INFORMÁTICA
TÉCNICA

Expert

Windows PowerShell

Administrar puestos cliente Windows

Prefacio de **Pierre Chesné**
Technology Solution Professional - Microsoft France

Archivos complementarios
para descarga

Julien MUSY

eni

Windows PowerShell

Administrar puestos cliente Windows

Prefacio de Pierre Chesné - Technology Solution Professional - Microsoft France

Este libro se dirige tanto a los **administradores de sistemas** Windows como a cualquier profesional que desee **administrar y gestionar un parque informático corporativo con Windows PowerShell**. Se basa en la **versión 5 de Windows PowerShell** y el autor presenta la administración de puestos cliente con **Windows 10**. Las operaciones de administración descritas son compatibles, a su vez, con **Windows 8.1**, y también en gran parte con **Windows 7**. Los abundantes ejemplos que se presentan le permiten desarrollar sus propios scripts para automatizar las numerosas tareas administrativas y mejorar la productividad y la eficacia de sus acciones cotidianas.

Tras revisar los cmdlets de PowerShell básicos para **manipular los recursos** (archivos, registros, certificados, etc.), el autor describe los muchos otros que permiten **administrar el sistema operativo** Windows (incluyendo sus parámetros y la seguridad), y también la **gestión de las aplicaciones** o incluso la de los **dispositivos** conectados al equipo.

Se dedica un capítulo a la **búsqueda y recopilación de información**: explotación de WMI para recuperar toda la información que puede utilizarse para construir estructuras condicionales en los scripts, o simplemente para auditar puestos de trabajo.

Como la versión 5 de Windows PowerShell integra un **administrador de paquetes** dentro de Windows, el autor desarrolla la instalación de aplicaciones a través de esta nueva funcionalidad, que facilita todavía más la preparación de los puestos de trabajo en el mundo profesional.

Se describe la administración remota de los puestos cliente con la **funcionalidad PowerShell Remoting**, que permite a los administradores enviar comandos, y también scripts, a uno o varios equipos remotos.

No se olvida una parte dedicada al scripting: el autor presenta el **desarrollo y la depuración de scripts en Windows PowerShell ISE**, y también el **despliegue de scripts a través de GPO**. Se proporcionan numerosos consejos y recomendaciones, como el uso de archivos XML para almacenar información relativa a los distintos entornos de ejecución de estos scripts.

Los scripts presentados en el libro pueden descargarse en esta página y pueden adaptarse fácilmente a su infraestructura.

Los capítulos del libro:
Prefacio – Prólogo – Presentación de Windows PowerShell – Las unidades de Windows PowerShell – Sistema y características – Configurar el sistema y las herramientas – Seguridad y protección – Buscar y recopilar información – Administrar el hardware y los dispositivos – Administrar aplicaciones – Administrador de paquetes – Administración remota de puestos de trabajo – Scripting – Casos prácticos y otras características – Conclusión – Anexos

---

Julien MUSY

Ingeniero de Sistemas, especializado en soluciones de infraestructura de Microsoft (Windows Server, SharePoint), **Julien MUSY** posee varias certificaciones de Microsoft (MCSA, MCITP et MCTS). Comparte regularmente su conocimiento y su pasión mediante artículos técnicos. Su experiencia y su destreza en la administración de sistemas Windows le han permitido escribir un libro eficaz para proporcionar a los lectores la información necesaria para la administración de los puestos cliente con Windows PowerShell.

Introducción

¡Cómo pasa el tiempo! Ya vamos por la quinta versión mayor de Windows PowerShell, incluido de manera nativa en Windows 10 y Windows Server 2016. Desde 2007, Windows PowerShell ha sabido imponerse como herramienta imprescindible para la administración de entornos de Microsoft.

En efecto, Windows PowerShell no es un simple «Shell»: es también un lenguaje de script orientado a objetos basado en el Framework .NET. Además de presentar innumerables comandos que permiten llevar a cabo la administración por línea de comandos, también permite la creación y la manipulación de los objetos propios de esta librería de clases, abriendo así el uso de Windows PowerShell a los desarrolladores.

Además de extender y simplificar su uso, se han integrado varias novedades funcionales importantes en Windows 10 que permiten, en lo sucesivo, controlar y administrar entornos Windows más fácilmente y de una manera mucho más potente. El administrador de paquetes, aparecido con la versión 5 de Windows PowerShell, permite asimismo instalar nuevos módulos de PowerShell, tales como Azure Resource Manager, por ejemplo, y también instalar aplicaciones a través de la descarga de paquetes desde los repositorios. La extensión DSC Windows PowerShell, menos conocida pero, no obstante, importante, ha aparecido con la versión 4 de Windows PowerShell y proporciona respuestas a las problemáticas de DevOps, de configuración de sistemas, al despliegue de plataformas y a la gestión de la conformidad.

¿PowerShell no solo para entornos Microsoft?

Si bien en la actualidad PowerShell está presente en todos los entornos On-Premise y Cloud de Microsoft, los cuales disponen de sus propios módulos, el alcance de PowerShell no se detiene aquí...

En efecto, los fabricantes de software y de hardware del ecosistema Microsoft proporcionan también sus propios módulos de **PowerShell** para facilitar sus despliegues. Muchas empresas han adoptado, por tanto, Windows PowerShell, ¡permitiendo así una unificación y una apertura mucho mayores de la administración por línea de comandos!

La estrategia de Microsoft se dirige hacia su apertura a otras plataformas que contienen el «.NET Core» en Linux y macOS. En

2014, Sataya Nadella (CEO de Microsoft) anunció que «**Microsoft Love Linux!**» y, desde el 18 de agosto de 2016, PowerShell ha pasado a ser de código abierto (licencia MIT). El padre de PowerShell, Jeffrey Snover, dijo el día del anuncio: «**Los usuarios actuales de PowerShell que necesiten administrar sus dominios heterogéneos lo quieren. Las personas que construyan herramientas de administración como nosotros con Operations Management Suite (OMS) lo quieren. Los que quieran trabajar con un estándar basado e un conjunto específico de herramientas lo quieren…»**. Y, en la actualidad, podemos ver cómo PowerShell no solo está presente en los propios sistemas On-Premise o Cloud de Microsoft; sino que también está presente en entornos de terceros, y se ha convertido en una herramienta interplataforma.

Hoy en día, ya no es necesario demostrar el interés de utilizar Windows PowerShell. ¡Con su libro, Julien Musy acompaña eficazmente a los profesionales de la informática en este camino hacia la automatización y la optimización! Ayuda a los administradores del sistema a adquirir los reflejos adecuados en el dominio del despliegue y de la configuración de los puestos cliente de Windows, liberándolos de tareas repetitivas sin ningún valor añadido.

Pierre Chesné, Technology Solution Professional para Microsoft Francia

Introducción

Este libro pertenece a la colección Expert IT de Ediciones ENI, y trata en particular de la administración de puestos de trabajo Windows mediante Windows PowerShell. Esta segunda edición del libro cubre todas las versiones del sistema operativo, desde Windows 7 hasta Windows 10, que incorporan la versión 5 de PowerShell. Esta nueva versión mayor de PowerShell, que puede instalarse también en las versiones anteriores de Windows, extiende los límites de la administración del sistema operativo editado por Microsoft.

Windows PowerShell es un intérprete de comandos interactivo, que se utiliza para administrar no solo puestos de trabajo, sino también de servidores y de varias aplicaciones editadas por Microsoft. Esta herramienta está presente por defecto en los sistemas operativos Windows, desde Windows 7, en versión 2.0. Actualmente, se incluye la versión 5 con Windows 10, incorporando novedades funcionales. En la empresa, Windows PowerShell es, a día de hoy, una herramienta prácticamente imprescindible para la administración de un parque informático, ya esté compuesto por una decena de puestos o por varias decenas de miles de puestos.

Windows PowerShell es una herramienta potente, tanto por línea de comandos, que puede volverse bastante compleja, como para los scripts. De manera similar a los scripts batch que pueden interpretarse desde el símbolo del sistema (cmd.exe), también es posible construir scripts PowerShell.

Así, para un administrador de sistemas que quiera actualizar cierta configuración en un conjunto de puestos de trabajo de un gran grupo internacional, el desarrollo y el despliegue de un script que automatice el conjunto de operaciones es algo muy recomendable. ¡Todo esto sin tener que entrar en cada puesto de trabajo de los empleados!

Si bien las ventajas de Windows PowerShell son numerosas, este intérprete de comandos puede llegar a desanimar: aprender un nuevo lenguaje no es sencillo cuando falta tiempo disponible para formarse o seguir una formación. Y como todo lenguaje informático o idioma extranjero, hay que practicarlo de manera regular para no olvidar lo aprendido.

Este libro muestra cómo administrar el conjunto de equipos de un parque informático únicamente con Windows PowerShell. Se

describen las tareas de administración más comunes y se amplían con numerosos ejemplos de comandos PowerShell. Esto le permitirá hacerse una idea rápidamente para aprender las tareas de administración correspondientes.

Para desarrollar scripts, Microsoft pone a su disposición una herramienta: Windows PowerShell ISE (*Integrated Scripting Environment*). Esta herramienta facilita enormemente la creación y el desarrollo de scripts. Se incluye de manera nativa con Windows, aunque existen también editores de scripts y herramientas de terceros compatibles con el lenguaje PowerShell.

¿A quién se dirige este libro?

Este libro se dirige principalmente a todos los administradores de puestos de trabajo de empresas (pymes o grandes grupos internacionales). Le ayudará en las tareas administrativas sobre los puestos de trabajo, tanto si sus conocimientos sobre Windows PowerShell son limitados como si no. Este libro le ofrecerá soluciones prácticas que bastará con adaptar a su entorno de trabajo y a la configuración de sus puestos de trabajo. Para la gran mayoría de los comandos que verá, no olvide que es necesario disponer de permisos de administrador y ejecutar Windows PowerShell como administrador.

¿Cómo se estructura este libro?

Este libro está compuesto de seis grandes temas correspondientes a la administración de los puestos cliente en Windows y los medios para llevarla a cabo. Antes de abordar la administración de Windows, se realiza una presentación de Windows PowerShell y los cmdlets básicos.

El primer gran tema corresponde, como es natural, a la administración de Windows. El capítulo Sistema y características está dedicado a la gestión y la administración global del sistema operativo, incluyendo los recursos y su protección. Un segundo capítulo, titulado Configurar el sistema y las herramientas, aborda la gestión de tareas planificadas, de los registros de eventos, y también los parámetros regionales y de idioma. Para terminar, el capítulo Seguridad y protección aborda el tema de la seguridad en Windows con la gestión del Firewall de Windows y de Windows Defender. Se ofrece también una visión general de las cuentas de usuario y los grupos locales y de la compartición de carpetas con Windows PowerShell.

El segundo tema se corresponde con la recuperación de información en un puesto de trabajo. Este tema es necesario para encontrar información y construir estructuras condicionales (If, Else, ElseIf), y también para llevar a cabo una auditoría de los equipos. Esta información se recupera mediante consultas WMI, que se describen en el capítulo Buscar y recopilar información.

El siguiente tema presenta cómo Administrar el hardware y los dispositivos. Configurar las tarjetas de red, administrar los discos e incluso configurar las impresoras en Windows son asuntos que dejarán de ser un secreto para usted con Windows PowerShell.

Otro tema imprescindible: la administración de aplicaciones con Windows PowerShell. Abordada en los capítulos Administrar aplicaciones y Administrador de paquetes, será capaz de instalar y auditar cualquier tipo de aplicación y de actualización, sin tener que interactuar con los usuarios.

Se realiza una presentación y demostración de administración remota con Windows PowerShell: PowerShell Remoting. Este último permite administrar varios puestos de trabajo remotos a través del intérprete de comandos. Todo ello se explica en el capítulo Administración remota de puestos de trabajo.

El último tema de este libro aborda un asunto importante: los scripts. Cualquier administrador de la empresa deberá escribir scripts para automatizar y llevar a cabo tareas administrativas. El capítulo Scripting detalla los consejos para el desarrollo y el despliegue de scripts.

Requisitos previos

Para la lectura de este libro se recomienda poseer unos conocimientos mínimos acerca del lenguaje PowerShell y sobre Windows, aunque es accesible a todos los debutantes. Gracias a su tabla de contenidos detallada, encontrará fácilmente el comando o los comandos para alcanzar el objetivo deseado.

Sistemas necesarios y recomendaciones

Conviene, antes de desplegar un script en producción, probarlo previamente en un entorno de certificación o de preproducción varias veces. No dude en crear un «laboratorio» basado en máquinas virtuales o en máquinas físicas para probar los comandos y los scripts antes de llevar a cabo su despliegue.

Como ocurre con todo lenguaje de programación, es posible obtener el mismo resultado utilizando diversos comandos o scripts. Los ejemplos presentados en este libro son lo más explícitos posibles, con el objetivo de facilitar la lectura, principalmente a los administradores que descubren Windows PowerShell.

Los ejemplos y las explicaciones se basan en la versión 5 de Windows PowerShell, versión instalada por defecto con Windows 10 y que puede instalarse en los sistemas operativos Windows 7 y Windows 8.1 (descargando una actualización).

Introducción

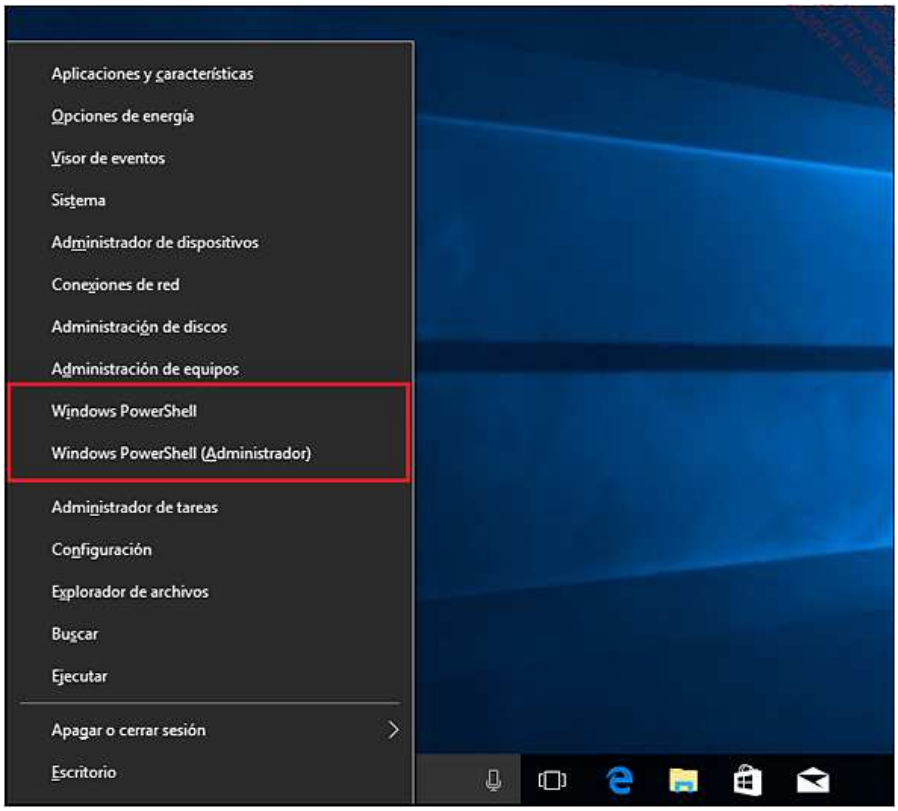
El nombre en clave «Monad» hace su aparición en septiembre de 2003, durante la PDC (*Professional Developers Conference*), una conferencia anual organizada por Microsoft. Detrás de su nombre en clave, se esconde en realidad el futuro de Windows PowerShell, tal y como lo conocemos en la actualidad. Tres años más tarde, en 2006, Windows PowerShell aparece en su versión final. Se pone a disposición para Windows XP y Windows Server 2003, y hay que descargarla de Internet e instalarla para poder utilizar Windows en estos sistemas operativos.

Pero, para interactuar con esta nueva herramienta dedicada a la administración de las máquinas, hay que aprender nuevos cmdlets, pues los comandos son algo diferentes en comparación con los de la consola de comandos existente desde Windows 95. Es necesario, por tanto, volver a aprender el lenguaje y la sintaxis. A su vez, la ausencia de un editor de script en la primera versión de Windows PowerShell frena el entusiasmo hacia esta herramienta.

Estas carencias se cubren a partir de la versión 2.0 de Windows PowerShell: Microsoft desarrolla un entorno de escritura de scripts, y además el intérprete de comandos se instala por defecto con Windows 7 y Windows Server 2008 R2. Junto al éxito de Windows 7, tanto en el dominio público como en el mundo empresarial, Windows PowerShell comienza realmente su ascenso. Con Windows 8 y Windows 10, Microsoft fuerza también su integración con el sistema: se proporciona un acceso directo a Windows PowerShell presente en el menú **Archivo** del Explorador de archivos de Windows. Se abre directamente en la carpeta seleccionada desde el Explorador.

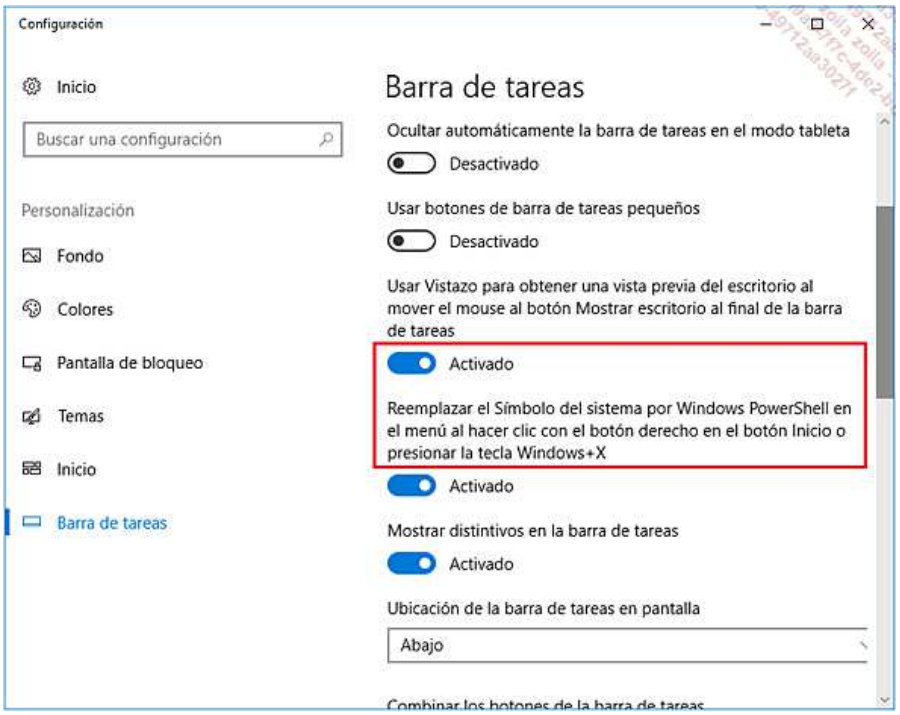
*Menú **Archivo** de la aplicación **Explorador de archivos***

A su vez, existe una opción que permite definir Windows PowerShell como aplicación por defecto en los accesos directos del menú. Así, cuando haga clic con el botón derecho en el menú **Inicio** (combinación de teclas [Windows] X), se propondrá la aplicación Windows PowerShell en lugar del Símbolo del sistema.



