

# Llamar a un método
$newDate = $date.AddDays(5)  # Añade 5 días a la fecha actual
```

Podemos ver todas las propiedades de un cmdlet con **Get-Member**:

```
PS C:\> Get-Member -InputObject (Get-Date) -MemberType Properties


   TypeName: System.DateTime

Name        MemberType     Definition
----        ----------     ----------
DisplayHint NoteProperty   DisplayHintType DisplayHint=DateTime
Date        Property       datetime Date {get;}
Day         Property       int Day {get;}
DayOfWeek   Property       System.DayOfWeek DayOfWeek {get;}
DayOfYear   Property       int DayOfYear {get;}
Hour        Property       int Hour {get;}
Kind        Property       System.DateTimeKind Kind {get;}
Millisecond Property       int Millisecond {get;}
Minute      Property       int Minute {get;}
Month       Property       int Month {get;}
Second      Property       int Second {get;}
Ticks       Property       long Ticks {get;}
TimeOfDay   Property       timespan TimeOfDay {get;}
Year        Property       int Year {get;}
DateTime    ScriptProperty System.Object DateTime {get=if ((& { Set-StrictMode -Vers
```

También los metodos:

```
PS C:\Users\Administrador> Get-Member -InputObject (Get-Date) -MemberType Method


   TypeName: System.DateTime

Name                 MemberType Definition
----                 ---------- ----------
Add                  Method     datetime Add(timespan value)
AddDays              Method     datetime AddDays(double value)
AddHours             Method     datetime AddHours(double value)
AddMilliseconds      Method     datetime AddMilliseconds(double value)
AddMinutes           Method     datetime AddMinutes(double value)
AddMonths            Method     datetime AddMonths(int months)
AddSeconds           Method     datetime AddSeconds(double value)
AddTicks             Method     datetime AddTicks(long value)
AddYears             Method     datetime AddYears(int value)
CompareTo            Method     int CompareTo(System.Object value), int CompareTo(da
Equals               Method     bool Equals(System.Object value), bool Equals(dateti
GetDateTimeFormats   Method     string[] GetDateTimeFormats(), string[] GetDateTimeF
GetHashCode          Method     int GetHashCode()
GetObjectData        Method     void ISerializable.GetObjectData(System.Runtime.Seri
GetType              Method     type GetType()
GetTypeCode          Method     System.TypeCode GetTypeCode(), System.TypeCode IConv
IsDaylightSavingTime Method     bool IsDaylightSavingTime()
Subtract             Method     timespan Subtract(datetime value), datetime Subtract
ToBinary             Method     long ToBinary()
ToBoolean            Method     bool IConvertible.ToBoolean(System.IFormatProvider p
ToByte               Method     byte IConvertible.ToByte(System.IFormatProvider prov
ToChar               Method     char IConvertible.ToChar(System.IFormatProvider prov
ToDateTime           Method     datetime IConvertible.ToDateTime(System.IFormatProvi
ToDecimal            Method     decimal IConvertible.ToDecimal(System.IFormatProvide
ToDouble             Method     double IConvertible.ToDouble(System.IFormatProvider
ToFileTime           Method     long ToFileTime()
ToFileTimeUtc        Method     long ToFileTimeUtc()
ToInt16              Method     int16 IConvertible.ToInt16(System.IFormatProvider pr
ToInt32              Method     int IConvertible.ToInt32(System.IFormatProvider prov
ToInt64              Method     long IConvertible.ToInt64(System.IFormatProvider pro
ToLocalTime          Method     datetime ToLocalTime()
ToLongDateString     Method     string ToLongDateString()
ToLongTimeString     Method     string ToLongTimeString()
ToOADate             Method     double ToOADate()
ToSByte              Method     sbyte IConvertible.ToSByte(System.IFormatProvider pr
ToShortDateString    Method     string ToShortDateString()
ToShortTimeString    Method     string ToShortTimeString()
ToSingle             Method     float IConvertible.ToSingle(System.IFormatProvider p
ToString             Method     string ToString(), string ToString(string format), s
ToType               Method     System.Object IConvertible.ToType(type conversionTyp
ToUInt16             Method     uint16 IConvertible.ToUInt16(System.IFormatProvider
ToUInt32             Method     uint32 IConvertible.ToUInt32(System.IFormatProvider
ToUInt64             Method     uint64 IConvertible.ToUInt64(System.IFormatProvider
ToUniversalTime      Method     datetime ToUniversalTime()
```

Control de procesos y servicios

- **ps -> Get-Process** ver procesos
- **kill -> Stop-Process** mata procesos
- **Get-Service -ProcessName <servicio>**
- **Stop-Service -ProcessName <servicio>**
- **Start-Service -ProcessName <servicio>**
- **Suspend-Service -ProcessName <servicio>**

Ejemplo:

