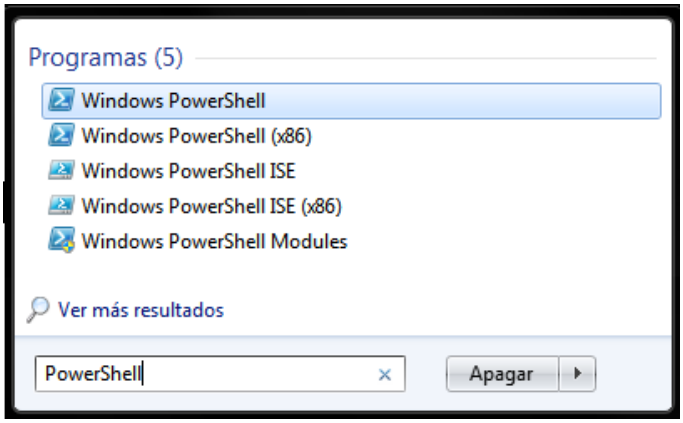
Autor: Darío Hirschfeldt

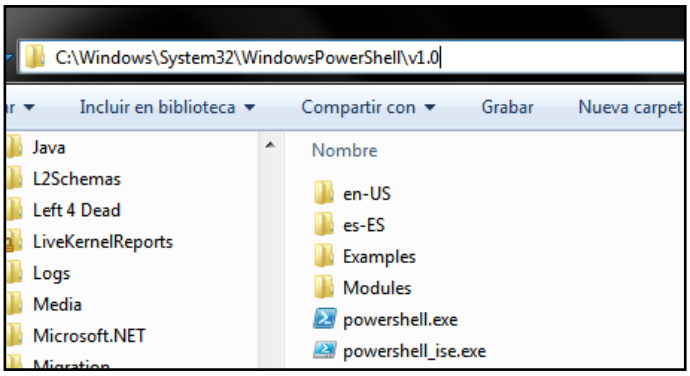
PowerShell

Es una interfaz de línea de comandos originalmente para Sistemas Operativos Windows, su principal característica es que es un intérprete orientado a objetos, a diferencia de bash por ejemplo que recibe y retorna texto. Existe una versión multiplataforma (Windows, macOS y Linux) de código libre y abierto llamada PowerShell Core.

Iniciando PowerShell



Desde el cuadro de texto para Buscar programas y archivos, escribimos PowerShell y seleccionamos Windows PowerShell, existe un ide para desarrollar que es el ISE, el cuál veremos mas adelante. Si no podemos encontrarlo de esta manera, debería encontrarse el ejecutable en la carpeta que se muestra a continuación (puede variar según sistema operativo y versión):



Comandos de ayuda

PowerShell acepta como intrucciones, cmdlets (comandos), cuyo formato es **Verbo-Nombre**. Veamos un ejemplo citando los cmdlets de ayuda que nos provee PowerShell:

GET-HELP **[**Muestra ayuda sobre cmdlets y tópicos conceptuales**]**

```
PS C:\Users\master> Get-Help
TEMA
    Get-Help

DESCRIPCIÓN BREVE
    Muestra ayuda sobre cmdlets y conceptos de Windows PowerShell.
```

Los tópicos conceptuales nos brindan ayuda sobre ciertos temas, que no son comandos, como por ejemplo; cómo maneja PowerShell los arrays o cómo es la sintáxis de las sentencias de control, tipos de datos soportados, etc.

```
PS C:\Users\master> Get-Help about_arrays
TEMA
    about_Arrays

DESCRIPCIÓN BREVE
    Estructura de datos compacta para almacenar elementos de datos

DESCRIPCIÓN DETALLADA
    Una matriz consiste en una estructura de datos para almacenar una
    colección de elementos de datos del mismo tipo. Windows
    PowerShell admite elementos de datos, como string, int (entero de
    32 bits), long (entero de 64 bits), bool (booleano), byte y otros
    tipos de objetos de Microsoft .NET Framework .

CREAR E INICIALIZAR UNA MATRIZ
    Para crear e inicializar una matriz, debe asignar varios valores a una
    variable.
    Los valores almacenados en la matriz están delimitados con una coma y
    separados del nombre de la variable por el operador de asignación (=).

    Por ejemplo, para crear una matriz denominada $A que contenga los
    siete valores numéricos (int) 22, 5, 10, 8, 12, 9 y
    80, escriba:

        $A = 22,5,10,8,12,9,80
```

