
ADMINISTRACIÓN DE SISTEMAS OPERATIVOS



UNIVERSIDADE DA CORUÑA



GRUPO: JFF

Brais Vázquez Villa

Xián García Ferreiro

Julio Fernández Fernández

Tomé Maseda Dorado

Sergio Rega Domato

Lucía Pérez Rego

POWERSHELL: ¿QUE ES?

PowerShell, llamado Windows PowerShell originariamente, es una interfaz de línea de comandos o CLI (Command-Line Interface) que tiene la posibilidad de ejecutar Scripts (unión de comandos) y que facilita la configuración, administración y automatización de tareas multiplataforma, además dispone de un lenguaje de scripting. Powershell tiene una salida basada en objetos por lo que acepta y devuelve objetos de .NET ya que está basado en .NET CLR (Common Language Runtime), esto es ideal para la automatización de procesos.

La principal diferencia de PowerShell respecto a otras interfaces de línea de comandos es que PowerShell puede tratar con objetos, en comparación con otros CLI que solo devuelven texto. Estos comandos, llamados cmdlet, devuelven una instancia de un objeto que da lugar a una información de salida mucho más completa que los demás intérpretes de comandos, además los otros CLI necesitan de este para poder obtener la información de salida, mientras que un cmdlet obtiene la información de salida por sí mismo.

PowerShell permite un conjunto de comandos extensible a diferencia de otros CLI que su conjunto de comandos está integrado, se pueden crear nuevos cmdlets a partir de scripts o código compilado, también si necesitamos nuestro propio cmdlet podemos crearlo. Este CLI trabaja con alias de comandos para facilitar su manejo, un alias es un nombre que designa a un comando, así si estamos acostumbrados a trabajar con otros CLI, podemos asignar alias a los comandos de PowerShell y seguir usando los mismos comandos. Por ejemplo, podemos asignar el alias listar-directorios al comando dir que nos mostrará una lista de directorios, cuando ejecutemos uno u otro nombre, siempre mostrará el mismo resultado.

El código fuente de PowerShell se liberó el 15 de agosto de 2016 y Microsoft, que es su desarrollador, lo publicó en GitHub, que ahora le pertenece al haberlo comprado un par de años más tarde. La versión inicial de PowerShell fue lanzada en noviembre de 2006 para Windows XP Service Pack 2, Windows Vista y Windows Server 2003 Service Pack 1, la versión actual, después de algunos cambios de nombre intermedios, es la PowerShell 7.0 que está construida sobre .NET Core 3.1.

PLATAFORMAS SOPORTADAS

Las plataformas soportadas por Powershell son las siguientes:

- Windows 8.1 y 10
- Windows Server 2012 R2, 2016, 2019
- Ubuntu 16.04, 18.04, 20.04 y Ubuntu 19.10 y 20.10 mediante paquete Snap
- Debian 9 y 10
- CentOS 7 y 8
- Red Hat Enterprise Linux 7 y 8
- Fedora 31 y superiores
- Alpine 3.10, 3.11 y superiores
- MacOS 10.13+
- Arco
- Raspbian

- Kali
- Appliance (funciona en varias plataformas Linux)

SEGURIDAD

Dentro de lo referido a la seguridad, el cmdlet Get-ExecutionPolicy simplemente le indica en cuál de las cuatro políticas de ejecución (políticas que determinan cuáles son los scripts de Windows PowerShell, en su caso, se ejecutarán en su ordenador) se encuentra actualmente en la fuerza.

Las directivas de ejecución de Windows PowerShell se incluyen los siguientes:

- **Restricted (restringido):** No hay secuencias de comandos se pueden ejecutar. Windows PowerShell sólo se puede utilizar en modo interactivo.
- **AllSigned (todos firmados):** Sólo guiones firmados por un editor de confianza se pueden ejecutar.
- **RemoteSigned (firmados remotamente):** Guiones descargados deben ser firmados por un editor de confianza antes de que se pueden ejecutar.
- **Unrestricted (sin restricción):** No hay restricciones de libre disposición; todos los scripts de Windows PowerShell se pueden ejecutar.