*Accesos directos del menú **Inicio** con Windows PowerShell*

Esta opción está presente en la aplicación **Configuración** de Windows 10, en la sección **Personalización**, y luego en la entrada **Barra de tareas**.



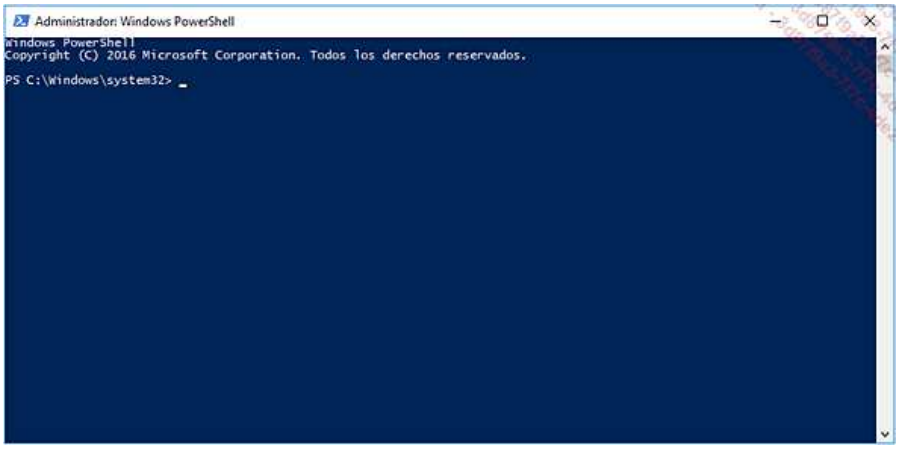
Las distintas opciones de personalización de la barra de tareas

Windows 10 integra a su vez una nueva versión de Windows PowerShell: la versión 5. Esta nueva versión del intérprete de comandos aporta su conjunto de novedades y de funcionalidades. Además de ofrecer una compatibilidad con los comandos, proveedores, scripts, etc., hasta la versión 2.0, se colorea la sintaxis en la consola automáticamente, ofreciendo así una lectura más fácil de las líneas de comandos que se introduzcan. A su vez, la consola ofrece, ahora, todos los aspectos prácticos que es posible tener con un procesador de texto: seleccionar, copiar y pegar mediante las combinaciones de teclas clásicas. ¡El ratón no es más que un accesorio!

¿Qué es Windows PowerShell?

Windows PowerShell es un intérprete de comandos, orientado a la administración del sistema de los productos de Microsoft. Si bien en este libro verá la administración del sistema para Windows, conviene saber que Windows PowerShell no se limita únicamente a sistemas operativos orientados a cliente. He aquí el conjunto de productos editados por la empresa de Redmond que pueden administrarse a través de PowerShell actualmente: productos orientados a servidor, como Windows Server, Exchange Server, etc., y también productos cloud, como Office 365 y Azure.

Microsoft comenzó en torno a los años 2004 y 2005 a revisar su estrategia para la administración del sistema en sus productos. En lo sucesivo, todo lo que puede realizar a través de la interfaz gráfica también es posible realizarlo mediante Windows PowerShell... ¡e incluso más! La interfaz gráfica de Windows, que ha contribuido a su éxito, se encuentra ahora limitada en comparación con las posibilidades que ofrece Windows PowerShell. Esta herramienta se posiciona también en la gama Windows Server: desde la versión 2012, se propone una instalación de Windows Server por defecto sin interfaz gráfica. La administración en su conjunto se realiza por línea de comandos con Windows PowerShell.



Consola Windows PowerShell

Seamos claros, la interfaz gráfica no está próxima a desaparecer: ofrece un acceso sencillo y ergonómico a los usuarios de Windows, tanto para un uso doméstico como en la empresa. Pero la presencia

de Windows PowerShell en el conjunto de productos de Microsoft demuestra su importancia.

Para abrir Windows PowerShell, siga estas instrucciones:

- Windows 7: abra el menú **Inicio** y, a continuación, haga clic en **Todos los programas**, luego en **Accesorios**, **Windows PowerShell**, y abra la consola **Windows PowerShell**.
- Windows 8.1: desde el Escritorio, abra la Charm Bar (teclas [Windows] C o apuntando el cursor del ratón hacia la esquina superior o inferior derecha de la pantalla) y, a continuación, haga clic en **Buscar**. Escriba **Windows PowerShell** y presione la tecla [Intro].
- Windows 10: abra el menú **Inicio** y, a continuación, escriba **Windows PowerShell** y presione la tecla [Intro].

¿Y los scripts, en todo esto?

¿Qué diría si pudiera automatizar todas las tareas de administración sobre el conjunto de puestos de trabajo presentes en su empresa? Y, todo ello, sin moverse de su escritorio. Imagine esto en un parque de ordenadores de más de diez mil puestos, en ocasiones con restricciones de idioma, de interfaz, de dispositivos de entrada (teclados QWERTY, AZERTY…), e incluso con diferencia horaria.

Windows PowerShell permite abstraerse de todas estas restricciones y automatizar el conjunto de sus tareas sobre un número indefinido de puestos de trabajo. El tiempo que ahorrará es, simplemente, extraordinario en comparación con el tiempo que le tomaría elaborar dichos scripts.

El interés de los scripts no se limita a un ahorro de tiempo. Permiten, asimismo, extender sus conocimientos sobre la sintaxis de Windows PowerShell, y también reducir los riesgos y aumentar la calidad y la eficacia de las acciones llevadas a cabo sobre los sistemas de información: un script ejecuta el conjunto de comandos escritos, mientras que, si debe realizar las mismas operaciones de forma manual decenas o centenares de veces (generalmente con la ayuda de más personas), esto produce errores humanos.

Otra ventaja de los scripts es su simplicidad en la escritura y la edición, en comparación con un programa. Como todo script, sea cual sea su lenguaje, está compuesto simplemente de un archivo de texto ASCII (*American Standard Code for Information Interchange*). De modo que es posible escribir o editar un script de PowerShell simplemente con el Bloc de notas de Windows o con cualquier otro editor de texto. Esto puede resultar muy práctico si está de viaje, o si no se encuentra en su ordenador habitual.

A partir de los comandos descritos en este libro, podrá crear un script que automatizará el conjunto de sus tareas administrativas, en función de sus necesidades: para el despliegue de un nuevo puesto de trabajo, para una migración, una operación de mantenimiento, o incluso simplemente para el despliegue de una configuración específica sobre el conjunto de puestos de trabajo de una empresa.

La sintaxis de Windows PowerShell

El lenguaje PowerShell está formado por comandos (utilizaremos en este libro el término cmdlets). Se trata de comandos descritos con la forma de un verbo y, a continuación, un nombre, separados por un guion. Para hacerse una idea, he aquí algunos cmdlets:

Cmdlet	Descripción
New-Item	Crear (New) un nuevo objeto (Item).
Get-Service	Obtener (Get) información vinculada a los servicios (Service) de Windows.
Set-ExecutionPolicy	Definir (Set) la política de ejecución (ExecutionPolicy) de sus scripts PowerShell.

Existen muchos cmdlets, ¡y con cada versión de Windows PowerShell se incorporan nuevos! De modo que se recomienda encarecidamente trabajar con la última versión de Windows PowerShell. Cada nueva versión aporta también nuevas funcionalidades: PowerShell Remoting (administración remota), Windows PowerShell ISE y Script Debugging (que permiten la depuración remota y la ejecución por línea de comandos paso a paso), Session Connectivity (para establecer una sesión en un equipo remoto), Help Update (actualización de la documentación de los comandos), y también nuevos parámetros a los cmdlets ya existentes. Esto es solo un pequeño vistazo de la evolución de Windows PowerShell desde la versión 1.0.

Para hacerse una idea de los cmdlets existentes en su puesto de trabajo, abra una ventana de Windows PowerShell y ejecute la siguiente línea de comandos:

```
PS C:\Windows\system32> Get-Command

CommandType      Name                                               Version
Source
-----------      ----                                               -------
-----
Alias            Add-ProvisionedAppxPackage        3.0         Dism
Alias            Apply-WindowsUnattend             3.0         Dism
Alias                 Disable-PhysicalDiskIndication      2.0.0.0
Storage
Alias                Disable-StorageDiagnosticLog         2.0.0.0
Storage
```

```
Alias                 Enable-PhysicalDiskIndication        2.0.0.0
Storage
Alias                 Enable-StorageDiagnosticLog          2.0.0.0
Storage
Alias              Flush-Volume                             2.0.0.0
Storage
Alias              Get-DiskSNV                              2.0.0.0
Storage
Alias               Get-PhysicalDiskSNV                     2.0.0.0
Storage
Alias            Get-ProvisionedAppxPackage         3.0         Dism
Alias               Get-StorageEnclosureSNV                2.0.0.0
Storage
Alias               Initialize-Volume                       2.0.0.0
Storage
Alias              Move-SmbClient                           2.0.0.0
SmbWitness
Alias            Remove-ProvisionedAppxPackage      3.0         Dism
Alias               Write-FileSystemCache                  2.0.0.0
Storage
Function         A:
[...]
```

Y para conocer el número exacto, escriba:

```
PS C:\Windows\system32> (Get-Command).Count
```

Para simplificar la lectura de este libro, todos los comandos de tipo **`Verbo-Sustantivo`** se llaman cmdlets. En realidad, solo una parte de ellos son nativamente cmdlets, mientras que otros son funciones o alias. Es posible conocer el tipo de comando en la columna `CommandType`.

El número de cmdlets puede variar de un puesto de trabajo a otro, pues es algo que depende de la versión de Windows, de la versión de PowerShell y también de las aplicaciones o de las funcionalidades instaladas en Windows. También es posible importar módulos en Windows PowerShell que agregan cmdlets suplementarios. Por ejemplo, si ha instalado la característica IIS (*Internet Information Services*) en su puesto de trabajo, o incluso RSAT (*Remote Server Administration Tools*), se agregan cmdlets dedicados a estas aplicaciones. Es posible detectarlos en la columna `Source` presente en el resultado del comando **`Get-Command`**.

1. Encontrar rápidamente un cmdlet

Siempre con **`Get-Command`**, puede encontrar rápidamente el cmdlet o los cmdlets que responden a un nombre común. Esto permite obtener una visión general de los comandos disponibles para gestionar alguna tarea de administración.

Ejemplo: encontrar los cmdlets correspondientes a la administración de servicios de Windows

```
PS C:\Windows\system32> Get-Command -Noun Service

CommandType     Name                Version    Source
-----------     ----                -------    ------
Cmdlet                   Get-Service                    3.1.0.0
Microsoft.PowerShell.Management
Cmdlet                   New-Service                    3.1.0.0
Microsoft.PowerShell.Management
Cmdlet                     Restart-Service            3.1.0.0
Microsoft.PowerShell.Management
Cmdlet                    Resume-Service              3.1.0.0
Microsoft.PowerShell.Management
Cmdlet                   Set-Service                    3.1.0.0
Microsoft.PowerShell.Management
Cmdlet                    Start-Service               3.1.0.0
Microsoft.PowerShell.Management
Cmdlet                   Stop-Service                   3.1.0.0
Microsoft.PowerShell.Management
Cmdlet                    Suspend-Service            3.1.0.0
Microsoft.PowerShell.Management
```

Si tiene dudas acerca del nombre de un cmdlet, utilice el carácter * (*wildcard*) para realizar una búsqueda más extensa, pues en caso contrario **`Get-Command`** buscará el nombre exacto indicado con el parámetro **`-Noun`**.

Ejemplo: búsqueda extendida basada en la palabra ScheduledTask

```
PS C:\Windows\system32> Get-Command -Noun *ScheduledTask*

CommandType      Name                                     Version
Source
-----------     ----                                    -------    ---
---
Function         Disable-ScheduledTask                    1.0.0.0
ScheduledTasks
Function         Enable-ScheduledTask                     1.0.0.0
ScheduledTasks
Function         Export-ScheduledTask                     1.0.0.0
ScheduledTasks
Function         Get-ClusteredScheduledTask               1.0.0.0
ScheduledTasks
Function         Get-ScheduledTask                        1.0.0.0
ScheduledTasks
Function         Get-ScheduledTaskInfo                    1.0.0.0
ScheduledTasks
Function         New-ScheduledTask                        1.0.0.0
ScheduledTasks
Function         New-ScheduledTaskAction                  1.0.0.0
ScheduledTasks
Function         New-ScheduledTaskPrincipal               1.0.0.0
ScheduledTasks
Function         New-ScheduledTaskSettingsSet             1.0.0.0
ScheduledTasks
Function         New-ScheduledTaskTrigger                 1.0.0.0
```

```
ScheduledTasks
Function          Register-ClusteredScheduledTask        1.0.0.0
ScheduledTasks
Function          Register-ScheduledTask                 1.0.0.0
ScheduledTasks
Function          Set-ClusteredScheduledTask             1.0.0.0
ScheduledTasks
Function          Set-ScheduledTask                      1.0.0.0
ScheduledTasks
Function          Start-ScheduledTask                    1.0.0.0
ScheduledTasks
Function          Stop-ScheduledTask                     1.0.0.0
ScheduledTasks
Function          Unregister-ClusteredScheduledTask      1.0.0.0
ScheduledTasks
Function          Unregister-ScheduledTask               1.0.0.0
ScheduledTasks
```

El resultado devuelve los cmdlets que contienen el nombre ScheduledTask, efectivamente, ¡pero también ScheduledTaskAction, ScheduledTaskSettingsSet, ScheduledTaskTrigger y muchos otros!

2. Obtener la documentación de un cmdlet

Para conocer todos los detalles acerca de un cmdlet, existe el comando **Get-Help**. Este comando permite obtener toda la documentación correspondiente al cmdlet que desea utilizar. No olvide incluir el parámetro **-Full**, que agrega el detalle de cada parámetro que puede utilizarse, seguido de algunos ejemplos.

*Ejemplo: obtener la ayuda del cmdlet **Out-File***

```
PS C:\Windows\system32> Get-Help Out-File -Full

NOMBRE
    Out-File

SINOPSIS
    Sends output to a file.


SINTAXIS
     Out-File [-FilePath] <String> [[-Encoding] {unknown | string |

      unicode | bigendianunicode | utf8 | utf7 | utf32 | ascii |
default |
      oem}] [-Append] [-Confirm] [-Force] [-InputObject <PSObject>]

    [-NoClobber] [-NoNewline] [-WhatIf] [-Width <Int32>]
    [<CommonParameters>]

         Out-File  [[-Encoding]  {unknown  |  string  |  unicode  |
bigendianunicode |
      utf8 | utf7 | utf32 | ascii | default | oem}] [-Append] [-
Confirm]
```

```
      [-Force] [-InputObject <PSObject>] -LiteralPath <String> [-
NoClobber]
    [-NoNewline] [-WhatIf] [-Width <Int32>] [<CommonParameters>]


DESCRIPCIÓN
     The Out-File cmdlet sends output to a file. You can use this
cmdlet
     instead of the redirection operator (>) when you need to use
its
    parameters.


PARÁMETROS
    -Append [<SwitchParameter>]
         Indicates that the cmdlet adds the output to the end of an
existing
        file, instead of replacing the file contents.

        ¿Requerido?                    false
        ¿Posición?                     named
        Valor predeterminado           false
        ¿Aceptar canalización?         false
        ¿Aceptar caracteres comodín?   false

[...]

    Ejemplo 1: Send output to a file

    PS C:\>Get-Process | Out-File -filepath C:\Test1\process.txt

     This command sends a list of processes on the computer to the

     Process.txt file. If the file does not exist, Out-File creates
it.
    Because the name of the FilePath parameter is optional, you can
omit it
    and submit the equivalent command `Get-Process | Outfile
    C:\Test1\process.txt`.

[...]
```

3. Actualizar la documentación

Desde Windows PowerShell versión 3.0, es posible actualizar la documentación de los cmdlets gracias al comando **Update-Help**. Este cmdlet comprueba, para cada uno de los módulos presentes, si se posee la última versión del archivo de ayuda asociado:

```
PS C:\Windows\system32> Update-Help
```

Si no posee la última versión, **Update-Help** descargará desde Internet el archivo de ayuda y, a continuación, lo instalará. Verá también, en la parte superior de la pantalla, la actualización del contenido de ayuda para los módulos instalados en Windows PowerShell:

```
Actualizando Ayuda para el módulo Dism
    Instalando contenido de Ayuda...

[ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo]
```

Las distintas versiones de Windows PowerShell

Actualmente, existen cinco versiones mayores de Windows PowerShell. He aquí un resumen de cada una de estas versiones, de las principales novedades que aportan y de los enlaces para su descarga.

1. Versión 1.0

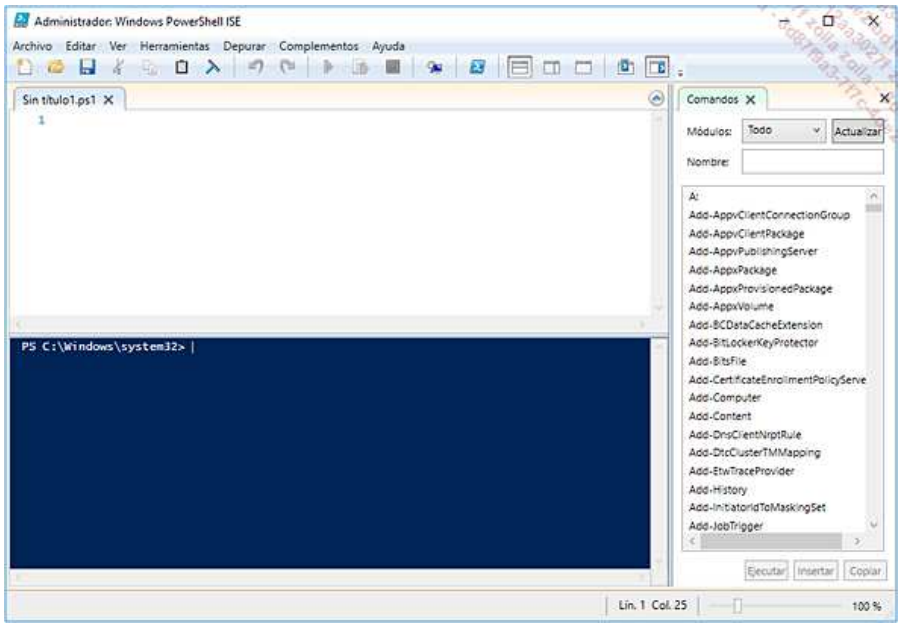
Windows PowerShell aparece, en su versión final, en 2006. Se construye para los sistemas operativos Windows XP, Windows Vista y Windows Server 2003. Está integrado también en Windows Server 2008 como una característica para instalar.

Esta primera versión ya permite administrar varios tipos de objeto: .NET, COM, Active Directory, WMI... además de los cmdlets básicos para la manipulación de objetos tales como archivos, claves de registro, etc.

2. Versión 2.0

La versión 2.0 está integrada en Windows 7 y Windows Server 2008 R2. Instalada por defecto en estos sistemas operativos, Windows PowerShell está disponible directamente haciendo clic en **Inicio**, y a continuación en **Todos los programas**, **Accesorios**, **Windows PowerShell**.