GET-COMMAND [Información básica sobre cmdlets]

```
PS C:\Users\master> Get-Command

CommandType     Name                Definition
-----------     ----                ----------
Alias           %                   ForEach-Object
Alias           ?                   Where-Object
Function        A:                  Set-Location A:
Alias           ac                  Add-Content
Cmdlet          Add-Computer        Add-Computer [-DomainName] <String> [
Cmdlet          Add-Content         Add-Content [-Path] <String[]> [-Valu
Cmdlet          Add-History         Add-History [[-InputObject] <PSObject
```

Como podemos ver en la salida del comando, existen **Alias** a comandos de otros interpretes como por ejemplo bash, entonces podemos utilizar la sintáxis de PowerShell o utilizar el alias, por ejemplo para cambiar de directorio se utiliza el comando **Set-Location**, en su lugar podemos utilizar el alias **cd**. Tambien podemos ver que existen funciones, por ejemplo **A:** es lo mismo que ejecutar el comando **Set-Location A:** y sería cambiar a la unidad de disco A.

GET-MEMBER [Conocer los métodos y propiedades de los objetos]

```
PS C:\Users\master> "HOLA" | Get-Member


   TypeName: System.String

Name             MemberType           Definition
----             ----------           ----------
Clone            Method               System.Object Clone()
CompareTo        Method               int CompareTo(System.Object value), int CompareTo(string strB)
Contains         Method               bool Contains(string value)
```

Comandos básicos

SET-LOCATION [Cambiar de directorio de trabajo]

```
PS C:\> Set-Location A:
PS A:\>
```

Es el clásico comando **cd**.

GET-CHILDITEM [Listar el contenido de un directorio]

```
PS C:\> Get-ChildItem -Force


    Directorio: C:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d--h-       29/04/2017  03:04 p.m.            $AV_ASW
d--hs       11/06/2019  03:54 p.m.            $Recycle.Bin
d----       08/02/2020  11:32 p.m.            08_02_20
```

En este caso se ejecuta con el opcional –Force para que incluya en el listado archivos ocultos y/o del sistema. El modo representa atributos del elemento listado, si el atributo no aplica encontraremos un guión medio (-), de lo contrario podemos encontrar:

d	a	r	h	s	l
Directorio	Archivo	Solo lectura	Oculto	Sistema	Acceso directo (5.0 o superior)

GET-CONTENT [Obtener el contenido de un elemento]

```
PS C:\> Get-Content .\Frase.txt
"He aquí mi secreto, que no puede ser
más simple: solo con el corazón se puede
ver bien; lo esencial es invisible a los ojos".
PS C:\>
```

Cada línea es un objeto.

NEW-ITEM [Crear un nuevo elemento]

```
PS C:\> New-Item texto.txt -type "File"

    Directorio: C:\

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---       25/09/2020  06:16 a.m.          0 texto.txt
```

Se debe especificar el tipo de elemento con **–type**, en el ejemplo anterior se crea un archivo, para crear un directorio es:

```
PS C:\> New-Item Carpeta -type "Directory"

    Directorio: C:\

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----       25/09/2020  06:23 a.m.            Carpeta
```

Tambien se puede especificar una ruta donde se creará el archivo con el opcional **–path**, incluso se puede incluir el nombre del archivo en la ruta y automáticamente se creará el archivo con ese nombre, a continuación se ejemplifica dicha situación:

```
PS C:\> New-Item -path Carpeta\archivo.txt -type "File"
    Directorio: C:\Carpeta

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---       25/09/2020  06:26 a.m.          0 archivo.txt
```

En el caso de un archivo, se puede agregar contenido al crearlo con el opcional **–value**

```
PS C:\> New-Item HolaMundo.txt -type "File" -value "Hola mundo"
    Directorio: C:\

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---       25/09/2020  06:33 a.m.         10 HolaMundo.txt
```

REMOVE-ITEM [Eliminar un elemento **]**

```
PS C:\> Remove-Item .\Carpeta -recurse
PS C:\> Remove-Item .\archivo
```

COPY-ITEM [Copiar un elemento **]**

```
PS C:\> Copy-Item Carpeta CarpetaB
PS C:\> Copy-Item Frase.txt FraseB.txt
```

Para copiar un directorio con todo su contenido se debe agregar **-recurse**

MOVE-ITEM [Mover un elemento **]**

```
PS C:\> Move-Item Frase.txt FraseC.txt
PS C:\> Move-Item Carpeta CarpetaB
```

La diferencia con Copy-Item es que se elimina el elemento origen, además tratandosé de un directorio se mueve con todo su contenido sin necesidad de opciones extra.