Mostramos a continuación un ejemplo de cómo se haría un cambio del tipo de la política de seguridad.

```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/powershell

PS C:\WINDOWS\system32> Get-ExecutionPolicy
Restricted
PS C:\WINDOWS\system32> Set-ExecutionPolicy Unrestricted

Cambio de directiva de ejecución
La directiva de ejecución te ayuda a protegerte de scripts en los que no confías. Si cambias dicha directiva, podrías exponerte a los riesgos de seguridad descritos en el tema de la Ayuda about_Execution_Policies en https://go.microsoft.com/fwlink/?LinkId=135170. ¿Quieres cambiar la directiva de ejecución?
[S] Sí [O] Sí a todo [N] No [T] No a todo [U] Suspender [?] Ayuda (el valor predeterminado es "N"): O
PS C:\WINDOWS\system32> Set-ExecutionPolicy RemoteSigned

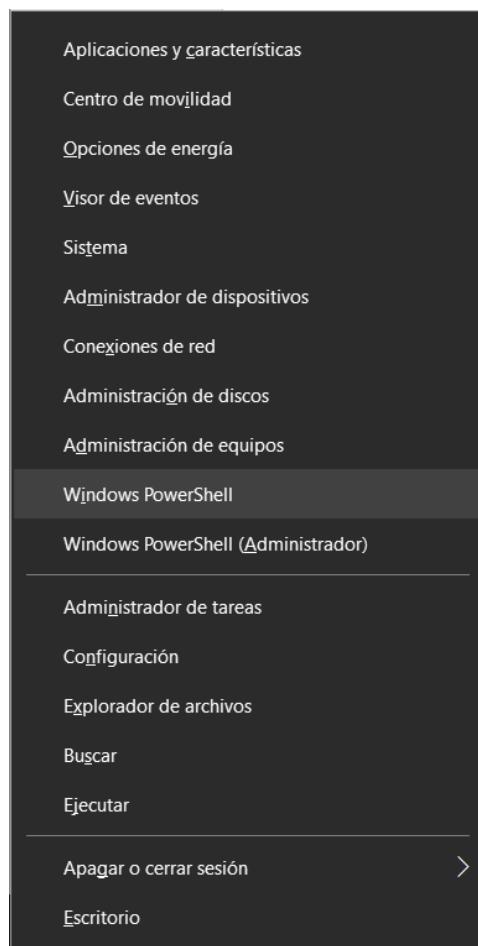
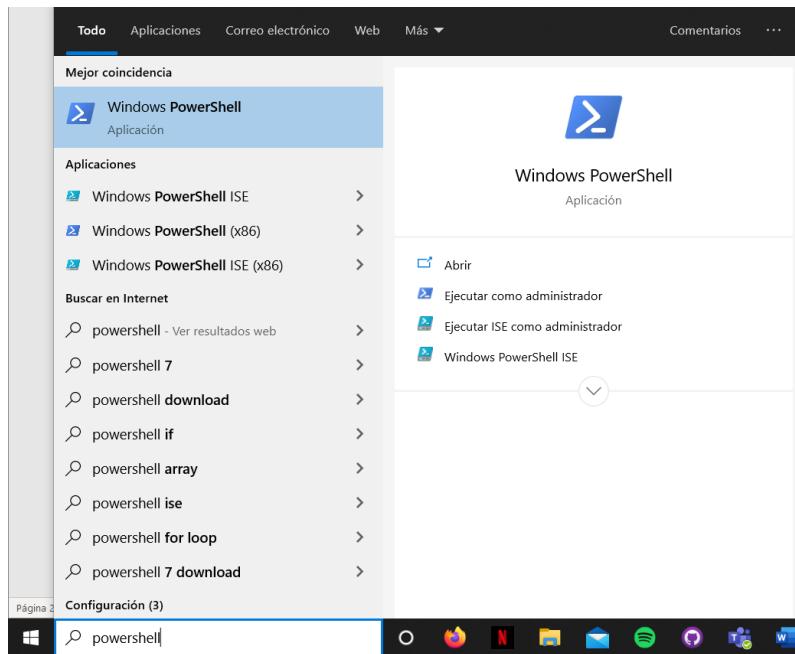
Cambio de directiva de ejecución
La directiva de ejecución te ayuda a protegerte de scripts en los que no confías. Si cambias dicha directiva, podrías exponerte a los riesgos de seguridad descritos en el tema de la Ayuda about_Execution_Policies en https://go.microsoft.com/fwlink/?LinkId=135170. ¿Quieres cambiar la directiva de ejecución?
[S] Sí [O] Sí a todo [N] No [T] No a todo [U] Suspender [?] Ayuda (el valor predeterminado es "N"): N
PS C:\WINDOWS\system32> Set-ExecutionPolicy Restricted

Cambio de directiva de ejecución
La directiva de ejecución te ayuda a protegerte de scripts en los que no confías. Si cambias dicha directiva, podrías exponerte a los riesgos de seguridad descritos en el tema de la Ayuda about_Execution_Policies en https://go.microsoft.com/fwlink/?LinkId=135170. ¿Quieres cambiar la directiva de ejecución?
[S] Sí [O] Sí a todo [N] No [T] No a todo [U] Suspender [?] Ayuda (el valor predeterminado es "N"): O
PS C:\WINDOWS\system32>

```

COMO INICIAR POWERSHELL

La consola de Windows Powershell es accesible a través de la búsqueda de Windows:



Se puede acceder a ella también a través de las teclas rápidas “Windows+X” o usando “Windows+R” y escribiendo powershell en la ventana de comandos Ejecutar

En otros sistemas operativos como mac o distribuciones linux podemos descargar powershell a través del repositorio de github:
<https://github.com/PowerShell/PowerShell>

SISTEMA DE AYUDA

Get-Command se usa para buscar cmdlets que son comandos que realizan una acción y normalmente devuelven un objeto Microsoft .NET. Son comandos únicos que participan en la semántica de canalización de PowerShell.

```
PS C:\Users\super> Get-service | Get-member

TypeName: System.ServiceProcess.ServiceController

Name          MemberType  Definition
----          -----      -----
Name          AliasProperty  Name = ServiceName
RequiredServices AliasProperty  RequiredServices = ServicesDependedOn
Disposed       Event       System.EventHandler Disposed(System.Object, System.EventArgs)
Close          Method      void Close()
Continue      Method      void Continue()
CreateObjRef   Method      System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose        Method      void Dispose(), void IDisposable.Dispose()
Equals         Method      bool Equals(System.Object obj)
ExecuteCommand Method      void ExecuteCommand(int command)
GetHashCode    Method      int GetHashCode()
GetLifetimeService Method      System.Object GetLifetimeService()
GetType        Method      type GetType()
InitializeLifetimeService Method      System.Object InitializeLifetimeService()
Pause          Method      void Pause()

PS C:\Users\super> get-command get-volume

 CommandType     Name           Version   Source
-----      ----           -----   -----
 Function      Get-Volume      2.0.0.0  Storage
```

Get-Help nos sirve para obtener información sobre un cmdlet determinado. Este cmdlet dará información sobre otros cmdlets, sobre todo la sintaxis a usar.

```
PS C:\Users\super> get-help get-process

NOMBRE
    Get-Process

SINTAXIS
    Get-Process [[-Name] <string[]>] [<CommonParameters>]
    Get-Process [[-Name] <string[]>] [<CommonParameters>]
    Get-Process [<CommonParameters>]
    Get-Process [<CommonParameters>]
    Get-Process [<CommonParameters>]

ALIAS
    gps
    ps

NOTAS
    Get-Help no encuentra los archivos de Ayuda para este cmdlet en el equipo. Mostrará solo una parte de la Ayuda.
    -- Para descargar e instalar los archivos de Ayuda para el módulo que incluye este cmdlet, use Update-Help.
    -- Para ver en línea el tema de Ayuda de este cmdlet, escriba "Get-Help Get-Process -Online" o
    vaya a https://go.microsoft.com/fwlink/?LinkID=113324.
```

Get-Process sirve para mostrar una lista completa de procesos que se están ejecutando en la computadora y si queremos saber los procesos que pertenecen a una aplicación específica le añadimos un argumento.