Esta nueva versión incorpora un número considerable de cmdlets y de funcionalidades, como la administración remota de máquinas (PowerShell Remoting). Pero esto no es todo, pues integra también un entorno y un editor de scripts: Windows PowerShell Integrated Scripting Environment (ISE), que era una carencia considerable de la versión 1.0. Windows PowerShell ISE aporta una gran facilidad para escribir scripts y herramientas de despliegue, como el depurador.



Windows PowerShell ISE

Esta versión empieza a ser interesante no solo desde un punto de vista técnico, sino también desde un punto de vista del scripting, gracias al editor de scripts.

3. Versión 3.0

La versión 3.0 de Windows PowerShell está, por su parte, integrada en Windows 8 y Windows Server 2012. También está disponible para Windows 7, Windows Server 2008 y Windows Server 2008 R2. Para ello, debe descargarla e instalar el paquete Windows Management Framework 3.0, que contiene Windows PowerShell versión 3.0. Es posible descargarlo de la siguiente dirección: http://www.microsoft.com/en-us/download/details.aspx?id=34595

En función del sistema operativo instalado, descargue el archivo indicado en la siguiente tabla:

Versión de Windows	Archivo para descargar
Windows 7 con Service Pack 1 (32 bits)	Windows6.1-KB2506143-x86.msu
Windows 7 con Service Pack 1 (64 bits)	Windows6.1-KB2506143-x64.msu

Versión de Windows	Archivo para descargar
Windows Server 2008 con Service Pack 1 (32 bits)	Windows6.0-KB2506146-x86.msu
Windows Server 2008 con Service Pack 1 (64 bits)	Windows6.0-KB2506146-x64.msu
Windows Server 2008 R2 con Service Pack 1	Windows6.1-KB2506143-x64.msu

Para instalar Windows Management Framework 3.0, hay que descargar previamente el .NET Framework versión 4 o 4.5. Puede descargarlo en los siguientes enlaces:

Versión 4: http://www.microsoft.com/es-ES/download/details.aspx?id=17718

Versión 4.5: http://www.microsoft.com/es-ES/download/details.aspx?id=42642

Esta versión incluye sus novedades: mejora de la administración remota de equipos, planificación de la ejecución, actualización de la documentación a través de Internet… y, por supuesto, nuevos cmdlets.

4. Versión 4.0

La versión 4.0 de Windows PowerShell, integrada en Windows 8.1 y Windows Server 2012 R2, incluye nuevos cmdlets y también mejora los ya existentes proponiendo nuevos parámetros. Esta versión está también disponible para los siguientes sistemas operativos: Windows 7, Windows Server 2008 R2 y Windows Server 2012.

Como con la versión anterior de Windows PowerShell, hay que descargar Windows Management Framework, esta vez la versión 4.0. Consulte la siguiente dirección para descargarlo: http://www.microsoft.com/en-us/download/details.aspx?id=40855

A continuación, descargue el archivo indicado en la siguiente tabla en función de la versión de Windows instalada:

Versión de Windows	Archivo para descargar

Versión de Windows	Archivo para descargar
Windows 7 con Service Pack 1 (32 bits)	Windows6.1-KB2819745-x86-MultiPkg.msu
Windows 7 con Service Pack 1 (64 bits)	Windows6.1-KB2819745-x64-MultiPkg.msu
Windows Server 2008 R2 con Service Pack 1	Windows6.1-KB2819745-x64-MultiPkg.msu

Para instalar Windows Management Framework versión 4.0, hay que instalar previamente el .NET Framework versión 4.5. Puede descargarlo de la siguiente dirección: http://www.microsoft.com/es-ES/download/details.aspx?id=42642

Windows Management Framework 4.0 no puede instalarse en Windows 8, de modo que debe actualizarse obligatoriamente a Windows 8.1 para obtener Windows PowerShell versión 4.0.

5. Versión 5.0 y 5.1

La versión 5.0 se incluye con Windows 10. Esta nueva versión aporta numerosas novedades, como la manipulación de archivos comprimidos ZIP o la administración de paquetes (permite encontrar paquetes de aplicaciones en los repositorios e instalarlos en los equipos). También se incorporan nuevos cmdlets, ofreciendo así siempre más posibilidades en la gestión del puesto de trabajo.

También es posible instalar la versión 5.0 de Windows PowerShell en las versiones anteriores de Windows, mediante la actualización Windows Management Framework 5.0, que debe descargarse e instalarse. Está disponible en la siguiente dirección: https://www.microsoft.com/en-us/download/details.aspx?id=50395

Consulte la siguiente tabla para descargar el archivo correcto en función de la versión de Windows sobre la que desee instalar la actualización:

Versión de Windows	Archivo para descargar

Versión de Windows	Archivo para descargar
Windows 7 con Service Pack 1 (32 bits)	Win7-KB3134760-x86.msu
Windows 7 con Service Pack 1 (64 bits)	Win7AndW2K8R2-KB3134760-x64.msu
Windows 8.1 (32 bits)	Win8.1-KB3134758-x86.msu
Windows 8.1 (64 bits)	Win8.1AndW2K12R2-KB3134758-x64.msu
Windows Server 2008 R2 con Service Pack 1	Win7AndW2K8R2-KB3134760-x64.msu
Windows Server 2012	W2K12-KB3134759-x64.msu
Windows Server 2012 R2	Win8.1AndW2K12R2-KB3134758-x64.msu

Antes de instalar Windows Management Framework 5.0, el .NET Framework versión 4.5 o superior debe estar previamente instalado en el puesto de trabajo. La versión 4.5 está disponible en la siguiente dirección: https://www.microsoft.com/es-es/download/details.aspx?id=42642

Por su parte, la versión 4.6, está disponible aquí: https://www.microsoft.com/es-es/download/details.aspx?id=53344

Versión 5.1

Esta actualización menor de la versión 5 de Windows PowerShell hizo su aparición con la actualización *Anniversary Update* (nombre en clave *Redstone 1*) de Windows 10, y más adelante con la aparición oficial de Windows Server 2016. Para las versiones anteriores de Windows y Windows Server, el paquete de actualización de Windows Management Framework se publicó el 19 de enero de 2017.

La versión 5.1 aporta novedades y mejoras sobre varias funcionalidades, como el administrador de paquetes o también en la depuración de scripts.

Para obtener la versión 5.1 de Windows PowerShell, he aquí el procedimiento en función de la versión de Windows:

- Windows 10: actualizar Windows 10 para obtener la actualización *Anniversary Update*.
- Windows Server 2016: se incluye la versión 5.1 de manera nativa.
- Para las versiones previas de Windows y Windows Server: es preciso descargar Windows Management Framework 5.1, disponible en la siguiente dirección: https://www.microsoft.com/en-us/download/details.aspx?id=54616

  Consulte a continuación la siguiente tabla para descargar el archivo correcto en función de la versión de Windows:

Versión de Windows	Archivo para descargar
Windows 7 con Service Pack 1 (32 bits)	Win7-KB3191566-x86.zip
Windows 7 con Service Pack 1 (64 bits)	Win7AndW2K8R2-KB3191566-x64.zip
Windows 8.1 (32 bits)	Win8.1-KB3191564-x86.msu
Windows 8.1 (64 bits)	Win8.1AndW2K12R2-KB3191564-x64.msu
Windows Server 2008 R2 con Service Pack 1	Win7AndW2K8R2-KB3191566-x64.zip
Windows Server 2012	W2K12-KB3191565-x64.msu
Windows Server 2012 R2	Win8.1AndW2K12R2-KB3191564-x64.msu

Antes de instalar Windows Management Framework 5.1, el .NET Framework versión 4.5 o superior debe estar previamente instalado en el puesto de trabajo. La versión. 4.5 está disponible en la siguiente dirección: https://www.microsoft.com/es-es/download/details.aspx?id=42642

Por su parte, la versión 4.6, está disponible aquí: https://www.microsoft.com/fr-fr/download/details.aspx?id=53344

6. Evolución continua de Windows PowerShell

Con la aparición de Windows 10, Microsoft también ha cambiado el ciclo de desarrollo de su sistema operativo. En efecto, la incorporación de novedades en Windows ya no se realiza a través de Service Packs o con una nueva versión de Windows. En lo

sucesivo, Windows 10 pasa a un esquema de desarrollo continuo, donde pueden aparecer nuevas funcionalidades con regularidad durante las actualizaciones proporcionadas por el editor. Windows PowerShell evoluciona, así, periódicamente con el paso del tiempo.

Por su parte, Microsoft sigue avanzando en el terreno del open source y anunció, en agosto de 2016, que pasaría PowerShell a licencia open source (licencia MIT). Sin duda, esta acción ayudará a realzar la importancia de la comunidad en torno a PowerShell y la difusión de la herramienta en otras plataformas. En lo sucesivo, hay versiones de PowerShell disponibles para Linux y macOS, basadas en el núcleo .NET Core, que también es open source.

Conocer la versión de PowerShell instalada

Dadas las distintas versiones de PowerShell existentes hasta la fecha, y las novedades que aportan, es importante conocer la versión de Windows PowerShell instalada en el puesto de trabajo.

Para ello, una vez abierta la interfaz de Windows PowerShell, puede ejecutar el siguiente comando: **`$PSVersionTable`**, que devolverá el contenido de la variable:

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

Name                           Value
----                           -----
PSVersion                      5.1.14393.693
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.14393.693
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

Como podrá observar, aquí se trata de Windows PowerShell versión 5.1 (valor representado por `PSVersion`).

Si este comando devolviera un error, ello se debería a que la versión de PowerShell instalada en el equipo es la versión 1.0. En efecto, la variable **`$PSVersionTable`** apareció con la versión 2.0. Se define automáticamente cada vez que se lanza Windows PowerShell.

He aquí un segundo método para conocer la versión de PowerShell, utilizando en esta ocasión la variable **`$Host`**:

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

Name             : ConsoleHost
Version          : 5.1.14393.693
InstanceId       : 219b30cb-f6b9-4b6e-ba3d-b58885b83c7b
UI               : System.Management.Automation.Internal.Host.
                     InternalHostUserInterface
CurrentCulture   : es-ES
CurrentUICulture: es-ES
PrivateData                                                    :
Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
```

```
DebuggerEnabled : True
IsRunspacePushed: False
Runspace                                                                :
System.Management.Automation.Runspaces.LocalRunspace
```

Verá, como con la variable **$PSVersionTable**, la versión 5.1.

Puede que haya observado, en el resultado que devuelve **$PSVersionTable**, que el campo `PSCompatibleVersions` contiene todas las versiones de Windows PowerShell: 1.0, 2.0, 3.0, 4.0, 5.0 y 5.1. De hecho, es posible ejecutar Windows PowerShell en modo de compatibilidad con cualquier versión anterior a la versión actualmente instalada.

```
PS C:\Users\Admin> $PSVersionTable.PSCompatibleVersions

Major  Minor  Build  Revision
-----  -----  -----  --------
1      0      -1     -1
2      0      -1     -1
3      0      -1     -1
4      0      -1     -1
5      0      -1     -1
5      1      14393  693
```

El motivo es muy simple: si desea ejecutar un script PowerShell desarrollado específicamente con la versión 2.0 de Windows PowerShell, puede ser conveniente lanzarlo en modo de compatibilidad. Para pasar a una versión anterior de Windows PowerShell, hay que llamar al ejecutable powershell.exe, incluyendo el parámetro **-version** e indicando el número de versión correspondiente:

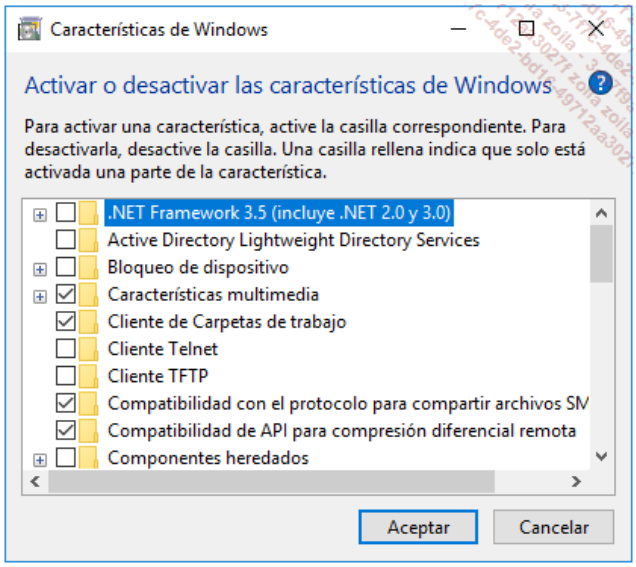
```
PS C:\Windows\system32> powershell.exe -version 2.0
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> $PSVersionTable

Name                           Value
----                           -----
CLRVersion                     2.0.50727.8745
BuildVersion                   6.1.7600.16385
PSVersion                      2.0
WSManStackVersion              2.0
PSCompatibleVersions           {1.0, 2.0}
SerializationVersion           1.1.0.1
PSRemotingProtocolVersion      2.1
```

Así, sin tener que abrir una nueva ventana, Windows PowerShell funciona como si hubiera abierto la versión 2.0 de Windows PowerShell instalada en su puesto de trabajo.

En caso de trabajar con Windows 8.1 o Windows 10, puede que obtenga un error si intenta lanzar el modo de compatibilidad con la versión 2.0.

```
PS C:\Windows\system32> powershell.exe -version 2.0
La versión v2.0.50727 de .NET Framework no está instalada.
Esta versión es necesaria para ejecutar la versión 2.0 de
Windows PowerShell.
```

Para resolver este inconveniente, debe habilitar la característica de Windows **.NET Framework 3.5 (incluye .NET 2.0 y 3.0)** presente en **Programas y características** del Panel de control de Windows.



Ventana de activación o desactivación de características de Windows

Preste atención, sin embargo, pues las fuentes para habilitar esta característica no están incluidas directamente en el sistema operativo. Existen dos posibles soluciones:

- El puesto de trabajo debe poseer una conexión a Internet activa para poder descargar los archivos desde Windows Update.
- Poseer las fuentes de instalación de Windows (en un medio extraíble o en un archivo ISO montado previamente) y ejecutar el cmdlet **`Enable-WindowsOptionalFeature`** especificando la ruta de acceso a los archivos fuentes mediante el parámetro **`-Source`**.

```
PS C:\Windows\system32> Enable-WindowsOptionalFeature
-FeatureName NetFx3 -Source D:\sources\sxs -Online
```

Para obtener más detalles acerca de la activación y la desactivación de las características de Windows directamente por línea de comandos PowerShell, consulte el capítulo Sistema y características.

Las unidades de Windows PowerShell y la navegación

En este capítulo veremos el conjunto de unidades de Windows PowerShell a los que se tiene acceso desde el intérprete de comandos. Sistema de archivos, registros o certificados son, por citar solamente algunos, almacenes de datos accesibles con Windows PowerShell. En cada uno de ellos es posible navegar y realizar acciones sobre los elementos que los componen.

1. ¿Qué es una unidad Windows PowerShell?

Una unidad Windows PowerShell es la ubicación de algún almacén de datos, como por ejemplo una unidad de sistema de archivos. Sin embargo, los proveedores de Windows PowerShell crean por defecto varias unidades en los distintos almacenes de datos, y le ofrecen así la posibilidad de acceder a las unidades de sistema de archivos (C:), las unidades de registro (HKCU: y HKLM:), la unidad de certificados (Cert:) y otras. En todas estas unidades de Windows PowerShell, es posible ejecutar por línea de comandos el conjunto de acciones que se llevan a cabo habitualmente a través de la interfaz gráfica de Windows.

He aquí una tabla de las unidades de Windows PowerShell más importantes:

Nombre	Tipo	Ruta raíz
C	Sistema de archivos	C:\
Env	Variables de entorno	
HKCU	Registro	HKEY_CURRENT_USER
HKLM	Registro	HKEY_LOCAL_MACHINE

Y… ¡puede haber otras! En efecto, los proveedores de Windows PowerShell pueden crear nuevas unidades hacia otros almacenes de datos. Es el caso, por ejemplo, de IIS (servidor web de Microsoft), que puede instalar en un puesto de trabajo. Tras importar el módulo correspondiente, se creará una nueva unidad Windows PowerShell que estará accesible (IIS:).

Como administrador de puestos de trabajo, resulta imprescindible saber realizar operaciones sobre estas unidades de Windows PowerShell. Copiar, mover, renombrar o simplemente eliminar

archivos son tareas habituales. Obtener un comportamiento concreto de una aplicación o de Windows sobre un puesto de trabajo es, a menudo, una exigencia de los usuarios y, para ello, poder modificar valores en el registro o en un archivo resulta esencial. Migrar cientos de puestos de trabajo tras la adquisición de una empresa e incluso realizar tareas administrativas comunes son ejemplos, entre otros, que obligan a los administradores de puestos de trabajo a realizar acciones sobre estas unidades de Windows PowerShell.

2. Conocer las unidades de Windows PowerShell accesibles

Los accesos a las unidades de Windows PowerShell pueden variar de un puesto a otro. Existe un cmdlet que permite enumerar estos accesos: **`Get-PSDrive`**. He aquí una demostración sobre un puesto de trabajo:

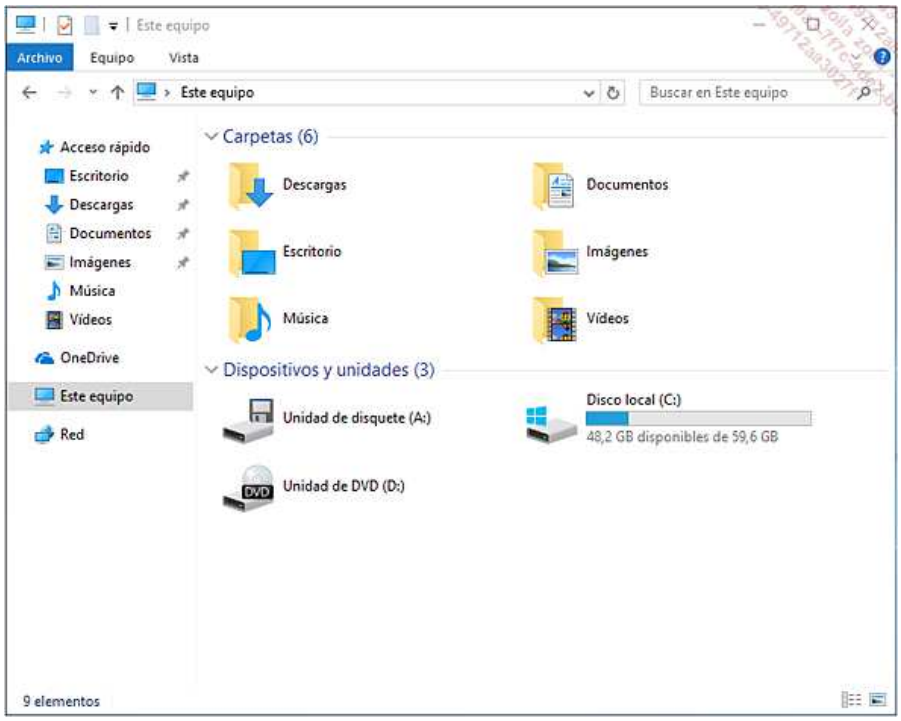
```
PS C:\Windows\system32> Get-PSDrive