Comandos de formato

Existen algunos comandos que nos permiten formatear la salida por pantalla, por ejemplo para la siguiente salida del comando Get-ChildItem, aplicaremos los distintos comandos:

```
PS C:\> Get-ChildItem Carpeta
    Directorio: C:\Carpeta
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----       25/09/2020  07:32 a.m.            bin
-a---       18/05/2018  04:42 p.m.       2499 bdd.cpp
-a---       22/06/2018  06:29 p.m.        890 bdd.h
-a---       23/01/2020  01:45 a.m.       1731 generador.cbp
-a---       15/08/2019  06:42 p.m.    3688459 generador.exe
```

FORMAT-LIST [Aplica un formato tipo lista para mostrar los elementos **]**
En este caso vamos a quedarnos con el Nombre y el peso

```
PS C:\> Get-ChildItem Carpeta | Format-List -Property Name, Length
Name : bin

Name   : bdd.cpp
Length : 2499

Name   : bdd.h
Length : 890

Name   : generador.cbp
Length : 1731

Name   : generador.exe
Length : 3688459
```

Por un lado vemos que al igual que bash, utilizamos la salida de un comando como entrada de otro a través de una canalización con pipe (|). Por otro lado vemos que la salida se muestra en forma de lista, y por último de cada elemento se muestra solo el nombre y peso, esto lo especificamos con el opcional **–Property** Name, Length.

FORMAT-TABLE [Aplica un formato tipo tabla para mostrar los elementos **]**

```
PS C:\> Get-ChildItem Carpeta | Format-Table -Property Name, Length

Name                                                           Length
----                                                           ------
bin
bdd.cpp                                                          2499
bdd.h                                                             890
generador.cbp                                                    1731
generador.exe                                                 3688459
```

FORMAT-WIDE [Aplica un formato tipo tabla, pero solo mostrando una propiedad **]**

```
PS C:\> Get-ChildItem Carpeta | Format-Wide -Property Name -Colum 3
bin                                    bdd.cpp                                bdd.h
generador.cbp                          generador.exe
```

Se especifica –colum para fijar la cantidad de columnas que tendra la tabla, en este caso 3.

Comandos de filtrado

Tenemos comandos que permiten filtrar o restringir la cantidad de elementos o propiedades que deseamos obtener como resultado final, por ejemplo siguiendo con el ejemplo del Get-ChildItem:

WHERE-OBJECT [Filtra los objetos que cumplen con una condición dada **]**

```
PS C:\> Get-ChildItem Carpeta | Where-Object { $_.Length -gt 1024 }

    Directorio: C:\Carpeta

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---       18/05/2018  04:42 p.m.       2499 bdd.cpp
-a---       23/01/2020  01:45 a.m.       1731 generador.cbp
-a---       15/08/2019  06:42 p.m.    3688459 generador.exe
```

En este ejemplo solo deseamos listar los archivos que pesan mas de 1024 bytes. Por un lado vemos que la condición del debe ir entre llaves { }. También vemos la variable $_, dicha variable es automática y se la utiliza para referenciar al objeto que se está procesando en cada pasada del Where-Object, pensemos a este comando como si fuera un ciclo for, while, etc. que ejecutará la

condición para cada objeto de la colección de objetos pasados desde la entrada. Por lo tanto cuando hacemos **$_.Length** estamos accediendo al atributo Length (Peso en bytes) de cada archivo o carpeta. Por último se utiliza el comparador –gt (Mayor que), PowerShell implementa estos comparadores al estilo de bash.

SELECT-OBJECT [Selecciona propiedades especificas de un conjunto de elementos **]**

```
PS C:\> Get-ChildItem Carpeta | Select-Object -Property Name

Name
----
bin
bdd.cpp
bdd.h
generador.cbp
generador.exe
```

En este caso solo seleccionamos la propiedad Name de la colección de elementos.