Ejemplo: get-process winword obtendrá todos los procesos que Word esté ejecutando.

```
PS C:\Users\super> get-process winword

Handles  NPM(K)    PM(K)    WS(K)    CPU(s)   Id  SI ProcessName
-----  -----    -----    -----    -----   --
1844      77    158472    283424   152,41  4136  5 WINWORD
```

Get-Member nos proporciona información sobre los miembros, las propiedades y métodos de un objeto. Ejemplo de uso con get-service:

```
PS C:\Users\super> Get-service | Get-member

TypeName: System.ServiceProcess.ServiceController

Name          MemberType      Definition
----          -----          -----
Name          AliasProperty  Name = ServiceName
RequiredServices AliasProperty RequiredServices = ServicesDependedOn
Disposed       Event          System.EventHandler Disposed(System.Object, System.EventArgs)
Close         Method          void Close()
Continue      Method          void Continue()
CreateObjRef   Method          System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose        Method          void Dispose(), void IDisposable.Dispose()
Equals         Method          bool Equals(System.Object obj)
ExecuteCommand Method          void ExecuteCommand(int command)
GetHashCode    Method          int GetHashCode()
GetLifetimeService Method          System.Object GetLifetimeService()
GetType        Method          type GetType()
InitializeLifetimeService Method          System.Object InitializeLifetimeService()
Pause          Method          void Pause()
Refresh        Method          void Refresh()
Start          Method          void Start(), void Start(string[] args)
Stop           Method          void Stop()
WaitForStatus  Method          void WaitForStatus(System.ServiceProcess.ServiceControllerStatus desiredStat..
CanPauseAndContinue Property     bool CanPauseAndContinue {get;}
CanShutdown    Property     bool CanShutdown {get;}
CanStop        Property     bool CanStop {get;}
Container      Property     System.ComponentModel.IContainer Container {get;}
DependentServices Property     System.ServiceProcess.ServiceController[] DependentServices {get;}
DisplayName    Property     string DisplayName {get;set;}
MachineName    Property     string MachineName {get;set;}
ServiceHandle  Property     System.Runtime.InteropServices.SafeHandle ServiceHandle {get;}
ServiceName    Property     string ServiceName {get;set;}
ServicesDependedOn Property     System.ServiceProcess.ServiceController[] ServicesDependedOn {get;}
ServiceType    Property     System.ServiceProcess.ServiceType ServiceType {get;}
Site           Property     System.ComponentModel.ISite Site {get;set;}
StartType      Property     System.ServiceProcess.ServiceStartMode StartType {get;}
Status         Property     System.ServiceProcess.ServiceControllerStatus Status {get;}
ToString       ScriptMethod System.Object ToString();
```

El comando Update-Help actualizará los temas de ayuda (requiere acceso a internet).

ESTRUCTURA DE COMANDOS

Los comandos de Powershell están estructurados de la siguiente manera: Un verbo y un nombre en singular separados por un guión (verbo-nombre). Ejemplo: Get-command

El verbo describe la acción a realizar sobre el nombre. En el ejemplo anterior recuperamos (Get) los comandos (Command)

En Powershell hay varios verbos genéricos como Get, Set, Add, Remove... que se combinan con nombres como Path, Variable, Item, Objetc, Computer...

Estos comandos pueden escribirse indistintamente en mayúsculas o minúsculas, por lo que el analizador sintáctico de PowerShell no es case sensitive.

POWERSHELL: ¿PARA QUE SIRVE?

PowerShell sirve para facilitar a los administradores de sistemas tareas de automatización, administración y configuración de sistemas Windows, aunque también sirve para otros programas de Microsoft como SQL Server, Exchange o IIS. Con PowerShell puedes declarar variables de todo tipo, incluye operadores aritméticos y de asignación, se pueden crear vectores y compararlos, crear hashtables y un sinfín de opciones más que permitirán que la automatización de tareas sea aún más sencilla, es por esto por lo que es mayormente usada por los administradores de sistemas.

También puedes usar PowerShell en tu PC con Windows, tan solo tienes que escribir PowerShell en el recuadro de búsqueda de la barra de tareas de Windows, incluso puedes poner a PowerShell como tu consola por defecto para Windows 10, con esta CLI además podrás gestionar el sistema operativo o incluso gestionar unidades de discos, aunque esto también puedes hacerlo con Diskpart, dispone de comandos y utilidades dedicadas para la gestión de las unidades de tu PC.

Otra de las utilidades es realizar script para automatizar las tareas normales, no necesariamente tenemos que ser un administrador de sistemas o ejecutar Windows Server para realizar esto. Por ejemplo, podemos usar PowerShell para conectar y realizar tareas administrativas o crear informes de Microsoft Office 365 o Azure. También puedes crear tus propios alias para hacer funcionar los comandos de PowerShell si estás acostumbrado a unos comandos concretos usados en cualquier otro CLI.

Pero PowerShell está más orientado a facilitar la administración, gestión y configuración de sistemas Windows o programas de Microsoft. Los administradores de sistemas pueden crear sus propios scripts para automatizar las tareas que desempeñen más habitualmente y siempre hay margen para ir añadiendo o mejorando las que ya disponen, para finalmente dejar un sistema perfectamente automatizado y vigilado muy de cerca por el administrador por si surgen fallos.

CARACTERÍSTICAS

Este shell de comandos modernos cuenta con varias características a resaltar:

- Lenguaje de scripting.
- Diseñado para administradores de sistemas.
- Entorno interactivo y shell básica.
- Basado en .NET (.NET es una plataforma de desarrollo para la creación de todas las aplicaciones: web, para dispositivos móviles, escritorio, juegos, IoT y mucho más. Se admite en Windows, Linux y macOS).
- Controla y automatiza el SO.
- Permite controlar también aplicaciones de Windows.
- Mejora de la antigua shell de Windows y elimina problemas antiguos.
- Incorpora conceptos y ventajas de distintos entornos.
- Realizar tareas de administración relacionadas con el registro, procesos, servicios, eventos, etc.
- Gestión de WMI (Windows Management Instrumentation).
- Diseño sencillo.

- Seguridad. Tiene sistemas que controlan la ejecución de scripts y así se evita la ejecución de scripts no deseados.
- Orientación a objetos. Aunque los cmdlets se escriben como texto, se comportan como objetos.
- Se pueden administrar remotamente.
- Los proveedores de Windows PowerShell permiten obtener acceso a almacenes de datos con la misma simplicidad con que se obtiene acceso al sistema de archivos.
- Permite realizar automatizaciones al tener el control del sistema operativo.
- Se puede ejecutar en cualquier SO.

CMDLET

El intérprete de línea de comandos (en inglés, command-line interpreter o CLI) de PowerShell proporciona al usuario acceso a las funciones internas del sistema operativo mediante entradas de teclado. Un cmdlet es un comando que se usa en el entorno de PowerShell.

Existen canalizaciones entre varios cmdlets, esto es, la salida de un segmento del cmdlet es la entrada de otro. Es uno de los conceptos más útiles porque permite relaciones de unos cmdlets con otros. Se produce con objetos y no con textos. Las operaciones que se pueden hacer con cmdlet son: seleccionar, agrupar, ordenar, contar, comparar, dar formato...

Podemos ver algunos comandos que son comunes a varias terminales como pueden ser: **cd**, **ls/dir**, **rm**, **cp**, **echo**, **cat**... Pero esto no ocurre para todos los comandos, vamos a ver a continuación algunos ejemplos de comandos “cotidianos” que son diferentes en PowerShell.