```
calc.exe
Get-Process -ProcessName CalculatorApp
Stop-Process -ProcessName CalculatorApp
calc.exe
Stop-Process -Id 2828
```

Alias [1]

- **New-Alias -Name "ver" -Value Get-ChildItem**
- **Get-Alias** ver los alias que hay en el sistema

Ficheros y directorios

- **pwd -> Get-Location** donde te encuentras
- **cp -r -> Copy-Item** copiar
- **mv -> Move-Item** mover, renombrar
- **rm, rm -r -> Remove-Item** borrar
- **mkdir** crear directorio
- **ls -> Get-ChildItem** listar archivos y carpetas, para ver los archivos ocultos -h
  - l (vínculo)
  - d (directorio)
  - a (archivo)
  - r (solo lectura)
  - h (oculto)
  - s (sistema)

- **echo -> Write-Output** repetir salida estándar
- **Test-Path -Path <archivo>** nos dice si exite el archivo o carpeta
- **Get-Help -Name Get-ChildItem** obtener ayuda

Ejemplo:

```
PS C:\> pwd

Path
----
C:\

PS C:\> mkdir A

Directorio: C:\
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        29/03/2024      9:27                A

PS C:\> cd A

PS C:\A> mkdir B

Directorio: C:\A
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        29/03/2024      9:27                B

PS C:\A> mkdir C

Directorio: C:\A
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        29/03/2024      9:27                C

PS C:\A> ls

Directorio: C:\A

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        29/03/2024      9:27                B
d-----        29/03/2024      9:27                C
-a----        29/03/2024      9:28              8 archivo.dat

PS C:\A> Test-Path D
False
PS C:\A> Test-Path B
True
PS C:\A> pwd

Path
----
C:\A

PS C:\A> mv B D
PS C:\A> cp -r D F
PS C:\A> ls

Directorio: C:\A
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        29/03/2024      9:27                C
d-----        29/03/2024      9:27                D
d-----        29/03/2024      9:29                F
```

```
-a----        29/03/2024      9:28              8 archivo.dat

PS C:\A> rm F
PS C:\A> ls

Directorio: C:\A
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        29/03/2024      9:27                C
d-----        29/03/2024      9:27                D
-a----        29/03/2024      9:28              8 archivo.dat
```

Caracteres especiales

- ***** (Asterisco):

  Se utiliza como comodín para hacer coincidir cero o más caracteres en una ruta o nombre de archivo, por ejemplo, para listar todos los archivos .txt en un directorio, puedes usar:

  ```
  Get-ChildItem C:\Directorio\*.txt
  ```

- **?** (Signo de interrogación):

  Se utiliza como comodín para hacer coincidir un único carácter en una ruta o nombre de archivo, por ejemplo, para listar todos los archivos que tengan una extensión de tres caracteres en un directorio, puedes usar:

  ```
  Get-ChildItem C:\Directorio\???.*
  ```

- **** (Barra invertida):

  Se utiliza como separador de ruta en las rutas de archivo y directorio en Windows.