Otros comandos útiles

FOREACH-OBJECT [Realizar operaciones sobre un conjunto de elementos **]**

```
PS C:\> Get-ChildItem Carpeta | Foreach-Object { $sum=0 }{ $sum+=$_.Length }{ [math]::round($sum/1MB,2) }
3,52
```

Lo que hacemos aquí es calcular y mostrar por pantalla el tamaño total en MB de los archivos contenidos en la raíz del directorio Carpeta.

Vemos que el Foreach-Object tiene tres partes entre llaves { }, funciona de manera análoga a lo que sería AWK, con un Begin, Process y End, ejecutandosé el Begin y End una sola vez, el Begin antes de empezar a procesar y el End al finalizar el procesamiento.

En PowerShell existe una clase estática llamada **math**, que en este caso utilizamos para mostrar solo dos decimales. A continuación se muestra como obtener una lista de los métodos disponibles de la clase math:

```
PS C:\> [math].GetMethods() | select name -Unique | Format-Wide -Colum 3


Acos                                Asin                                Atan
Atan2                               Ceiling                             Cos
Cosh                                Floor                               Sin
Tan                                 Sinh                                Tanh
Round                               Truncate                            Sqrt
Log                                 Log10                               Exp
Pow                                 IEEERemainder                       Abs
Max                                 Min                                 Sign
BigMul                              DivRem                              ToString
Equals                              GetHashCode                         GetType
```

SORT-OBJECT [Ordena un conjunto de elementos según uno o mas propiedades **]**

```
PS C:\> Get-ChildItem Carpeta | Sort-Object -Property LastWriteTime -descending
    Directorio: C:\Carpeta
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----      25/09/2020  07:32 a.m.            bin
-a---      23/01/2020  01:45 a.m.       1731 generador.cbp
-a---      15/08/2019  06:42 p.m.    3688459 generador.exe
-a---      22/06/2018  06:29 p.m.        890 bdd.h
-a---      18/05/2018  04:42 p.m.       2499 bdd.cpp
```

En este caso ordenamos por fecha de última modificación de manera descendente.

WRITE-OUTPUT [Imprimir por pantalla **]** - **READ-HOST [** Leer desde teclado **]**

```
PS C:\> $nombre = Read-Host "Ingrese su nombre"
Ingrese su nombre: Pepe
PS C:\> Write-Host $nombre -foregroundcolor Green
Pepe
```

Vemos que lo ingresado desde teclado se almacena en la variable **nombre,** en PowerShell tanto para referenciar una variable como para asignarle valor se debe escribir con $. También vemos que con el Write-Host se puede colorear la impresión por pantalla.

TEST-PATH [Validar si existe una ruta dada **]**

```
PS C:\> Test-Path .\Carpeta
True
PS C:\> Test-Path .\Carpeta\pepe
False
```

SPLIT-PATH [Obtener parte de una ruta dada **]**

```
PS C:\> Split-Path .\08_02_20\generador\generador.exe
.\08_02_20\generador
PS C:\> Split-Path .\08_02_20\generador\generador.exe -leaf
generador.exe
```

Por defecto Split-Path corta toda la ruta del directorio padre. Si se ejecuta con el modificador **–leaf** se queda con la última parte de la ruta.

Comandos de administración

GET-PROCESS [Obtener un pantallazo de los procesos que se estan ejecutando **]**

```
PS C:\> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    467      30    41288     100724   254    16,50   2148 aswEngSrv
    481      27    42256      35056   171    74,29   3860 aswidsagent
    309      12    22868      23308    63           4236 audiodg
   5229     939   205184     252140   679    81,90   1428 AvastSvc
   1952      54    33136      33964   399   180,87   1404 AvastUI
```

GET-SERVICE [Obtener un listado de los servicios del equipo y su estado **]**

```
PS C:\> get-service

Status   Name               DisplayName
------   ----               -----------
Stopped  AdobeFlashPlaye... Adobe Flash Player Update Service
Stopped  AeLookupSvc        Experiencia con aplicaciones
Stopped  ALG                Servicio de puerta de enlace de niv...
Stopped  AMD External Ev... AMD External Events Utility
```

STOP-PROCESS [Detiene uno o más procesos en ejecución **]**

```
PS C:\> Stop-Process 3948
```

Como parámetro se debe especificar el ID del proceso que queremos detener, en este ejemplo es el 3948.