Get-Service

Comando que se utiliza para ver todos los procesos con estado RUNNING, STOPPED... Este comando es similar al comando *top* de Unix. Si queremos profundizar un poco más, podemos filtrar por estados los procesos con un comando similar:

- `Get-Service | Where-Object {$_._Status -eq 'Running'}` refleja sólo los comandos en estado RUNNING.

Status	Name	DisplayName
Stopped	AarSvc_baell	Agent Activation Runtime_baell
Running	AdobeAcMService	Adobe Acrobat Update Service
Stopped	AJRouter	Servicio de enrutador de AllJoyn
Stopped	ALG	Servicio de puerta de enlace de niv...
Stopped	AppInfoDsvc	Identificación de aplicación
Running	AppInfo	Información de aplicación
Stopped	AppPgmt	Administración de aplicaciones
Stopped	AppReadiness	Preparación de aplicaciones
Stopped	AppClient	Microsoft App-V Client
Running	AppX	Servicio de implementación de AppX ...
Stopped	AppxRedAccessM...	Servicio de acceso a AppAccessManager
Running	AudioEndpointBu...	Compilador de extremo de audio de W...
Running	Audiosrv	Audio de Windows
Stopped	autotimesvc	Hora de la red de telefonía móvil
Stopped	AxInstSV	Instalador de ActiveX (AxinstSV)
Stopped	BroadcastUserSer...	Servicio de difusión de difusión y G...
Stopped	BFE	Servicio Cifrado de Unidad BitLocker
Running	BFE	Motor de filtrado de base
Stopped	BITS	Servicio de transferencia inteligen...
Running	BluetoothUserSe...	Servicio de soporte técnico de usu...
Running	BrokerInfrastru...	Servicio de infraestructura de tare...
Running	BtHwVctpsvc	Servicio de interrupción de enlace de aud...
Running	BtHwVctpsvc	Servicio AVC
Running	bthserv	Servicio de compatibilidad con Blue...
Running	camsvc	Servicio Administrador de Funcional...
Stopped	CaptureService...	CaptureService_baell
Running	cddhsvc_baell	Servicio de usuario del portapapeles...
Running	CertPropSvc	Servicio de certificado de portapapeles...
Running	COPUserSvcs_baell	Servicio de propagación de plataforma d...
Running	COPUserSvcs_baell	Propagación de certificados
Running	CryptSvc	Servicio Hacer clic y ejecutar de M...
Running	ClickToRunsvc	Vmware Horizon Client
Stopped	client_service	Servicio de licencia de cliente (Cl...
Stopped	COMSysApp	Aplicación del sistema COM+
Stopped	ConsentDUserSv...	ConsentUX_baell
Running	CoreMessagingRe...	CoreMessaging
Stopped	CredentialInrol...	CredentialInrollmentManagerUserSvc_...
Running	CryptSvc	Servicios de cifrado
Stopped	DCCService	Área de trabajo sin conexión
Running	DcomLaunch	Iniciador de procesos de servidor DCOM
Stopped	defragsvc	Optimizar unidades

Tasklist

Este comando muestra una lista de los procesos que se encuentran actualmente en el equipo local o en un equipo remoto.

Nombre de imagen	PID	Nombre de sesión	Núm. de ses	Uso de memor
System Idle Process	0	Services	0	8 KB
System	4	Services	0	2.188 KB
Registry	140	Services	0	83.912 KB
smss.exe	496	Services	0	1.056 KB
csrss.exe	744	Services	0	5.528 KB
wininit.exe	844	Services	0	6.708 KB
csrss.exe	852	Console	1	13.948 KB
ServiceHost.exe	910	Services	0	3.024 KB
lsass.exe	936	Services	0	23.840 KB
winlogon.exe	1008	Console	1	11.920 KB
svchost.exe	1040	Services	0	27.336 KB
Fontdrvhost.exe	1068	Console	1	10.512 KB
Fontdrvhost.exe	1076	Services	0	3.380 KB
svchost.exe	1156	Services	0	17.924 KB
svchost.exe	1204	Services	0	8.412 KB
dwm.exe	1288	Console	1	89.532 KB
svchost.exe	1392	Services	0	8.192 KB
svchost.exe	1400	Services	0	11.672 KB
svchost.exe	1408	Services	0	11.004 KB
svchost.exe	1520	Services	0	15.984 KB
svchost.exe	1528	Services	0	10.076 KB
svchost.exe	1536	Services	0	13.408 KB
svchost.exe	1544	Services	0	10.184 KB
svchost.exe	1756	Services	0	10.176 KB
svchost.exe	1776	Services	0	6.384 KB
svchost.exe	1804	Services	0	7.116 KB
svchost.exe	1832	Services	0	16.320 KB
svchost.exe	1880	Services	0	8.220 KB
svchost.exe	1964	Services	0	5.936 KB
svchost.exe	2068	Services	0	8.284 KB
NVDisplay.Container.exe	2104	Services	0	16.024 KB
svchost.exe	2132	Services	0	7.016 KB
svchost.exe	2204	Services	0	7.712 KB
svchost.exe	2264	Services	0	11.740 KB
svchost.exe	2272	Services	0	5.672 KB
svchost.exe	2280	Services	0	7.480 KB
svchost.exe	2288	Services	0	11.940 KB
svchost.exe	2344	Services	0	6.760 KB
dasHost.exe	2392	Services	0	9.640 KB
Memory Compression	2464	Services	0	78.912 KB
svchost.exe	2512	Services	0	7.896 KB
svchost.exe	2536	Services	0	9.000 KB
svchost.exe	2564	Services	0	7.716 KB
svchost.exe	2600	Services	0	9.200 KB
NVDisplay.Container.exe	2820	Console	1	49.912 KB
svchost.exe	3036	Services	0	8.240 KB

Taskkill

Este comando mata a un proceso que se encuentre actualmente en ejecución. Podemos pasarle tanto el pid del proceso como el nombre del proceso. Usamos además varios parámetros:

- /PID: Especificamos el pid del proceso, el cual podemos obtener con el comando *tasklist* mencionado anteriormente.
- /F: Fuerza a acabar el proceso.
- /IM: Nombre de la imagen del proceso que queremos cerrar. Si ponemos un * podríamos cerrar todos los procesos.

```
PS C:\WINDOWS\system32> taskkill /F /PID 9576
Correcto: se terminó el proceso con PID 9576.
PS C:\WINDOWS\system32> taskkill /IM "Spotify.exe" /F
Correcto: se terminó el proceso "Spotify.exe" con PID 12504.
Correcto: se terminó el proceso "Spotify.exe" con PID 14056.
Correcto: se terminó el proceso "Spotify.exe" con PID 8888.
Correcto: se terminó el proceso "Spotify.exe" con PID 14932.
Correcto: se terminó el proceso "Spotify.exe" con PID 17956.
Correcto: se terminó el proceso "Spotify.exe" con PID 12116.
PS C:\WINDOWS\system32>
```

Gestión de usuarios y grupos

Ya que hemos visto como se llevan a cabo esta tarea en distintos sistemas operativos, queremos hacer referencia a como se realizaría en PowerShell, para de esta forma poder apreciar las diferencias entre ellos.