  ```
  cd C:\Directorio
  ```

- **"** (Comillas dobles):

  Se utilizan para delimitar cadenas de texto que contienen espacios u otros caracteres especiales, por ejemplo, para especificar un nombre de archivo con espacios al usar un comando como Get-ChildItem:

  ```
  Get-ChildItem "C:\Directorio con Espacios\Archivo.txt"
  ```

- **>** (Redireccionamiento de salida):

  Se utiliza para redirigir la salida de un comando hacia un archivo (sobrescribiendo el archivo si ya existe), por ejemplo, para guardar la salida de un comando en un archivo de texto:

  ```
  Get-Process > procesos.txt
  ```

- **>>** (Redireccionamiento de salida, añadir al final del archivo):

  Se utiliza para redirigir la salida de un comando y agregarla al final de un archivo (sin sobrescribir el contenido existente), por ejemplo, para agregar la salida de un comando al final de un archivo de registro:

  ```
  Get-Date >> registro.txt
  ```

- **$_** (Token) y la **|** (Tubería o pipe):

  El token es una variable automática del objeto actual y la tubería se utiliza para pasar la salida de un comando como entrada a otro comando, por ejemplo, para filtrar la salida de un comando usando Where-Object, puedes usar:

  ```
  Get-Process | Where-Object { $_.Name -eq "explorer" }
  Get-Process | Where-Object { $_.CPU -gt 100 }
  ```

Trabajando con objetos

Hemos visto algunos ejemplos como **Get-Command**, **Get-Member**, encontramos mas cmdlets interesante para la manipulació de objetos:

- **Select-Object** Selecciona propiedades específicas de un objeto o un conjunto de objetos, permitiendo filtrar y proyectar datos.

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

# Name                                           CPU
# ----                                           ---
# cmd                                              0
# conhost                                     0,0625
# conhost                                        0,5
# csrss                                      0,546875
# csrss                                         0,125
# .
# .
# .
```

- **Where-Object** Filtra objetos en una colección basándose en una condición lógica.

```
# Filtrar los procesos que están usando más de 100 segundos de tiempo de CPU
PS C:\A> Get-Process | Where-Object { $_.CPU -gt 100 }

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
1523     113     56432      62976     179,81    528   0 lsass
 688     199    220020     129600     161,33   1800   0 MsMpEng
 280      20     10876      12840     119,36   1604   0 svchost


# Obtener los procesos que han estado ejecutándose por más de una hora
PS C:\> Get-Process | Where-Object { $_.StartTime -lt (Get-Date).AddHours(-1) }

Handles  NPM(K)    PM(K)      WS(K)     CPU(s) Id
-------  ------    -----      -----     ------ --
     71       5     2216       4060       0,00 04
    151      11     6704      16640       0,06 72
     89       7     1172       5524       0,25 88
    333      15     1796       5992       0,59 68
    190      12     1744       8884       0,13 44
    402      33    21376      29240      40,86 32

# Filtrar los procesos que están utilizando más de 100 MB de memoria
Get-Process | Where-Object { $_.WorkingSet -gt 100MB }

# Filtrar procesos ejecutados por un usuario específico
Get-Process | Where-Object { $_.StartInfo.UserName -eq "Usuario1" }

# Filtrar procesos que no responden
Get-Process | Where-Object { $_.Responding -eq $false }
```

- **Sort-Object** Ordena una colección de objetos según una o más propiedades.

```
Get-Process | Sort-Object CPU -Descending

# Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
# -------  ------    -----      -----     ------     --  -- -----------
# 1521     113    56476      63008     179,89    528   0 lsass
#  687     198   224592     141488     161,36   1800   0 MsMpEng
#  253      14    10544      12600     119,41   1604   0 svchost
#  993      39    22360      44860      95,70    360   0 svchost
#  401      33    20508      28288      30,86   1632   0 dfsrs
#  760      43     8184      24432      18,48    836   0 svchost
# 1055       0       40        140       8,22      4   0 System
#  470      19     4100      10436       5,36    844   0 svchost
# .
# .
# .
```

- **New-Object** Crea una instancia de un objeto .NET u otro tipo de objeto.

```
$obj = New-Object -TypeName PSObject -Property @{Name="Tutankamón"; Age=3358}

# PS C:> $obj
#
# Age Name
#  --- ----
# 3358 Tutankamón
#
# PS C:> $obj.Name
# Tutankamón
```

- **ForEach-Object** Ejecuta un bloque de script en cada objeto de una colección que pasa a través del pipeline.

```
Get-Process | ForEach-Object { $_.Name.ToUpper() }
```

- **Measure-Object** Calcula propiedades estadísticas de objetos como el recuento, suma, promedio, etc.

```
Get-Process | Measure-Object CPU -Sum
```

- **Group-Object** Agrupa los procesos por su nombre.

```
Get-Process | Group-Object Name
```

Visualizadores de archivos, filtros y búsqueda de información

- **more** mostrar archivos haciendo pausa en cada pantalla
- **cat -> Get-Content** visualizar el contenido archivo

  Get-Content archivo.dat -tail 10 -wait es como el comando tail -f en GNULinux

  (Get-Content archivo.dat)[2] podemos ver la linea 3

- **select -> Select-Object** se utiliza para procesar cada objeto en un flujo de datos.
- **% -> ForEach-Object** se utiliza para seleccionar y proyectar propiedades específicas de un objeto.