Name         Used (GB)    Free (GB)  Provider      Root
----         ---------    ---------  --------      ----
Alias                                 Alias
C                11,84       114,64  FileSystem    C:\
Cert                                  Certificate   \
D                 3,62               FileSystem    D:\
Env                                   Environment
Function                              Function
HKCU                                                Registry
 HKEY_CURRENT_USER
HKLM                                                Registry
 HKEY_LOCAL_MACHINE
Variable                              Variable
WSMan                                 WSMan
X               386,40        13,59  FileSystem    \\server01\D$
```

Estas unidades de Windows PowerShell, resultado del comando **`Get-PSDrive`**, están accesibles desde el intérprete de comandos. He aquí una lista explicativa para cada uno de los distintos proveedores (columna `Provider`):

- **Alias**: aquí se trata de alias que se implementan para invocar cmdlets PowerShell. Así, `del` invoca realmente al cmdlet **`Remove-Item`**. Es posible, por supuesto, agregar nuevos alias.
- **FileSystem**: se trata de unidades que contienen un sistema de archivos. Estas unidades pueden ser locales o estar en red.

- **Certificate**: se corresponde con el almacén de certificados del puesto de trabajo. Este almacén de certificados está accesible a través de la interfaz gráfica de la consola Microsoft Management Console (MMC) y agregando a continuación el componente llamado Certificados.
- **Environment**: contiene la lista de variables de entorno. Estas variables dinámicas las utilizan los distintos procesos del sistema operativo. Así, la variable de entorno `$env:COMPUTERNAME` designa el nombre del equipo debidamente registrado en Windows.
- **Function**: contiene el conjunto de funciones disponibles en Windows PowerShell. Estas funciones contienen código PowerShell que se ejecuta cuando las invoca. Por ejemplo, cuando introduce el comando `D:` en Windows PowerShell, este último ejecuta en realidad: `Set-Location D:`. Por supuesto, es posible registrar nuevas funciones.
- **Registry**: designa el registro de Windows. Este registro contiene toda la configuración del sistema operativo y de las aplicaciones instaladas. El registro está accesible a través de una interfaz gráfica abriendo el ejecutable regedit.exe.
- **Variable**: contiene el conjunto de variables durante el tiempo de vida del proceso Windows PowerShell. Algunas variables se construyen previamente, como `$PSVersionTable` y `$Host`, que vimos en el capítulo Presentación de Windows PowerShell, y también es posible crear otras nuevas.
- **WSMan**: contiene los parámetros WS-Management. Estos parámetros se utilizan principalmente cuando se trabaja con la funcionalidad PowerShell Remoting.

El trabajo cotidiano de un administrador de equipos de trabajo se realiza en gran parte sobre unidades de Windows PowerShell de tipo sistema de archivos y registro. Lo veremos en los siguientes capítulos. Por su parte, las variables de entorno tienen una gran importancia, sobre todo en el desarrollo de scripts. Este tema se abordará en el capítulo Buscar y recopilar información.



El Explorador de Windows permite acceder a los sistemas de archivos

Se dedicará también una sección a cada una de las unidades de Windows PowerShell siguientes: unidad de certificados (Cert:), unidad WS-Management (WSMan:) e IIS (IIS:). Las distintas secciones nos permitirán conocer perfectamente las posibilidades ofrecidas por Windows PowerShell sobre estas unidades.

El sistema de archivos

El sistema de archivos permite almacenar archivos, y por tanto datos. Estos archivos pueden estar organizados en una o varias carpetas, de manera jerárquica.

Existen dos tipos de cmdlet: los cmdlets utilizados para navegar para situarse en una unidad o en una carpeta deseada, y aquellos que permiten manipular objetos (como, por ejemplo, renombrar una carpeta, copiar un archivo, etc.).

1. Cmdlets dedicados a la navegación

En esta sección, verá la gran mayoría de los cmdlets necesarios para navegar por las unidades de Windows PowerShell. Pues en lo sucesivo, con Windows PowerShell, ya no existen comandos específicos destinados a un único tipo de unidad. En efecto, con la consola de comandos de Windows, comandos tales como `cd`, `copy` o incluso `delete` funcionaban solamente con el sistema de archivos. El comando `reg` está dedicado, por su parte, únicamente al registro de Windows.

Con Windows PowerShell, cmdlets tales como **`Set-Location`**, **`Copy-Item`** o incluso **`Get-Content`** son válidos tanto en la unidad de Windows PowerShell de tipo sistema de archivos como en la del registro… ¡y en las demás unidades de Windows PowerShell que hemos indicado previamente!

a. Get-Location

`Get-Location` permite, simplemente, obtener la carpeta activa en la ejecución del comando:

```
PS C:\Windows\system32> Get-Location

Path
----
C:\Windows\system32
```

b. Set-Location

Todo administrador de sistemas conoce el comando `cd` (Change Directory) de la consola de comandos. En Windows PowerShell, si

bien sigue funcionando `cd`, en realidad es porque existe un alias que apunta al cmdlet **`Set-Location`**.

Así, para situarse en una carpeta específica, escriba:

```
PS C:\Windows\system32> Set-Location D:\
PS D:\>
```

Si la ruta de acceso contiene espacios, hay que escribirla entre comillas (simples o dobles):

```
PS D:\> Set-Location 'C:\Program Files'
PS C:\Program Files>
```

También es posible utilizar rutas relativas: «.», «..», «\». Así, para acceder a una subcarpeta, escriba:

```
PS C:\Program Files> Set-Location '.\Common Files'
PS C:\Program Files\Common Files>
```

c. Get-ChildItem

El cmdlet **`Get-ChildItem`** permite enumerar el contenido de una carpeta, y le indica también los atributos de cada uno de sus elementos (archivos y carpetas). Se trata del equivalente al comando `dir` (Directory) en la consola de comandos.

Veamos un ejemplo:

```
PS C:\Program Files> Get-ChildItem


    Directorio: C:\Program Files


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----         15/06/2014     16:05            Common Files
d----         02/06/2014     07:36            Internet Explorer
d----          15/06/2014      16:01            Microsoft Analysis
Services
d----         15/06/2014     16:03            Microsoft Office
d----           15/06/2014       16:03                Microsoft SQL
Server
d----         15/06/2014     16:04            Microsoft.NET
d----         02/06/2014     08:22            Windows Defender
d----         30/09/2013     06:03            Windows Journal
d----         30/09/2013     05:59            Windows Mail
d----           02/06/2014       07:35                Windows Media
Player
d----          02/06/2014      07:35            Windows Multimedia
Platform
d----         01/06/2014     15:32            Windows NT
d----           30/09/2013       05:59                Windows Photo
```

```
Viewer
d----           02/06/2014      07:35                Windows Portable
Devices
d----         22/08/2013     17:36             WindowsPowerShell
```

En el ejemplo anterior, Windows PowerShell muestra el contenido de la carpeta en la que se ha ejecutado el comando. Por defecto, **Get-ChildItem** no muestra los archivos ocultos. Para mostrar los archivos ocultos, es preciso agregar el parámetro **-Hidden** al comando.

He aquí un resumen de la información devuelta por el comando **Get-ChildItem**:

Columna	Descripción
Modo: d a r h s	Atributos de los objetos: Directory (carpeta) Archive Read-Only (solo lectura) Hidden (objeto oculto) System (objeto del sistema)
LastWriteTime	Fecha de última modificación.
Length	Tamaño del archivo (en bytes).
Name	Nombre del objeto.

He aquí los parámetros más importantes de **Get-ChildItem**:

Parámetro	Descripción
-Attributes <String[]>	Permite seleccionar únicamente los objetos con uno o varios atributos (consulte la tabla anterior).
-Recurse	Llamada recursiva (muestra el contenido de las subcarpetas).
-Filter <String>	Permite filtrar los objetos.
-Path <String[]>	Especifica una o varias rutas de acceso. Están permitidos los caracteres comodín. La ubicación por defecto es la carpeta activa.
-Hidden	Muestra únicamente los archivos y carpetas ocultas.

He aquí otro ejemplo que permite mostrar los archivos ocultos, filtrados por la extensión de archivo *.dat:

```
PS C:\Users\Admin> Get-ChildItem -Path . -Hidden -Filter *.dat


    Directorio: C:\Users\Admin


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a-hs         02/06/2014     19:18     524288 NTUSER.DAT
```

2. Cmdlets dedicados a la manipulación

Más importantes, los cmdlets dedicados a la manipulación de objetos permiten modificar los valores de cada una de las unidades de Windows PowerShell.

a. Get-Item

`Get-Item` permite recuperar el objeto indicado en la línea de comandos, gracias al parámetro **`-Path`**. Este cmdlet es útil, sobre todo, en combinación con una barra, o *pipe* en inglés (barra vertical, correspondiente al carácter |), que permite redirigir a otro cmdlet el flujo de información del objeto recuperado.

He aquí un resumen de los parámetros disponibles:

Parámetro	Descripción
`-Exclude` `<String[]>`	Permite excluir objetos devueltos por el parámetro **`-Path`**. Es posible utilizar el carácter *wildcard* (*).
`-Force`	Autoriza al cmdlet a recuperar un objeto que normalmente no es accesible, como por ejemplo archivos ocultos.
`-Include` `<String[]>`	Recupera únicamente los objetos especificados. Es posible utilizar el carácter *wildcard* (*).
`-Path` `<String[]>`	Indica la ruta de acceso del archivo o de los archivos y de las carpetas.

El siguiente ejemplo permite recuperar el conjunto de archivos con extensión *.txt presentes en una carpeta específica:

```
PS C:\Windows\system32> Get-Item -Path C:\Temp\*.txt


    Directorio: C:\Temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        01/10/2016     13:46            552 Archivo1.txt
-a----        01/10/2016     13:46            976 Archivo2.txt
```

Pero el interés de este cmdlet es utilizarlo junto a otro para realizar alguna acción, como por ejemplo una copia o un desplazamiento. He aquí un ejemplo:

```
PS C:\Windows\system32> Get-Item -Path C:\Temp\*.txt | Copy-Item

-Destination C:\Temp2
```

Los archivos de texto se copian, a continuación, en la carpeta de destino especificada por el parámetro **-Destination** del cmdlet **Copy-Item**.

Otro aspecto interesante de **Get-Item** es la posibilidad de recuperar las propiedades asociadas al objeto. He aquí un ejemplo con un archivo de imagen:

```
PS C:\Temp> Get-Item .\IMGP3532.JPG | Format-List *


PSPath                                                                :
Microsoft.PowerShell.Core\FileSystem::C:\Temp\IMGP3532.JPG
PSParentPath      : Microsoft.PowerShell.Core\FileSystem::C:\Temp
PSChildName       : IMGP3532.JPG
PSDrive           : C
PSProvider        : Microsoft.PowerShell.Core\FileSystem
PSIsContainer     : False
Mode              : -a----
VersionInfo       : File:             C:\Temp\IMGP3532.JPG
                    InternalName:
                    OriginalFilename:
                    FileVersion:
                    FileDescription:
                    Product:
                    ProductVersion:
                    Debug:            False
                    Patched:          False
                    PreRelease:       False
                    PrivateBuild:     False
                    SpecialBuild:     False
                    Language:

BaseName          : IMGP3532
```

```
Target           : {}
LinkType         :
Name             : IMGP3532.JPG
Length           : 11613776
DirectoryName    : C:\Temp
Directory        : C:\Temp
IsReadOnly       : False
Exists           : True
FullName         : C:\Temp\IMGP3532.JPG
Extension        : .JPG
CreationTime     : 01/10/2016 14:32:16
CreationTimeUtc  : 01/10/2016 12:32:16
LastAccessTime   : 01/10/2016 14:32:16
LastAccessTimeUtc: 01/10/2016 12:32:16
LastWriteTime    : 09/01/2011 16:48:24
LastWriteTimeUtc : 09/01/2011 15:48:24
Attributes       : Archive
```

Algunas propiedades pueden resultar muy útiles en la elaboración de scripts. Encontramos, por ejemplo, el nombre del objeto (`Name`), su tamaño (`Length`) expresado en bytes, así como las fechas de creación (`CreationTime`), de último acceso (`LastAccessTime`), y por último la fecha de última modificación (`LastWriteTime`).

Así, para conocer el tamaño del archivo en megabytes, hay que utilizar una fórmula matemática:

```
PS C:\Temp> (Get-Item .\IMGP3532.JPG).Length / 1MB
11,0757598876953
```

Este archivo posee un tamaño de 11 MB. Puede utilizar los operadores 1KB, 1MB, 1GB para convertir el resultado a kilobytes, megabytes y gigabytes, respectivamente.

En lo relativo a las fechas, puede explotarlas directamente recuperando la propiedad correspondiente:

```
PS C:\Temp> (Get-Item .\IMGP3532.JPG).LastWriteTime

domingo, 9 de enero de 2011 16:48:24
```

En efecto, las fechas no son aquí una cadena de caracteres, sino un objeto DateTime que puede utilizarse directamente y explotarse con el cmdlet **`Get-Date`**.

b. Get-Content

`Get-Content` permite recuperar el contenido de un objeto. Incluso aunque sea un archivo binario, **`Get-Content`** se adapta y devuelve un resultado.

He aquí un ejemplo con el archivo hosts:

```
PS                C:\Windows\system32>                Get-Content
C:\Windows\System32\drivers\etc\hosts
# Copyright (c) 1993-2009 Microsoft Corp.
#
#  This  is  a  sample  HOSTS  file  used  by  Microsoft  TCP/IP  for
Windows.
#
# This file contains the mappings of IP addresses to host names.
Each
#  entry  should  be  kept  on  an  individual  line.  The  IP  address
should
# be placed in the first column followed by the corresponding host
name.
# The IP address and the host name should be separated by at least
one
# space.
#
#  Additionally,  comments  (such  as  these)  may  be  inserted  on
individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#      ::1             localhost
```

He aquí dos parámetros que pueden utilizarse con **Get-Content**:

Parámetro	Descripción
-TotalCount <Int64>	Devuelve el número de filas desde el inicio del archivo.
-Tail <Int32>	Devuelve el número de filas desde el final del archivo.

El objetivo de **Get-Content** es, a menudo, recuperar un valor, o cuando se trabaja con archivos de texto muy grandes (con logs, por ejemplo), es necesario saber encontrar la información que se quiere recuperar. Para ello, usará una combinación de ambos cmdlets. El resultado del primero se enviará al segundo, utilizando un pipe «|»:

```
PS C:\Windows\system32> Get-Content
C:\Windows\System32\drivers\etc\hosts | Where-Object {$_ -match
"acme.com"}
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host
```

`Where-Object {$_ -match "acme.com"}` devuelve las filas que contienen el término `acme.com`.

Para ajustar todavía más la necesidad y recuperar, por ejemplo, la dirección IP correspondiente a la dirección x.acme.com, he aquí un ejemplo con el uso de varios pipes:

```
PS C:\Windows\system32> $ip = ((Get-Content
C:\Windows\System32\drivers\etc\hosts | Where-Object {$_ -match
"x.acme.com"}).Split("") | Where-Object {$_ -ne ""})[1]
PS C:\> $ip
38.25.63.10
```

Es importante saber acotar la información en un archivo de texto, pues puede necesitar cambiar el contenido de una fila o simplemente recuperar la información para utilizarla más adelante.

c. Set-Content

De manera inversa a **Get-Content**, **Set-Content** permite escribir o reemplazar datos en un objeto. He aquí algunos parámetros que puede explotar el cmdlet:

Parámetro	Descripción
-Path <String[]>	Ruta de acceso del objeto (archivo, registro, etc.).
-Value <Object[]>	Valor que se ha de definir en el objeto correspondiente mediante el parámetro **-Path**. Este valor puede ser una cadena de caracteres o también un objeto.

He aquí un ejemplo muy simple, donde **Set-Content** reemplaza un archivo de texto con la cadena de caracteres (valor) que se le ha pasado:

```
PS C:\Windows\system32> Set-Content -Path C:\Temp\LogFile.log
-Value "Nuevo archivo de log"
```

Se crea entonces un nuevo archivo LogFile.log si no existe. Si se ejecuta de nuevo el comando **Set-Content** con el archivo ya creado, su contenido se reemplazará con el nuevo valor.

Si la carpeta indicada no existe, se obtiene un error. Es preciso crear previamente la carpeta o las carpetas indicadas en

el parámetro **-Path** de **Set-Content**.

d. New-Item

Además de recuperar o definir la información de los objetos, es posible crearlos. Este es el rol de **New-Item**, que permite crear archivos y también carpetas. He aquí un resumen de los parámetros:

Parámetro	Descripción
-Force	Autoriza al cmdlet a escribir sobre un objeto ya existente, incluso aunque este sea de solo lectura.
-ItemType `<String>`	Tipo de objeto que se va a crear.
-Path `<String[]>`	Ruta de acceso del objeto que se va a crear.

Para crear un archivo, hay que indicar la ubicación y el nombre del archivo mediante el parámetro **-Path**, así como el tipo de elemento que se va a crear con el parámetro **-ItemType**, en este ejemplo de tipo `File`:

```
PS C:\Windows\system32> New-Item -Path C:\Temp\Test.txt -ItemType
File
```

Si el objeto (archivo) especificado en el comando **New-Item** ya existe, se obtiene un error. Indicando el parámetro **-Force**, se borra el objeto, y su contenido se elimina por completo.

Igual que con **Set-Content**, si la carpeta que debe contener el nuevo archivo no se ha creado, se obtiene un error. Hay que crear previamente la carpeta (consulte el siguiente ejemplo).

Para crear una carpeta, se procede de la misma manera, pero asignando a **-ItemType** el valor `Directory`:

```
PS C:\Windows\system32> New-Item C:\Temp\RepTest -ItemType
Directory
```

e. Remove-Item

Remove-Item elimina archivos o carpetas. He aquí algunos de sus parámetros:

Parámetro	Descripción
-Force	Autoriza al cmdlet a eliminar los objetos ocultos o de solo lectura.
-Path <String[]>	Ruta de acceso del objeto que debe eliminarse.
-Recurse	Elimina los objetos y los subobjetos presentes en la ruta de acceso.

El siguiente comando elimina únicamente los archivos con extensión .log presentes en la carpeta especificada por el parámetro **-Path**:

```
PS C:\Windows\system32> Remove-Item -Path D:\Temp\*.log
```

Para eliminar la carpeta y todo su árbol (subcarpetas y archivos) basta con indicar la carpeta que se ha de eliminar:

```
PS C:\Windows\system32> Remove-Item -Path D:\Temp
```

Se pide una confirmación. Es posible omitirla agregando el parámetro **-Recurse**.

Si solo desea eliminar el conjunto de subcarpetas y archivos contenidos, sin eliminar la carpeta de base, escriba:

```
PS C:\Windows\system32> Remove-Item -Path D:\Temp\*
```

Preste atención durante la ejecución del cmdlet **Remove-Item**. Este elimina los archivos y carpetas sin dejarlos en la Papelera de reciclaje de Windows. Esta acción es, por tanto, irreversible. Por este motivo puede resultar interesante, antes de ejecutar realmente el comando, realizar una simulación: si se especifica el parámetro **-WhatIf**, se obtiene una vista previa del resultado de la ejecución del comando:

```
PS C:\Windows\system32> Remove-Item -Path C:\Temp\*.vbs -Recurse
-WhatIf
WhatIf: Se está realizando la operación "Quitar archivo" en el
destino
"C:\Temp\Audit.vbs".
WhatIf: Se está realizando la operación "Quitar archivo" en el
destino
"C:\Temp\Serial.vbs".
```