En las siguientes imágenes se aprecia como creamos un usuario y un grupo, añadimos el usuario al grupo y posteriormente eliminamos todo lo que hemos creado. Además, para apreciar que se ha creado correctamente, listamos los usuarios y los grupos.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\WINDOWS\system32> $pass=ConvertTo-SecureString "1234" -asplaintext -force
PS C:\WINDOWS\system32> New-LocalUser Antonio -Password $pass

Name      Enabled Description
----      --     -----
Antonio   True

PS C:\WINDOWS\system32> New-LocalGroup amarillo

Name      Description
----      -----
amarillo

PS C:\WINDOWS\system32> Add-LocalGroupMember -Member Antonio -Group amarillo
```

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> Get-LocalGroupMember Amarillo
ObjectClass Name          PrincipalSource
----- ----          -----
Usuario    DESKTOP-SFDA4LQ\Antonio Local

PS C:\WINDOWS\system32> Remove-LocalGroupMember -Group "amarillo" -Member "Antonio"
PS C:\WINDOWS\system32> Get-LocalGroupMember Amarillo
PS C:\WINDOWS\system32> Remove-LocalGroup -Name amarillo
PS C:\WINDOWS\system32> Get-LocalGroupMember Amarillo
Get-LocalGroupMember : No se encontró el grupo Amarillo.
En linea: 1 Carácter: 1
+ Get-LocalGroupMember Amarillo
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (Amarillo:String) [Get-LocalGroupMember], GroupNotFoundException
+ FullyQualifiedErrorId : GroupNotFound,Microsoft.PowerShell.Commands.GetLocalGroupMemberCommand

PS C:\WINDOWS\system32> Remove-LocalUser -Name "Antonio"
PS C:\WINDOWS\system32> Get-LocalUser

Name      Enabled Description
----      --     -----
Administrador False  Cuenta integrada para la administración del equipo o dominio
DefaultAccount False  Cuenta de usuario administrada por el sistema.
defaultuser0 False
Invitado    False  Cuenta integrada para el acceso como invitado al equipo o dominio
Sergio     True
WDAGUtilityAccount False Una cuenta de usuario que el sistema administra y usa para escenarios de Protección de ap...

PS C:\WINDOWS\system32>
```

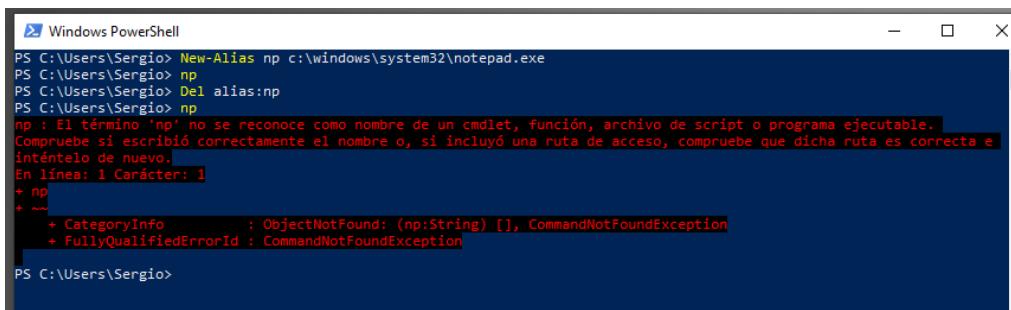
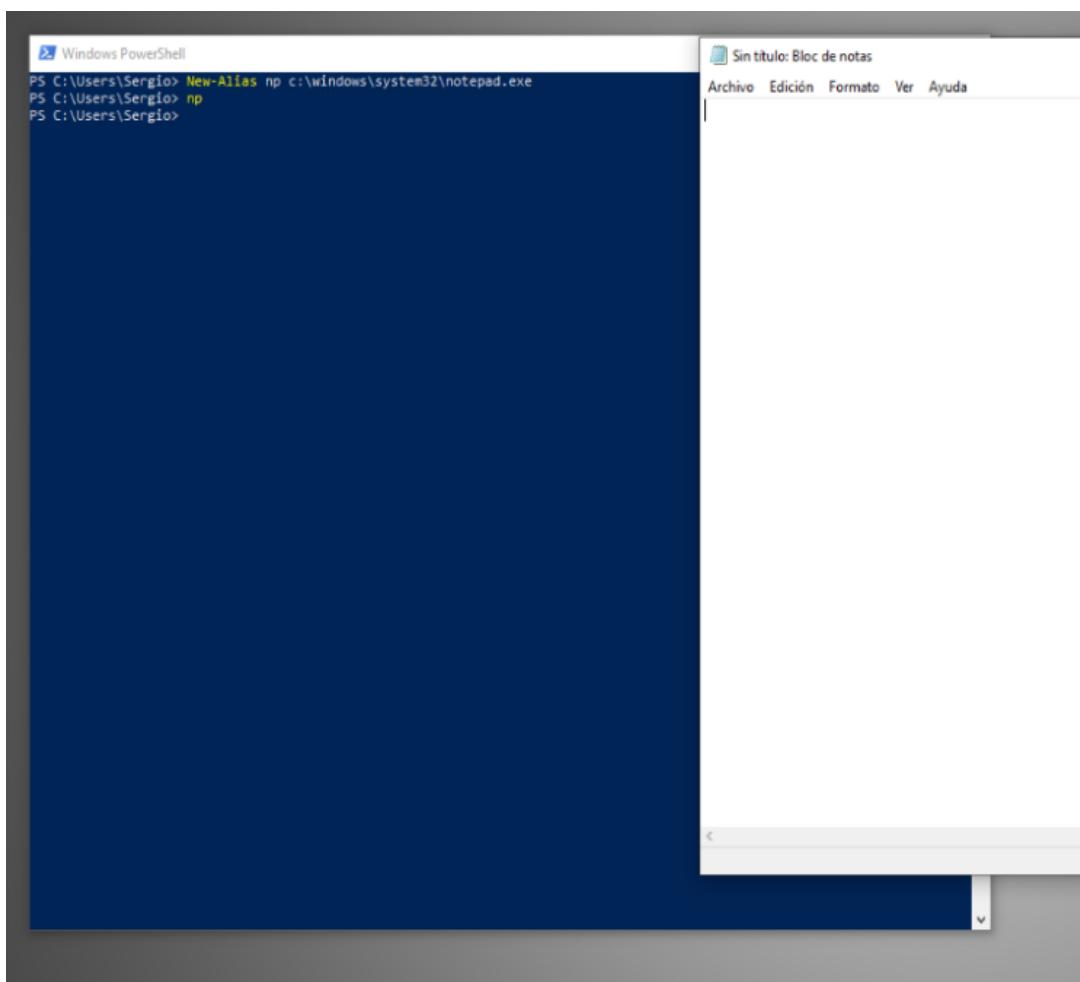
Alias