  Get-Content -head 3 archivo.dat | select -Last 1

- **sls -> Select-String = grep** filtrar,
- **Select-String -Pattern <texto>** -Quiet nos devuelve el texto o nada
- **ft -> Format-Table** dar a la salida formato de tabla :

  Get-Service | Format-Table -Property Name, DependentServices

Ejemplo:

```
PS C:\> cat archivo.dat
1 linea
2 linea
3 linea
4 linea
5 linea

PS C:\> (Get-Content archivo.dat)[2]
3 linea

PS C:\> Get-Content -head 3 archivo.dat | select -last 1
3 linea

PS C:\> Get-Content -head 3 archivo.dat | select -First 3
1 linea
2 linea
3 linea

PS C:\> vi .\archivo.dat          # % -> ForEach-Object
PS C:\>  Get-Content archivo.dat | %{ $_ -replace '2', 'B' }
1 linea
B linea
3 linea
4 linea
5 linea

PS C:\> Get-Content archivo.dat | %{ $_ -replace '2', 'B' } | sort
1 linea
3 linea
4 linea
5 linea
B linea

PS C:\> sls 2 archivo.dat

archivo.dat:2:2 linea

PS C:\> sls linea archivo.dat
archivo.dat:1:1 linea
archivo.dat:2:2 linea
archivo.dat:3:3 linea
archivo.dat:4:4 linea
archivo.dat:5:5 linea

PS C:\> sls linea archivo.dat -Quiet
True
PS C:\> sls J archivo.dat -Quiet
False

PS C:\> Get-Content archivo.dat | Select-String -Pattern  2

2 linea


PS C:\> Get-Content archivo.dat | Select-String -Pattern  liena
PS C:\> Get-Content archivo.dat | Select-String -Pattern  linea

1 linea
2 linea
```

```
3 linea
4 linea
5 linea

PS C:\> Get-Service | Format-Table -Property Name, Displayname | select -First 4
Name                                     DisplayName
----                                     -----------
ADWS                                     Servicios web de Active  Directory
AJRouter                                 Servicio de enrutador de AllJoyn
```

Información de harware

- **Get-PSDrive** cmdlet obtiene las unidades de la sesión actual.
- **Get-NetAdapter** en PowerShell te mostrará información sobre las interfaces de red
- **Get-WmiObject** optener información sobre el procesador
- **Get-CimInstance** se utiliza para recuperar instancias de una clase

Ejemplo:

```
PS C:\>  Get-PSDrive

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
Alias                                  Alias
C                   8,28         91,07 FileSystem    C:\
Cert                                   Certificate   \
D                                      FileSystem    D:\
Env                                    Environment
Function                               Function
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
Variable                               Variable
WSMan                                  WSMan

PS C:\> Get-PSDrive -PSProvider FileSystem

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
C                   8,28         91,07 FileSystem    C:\
D                                      FileSystem    D:\

PS C:\> Get-PSDrive -PSProvider FileSystem |  Select-Object Name, Used, Free

Name       Used        Free
----       ----        ----
C    8886775808 97782759424
D             0

PS C:\> Get-PSDrive -PSProvider FileSystem |  Select-Object Name, Used, Free |  Sele
Name       Used        Free
----       ----        ----
C    8886775808 97782759424


PS C:\> $particion_C=$(Get-PSDrive -PSProvider FileSystem |  Select-Object Name, Use
PS C:\> echo $particion_C.Used
8886775808

PS C:\> $porcentaje=100*$particion_C.Used/($particion_C.Used+$particion_C.Free)
PS C:\> echo $porcentaje
8,33112827263359

PS C:\> $porcentaje=[math]::Round(100*$particion_C.Used/($particion_C.Used+$particio
PS C:\> echo $porcentaje
8,33

PS C:\> echo "El $porcentaje % de la partición C esta ocupada"
El 8.33 % de la partición C esta ocupada
```