```
PS C:\Windows\system32> Get-ChildItem -Path C:\Temp


    Directorio: C:\Temp


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         28/11/2012     16:05       2976 Audit.vbs
-a---         29/01/2014     03:50       9510 CustomActionsPS.ps1
-a---         18/03/2014     12:38        684 ChangeSetting.reg
-a---         10/07/2012     14:40      25390 Serial.vbs
```

f. Copy-Item

Como su propio nombre indica, **Copy-Item** permite copiar uno o varios objetos. Se utiliza como el antiguo comando `copy`, pasando como parámetro el archivo (o grupo de archivos) o carpeta a copiar, y a continuación el destino.

Parámetro	Descripción
-Destination <String>	Copia los objetos y subobjetos presentes en la ruta de acceso.
-Force	Autoriza al cmdlet a copiar los objetos ocultos o de solo lectura.
-Path <String[]>	Ruta de acceso del objeto que se ha de copiar.
-Recurse	Especifica una copia recursiva.

Ejemplo 1: copia de un archivo

El siguiente comando copia el archivo Audit.vbs en la carpeta C:\Temp2:

```
PS C:\Windows\system32> Copy-Item -Path C:\Temp\Audit.vbs
-Destination C:\Temp2
```

Ejemplo 2: copia de un árbol de carpetas completo

Otro ejemplo: el siguiente comando copia el contenido de la carpeta C:\Temp en C:\Temp2. Si existen subcarpetas en la carpeta de origen, el conjunto también se copiará, y el árbol de carpetas y archivos se preservará:

```
PS C:\Windows\system32> Copy-Item -Path C:\Temp\* -Destination
C:\Temp2 -Recurse
```

Ejemplo 3: copia de un árbol de carpetas incluyendo la carpeta raíz

El resultado del siguiente comando es prácticamente idéntico al anterior. Sin embargo, la carpeta Temp se incluye en la copia. Esto produce el resultado final C:\Temp2\Temp:

```
PS C:\Windows\system32> Copy-Item -Path C:\Temp\ -Destination
C:\Temp2 -Recurse
```

Ejemplo 4: copia de un archivo renombrándolo

Copy-Item permite renombrar un archivo copiándolo. Basta con especificar el nuevo nombre del archivo en el campo de destino del comando:

```
PS C:\Windows\system32> Copy-Item -Path C:\Temp\Audit.vbs
-Destination C:\Temp2\Audit_v2.vbs
```

Como con **New-Item**, si existe en la carpeta de destino algún archivo con el mismo nombre que el archivo indicado, la copia no se lleva a cabo. El uso del parámetro **-Force** tiene como resultado eliminar los archivos existentes.

g. Rename-Item

Rename-Item permite renombrar un objeto. Hay que especificar el archivo (con su ruta de acceso relativa o absoluta) que desea renombrar y definir el nuevo nombre:

Parámetro	Descripción
-Force	Autoriza al cmdlet a renombrar los objetos ocultos o de solo lectura.
-NewName <String>	Nuevo nombre con el que se renombra el objeto.
-Path <String>	Ruta de acceso del objeto que se ha de renombrar.

Ejemplo: renombrar un archivo

```
PS C:\Temp> Rename-Item -Path .\ScriptPowerShell.ps1 -NewName
Migracion.ps1
```

h. Move-Item

Move-Item permite mover un archivo o una carpeta a otra ubicación. En primer lugar, hay que especificar el objeto o los objetos que se han de mover y, a continuación, el contenedor al que se moverán.

Parámetro	Descripción
-Destination <String>	Ruta de acceso del contenedor a donde se moverá el objeto.
-Force	Autoriza al cmdlet a escribir sobre un objeto existente, incluso aunque sea de solo lectura.
-Path <String[]>	Ruta de acceso del objeto que se ha de mover.

Ejemplo 1: mover un archivo

```
PS C:\Windows\system32> Move-Item -Path .\Migracion.ps1
-Destination C:\Temp
```

Ejemplo 2: forzar el desplazamiento de todos los archivos .ps1

Si en la carpeta de destino existen archivos con el mismo nombre que los presentes en la carpeta de origen, estos archivos no se mueven. Si desea reemplazarlos, debe especificar el parámetro **-Force**:

```
PS C:\Windows\system32> Move-Item -Path C:\Scripts\*.ps1
-Destination C:\Temp -Force
```

Ejemplo 3: mover y renombrar

Move-Item también puede renombrar un archivo moviéndolo. En este caso, basta con especificar la ruta de destino con el nuevo nombre del archivo:

```
PS C:\Windows\system32> Move-Item C:\Scripts\mig_v1.ps1
-Destination C:\Temp\mig_v2.ps1
```

i. Invoke-Item

Este cmdlet permite abrir un archivo y realizar su acción por defecto (ejecutando la aplicación asociada a la extensión del archivo). Por ejemplo, si se trata de un archivo ejecutable, se ejecuta. Para un archivo .doc, se abre la aplicación de

procesamiento de textos asociada a la extensión .doc y se muestra el documento.

Parámetro	Descripción
-Path <String[]>	Ruta de acceso del objeto que se ha de abrir o ejecutar.

Ejemplo: lanzar un archivo ejecutable

En el siguiente ejemplo, **Invoke-Command** abre la herramienta Conexión a escritorio remoto.

```
PS C:\Windows\system32> Invoke-Item C:\Windows\system32\mstsc.exe
```

Pero, en realidad, **Invoke-Item** no es obligatorio. Puede escribir, en efecto, `C:\Windows\system32\mstsc.exe` en Windows PowerShell, y se abrirá la aplicación. Ocurre lo mismo con cualquier otro documento.

El potencial de **Invoke-Item** reside en el hecho de poder abrir varios archivos con una única línea de comandos. Para ello, existen dos posibilidades:

- La primera es indicar los archivos que se han de abrir (separados por una coma).
- La segunda, utilizar el carácter comodín * (*wildcard*).

Ejemplo: abrir una colección de archivos

La siguiente línea de comandos abre el conjunto de archivos .txt presentes en la carpeta C:\Temp con el editor de archivos de texto configurado por defecto en su puesto de trabajo.

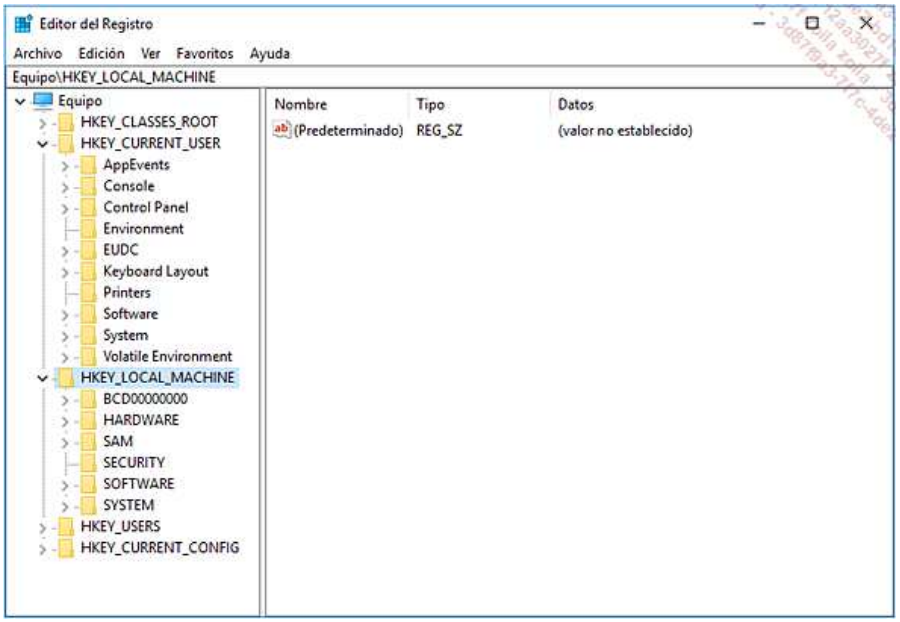
```
PS C:\Windows\system32> Invoke-Item C:\Temp\*.txt
```

El registro de Windows (HKCU y HKLM)

Los cmdlets descritos anteriormente permiten navegar y también realizar acciones sobre los sistemas de archivos. Sin embargo, no se limitan solo a las unidades de sistemas de archivos. En efecto, es posible interactuar con el registro de Windows utilizando estos mismos cmdlets.

Cualquier administrador de sistemas conoce el ejecutable regedit.exe (Registry Editor), que permite abrir la interfaz gráfica para navegar por el registro de Windows y realizar modificaciones (mantenimiento, comportamiento de aplicaciones, reparación del sistema, etc.).

He aquí algunos ejemplos de comandos que pueden ejecutarse en Windows PowerShell.



Editor del Registro de Windows

1. Moverse por el registro

Además de navegar por el sistema de archivos, es posible navegar por el registro mediante el comando **`Set-Location`**:

Por último, **`Get-ChildItem`** permite enumerar el conjunto de

```
PS C:\Windows\system32> Set-Location HKCU:
PS HKCU:\>
```

claves del registro presentes en la raíz HKEY_CURRENT_USER, así como los valores (columna `Property`) incluidos en cada una de ellas:

```
PS HKCU:\> Get-ChildItem


    Hive: HKEY_CURRENT_USER


Name                           Property
----                           --------
AppEvents
Console                        HistoryNoDup          : 0
                               FullScreen            : 0
                               ScrollScale           : 1
                               ExtendedEditKeyCustom : 0
                               CursorSize            : 25
                               FontFamily            : 0
                               ScreenColors          : 7
                               TrimLeadingZeros      : 0
                               WindowSize            : 1638480
[...]
```

De este modo, con estos cmdlets puede navegar por el registro de manera idéntica a como lo hace por el sistema de archivos, y explorar así el conjunto de una manera unificada con la interfaz de Windows PowerShell.

Sin embargo, el registro no se limita únicamente a la navegación. Veamos el conjunto de acciones y manipulaciones que pueden realizarse a través de la línea de comandos. Estas son comparables a las vistas anteriormente.

2. Crear una clave del registro

Empezaremos creando una nueva clave del registro con **`New-Item`**:

```
PS C:\Windows\system32> New-Item HKCU:\Temp


    Hive: HKEY_CURRENT_USER


Name                           Property
----                           --------
Temp
```

Se ha creado la clave del registro HKEY_CURRENT_USER\Temp.

Como puede observar en el ejemplo anterior, no es necesario estar en el interior de la unidad de datos HKCU o HKLM para poder manipular los objetos del registro.

3. Crear un valor del registro

Veamos ahora cómo crear un nuevo valor en HKCU:\Temp. Un valor del registro se compone de tres elementos:

- un nombre (nombre del valor),
- un tipo (tipo de valor),
- un dato (datos del valor).

La creación de valores en el registro de Windows se lleva a cabo mediante el cmdlet **`New-ItemProperty`**:

```
PS C:\Windows\system32> New-ItemProperty -Path HKCU:\Temp -Name
reg1
-Type String -Value 'Contoso'


reg1        : Contoso
PSPath                                                          :
Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Temp
PSParentPath:
Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER
PSChildName : Temp
PSDrive     : HKCU
PSProvider  : Microsoft.PowerShell.Core\Registry
```

Como con el sistema de archivos, debe crear previamente la clave del registro que contendrá el nuevo valor si esta no existe.

Observe aquí la aparición del nombre ItemProperty en el cmdlet. Describe las propiedades de un objeto (en este caso: los valores) si el objeto (Item) es una clave del registro.

Para comprender mejor los distintos parámetros de **`New-ItemProperty`**, he aquí una tabla explicativa:

Parámetro	Descripción
`-Name` `<String>`	Define el nombre de la nueva propiedad.

Parámetro	Descripción
-Path <String[]>	Especifica la ruta de acceso al objeto.
-Type <String>	Indica el tipo de propiedad que se va a agregar. Consulte la siguiente tabla para conocer los distintos tipos posibles.
-Value <Object>	Especifica el valor de la propiedad.

En el ejemplo anterior, el nuevo valor creado es de tipo cadena. Sin embargo, los valores no se limitan únicamente a este tipo en el registro. He aquí una tabla que resume los distintos tipos de valores que es posible crear, y la palabra clave correspondiente que debe utilizarse en Windows PowerShell:

Tipo de valor	Correspondencia en PowerShell
Valor cadena	String
Valor binario	Binary
Valor DWORD (32 bits)	DWORD
Valor QWORD (64 bits)	QWORD
Valor de cadenas múltiples	MultiString
Valor de cadena extensible	ExpandString

4. Recuperar el dato de un valor del registro

Windows PowerShell v5 incluye un nuevo cmdlet que permite recuperar directamente el dato de un valor del registro sin tener que utilizar la notación por puntos vista hasta ahora.

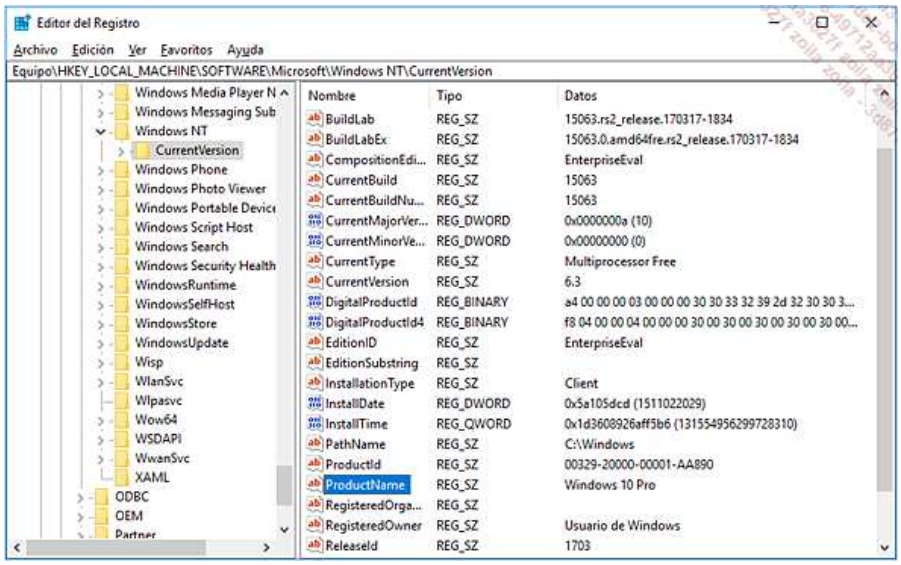
Este cmdlet, **Get-ItemPropertyValue**, se utiliza con dos parámetros:

Parámetro	Descripción
-Name <String[]>	Especifica el nombre de la propiedad.
-Path <String[]>	Especifica la ruta de acceso al objeto.

Ejemplo: recuperar el dato de un valor del registro

```
PS C:\Windows\system32> Get-ItemPropertyValue -Path
"HKLM:\SOFTWARE\Microsoft\Windows      NT\CurrentVersion"      -Name
ProductName
Windows 10 Pro
```

En este ejemplo, `ProductName` representa el nombre del sistema operativo Windows instalado en el puesto de trabajo. El valor del registro `ProductName` se devuelve correctamente, conforme a lo que se puede encontrar con el editor del registro.



Lista de valores presentes en la clave del registro CurrentVersion

Existen numerosos valores presentes en el registro que le permiten recuperar información importante, que puede utilizarse durante la elaboración de scripts.

5. Borrar el dato de un valor del registro

`Clear-ItemProperty` permite borrar el dato de un valor sin eliminar este último.

Parámetro	Descripción
`-Name` `<String>`	Designa el nombre de la propiedad. El dato contenido se elimina.
`-Path` `<String[]>`	Especifica la ruta de acceso a la propiedad.

Ejemplo: borrar el dato de un valor del registro

```
PS C:\Windows\system32> Clear-ItemProperty -Path HKCU:\Temp
-Name reg1
```

Así, el valor reg1 sigue existiendo, pero no hay ningún dato presente.

6. Renombrar un valor del registro

El cmdlet **Rename-ItemProperty** se utiliza para renombrar un valor.

Parámetro	Descripción
-Name <String>	Designa el nombre de la propiedad.
-NewName <String>	Indica el nuevo nombre que se da a la propiedad.
-Path <String>	Especifica la ruta de acceso a la propiedad.

Ejemplo: renombrar un valor del registro

```
PS C:\Windows\system32> Rename-ItemProperty -Path HKCU:\Temp
-Name reg1 -NewName reg2
```

El nombre del valor reg1 pasa a ser reg2.

7. Mover un valor del registro

Para mover un valor, hay que llamar a **Move-ItemProperty**.

Parámetro	Descripción
-Destination <String>	Especifica la ruta de acceso a la clave del registro donde se mueve la propiedad.
-Name <String[]>	Indica el nombre de la propiedad.
-Path <String[]>	Especifica la ruta de acceso a la propiedad.

Ejemplo: mover un valor del registro

```
PS C:\Windows\system32> Move-ItemProperty -Path HKCU:\Temp
-Name reg1 -Destination HKCU:\Temp2
```

La clave del registro de destino debe existir. Si no se ha creado previamente, se obtiene un error.

El valor reg1 que estaba presente en la clave del registro HKCU:\Temp se encuentra en HKCU:\Temp2.

8. Copiar un valor del registro

Copy-ItemProperty permite copiar un valor. Como en los ejemplos anteriores, debe existir la clave del registro de destino.

Parámetro	Descripción
-Destination <String>	Especifica la ruta de acceso a la clave del registro donde se copia la propiedad.
-Name <String>	Indica el nombre de la propiedad
-Path <String[]>	Especifica la ruta de acceso a la propiedad.

Ejemplo: copiar un valor del registro

```
PS C:\Windows\system32> Copy-ItemProperty -Path HKCU:\Temp2
-Name reg1 -Destination HKCU:\Temp
```

El valor reg1, presente en HKCU:\Temp2, se copia en HKCU:\Temp.

9. Eliminar un valor del registro

Para eliminar un valor en el registro, hay que ejecutar el cmdlet **Remove-ItemProperty** pasándole como parámetro la ruta de acceso de la clave del registro y, a continuación, el nombre del valor que se ha de eliminar.

Parámetro	Descripción
-Name <String[]>	Indica el nombre de la propiedad.
-Path <String[]>	Especifica la ruta de acceso a la propiedad.

Ejemplo: eliminar un valor del registro

```
PS C:\Windows\system32> Remove-ItemProperty -Path HKCU:\Temp -Name
reg1
```

El valor reg1, presente en HKCU:\Temp antes de ejecutar el comando, se ha eliminado.

10. Eliminar una clave del registro

Para eliminar una clave del registro, hay que invocar al cmdlet **Remove-Item**, indicando simplemente la ruta de acceso a la clave del registro:

```
PS C:\Windows\system32> Remove-Item HKCU:\Temp
```

En caso de que existan subclaves, se pide una confirmación de la eliminación. Una vez validada, la clave de registro Temp y sus subclaves se eliminan. Puede agregar el parámetro **-Recurse** para omitir la petición de confirmación.

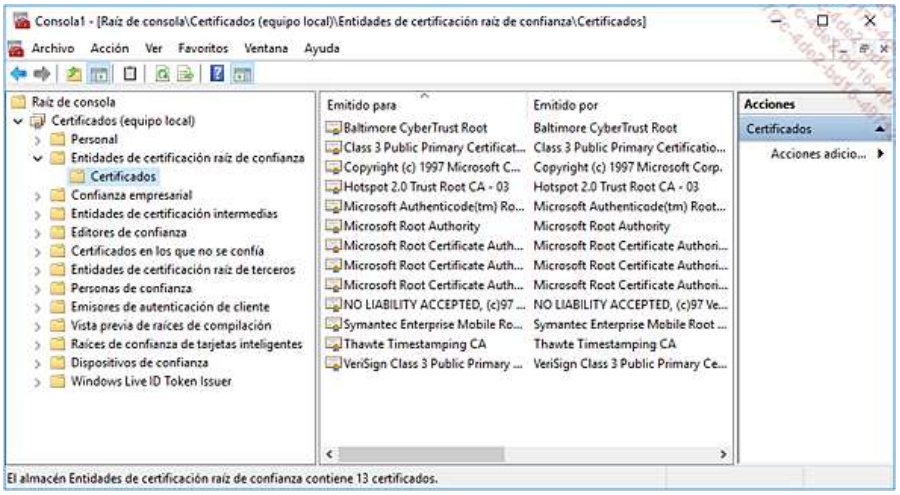
Si solo existen valores en la clave del registro, no se pide ninguna confirmación de la eliminación.

Preste atención: la eliminación de la clave del registro elimina todos los valores y todas las subclaves. ¡Esta acción es irreversible! Sea muy prudente cuando manipule las claves y los valores del registro de Windows.

Por supuesto, los cmdlets **Copy-Item**, **Move-Item** y **Rename-Item** también funcionan, y permiten respectivamente copiar, mover y renombrar una clave del registro.

Los certificados

Igual que con el sistema de archivos y el registro, la gestión de certificados puede llevarse a cabo a través de Windows PowerShell. Esto se ha mejorado enormemente desde Windows 8, con la creación de cmdlets suplementarios dedicados. Algunos cmdlets que se abordan aquí no están disponibles en Windows 7.



La consola MMC con el componente Certificados

1. Gestionar los certificados con Windows PowerShell

La gestión de certificados también puede hacerse a través de Windows PowerShell. Gracias al acceso a la unidad de certificados (Cert:), es posible interactuar directamente con los certificados existentes, agregar nuevos y también eliminarlos.

El acceso a los certificados con Windows PowerShell se hace de la misma manera que con el registro:

```
PS C:\Windows\system32> Set-Location Cert:
PS Cert:\> Get-ChildItem


Location   : CurrentUser
StoreNames: {Local NonRemovable Certificates, SmartCardRoot, Root,
Trust...}

Location   : LocalMachine
```

```
StoreNames: {TrustedPublisher, ClientAuthIssuer, Remote Desktop,
Root...}
```

Como muestra el ejemplo de código anterior, es posible gestionar el conjunto de certificados correspondientes al puesto de trabajo (`LocalMachine`), y también los correspondientes al usuario autenticado en la máquina (`CurrentUser`). Los ejemplos y explicaciones que se proveen en esta sección corresponden a los certificados de equipo.

He aquí un resumen de los distintos almacenes de certificados:

```
PS Cert:\> Set-Location .\LocalMachine
PS Cert:\LocalMachine> Get-ChildItem


Name: TrustedPublisher

Name: ClientAuthIssuer

Name: Remote Desktop

Name: Root

Name: TrustedDevices

Name: WebHosting

Name: CA

Name: Windows Live ID Token Issuer

Name: AuthRoot

Name: FlightRoot

Name: TrustedPeople

Name: My

Name: SmartCardRoot

Name: Trust

Name: Disallowed
```

Para establecer cierto paralelismo con las carpetas presentes en la consola MMC, he aquí una tabla resumen:

Nombre en PowerShell	MMC
`TrustedPublisher`	Editores de confianza
`ClientAuthIssuer`	Emisores de autenticación de cliente
`Remote Desktop`	Escritorio remoto

Nombre en PowerShell	MMC
`Root`	Entidades de certificación raíz de confianza
`TrustedDevices`	Dispositivos de confianza
`WebHosting`	Hospedaje web
`CA`	Entidades de certificación intermedias
`Windows Live ID Token Issuer`	Windows Live ID Token Issuer
`AuthRoot`	Entidades de certificación raíz de terceros
`FlightRoot`	Vista previa de raíces de compilación
`TrustedPeople`	Personas de confianza
`My`	Personal
`SmartCardRoot`	Raíces de confianza de tarjetas inteligentes
`Trust`	Confianza empresarial
`Disallowed`	Certificados no permitidos

La siguiente tabla detalla el rol de los almacenes de certificados más importantes:

Almacén de certificados	Descripción
Personal	Certificados asociados a las claves privadas que usted posee, y a las que tiene acceso.
Entidades de certificación raíz de confianza	Contiene el conjunto de entidades de certificación en las que se confía de manera implícita. Agrupa los certificados emitidos por Microsoft, su empresa y todos los presentes en el almacén de certificados «Entidades de certificación raíz de terceros».
Confianza empresarial	Lista de certificados de confianza empresarial aprobados y autofirmados por otras organizaciones.

Almacén de certificados	Descripción
Entidades de certificación intermedias	Certificados entregados a las entidades de certificación secundarias.
Editores de confianza	Certificados emitidos por entidades de certificación. Estos se aprueban mediante las directivas de restricción de aplicaciones.
Certificados no permitidos	Certificados en los que no se confía de forma explícita.
Entidades de certificación raíz de terceros	Certificados raíz de confianza que se emiten por entidades de certificación diferentes a Microsoft o su empresa.
Personas de confianza	Certificados autofirmados o aprobados explícitamente, entregados a personas o a entidades finales.
Raíces de confianza de tarjetas inteligentes	Certificados raíz de confianza autorizados a emitir certificados sobre tarjetas inteligentes.

Para obtener información suplementaria sobre un certificado concreto, puede pasar el objeto (un certificado, en este caso) recuperado con **`Get-Item`** al cmdlet **`Format-List`**. Puede, con este último, ver las propiedades contenidas en el objeto fila a fila. He aquí un ejemplo, con un certificado autofirmado por el servidor web IIS:

```
PS                    Cert:\LocalMachine\My>                      Get-
Item .\E02F9E498A1C5C81A935840257004B4933994247 |
Format-List -Property *


PSPath                                                                :
Microsoft.PowerShell.Security\Certificate::LocalMachine\

 My\E02F9E498A1C5C81A935840257004B4933994247
PSParentPath            : Microsoft.PowerShell.Security\
                          Certificate::LocalMachine\My
PSChildName                                                           :
E02F9E498A1C5C81A935840257004B4933994247
PSDrive                 : Cert
PSProvider                                                            :
Microsoft.PowerShell.Security\Certificate
PSIsContainer           : False
EnhancedKeyUsageList            :   {Authenticación   del   servidor
(1.3.6.1.5.5.7.3.1)}
```

```
DnsNameList             : {LAB-WS01}
SendAsTrustedIssuer     : False
EnrollmentPolicyEndPoint: Microsoft.CertificateServices.Commands.
                          EnrollmentEndPointProperty
EnrollmentServerEndPoint: Microsoft.CertificateServices.Commands.
                          EnrollmentEndPointProperty
PolicyId                :
Archived                : False
Extensions              : {System.Security.Cryptography.Oid,
                          System.Security.Cryptography.Oid}
FriendlyName            : LAB-WS01 IIS Certificat
IssuerName                                                      :
System.Security.Cryptography.X509Certificates.
                          X500DistinguishedName
NotAfter                : 25/06/2015 02:00:00
NotBefore               : 25/06/2014 17:04:37
HasPrivateKey           : True
PrivateKey                                                      :
System.Security.Cryptography.RSACryptoServiceProvider
PublicKey                                                       :
System.Security.Cryptography.X509Certificates.PublicKey
RawData                 : {48, 130, 2, 202...}
SerialNumber            : 1A4151F2F975DCB242776CFE26CBD72F
SubjectName                                                     :
System.Security.Cryptography.X509Certificates.
                          X500DistinguishedName
SignatureAlgorithm      : System.Security.Cryptography.Oid
Thumbprint                                                      :
E02F9E498A1C5C81A935840257004B4933994247
Version                 : 3
Handle                  : 699913600448
Issuer                  : CN=LAB-WS01
Subject                 : CN=LAB-WS01
```

La propiedad `FriendlyName` devuelve el nombre del certificado, que permite identificarlo más fácilmente. Sin embargo, el único identificador verdadero de un certificado es su firma digital `Thumbprint`. Este identificador es un número único creado por una función de hash durante la firma del certificado por parte de la entidad de certificación. Es por ello por lo que podemos tener varios certificados con el mismo nombre, pero es teóricamente imposible tener dos certificados con la misma firma digital. Por ello, este es el único identificador que permite administrar los certificados con Windows PowerShell.

En la unidad de certificados, las acciones están más limitadas que en las otras dos unidades vistas anteriormente. En lo que respecta a los almacenes de certificados, es posible crear nuevos y eliminarlos, pero en ningún caso renombrarlos o moverlos.

En lo que respecta a los propios certificados, debido a su seguridad, tampoco es posible realizar ciertas acciones: copiarlos (**`Copy-Item`**), renombrarlos (**`Rename-Item`**), borrar su contenido (**`Clear-Item`**). Sin embargo, sí es posible crear

nuevos (aunque no con **New-Item**), moverlos (**Move-Item**) y eliminarlos (**Remove-Item**).

Si intenta realizar alguna de las acciones no permitidas, Windows PowerShell se lo indica con claridad:

```
PS Cert:\LocalMachine\My> Copy-Item
.\E02F9E498A1C5C81A935840257004B4933994247
Cert:\LocalMachine\WebHosting
Copy-Item: Se ha detenido la ejecución del proveedor porque éste
no admite  esta operación.
```

2. Crear certificados autofirmados

El cmdlet **New-SelfSignedCertificate** permite crear certificados autofirmados. Este comando puede utilizarse de dos formas: bien creando un nuevo certificado, o bien clonando un certificado existente.

a. Crear un nuevo certificado

Para crear un nuevo certificado autofirmado, hay que especificar los parámetros que se indican en la siguiente tabla:

Parámetro	Descripción
-CertStoreLocation <String>	Indica la ubicación del almacén de certificados donde se crea el nuevo certificado.
-DnsName <String>	Especifica uno o varios nombres DNS para informar en el objeto «Otro nombre» del certificado.

Ejemplo: creación de un certificado autofirmado

```
PS Cert:\LocalMachine> New-SelfSignedCertificate -DnsName
LAB-WS01.CONTOSO.LOCAL -CertStoreLocation Cert:\LocalMachine\My


    Directorio:
Microsoft.PowerShell.Security\Certificate::LocalMachine\My


Thumbprint                                Subject
----------                                -------
737F0E5432EE51957FB91E93B6AC18E2A8D566ED                    CN=LAB-
WS01.CONTOSO.LOCAL
```

b. Clonar un certificado

Durante el clonado de un certificado existente, el nuevo certificado clonado poseerá los mismos atributos que el original, a excepción de la clave pública: esta se regenerará. He aquí los parámetros utilizados para realizar esta operación:

Parámetro	Descripción
-CertStoreLocation <String>	Indica la ubicación del almacén de certificados donde se crea el nuevo certificado.
-CloneCert <Certificate>	Identifica el certificado que se va a clonar durante la creación de un nuevo certificado.

Ejemplo: clonado de un certificado

```
PS Cert:\LocalMachine> $originalCert = Get-Item
.\my\737F0E5432EE51957FB91E93B6AC18E2A8D566ED
PS Cert:\LocalMachine> New-SelfSignedCertificate -CloneCert
$originalCert -CertStoreLocation .\My


    Directorio:
Microsoft.PowerShell.Security\Certificate::LocalMachine\My


Thumbprint                                Subject
----------                                -------
FD7396FDE480F72140E65F9A8BE81F0F801C86FD                    CN=LAB-
WS01.CONTOSO.LOCAL
```

En los dos ejemplos anteriores, el parámetro **-CertStoreLocation** no es obligatorio si ya se encuentra en el almacén de certificados Personal (`.\My`): los certificados se crean directamente en este almacén. Se trata de un paso obligatorio para cualquier nuevo certificado generado con el cmdlet **New-SelfSignedCertificate**. Tras la creación, nada le impide mover los certificados a otro almacén, si así lo desea.

3. Exportar certificados

Debido a la importancia que tienen los certificados, conviene saber exportarlos con Windows PowerShell si desea implementar scripts

de copia de seguridad de los puestos de trabajo. Tomemos como ejemplo una actualización de Windows, una reinstalación completa o un cambio de puestos de trabajo, donde la pérdida de un certificado puede resultar problemática en ciertos casos.

Así, exportar un certificado es algo que adquiere todo el sentido del mundo. Para ello, existe el cmdlet **`Export-Certificate`**. Por supuesto, hay que indicar el certificado que desea exportar y, a continuación, la carpeta a la que se exportará como archivo. También se recomienda encarecidamente indicar el tipo de certificado que se va a exportar. Como mínimo, deben proveerse dos parámetros para el correcto funcionamiento de este comando: **`-Cert`** y **`-FilePath`**. Se detallan en la siguiente tabla:

Parámetro	Descripción
`-Cert` `<Certificate>`	Indica la ruta de acceso al certificado en la unidad de certificados (Cert:).
`-FilePath` `<String>`	Especifica la ruta de acceso a la carpeta y el nombre del archivo donde se exportará el certificado.
`-Type` `<CertType>`	Especifica el tipo de archivo para la exportación del certificado. Consulte la siguiente tabla para conocer los distintos tipos.

Ejemplo 1: exportar un certificado

```
PS C:\Windows\system32> Export-Certificate -Cert
Cert:\LocalMachine\My\FD7396FDE480F72140E65F9A8BE81F0F801C86FD
-FilePath C:\Temp\myCertificate.cer
```

El certificado indicado por el parámetro **`-Cert`** se exporta, a continuación, como archivo en la carpeta C:\Temp, con el nombre myCertificate.cer.

Los tipos de certificados exportados puede ser los siguientes (parámetro **`-Type`**):

Tipo de certificado exportable	Descripción
SST	Se trata de un contenedor de certificados. Microsoft (.sst). Este formato de archivo puede incluir uno o varios certificados.

Tipo de certificado exportable	Descripción
CERT	Archivo en formato .cer que contiene un único certificado codificado en formato DER. Es el valor por defecto cuando se exporta un único certificado.
P7B	Puede contener uno o varios certificados integrados en un archivo en formato PKCS#7 (.p7b).

Ejemplo 2: exportar un conjunto de certificados en un único archivo

Para exportar el conjunto de certificados de un almacén de certificados e incluirlos en un único archivo en formato .sst, escriba:

```
PS C:\Windows\system32> Get-ChildItem -Path Cert:\LocalMachine\
My | Export-Certificate -FilePath C:\Temp\myCertsMachine.sst
-Type SST
```

Exportar un certificado con clave privada

Export-Certificate exporta un certificado a un archivo, pero la clave privada no se incluye con esta operación. Si desea exportar la clave privada, debe utilizar el cmdlet **Export-PfxCertificate**, que permite crear un archivo en formato Personal Information Exchange (.pfx). Por defecto, este comando exporta las propiedades extendidas y toda la cadena de certificados asociados.

Parámetro	Descripción
-Cert <Certificate>	Indica la ruta de acceso al certificado en la unidad de certificados (Cert:).
-FilePath <String>	Especifica la ruta de acceso a la carpeta y el nombre del archivo donde se exportará el certificado.
-Password <SecureString>	Define la contraseña que sirve para proteger el archivo .pfx.

El comando es muy parecido al visto anteriormente, con la única diferencia de una contraseña obligatoria para proteger el

certificado. Contraseña que deberá utilizarse durante la operación de importación. Para ello, hay que definir la contraseña, e indicarla mediante el parámetro **-Password** en la ejecución del comando.

Ejemplo: exportar un certificado con una clave privada

```
PS C:\Windows\system32> $contraseña = ConvertTo-SecureString
"P@$$w0rd!" -AsPlainText -Force
PS C:\Windows\system32> Get-ChildItem
Cert:\LocalMachine\My\FD7396FDE480F72140E65F9A8BE81F0F801C86FD |
Export-PfxCertificate -FilePath C:\Temp\myPfxCert.pfx -Password
$contraseña
```

¡La exportación de certificados resulta muy sencilla con Windows PowerShell! Y, quien dice exportación, dice importación… Veámosla sin más dilación.

4. Importar certificados

La importación de certificados se realiza con el cmdlet **Import-Certificate**. He aquí un resumen de los parámetros disponibles para este cmdlet:

Parámetro	Descripción
-CertStoreLocation <String>	Indica el almacén de certificados donde se importará el certificado.
-FilePath <String>	Especifica la ruta de acceso al archivo de certificado que se va a importar.

Ejemplo 1: importar un certificado

```
PS C:\Windows\system32> Import-Certificate -FilePath
C:\Temp\myCertificate.cer -CertStoreLocation
Cert:\LocalMachine\My
```

Ejemplo 2: importar un conjunto de certificados

Para importar varios certificados que están presentes en una misma carpeta, recupere el contenido de la carpeta y páseselo al cmdlet **Import-Certificate** con un pipe.