El usuario puede llamar al comando como él quiera. Esto se proporciona con el fin de permitir a los usuarios nuevos, la capacidad de interactuar rápidamente con el sistema. Existen dos tipos de alias principales:

- Alias predefinidos: Nombres predefinidos alternativos para Windows, Unix...
- Altas definidos por el usuario: Nombres personalizados alternativos creados por el usuario.

Para obtener una lista de todos los alias predefinidos existentes en PowerShell, usemos el siguiente cmdlet: **Get-Alias**.

A continuación, mostramos un ejemplo de cómo crear y eliminar un alias:



Además, los alias pueden importarse y exportarse, el propósito de esto es hacer que los alias definidos por el usuario estén disponibles en múltiples máquinas. Esto se puede conseguir de varias maneras:

La primera de ellas es válida en los casos en los que queramos exportar a una máquina que no tiene los mismos alias que la máquina origen.

- Ejecutamos en la máquina origen: **Export-Alias –Path <fichero.txt>**. Con este comando exportamos todos los alias al documento.
- Ejecutamos en la máquina destino: **Import-Alias –Path <fichero.txt>**. En este caso importamos el contenido de los alias al PowerShell.

En caso de querer exportar tan sólo un alias, haremos lo siguiente:

- Ejecutamos en la máquina origen: **Export-Alias –Path <fichero.txt> -Name <nombre_alias>**. Gracias a esto conseguimos exportar tan sólo ese alias.
- Ejecutamos en la máquina destino: **Import-Alias –Path <fichero.txt>**.

El fichero.txt se genera automáticamente cuando hacemos el export y podemos darle el nombre que queramos.

Si hacemos estos pasos y cerramos PowerShell, nos daremos cuenta de que al volver a abrirlo habremos perdido esta configuración. Para hacer que los alias exportados se guarden permanentemente tendremos que crear perfiles. Los perfiles permiten personalizar el entorno de PowerShell en tiempo de lanzamiento. Por tanto, para crear un nuevo perfil, debemos hacer lo siguiente:

- Lo primero es comprobar como tenemos nuestra política de ejecución, esto lo haremos con **Get-ExecutionPolicy**.
- Posteriormente, cambiamos nuestra política a Unrestricted.
- A continuación, podemos comprobar si hay un perfil creado mediante **test-path \$Profile**. Si el resultado es false, significa que no existe el perfil por lo que se puede crear. En caso de que sea true, el perfil ya existe y, por tanto, hay que tener cuidado ya que, si creamos uno nuevo, se borrará el existente.
- Ejecutamos el comando **New-Item –Path \$Profile –ItemType file –Force**.
- Por último, hacemos **notepad \$Profile**, y escribiremos en él la sentencia de creación del alias que queramos que sea permanente.

Una vez hecho esto, podemos cerrar y abrir de nuevo el PowerShell y comprobaremos que si ejecutamos ese alias que hemos creado, funcionará correctamente.

SCRIPTS

En PowerShell se puede escribir, ejecutar y probar scripts de maneras que no están disponibles en la consola de Windows de PowerShell. El entorno de scripting integrado se utiliza para crear, ejecutar y depurar comandos o scripts, en definitiva, es una mejora del símbolo del sistema.

Es un requisito fundamental que el PowerShell tenga permisos totales sobre el servidor para su utilización, por tanto, al abrirlo será necesario escoger la opción de “ejecutar como administrador”.

Por último, es importante destacar que dichos ficheros deben tener la extensión ‘.ps1’.

```
param
(
    [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
    [String[]]
    [AllowEmptyString()]
    $User,$Pass
)

begin
{
    Write-Host "Probando un script de login"
    $usercorrecto="Antonio"
    $passcorrecto="ASO"
}

process
{
    if($User -eq $usercorrecto -and $Pass -eq $passcorrecto)
    {
        $ok=1
    }
    else
    {
        $ok=0
    }
}
end
{
    if($ok)
    {
        Write-Host "User correcto"
    }
    else
    {
        Write-Host "User no correcto"
    }
    Write-Host "Fin de login"
}
```



```
PS C:\Users\Sergio\Desktop> .\holamundo.ps1
Hola Mundo!
PS C:\Users\Sergio\Desktop> .\login.ps1 Antonio ASO
Probando un script de login
User correcto
Fin de login
PS C:\Users\Sergio\Desktop>
```