Ejemplo:

```
PS C:\> (Get-WmiObject Win32_Processor).caption
Intel64 Family 6 Model 142 Stepping 10

PS C:\> (Get-WmiObject Win32_ComputerSystem).SystemType
x64-based PC

PS C:\> (Get-WmiObject Win32_Processor).name
Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

PS C:\> ((Get-WmiObject Win32_Processor).name).split("@")[1]
1.60GHz

PS C:\> Get-WmiObject -Class Win32_Processor | Select -Property Name, Number*

Name                                      NumberOfCores NumberOfEnabledCore NumberOf
----                                      ------------- ------------------- --------
Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz              2


PS C:\> Get-WmiObject -Class Win32_Processor | Select-Object NumberOfCores

NumberOfCores
-------------
            2

PS C:\> Get-WmiObject win32_processor | Select-Object LoadPercentage

LoadPercentage
--------------
            44


PS C:\> Get-WmiObject -class "Win32_Processor"| % {
>>     Write-Host "CPU ID: "
>>     Write-Host $_.DeviceID
>>     Write-Host "CPU Model: "
>>     Write-Host $_.Name
>>     Write-Host "CPU Cores: "
>>     Write-Host $_.NumberOfCores
>>     Write-Host "CPU Max Speed: "
>>     Write-Host $_.MaxClockSpeed
>>     Write-Host "CPU Status: "
>>     Write-Host $_.Status
>>     Write-Host
>> }
CPU ID:
CPU0
CPU Model:
Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
CPU Cores:
2
CPU Max Speed:
1800
CPU Status:
OK
```

Ejemplo:

```
PS C:\>  Get-CimInstance -ClassName Win32_OperatingSystem

SystemDirectory     Organization BuildNumber RegisteredUser     SerialNumber
---------------     ------------ ----------- --------------     ------------
C:\Windows\system32              20348       Usuario de Windows 00454-40000-00001-AA


PS C:\> $(Get-CimInstance -ClassName Win32_OperatingSystem).FreePhysicalMemory
986776

PS C:\> $(Get-CimInstance -ClassName Win32_OperatingSystem).TotalVirtualMemorySize
3276340

PS C:\> $(Get-CimInstance -ClassName Win32_OperatingSystem).NumberOfUsers
6

PS C:\> $(Get-CimInstance -ClassName Win32_OperatingSystem).BootDevice
\Device\HarddiskVolume1

PS C:\> $(Get-CimInstance -ClassName Win32_OperatingSystem).Version
10.0.20348

PS C:\> $(Get-CimInstance -ClassName Win32_OperatingSystem).WindowsDirectory
C:\Windows

PS C:\> $(Get-CimInstance -ClassName Win32_OperatingSystem).CountryCode
34
```

Configuración de Windows (PowerShell)

- **Reiniciar**

```
shutdown /r
shutdown /f #de forma forzosa
```

- **Apagar**

```
shutdown /s
```

- **Consultar IP**

```
ipconfig
```

- **Cambiar IP**

```
netsh interface ip set address name="Ethernet" source=static addr=10.4.104.100 
```

- **Cambiar y consultar el DNS**

```
ipconfig /all #consultar dns
netsh interface ip set dns "Ethernet" static 8.8.8.8
```

- **Cambiar el nombre del equipo**

```
Rename-Computer -NewName "WS22tunombre"
```

- **Habilitar ping**

```
netsh advfirewall firewall add rule name="Habilitar respuesta ICMP IPv4" protoc
```

Instalar el servidor ssh

```
#Primero buscamos características disponibles en línea que coincidan con el patrón
Get-WindowsCapability -Online | Where-Object Name -like 'OpenSSH*'

#Luego la añadimos:
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0

#Iniciar el servicio ssh :
Start-Service sshd

#Para reiniciarlo
Restart-Service sshd

#Para iniciar el servicio ssh durante el arranque de forma automática:
Set-Service -Name sshd -StartupType Automatic

#Para conectarse sin contraseña primero copia tu clave publica
scp -P22 .ssh/id_rsa.pub Administrador@IP:C:\Users\Administrador\.ssh\authorized_key

#Después ya te puedes conectar sin meter contraseña
ssh -X Administrador@IP
```