```
PS C:\Windows\system32> $certificados = Get-ChildItem -Path
C:\Temp\Certificados
PS C:\Windows\system32> $certificados | Import-Certificate
-CertStoreLocation Cert:\LocalMachine\My
```

Importar un certificado PFX

Utilice el cmdlet **Import-PfxCertificate** para importar certificados en formato PFX. Para llevar a cabo esta operación, debe conocer la contraseña indicada durante la exportación y pasársela al comando usando el parámetro **-Password**.

Parámetro	Descripción
-CertStoreLocation <String>	Indique el almacén de certificados donde se importará el certificado.
-FilePath <String>	Especifica la ruta de acceso al archivo de certificado que se va a importar.
-Password <SecureString>	Define la contraseña que sirve para importar el certificado.

Ejemplo: importar un certificado en formato PFX

```
PS C:\Windows\system32> $contraseña = ConvertTo-SecureString
"P@$$w0rd!" -AsPlainText -Force
PS C:\Windows\system32> Import-PfxCertificate -FilePath
C:\Temp\myPfxCert.pfx Cert:\LocalMachine\My -Password $contraseña
```

Web Services-Management

El almacén de datos WSMan contiene, por su parte, todos los parámetros vinculados con la gestión de Windows Remote Management (WinRM). Se trata de la implementación del protocolo de comunicación WS-Management.

La funcionalidad PowerShell Remoting, aparecida con Windows PowerShell versión 2.0, permite administrar una o varias máquinas remotas mediante WinRM.

Sin embargo, Windows PowerShell y su funcionalidad PowerShell Remoting no son los únicos que explotan WinRM para la administración remota de máquinas. Exchange Server, algunas herramientas de la gama System Center y muchas otras la utilizan; en ciertos casos se producen errores de conexión cuando el administrador desea ejecutar comandos PowerShell en una máquina remota. Uno de los errores más frecuentes es que se ha superado el número de usuarios diferentes conectados a la máquina. En este caso concreto, he aquí el mensaje de error que se obtiene:

```
PS C:\Windows\system32> Enter-PSSession -ComputerName LAB-WS01
enter-pssession: Error de conexión al servidor remoto LAB-WS01.
Mensaje de error: El servicio WS-Management no puede procesar
la solicitud. Se superó el número máximo de usuarios que
ejecutan operaciones remotas para este complemento.
Vuelva a intentar la solicitud más tarde o aumente
la cuota de usuarios simultáneos.
Para obtener más información, consulte
el tema de la Ayuda about_Remote_Troubleshooting.
```

Es posible modificar un parámetro de WS-Management para evitar este mensaje de error en los siguientes intentos de conexión. A continuación, hay que modificar el valor de la propiedad `MaxConcurrent Users`, presente en la unidad de datos WSMan de la máquina remota. Observe que, para acceder a WSMan, el servicio WinRM (Administración remota de Windows (Administración WSM)) de Windows debe estar arrancado, y también debe poseer permisos de administración en el arranque de Windows PowerShell. La propiedad `MaxConcurrent Users` se encuentra en la siguiente ubicación:

```
PS C:\Windows\system32> Set-Location WSMan:
PS WSMan:\> Set-Location .\localhost\Shell
PS WSMan:\localhost\Shell> Get-ChildItem
```

```
   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Shell

Type            Name                          SourceOfValue  Value
----            ----                          -------------  -----
System.String   AllowRemoteShellAccess                       true
System.String   IdleTimeout                                  7200000
System.String   MaxConcurrentUsers                           10
System.String    MaxShellRunTime
2147483647
System.String   MaxProcessesPerShell                         25
System.String   MaxMemoryPerShellMB                          1024
System.String   MaxShellsPerUser                             30
```

Para aumentar el número de usuarios diferentes que pueden conectarse a la vez, hay que modificar el valor de `MaxConcurrentUsers`. Esta acción se lleva a cabo mediante el cmdlet **`Set-Item`**:

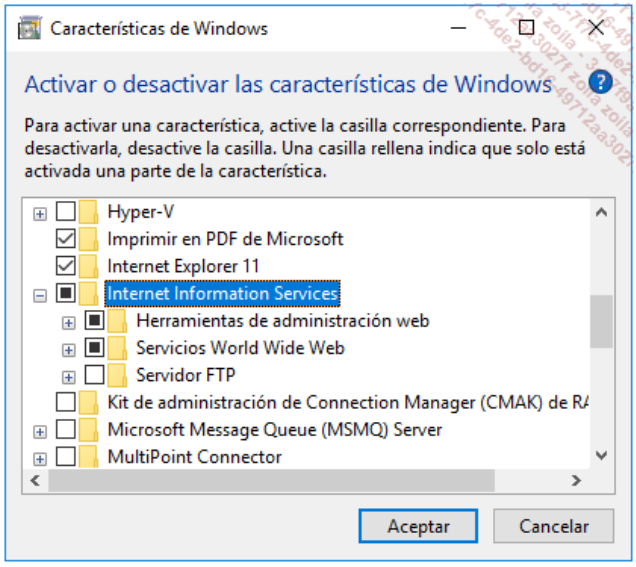
```
PS WSMan:\localhost\Shell> Set-Item .\MaxConcurrentUsers -Value 20
```

Las modificaciones se aplican inmediatamente. No es necesario reiniciar el servicio WinRM.

El conjunto de parámetros de configuración correspondientes a WS-Management sobre el que se basa PowerShell Remoting se encuentra en la unidad WSMan. Es posible modificar varios parámetros; por ejemplo, el tamaño máximo de memoria por interfaz PowerShell (`MaxMemoryPerShellMB`), o incluso el puerto de escucha de WinRM.

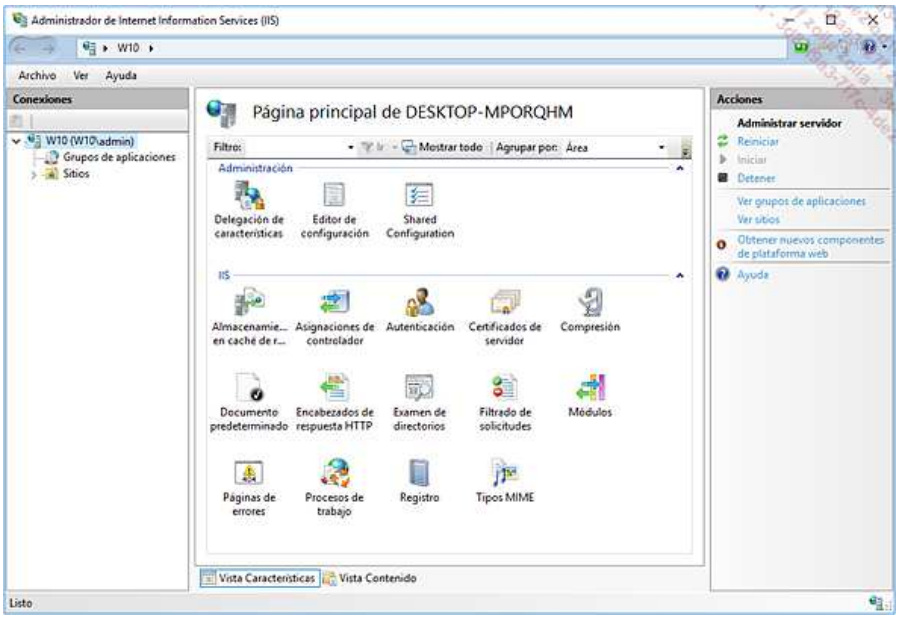
Internet Information Services (IIS)

IIS es un servidor web desarrollado por Microsoft. Está disponible en Windows como una característica para instalar: abra el **Panel de control**, seleccione **Programas y características** (categoría **Programas**) y a continuación, en el menú de la izquierda, seleccione **Activar o desactivar las características de Windows**. La característica que se ha de seleccionar se llama **Internet Information Services**.



Agregar el componente IIS en las características de Windows

Una vez instalado el componente, tiene acceso al **Administrador de IIS** escribiendo **inetmgr.exe**, bien utilizando la búsqueda de Windows (combinación de teclas [Windows] S) o a través de la ventana **Ejecutar** (combinación de teclas [Windows] R). Este administrador es la herramienta que permite administrar y configurar los diferentes sitios web alojados en su puesto.



El Administrador de IIS

Veamos la administración de este entorno con Windows PowerShell. Desde la administración de sitios web hasta la configuración de los parámetros, el conjunto está accesible a través de una unidad Windows PowerShell. En primer lugar, hay que importar el módulo `WebAdministration` para tener acceso al lector de IIS, así como a todos los cmdlets que permiten administrar el servidor web. Para ello, hay que utilizar el cmdlet **`Import-Module`**:

```
PS C:\Windows\system32> Import-Module WebAdministration
```

Para saber si el módulo se ha cargado correctamente, puede utilizar el cmdlet **`Get-Module`**. Este comando devuelve como resultado los módulos cargados en Windows PowerShell.

```
PS C:\Windows\system32> Get-Module

ModuleType  Version      Name
 ExportedCommands
----------- -------     ----                                 ----------
------
Binary      1.0.0.0     CimCmdlets                             {Export-
BinaryMiLog...
Script      3.0          Dism                                    {Add-
AppxProvisione...
Manifest    3.1.0.0       Microsoft.PowerShell.Management       {Add-
Computer, Add-...
Manifest      3.0.0.0        Microsoft.PowerShell.Security
{ConvertFrom-Secure...
```

```
Manifest    3.1.0.0      Microsoft.PowerShell.Utility           {Add-
Member, Add-Ty...
Manifest    3.0.0.0     Microsoft.WSMan.Management           {Connect-
WSMan, Dis...
Manifest    1.0.0.0     NetConnection                          {Get-
NetConnectionP...
Manifest    1.0.0.0     pki                                    {Add-
CertificateEnr...
Script      1.2          PSReadline                            {Get-
PSReadlineKeyH...
Manifest    1.0.0.0     WebAdministration                      {Add-
WebConfigurati...
```

Ahora, si realiza un **Get-PSDrive**, estará disponible el acceso a IIS:

```
PS C:\Windows\system32> Get-PSDrive

Name         Used (GB)     Free (GB)  Provider       Root
----         ---------     ---------  --------       ----
Alias                                 Alias
C                11,84       114,64  FileSystem     C:\
Cert                                  Certificate    \
D                 3,62                FileSystem     D:\
Env                                   Environment
Function                              Function
HKCU                                                  Registry
HKEY_CURRENT_USER
HKLM                                                  Registry
HKEY_LOCAL_MACHINE
IIS                                   WebAdminis...  \\LAB-WS01
Variable                              Variable
WSMan                                 WSMan
X               386,40        13,59  FileSystem     \\server01\D$


PS C:\Windows\system32> Set-Location IIS:
PS IIS:\> Get-ChildItem

Name
----
AppPools
Sites
SslBindings
```

Para conocer el conjunto de cmdlets que se han agregado tras la importación del módulo `WebAdministration`, puede utilizar el cmdlet **Get-Command**, indicando el nombre del módulo mencionado anteriormente al parámetro **-Module**:

```
PS IIS:\AppPools> Get-Command -Module WebAdministration

CommandType     Name                                       ModuleName
-----------     ----                                       ----------
Function           IIS:
 WebAdministration
Cmdlet                Add-WebConfiguration
 WebAdministration
Cmdlet                 Add-WebConfigurationLock
```

```
 WebAdministration
Cmdlet                  Add-WebConfigurationProperty
 WebAdministration
Cmdlet                Backup-WebConfiguration
WebAdministration
Cmdlet                  Clear-WebCentralCertProvider
 WebAdministration
Cmdlet                Clear-WebConfiguration
 WebAdministration
Cmdlet                    Clear-WebRequestTracingSetting
 WebAdministration
[...]
```

Este libro no aborda la administración de IIS, de modo que nos vamos a concentrar únicamente en algunos cmdlets y en las principales acciones, como el despliegue de un nuevo sitio o la copia de seguridad y su restauración. Esto nos dará una idea de la administración de IIS con Windows PowerShell.

1. Interactuar con los pools de aplicaciones

Los pools de aplicaciones están compuestos de una o varias aplicaciones. Están aislados los unos de los otros para permitir definir parámetros de configuración y un nivel de seguridad.

Así, cada pool de aplicaciones puede contener uno o varios sitios web o programas con parámetros concretos, y permisos de acceso específicos. Un sitio debe estar vinculado obligatoriamente a un pool de aplicaciones, o en caso contrario no funcionará.

La creación de un pool de aplicaciones se basa en el cmdlet **`New-WebAppPool`**. Basta con pasarle un nombre como parámetro.

Parámetro	Descripción
`-Name` `<String>`	Nombre del pool de aplicaciones que se ha de crear.

Ejemplo: creación de un pool de aplicaciones

```
PS IIS:> New-WebAppPool -Name "myWebAppPool"

Name                    State        Applications
----                    -----        ------------
myWebAppPool            Started
```

El pool de aplicaciones creado se ejecutará con la cuenta integrada Identidad del grupo de aplicaciones (ApplicationPoolIdentity).

Con la ayuda de varios cmdlets, podrá interactuar con los pools de aplicaciones.

- **Get-WebAppPoolState**: permite obtener el estado del pool de aplicaciones.
- **Start-WebAppPool**: arranca el pool de aplicaciones.
- **Stop-WebAppPool**: detiene el pool de aplicaciones.
- **Restart-WebAppPool**: reinicia el pool de aplicaciones.
- **Remove-WebAppPool**: elimina el pool de aplicaciones.

Su uso es muy sencillo. Como con la creación, basta con indicar el nombre del pool de aplicaciones (**-Name**) para llevar a cabo alguna acción sobre él. He aquí varios ejemplos que permiten ilustrar los cmdlets citados anteriormente.

Ejemplo 1: detener un pool de aplicaciones

Se utilizan dos cmdlets: el primero para detener el pool de aplicaciones, el segundo para recuperar su estado en un instante T.

```
PS IIS:\> Stop-WebAppPool -Name "myWebAppPool"
PS IIS:\> Get-WebAppPoolState -Name "myWebAppPool"

Value
-----
Stopped
```

Ejemplo 2: arrancar un pool de aplicaciones

Lo mismo que en el ejemplo anterior, pero esta vez se trata de un arranque.

```
PS IIS:\> Start-WebAppPool -Name "myWebAppPool"
PS IIS:\> Get-WebAppPoolState -Name "myWebAppPool"

Value
-----
Started
```

Ejemplo 3: reiniciar un pool de aplicaciones

```
PS IIS:\AppPools> Restart-WebAppPool -Name '.\myWebAppPool'
```

Ejemplo 4: eliminar un pool de aplicaciones

Tiene la posibilidad de eliminar un pool de aplicaciones si ya no es necesario.

```
PS IIS:\AppPools> Remove-WebAppPool -Name '.\myWebAppPool'
```

2. Los sitios web

Una vez creado el pool de aplicaciones, puede crear un sitio web. Su rol será alojar el contenido de los distintos archivos necesarios para su visualización y su funcionamiento, y también responder a los clientes que realicen alguna petición para interactuar con el sitio.

Para crear un sitio web, hay que utilizar el cmdlet **`New-Website`**. Son necesarios varios parámetros para crear un nuevo sitio. He aquí un resumen:

Parámetro	Descripción
`-ApplicationPool` `<String>`	Nombre del pool de aplicaciones al que está vinculado el sitio web.
`-Id` `<UInt32>`	Permite definir un ID para el sitio web.
`-HostHeader` `<String>`	Asigna un nombre de host al sitio web.
`-Name` `<String>`	Asigna un nombre al sitio web.
`-PhysicalPath` `<String>`	Ruta de acceso a la carpeta que contiene los recursos del nuevo sitio web.
`-Port` `<UInt32>`	Número de puerto de escucha asignado al sitio web.

Ejemplo: creación de un sitio web

El siguiente ejemplo crea un nuevo sitio web con el conjunto de parámetros descritos.

```
PS IIS:\> New-Website -Name myWebSite -Port 81 -PhysicalPath
C:\inetpub\wwwroot\myWebSite -ApplicationPool 'myWebAppPool'

Name               ID    State      Physical  Path
 Bindings
----               --    -----      -------------                    ------
--
myWebSite          2     Started    C:\inetpub\wwwroot\myWebSite     http
*:81:
```

Como con el pool de aplicaciones, la carpeta que contiene (o contendrá) los archivos necesarios para el sitio web debe existir antes de ejecutar el comando.

Para su información, y como ocurre con los pools de aplicaciones, es posible realizar tareas de administración sobre los sitios web a través de los siguientes cmdlets:

- **`Get-Website`**: obtiene la información vinculada al sitio web.
- **`Get-WebsiteState`**: obtiene el estado del sitio web.
- **`Start-Website`**: arranca el sitio web.
- **`Stop-Website`**: detiene el sitio web.
- **`Remove-Website`**: elimina el sitio web.

Una vez creado el sitio web, puede situar los archivos necesarios para su funcionamiento. En el caso anterior, conoce la ruta de acceso física del sitio, pero, si no la conociera, podría resultar interesante poder recuperarla. Así, a partir de las propiedades de un sitio web, puede encontrarla, gracias a la propiedad `physicalPath`:

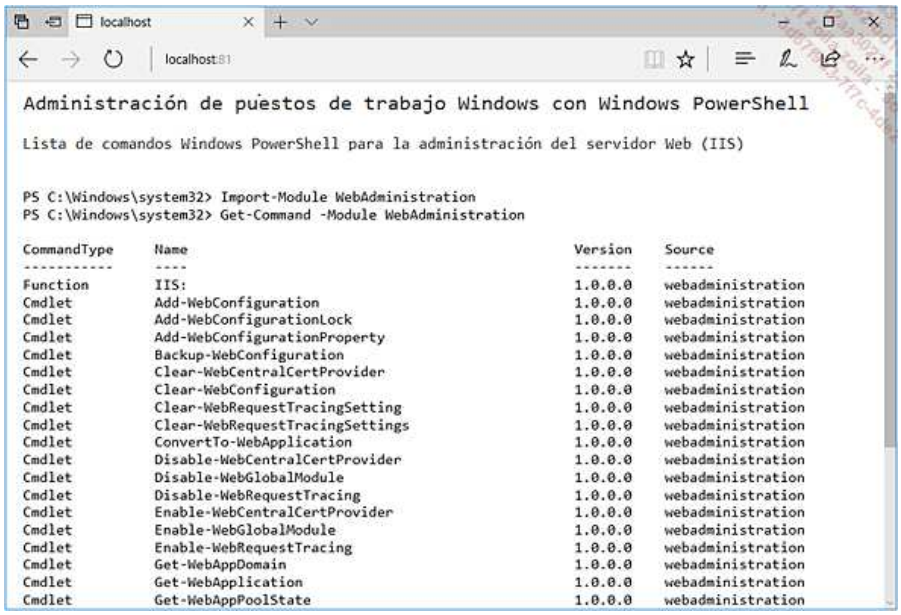
```
PS IIS:\Sites\myWebSite> Get-Website -Name myWebSite | Format-List
*


name                        : myWebSite
id                          : 2
serverAutoStart             : True
state                       : Started
bindings                                                          :
Microsoft.IIs.PowerShell.Framework.Config...
limits                                                            :
Microsoft.IIs.PowerShell.Framework.Config...
logFile                                                           :
Microsoft.IIs.PowerShell.Framework.Config...
traceFailedRequestsLogging:
Microsoft.IIs.PowerShell.Framework.Config...
applicationDefaults                                               :
Microsoft.IIs.PowerShell.Framework.Config...
virtualDirectoryDefaults                                          :
Microsoft.IIs.PowerShell.Framework.Config...
ftpServer                                                         :
Microsoft.IIs.PowerShell.Framework.Config...
Collection
 : {Microsoft.IIs.PowerShell.Framework.Confi...
applicationPool             : myWebAppPool
enabledProtocols            : http
physicalPath                : C:\inetpub\wwwroot\myWebSite
userName                    :
password                    :
```

```
Attributes                : {name, id, serverAutoStart, state}
ChildElements             : {bindings, limits, logFile,
                            traceFailedRequestsLogging...}
ElementTagName            : site
Methods                   : {Start, Stop}
Schema                                                        :
Microsoft.IIs.PowerShell.Framework.Config...
```

Así, para simplemente copiar (o mover) sus datos a la carpeta del sitio web, escriba:

```
PS C:\Windows\system32> Copy-Item -Path C:\Temp\* -Destination
(Get-Website -Name myWebSite).physicalPath -Recurse
```

Una vez copiados o movidos los archivos en la carpeta del sitio web, puede abrir un navegador de Internet e introducir la dirección del sitio. A continuación se mostrará la página de inicio del sitio:



Página de inicio del sitio web myWebSite configurada en IIS

He aquí un resumen de los demás cmdlets, con algunos ejemplos.

Ejemplo 1: detener un sitio web

Como con los pools de aplicaciones, es posible detener un sitio web y recuperar su estado.

```
PS IIS:\> Stop-Website -Name "myWebSite"
PS IIS:\> Get-WebsiteState -Name "myWebSite"

Value
```

```
-----
Stopped
```

Ejemplo 2: arrancar un sitio web

```
PS IIS:\> Start-Website -Name "myWebSite"
PS IIS:\> Get-WebsiteState -Name "myWebSite"

Value
-----
Started
```

Ejemplo 3: eliminar un sitio web

Preste atención: la eliminación del sitio web en el Administrador de IIS es irreversible.