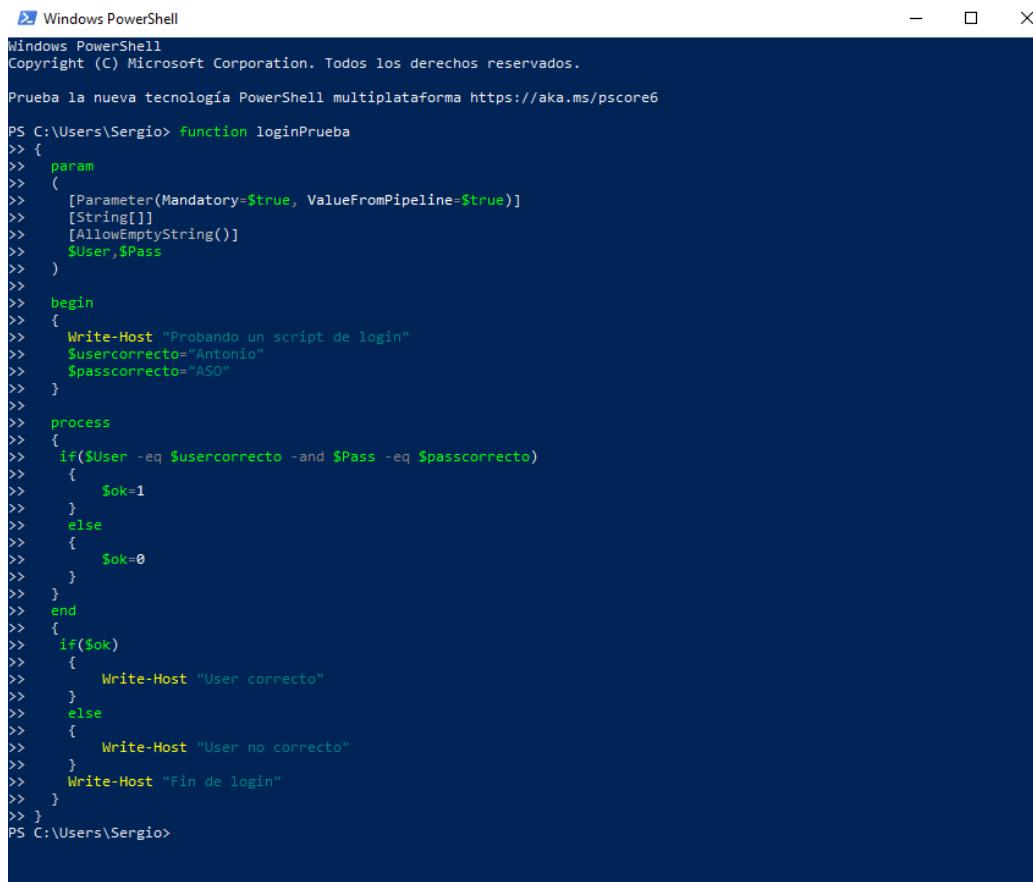
FUNCIONES

Las funciones se declaran con la palabra clave “function”, seguida del nombre de dicha función y unas llaves de apertura y cierre. Se podrá ejecutar una función invocando su nombre desde línea de comandos.

A las funciones no se les asignarán valores de forma estática, sino que usaremos parámetros y variables. Es un buen uso el asignar como nombre del parámetro, los cmdlets predeterminados siempre que sea posible.

En PowerShell existen funciones avanzadas. La diferencia entre las funciones estándar y las avanzadas es que esta última tiene una serie de parámetros comunes que se agregan automáticamente a la función. Algunos de estos parámetros son verbose y debug.

A continuación, veremos un ejemplo de una función en PowerShell y de su ejecución.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\Sergio> function loginPrueba
>> {
>>     param
>>     (
>>         [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
>>         [String[]]
>>         [AllowEmptyString()]
>>         $User,$Pass
>>     )
>>
>>     begin
>>     {
>>         Write-Host "Probando un script de login"
>>         $usercorrecto="Antonio"
>>         $passcorrecto="ASO"
>>     }
>>
>>     process
>>     {
>>         if($User -eq $usercorrecto -and $Pass -eq $passcorrecto)
>>         {
>>             $ok=1
>>         }
>>         else
>>         {
>>             $ok=0
>>         }
>>     end
>>     {
>>         if($ok)
>>         {
>>             Write-Host "User correcto"
>>         }
>>         else
>>         {
>>             Write-Host "User no correcto"
>>         }
>>         Write-Host "Fin de login"
>>     }
>> }
PS C:\Users\Sergio>
```



```
Windows PowerShell
PS C:\Users\Sergio> loginPrueba Antonio ASO
Probando un script de login
User correcto
Fin de login
PS C:\Users\Sergio> loginPrueba Anton ASO
Probando un script de login
User no correcto
Fin de login
PS C:\Users\Sergio>
```

VERSIONES DE POWERSHELL

¿Cómo puedo comprobar que versión de powershell tengo instalada?

Mostrando la variable `$PSVersionTable`.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\Usuario> $PSVersionTable

Name                Value
----               -----
PSVersion          5.1.19041.906
PSEdition          Desktop
PSCompatibleVersions {1.0, 2.0, 3.0, 4.0...}
BuildVersion       10.0.19041.906
CLRVersion         4.0.30319.42000
WSManStackVersion  3.0
PSRemotingProtocolVersion 2.3
SerializationVersion 1.1.0.1
```

PowerShell 1.0 (Noviembre 2006)

Esta es la versión inicial de PowerShell, fue lanzada para Windows XP SP2, Windows Server 2003 SP1 y Windows Vista. Es un componente opcional de Windows Server 2008.

Inicialmente PowerShell incluye:

- Más de 130 cmdlets para realizar tareas comunes de administración de sistema.
- Herramientas de línea de comandos que están diseñados para ser fáciles de aprender y fácil de usar.
- Compatibilidad con lenguajes de secuencias de comandos existentes en ese momento, las herramientas de línea de comandos existentes y varias versiones de Windows.
- Características que permiten a los usuarios navegar por los almacenes de datos, como el registro y los almacenes de certificados, como si se tratara de un sistema de archivos.
- Utilidades estándar para administrar datos de Windows en diferentes almacenes y formatos, como HTML y XML.
- Expresiones sofisticadas de análisis y manipulación de objetos .NET en la línea de comandos.
- Una interfaz extensible que permite a los proveedores de software independientes y desarrolladores empresariales generar cmdlets personalizados para satisfacer los requerimientos únicos de la administración de aplicaciones y del sistema.

PowerShell 2.0 (Octubre 2009)

PowerShell 2.0 está integrado con Windows 7 y Windows Server 2008 R2 y se lanza para Windows XP con Service Pack 3, Windows Server 2003 con Service Pack 2 y Windows Vista con Service Pack 1.

Esta versión incluye cambios en el lenguaje de scripts y en la API de hospedaje, además de incluir más de 240 nuevos cmdlets.

PowerShell 3.0 (Septiembre 2012)

PowerShell 3.0 está integrado con Windows 8 y con Windows Server 2012. Microsoft también ha puesto a disposición PowerShell 3.0 para Windows 7, Windows Server 2008 y Windows Server 2008 R2 con Service Pack 1.

PowerShell 3.0 es parte de un paquete más grande, Windows Management Framework 3.0 (WMF3), que también contiene el servicio WinRM para apoyar el remoting.

WinRM es equivalente a las llamadas RPC, se utiliza para la administración y ejecución de procesos sobre un equipo remoto, también permite recuperar información de estos.

PowerShell 4.0 (Octubre 2013)

PowerShell 4.0 está integrado con Windows 8.1 y con Windows Server 2012 R2. Microsoft también ha hecho que PowerShell 4.0 esté disponible para Windows 7 SP1, Windows Server 2008 R2 SP1 y Windows Server 2012.

PowerShell 5.0 (Febrero 2016)

Windows Management Framework (WMF) 5.0 RTM que incluye PowerShell 5.0 fue relanzado a la web el 24 de febrero de 2016, después de un lanzamiento inicial con un grave bug.