Para instalarlo con un solo comando:

```
Add-WindowsCapability -Online -Name $(Get-WindowsCapability -Online | Where-Object N
```

En el caso que que quieras conectarte a una sesión de powershell, abre el archivo .ssh/config. Si no existe, puedes crearlo y agrega las siguientes líneas, sonde la <ip> es la **ip o el nombre del equipo al que nos conectamos** y queremos loguarnos directamente con powershell:

```
Host <ip>
  RequestTTY force
  RemoteCommand powershell -NoLogo -NoProfile
```

Instalar edior vi

- Con Chocolatey:

```
#Instalar Chocolatey (si aún no lo tienes):
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManag

choco install vim
```

- Sin Chocolatey [1]:

```
# Visita el sitio oficial de Vim para Windows en https://www.vim.org/download.pl
# Descarga el instalador adecuado para tu sistema, en mi caso:
curl.exe https://ftp.nluug.nl/pub/vim/pc/gvim90.exe -o gvim90.exe

# Ejecutalo dentro de Windows, o desde una conexsión en la que se exporte el dis
./gvim90.exe

# Crea un alias:
Set-Alias -Name vi -Value 'C:\Program Files (x86)\Vim\vim90\vim.exe'
```

Footnotes

[1](1,2) Para crear un alias que esté disponible al principio de cada sesión de PowerShell, debes agregar el comando Set-Alias al archivo de perfil de PowerShell. El archivo de perfil es un script que se ejecuta automáticamente cada vez que inicias una nueva sesión de PowerShell.

Los perfiles pueden ser específicos del usuario o del sistema. Aquí te muestro cómo crear un alias en tu perfil de usuario:

Abre PowerShell como administrador (esto es necesario para modificar archivos en la ubicación del perfil).

Verifica la existencia del archivo de perfil. Puedes hacerlo ejecutando el siguiente comando:

```
Test-Path $PROFILE
```

Si el comando anterior devuelve False, significa que no tienes un archivo de perfil. En ese caso, puedes crear uno ejecutando el siguiente comando:

```
New-Item -Path $PROFILE -Type File -Force
```

Abre el archivo de perfil en tu editor de texto preferido. Puedes hacerlo ejecutando el siguiente comando:

```
C:\Program Files (x86)\Vim\vim90\vim.exe $PROFILE
```

```
notepad $PROFILE
```

Agrega el comando Set-Alias con el alias que deseas crear y el comando que deseas asociar. Por ejemplo:

```
Set-Alias -Name vi -Value 'C:\Program Files (x86)\Vim\vim90\vim.exe'
```

Guarda el archivo y cierra el editor de texto.

Cierra y vuelve a abrir PowerShell. El alias que agregaste debería estar disponible al principio de cada sesión.

Gestión de usuarios

Para ser administrador

```
start-process powershell -verb runas
```

- **Listar usuarios, grupos y usuarios del grupo**

```
Get-LocalUser
Get-LocalGroup
Get-LocalGroupMember -Name "nombre_grupo"
```

- **Crear un usuario con contraseña**

```
$Password = Read-Host -AsSecureString
New-LocalUser -Name nombre_usuario -Password $Password

#Sin que pida confirmación
$Password = ConvertTo-SecureString "alumno" -AsPlainText -Force
```

- **Crear un usuario sin contraseña**

```
New-LocalUser -Name "nombre_usuario" -NoPassword

#Se la podemos asignar después:
Set-LocalUser -Name "nombre_usuario" -Password $Password
```

- **Asignar usuario a un grupo**

```
Add-LocalGroupMember -Group "nombre_grupo" -Member "nombre_usuario"
```

- **Eliminar un usuario**

```
Remove-LocalUser -Name "nombre_usuario"
```

- **Crear y borrar un grupo**

```
New-LocalGroup -Name "nombre_grupo"
Remove-LocalGroup -Name "nombre_grupo"
```

- Configuración en AD