Las características clave incluyen cmdlets OneGet PowerShell para soportar la administración de paquetes basada en el repositorio de Chocolatey y la ampliación del soporte para la administración de comutadores a los comutadores de red de capa 2.

PowerShell 5.1 (Enero 2017)

WMF 5.1 cambia la detección automática de módulos para usar \$env:PSModulePath completamente.

Esto permite que un módulo creado por el usuario que define los comandos que proporciona PowerShell (por ejemplo, Get-ChildItem) se cargue automáticamente y reemplace correctamente el comando integrado.

Windows 10 agregó compatibilidad con secuencias de escape de VT100. PowerShell ignorará ciertas secuencias de escape con formato VT100 al calcular los anchos de tabla.

PowerShell Core 6.0 (Enero 2018)

PowerShell Core 6.0 fue anunciado por primera vez el 18 de agosto de 2016, cuando Microsoft dio a conocer PowerShell Core y su decisión de hacer que el producto sea multiplataforma, independiente de Windows, de código libre y abierto.

El cambio más significativo en esta versión de PowerShell es la expansión a las otras plataformas. Para los administradores de Windows, esta versión de PowerShell carece de nuevas características importantes.

PowerShell 7 (Marzo 2020)

PowerShell 7 es el producto de reemplazo para los productos PowerShell Core 6.x, así como para Windows PowerShell 5.1, que es la última versión de Windows PowerShell soportada. Para que PowerShell 7 sea un reemplazo viable para Windows PowerShell 5.1 debe tener casi paridad con Windows PowerShell en términos de compatibilidad con los módulos que se envían con Windows.

- Las nuevas características de PowerShell 7 incluyen:
 - Construido sobre .NET Core 3.1 (LTS)
 - `ForEach-Object -Parallel`
 - Envoltura de compatibilidad con Windows.
 - Notificación de nueva versión.
 - Nueva vista de error y cmdlet `Get-Error`.
 - Operadores de la cadena de oleoductos (`&&` y `||`)
 - ?: operador ternario (`a ? b : c`)
 - Asignación nula y operadores de coalescencia nula (`??=` y `??`)
 - Invocación de plataforma `Invoke-DscResource` (experimental).
 - `Out-GridView`, `-ShowWindow` y otros cmdlets GUI heredados están de vuelta en Windows.

RESUMEN DE LAS COMPATIBILIDADES POR VERSIÓN

PowerShell Version	Release Date	Default Windows Versions	Available Windows Versions
PowerShell 1.0	November 2006	Windows Server 2008 (*)	Windows XP SP2 Windows XP SP3 Windows Server 2003 SP1 Windows Server 2003 SP2 Windows Server 2003 R2 Windows Vista Windows Vista SP2
PowerShell 2.0	October 2009	Windows 7 Windows Server 2008 R2 (**)	Windows XP SP3 Windows Server 2003 SP2 Windows Vista SP1 Windows Vista SP2 Windows Server 2008 SP1 Windows Server 2008 SP2
PowerShell 3.0	September 2012	Windows 8 Windows Server 2012	Windows 7 SP1 Windows Server 2008 SP2 Windows Server 2008 R2 SP1
PowerShell 4.0	October 2013	Windows 8.1 Windows Server 2012 R2	Windows 7 SP1 Windows Server 2008 R2 SP1 Windows Server 2012
PowerShell 5.0	February 2016	Windows 10	Windows 7 SP1 Windows 8.1 Windows Server 2012 Windows Server 2012 R2
PowerShell 5.1	January 2017	Windows 10 Anniversary Update Windows Server 2016	Windows 7 SP1 Windows 8.1 Windows Server 2008 R2 SP1 Windows Server 2012 Windows Server 2012 R2
PowerShell Core 6	January 2018	N/A	Windows 7 SP1 Windows 8.1 Windows Server 2008 R2 SP1 Windows Server 2012 Windows Server 2012 R2
PowerShell 7	March 2020	N/A	Windows 7 SP1 Windows 8.1 Windows Server 2008 R2 SP1 Windows Server 2012 Windows Server 2012 R2

CURIOSIDADES SOBRE LAS VERSIONES

¿PowerShell 2.0 por defecto?

Windows 10 instala por defecto PowerShell para todos los usuarios, concretamente la versión 5.0 de esta consola, la última versión hasta la fecha. Sin embargo, además de instalar esta versión, el sistema operativo de Microsoft también deja habilitadas otras versiones de la misma, como PowerShell 2.0, versión ya obsoleta de esta consola.

Como ocurre con cualquier software, no se recomienda bajo ningún concepto utilizar o tener instalado software sin soporte ya que cualquier fallo de seguridad puede poner en peligro nuestro sistema.

Para comprobar si tenemos habilitada la consola PowerShell 2.0 en Windows 10, tan solo debemos abrir una ventana de esta consola con permisos de administrador y ejecutar el siguiente comando:

```
Get-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2

Administrator: Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Windows\system32> Get-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2

FeatureName      : MicrosoftWindowsPowerShellV2
DisplayName     : Motor de Windows PowerShell 2.0
Description     : Agrega o quita el motor de Windows PowerShell 2.0
RestartRequired : Possible
State          : Enabled
CustomProperties :
```



```
PS C:\Windows\system32>
```

Si el resultado es el mismo que podemos ver en la captura anterior, entonces tenemos esta versión habilitada, y podemos estar en peligro dado que mientras que **PowerShell 5.0 cuenta con protección contra malware**, la versión 2.0 no lo tiene, y los piratas informáticos fácilmente pueden hacer una llamada al motor de la versión 2.0 para llevar a cabo sus ataques.

¿Tiene Windows PowerShell futuro?

No, el futuro pertenece a PowerShell Core. Tal y como están las cosas ahora, Microsoft sólo proporcionará correcciones de características para PowerShell Core, Windows PowerShell solo recibirá correcciones de errores y actualizaciones de seguridad.

Aclaración:

Windows PowerShell es el nombre del producto hasta la versión PowerShell 5.1.

PowerShell Core es el nombre del producto desde la versión PowerShell Core 6.0 en adelante.

¿Debo seguir trabajando con Windows PowerShell o cambiar a PowerShell 7 ahora?

Esto depende de tu entorno, si todos sus sistemas tienen PowerShell 7 instalado, podrías considerar cambiarte ahora. Sin embargo, PowerShell 7 todavía tiene problemas de compatibilidad con algunos módulos. Por lo tanto, deberías asegurarte primero de que tienes todos los módulos necesarios disponibles.