```
PS IIS:\> Remove-Website -Name "myWebSite"
```

Remove-Website elimina el sitio web configurado en el Administrador de IIS. Los archivos utilizados para el funcionamiento del sitio (contenido de la carpeta representada por la propiedad `physicalPath`) no se eliminan.

3. Copia de seguridad y restauración

La copia de seguridad de sitios web es un elemento tan importante como la gestión y la administración en IIS. Esta copia de seguridad se realiza sobre dos partes bien diferentes: la configuración IIS de los pools de aplicaciones y de los sitios web, y los archivos de datos utilizados para el funcionamiento de cada uno de los sitios web.

a. Copia de seguridad de la configuración de IIS

Antes de efectuar modificaciones en IIS, es posible realizar una copia de seguridad de los parámetros para poder volver a un estado previo en caso necesario.

Esto se lleva a cabo mediante el cmdlet **Backup-WebConfiguration**. Se crea, a continuación, una carpeta compartida con el nombre indicado por el parámetro **-Name** en C:\Windows\System32\inetsrv\backup, que contiene el conjunto de la copia de seguridad.

Parámetro	Descripción

Parámetro	Descripción
-Name <String>	Nombre de la carpeta que se creará para almacenar la copia de seguridad.

Ejemplo: copia de seguridad de la configuración de IIS

```
PS IIS:> Backup-WebConfiguration -Name IISBackup_14-07-08
```

En lo relativo a la restauración de los parámetros de IIS, esto se realiza de una manera tan sencilla como la copia de seguridad. Basta con indicar el nombre de una copia de seguridad realizada previamente a través del parámetro **-Name** del cmdlet **Restore-WebConfiguration**.

Ejemplo: restaurar la configuración de IIS

```
PS IIS:> Restore-WebConfiguration -Name IISBackup_14-07-08
```

Se restaura así configuración y los parámetros gracias a la copia de seguridad IISBackup_14-07-08.

b. Copia de seguridad de los datos de los sitios web

Para realizar una copia de seguridad de cada uno de los sitios web en el Administrador de IIS, es posible escribir un bucle que recupere la ruta física de cada uno de los sitios web (physicalPath) y realice una copia de su contenido en un servidor de archivos.

Ejemplo: copia de seguridad de los archivos de los sitios web configurados en IIS

```
PS C:\Windows\system32> Get-ChildItem IIS:\Sites | Foreach
{Copy-Item -Path ((Get-Website -Name
$_.Name).physicalPath).replace("%SystemDrive%","$env:SystemDrive")

-Destination \\servidor01\d$\backup -Recurse -Force}
```

Preste atención: esta copia de seguridad realiza simplemente una copia del contenido de las carpetas representadas por la propiedad physicalPath de los sitios web. Si estos se encuentran vinculados a una base de datos, conviene realizar una copia de seguridad de dicha base de datos.

Introducción

Tras revisar el conjunto de cmdlets básicos y las distintas unidades Windows PowerShell, entremos de lleno en el asunto: la administración de los puestos de trabajo en Windows.

En este capítulo veremos cómo realizar tareas administrativas que afectan directamente al núcleo del sistema Windows: habilitar y deshabilitar características de Windows, administración de los procesos, de los servicios, etc.

Las características de Windows

Con Windows 7 y las versiones anteriores, es posible llevar a cabo tareas de administración desde la consola de comandos con el ejecutable Dism.exe. Es posible instalar actualizaciones, paquetes de idiomas, características de Windows, y mucho más. Además de un ejecutable, Dism está presente también como un módulo en Windows PowerShell. Esto quiere decir que las acciones que pueden realizarse con Dism.exe tienen su equivalente en varios cmdlets.

Es posible consultarlos gracias al siguiente comando:

```
PS C:\Windows\system32> Get-Command -Module Dism

CommandType     Name                                               Version
Source
-----------     ----                                               -------
------
Alias           Add-ProvisionedAppxPackage                         3.0
Dism
Alias           Apply-WindowsUnattend                              3.0
Dism
Alias           Get-ProvisionedAppxPackage                         3.0
Dism
Alias           Remove-ProvisionedAppxPackage                      3.0
Dism
Cmdlet          Add-AppxProvisionedPackage                         3.0
Dism
Cmdlet          Add-WindowsCapability                              3.0
Dism
Cmdlet          Add-WindowsDriver                                  3.0
Dism
Cmdlet          Add-WindowsImage                                   3.0
Dism
Cmdlet          Add-WindowsPackage                                 3.0
Dism
Cmdlet          Clear-WindowsCorruptMountPoint                     3.0
Dism
Cmdlet          Disable-WindowsOptionalFeature                     3.0
Dism
Cmdlet          Dismount-WindowsImage                              3.0
Dism
Cmdlet          Enable-WindowsOptionalFeature                      3.0
Dism
Cmdlet          Expand-WindowsCustomDataImage                      3.0
Dism
Cmdlet          Expand-WindowsImage                                3.0
Dism
Cmdlet          Export-WindowsDriver                               3.0
Dism
Cmdlet          Export-WindowsImage                                3.0
Dism
Cmdlet          Get-AppxProvisionedPackage                         3.0
```

```
Dism
Cmdlet          Get-WIMBootEntry                           3.0
Dism
Cmdlet            Get-WindowsCapability                          3.0
     Dism
Cmdlet          Get-WindowsDriver                          3.0
Dism
Cmdlet          Get-WindowsEdition                         3.0
Dism
Cmdlet          Get-WindowsImage                           3.0
Dism
Cmdlet          Get-WindowsImageContent                    3.0
Dism
Cmdlet          Get-WindowsOptionalFeature                 3.0
Dism
Cmdlet          Get-WindowsPackage                         3.0
Dism
Cmdlet          Mount-WindowsImage                         3.0
Dism
Cmdlet          New-WindowsCustomImage                     3.0
Dism
Cmdlet          New-WindowsImage                           3.0
Dism
Cmdlet          Optimize-WindowsImage                      3.0
Dism
Cmdlet          Remove-AppxProvisionedPackage              3.0
Dism
Cmdlet          Remove-WindowsCapability                   3.0
Dism
Cmdlet          Remove-WindowsDriver                       3.0
Dism
Cmdlet          Remove-WindowsImage                        3.0
Dism
Cmdlet          Remove-WindowsPackage                      3.0
Dism
Cmdlet          Repair-WindowsImage                        3.0
Dism
Cmdlet          Save-WindowsImage                          3.0
Dism
Cmdlet          Set-AppXProvisionedDataFile                3.0
Dism
Cmdlet          Set-WindowsEdition                         3.0
Dism
Cmdlet          Set-WindowsProductKey                      3.0
Dism
Cmdlet          Split-WindowsImage                         3.0
Dism
Cmdlet          Update-WIMBootEntry                        3.0
Dism
Cmdlet          Use-WindowsUnattend                        3.0
Dism
```

A continuación vamos a estudiar algunos cmdlets, comenzando por **Enable-WindowsOptionalFeature** y **Disable-WindowsOptionalFeature**. Algunos cmdlets hablan por sí solos y le invitamos a descubrirlos en la documentación que se proporciona para cada uno de ellos.

Recuerde que es posible consultar la documentación de los cmdlets con el comando **Get-Help**, especificando a continuación el cmdlet

correspondiente. Agregando el parámetro **-Full**, se muestra la documentación completa, generalmente con ejemplos.

```
PS C:\Windows\system32> Get-Help Enable-WindowsOptionalFeature -Full
```

Para instalar una característica de Windows directamente por línea de comandos PowerShell, utilice el cmdlet **Enable-WindowsOptionalFeature**. Pero, antes de utilizar este comando, conviene conocer el nombre de las características que desea instalar en el puesto de trabajo. Es posible consultar la lista de características mediante el cmdlet **Get-WindowsOptionalFeature**:

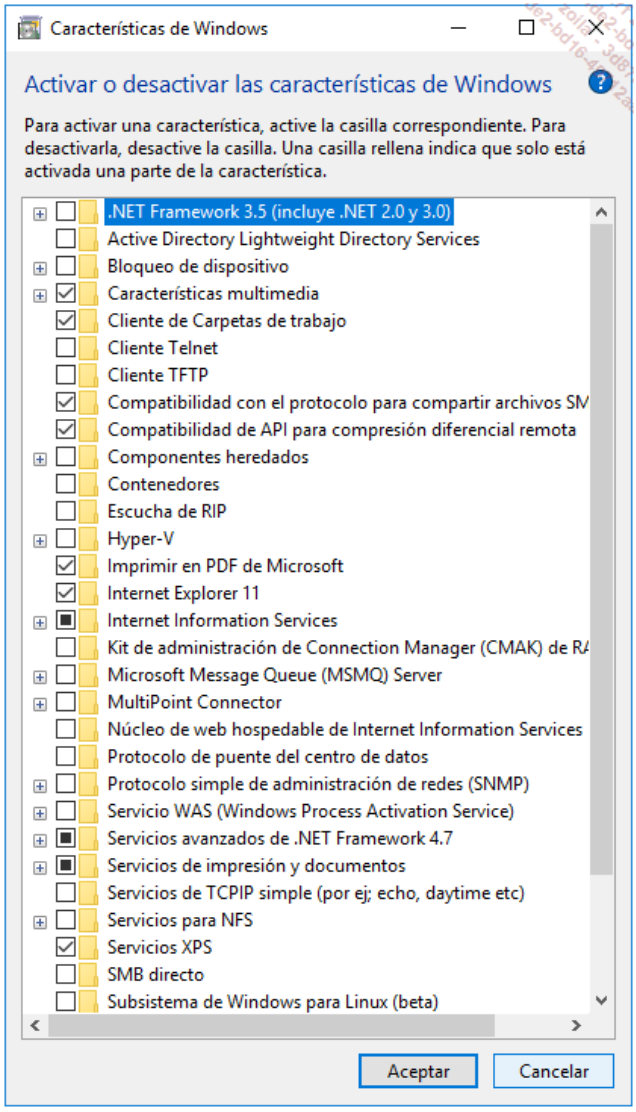
```
PS C:\Windows\system32> Get-WindowsOptionalFeature -Online |
Format-Table

FeatureName
                    State
-----------                                                    ---
--
Microsoft-Windows-Subsystem-Linux
Disabled
LegacyComponents
Disabled
DirectPlay
                                                  Disabled
SimpleTCP
Disabled
SNMP
Disabled
WMISnmpProvider
  Disabled
MicrosoftWindowsPowerShellV2Root
Enabled
MicrosoftWindowsPowerShellV2
Enabled
Windows-Identity-Foundation
Disabled
Internet-Explorer-Optional-amd64
Enabled
NetFx3
DisabledWithPayloadRemoved
IIS-WebServerRole
Disabled
IIS-WebServer
Disabled
[...]
```

El parámetro **-Online** indica el sistema Windows actualmente instalado y en ejecución en el puesto de trabajo. En efecto, Dism también puede utilizarse para la administración de archivos WIM y discos virtuales en escenarios de implementación de Windows.

La lista completa de características se presenta en los Anexos.

El resultado del comando **`Get-WindowsOptionalFeature`** devuelve el conjunto de características de Windows, e indica el estado para cada una de ellas: habilitada (`Enabled`), deshabilitada (`Disabled`) o incluso deshabilitada sin disponer de las fuentes de instalación (`DisableWithPayloadRemoved`).

La lista de características puede consultarse a través de la interfaz gráfica de Windows en la siguiente ubicación: **Panel de control**, **Programas**, **Programas y características**, **Activar o desactivar las características de Windows**.



Lista de características disponibles en Windows 10

1. Activar una característica

La activación de una característica se realiza fácilmente especificando su nombre mediante el parámetro **-FeatureName** del cmdlet **Enable-WindowsOptionalFeature**. He aquí un resumen de los parámetros disponibles:

Parámetro	Descripción
-FeatureName <String[]>	Nombre de la característica que se ha de activar.
-NoRestart	Elimina la petición de reinicio.
-Online	Indica que la acción se realiza en el sistema operativo arrancado en el puesto local.

Ejemplo: activación de una característica

La característica IIS-WebServerRole representa la instalación del servidor web IIS.

```
PS C:\Windows\system32> Enable-WindowsOptionalFeature -Online
-FeatureName IIS-WebServerRole


Path          :
Online        : True
Restart Needed: False
```

Verá, en la parte superior de la pantalla de Windows PowerShell, el progreso de la instalación de la característica:

```
Enable-WindowsOptionalFeature: IIS-WebServerRole
   Running
    [oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
    ]
```

Una vez terminada la instalación, si es necesario realizar un reinicio, Windows PowerShell se lo solicita. Para eliminar esta petición de reinicio, hay que indicar el parámetro **-NoRestart**. Se le muestra un resumen que indica si es necesario o no un reinicio de Windows (Restart Needed).

Es posible activar varias características en una única línea de comandos, indicando las distintas características separadas por una coma.

Caso particular: NetFx3

Para instalar la característica .NET Framework 3.5, los archivos de instalación no están cargados previamente en Windows. De modo que será preciso disponer de una conexión de Internet para poder descargar las fuentes, o bien un medio de instalación de Windows, y utilizar el parámetro **-Source** para indicar la ruta de acceso a la carpeta que contiene los archivos necesarios para activar esta característica.

```
PS C:\Windows\system32> Enable-WindowsOptionalFeature -FeatureName

NetFx3 -Source D:\sources\sxs -Online
```

En la siguiente línea de comandos, la unidad D representa la unidad de CD o el archivo ISO que se ha montado previamente. También puede tratarse de una unidad que disponga de una copia de los archivos de instalación de Windows.

2. Desactivar una característica

La desactivación de una característica de Windows se realiza mediante el cmdlet **Disable-WindowsOptionalFeature**, con los mismos parámetros anteriores, a saber: **-Online** y **-FeatureName**.

Ejemplo: desactivación de una característica

La característica `Microsoft-Hyper-V-All` indica el sistema de virtualización Hyper-V.

```
PS C:\Windows\system32> Disable-WindowsoptionalFeature -Online
-FeatureName Microsoft-Hyper-V-All
¿Desea reiniciar el equipo para completar esta operación ahora?
[Y] Yes  [N] No  [?] Ayuda (el valor predeterminado es "Y"):
```

Como hemos mencionado anteriormente, para omitir las peticiones de reinicio, hay que incluir el parámetro **-NoRestart**. En el caso de un script ejecutado en un equipo remoto, es importante especificar este parámetro, pues de no ser así el script puede quedarse bloqueado en espera de una respuesta.

Se muestra un mensaje de advertencia si es necesario un reinicio del puesto de trabajo para finalizar la operación.

Las aplicaciones de Windows Store

Con la aparición de Windows 8, Microsoft ha implementado una nueva categoría de aplicaciones, más conocida como apps. Sin embargo, este nuevo tipo de aplicaciones no llega solo: Microsoft ha inaugurado una tienda de aplicaciones que permite descargar e instalar nuevas apps. Con Windows 10, Microsoft ha lanzado una plataforma universal, llamada *Universal Windows Platform* (llamada habitualmente UWP), que permite ejecutar la versión PC y móvil de una misma app sobre arquitecturas de procesador muy diferentes (x86 para el PC y ARM para el móvil).

Incluida de manera nativa en Windows 10, la aplicación Store es el repositorio de aplicaciones que permiten, así, instalar en el equipo nuevas apps. Por defecto, Windows 10 autoriza la instalación de nuevas apps solo desde Windows Store. El conjunto de aplicaciones presentes en este repositorio están probadas, validadas y firmadas por el fabricante de Redmond.

Si una empresa desea desplegar una app (por ejemplo, una aplicación de negocio que ha desarrollado de manera interna) no puede permitirse esperar a esta validación, de modo que es posible instalar apps directamente a través de Windows PowerShell. Este procedimiento se denomina SideLoading.

Sin embargo, si desea utilizar SideLoading, puede que necesite una clave de licencia, así como cumplir ciertos requisitos previos a nivel técnico. Consulte la siguiente lista para conocer el conjunto de puntos en función del sistema operativo utilizado.

Windows 8/8.1

- Licencia:
  - Windows 8/8.1 Entreprise:
    - Si está unido a un dominio de Active Directory, no se necesita una licencia suplementaria.
    - Si no está unido a un dominio de Active Directory (WORKGROUP), se requiere una clave de activación SideLoading instalada y activada.
  - Windows 8/8.1 Profesional: se requiere una clave de activación SideLoading instalada y activada.

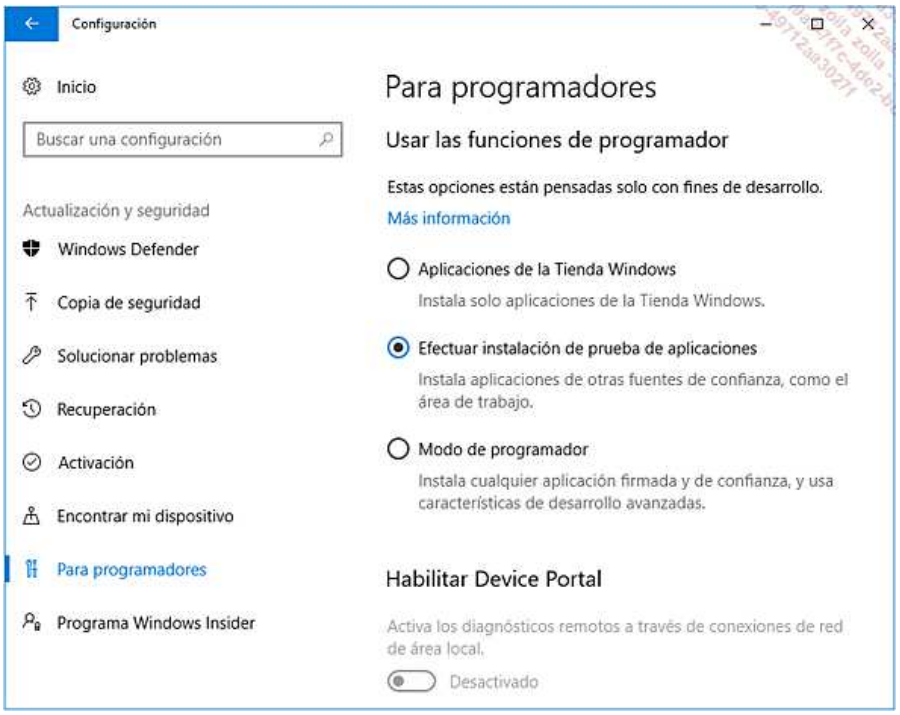
- Activar la directiva **Allow all trusted apps to install**: hay que habilitar la directiva por GPO (o directiva local). Esta directiva se encuentra en la siguiente ubicación: **Plantillas administrativas - Componentes Windows - App Package Deployment**.
- Instalar una app firmada: la aplicación que se va a instalar debe estar firmada por una entidad de certificación reconocida por los clientes, lo que significa que el certificado o la cadena de certificados utilizados para firmar la app debe estar presente en el almacén de certificados «Entidades de certificación raíz de confianza» del equipo local.

Windows 10

- No se requiere ninguna clave de licencia suplementaria, ya sea con Windows 10 edición Profesional o Entreprise.
- Activar uno de los dos modos siguientes: **Sideload apps** o **Modo de programador**. Esta opción se encuentra en **Configuración** - **Actualización y seguridad** - **Para programadores**. El modo **Sideload apps** permite instalar apps provenientes de otras fuentes diferentes a Windows Store.

El modo de desarrollador es más flexible y permite probar las apps durante su desarrollo con Visual Studio, lo que no permite el modo **Sideload Apps**. Este último se recomienda, por tanto, en un entorno de producción.

- Instalar una app firmada: la aplicación a instalar debe estar firmada por una entidad de certificación reconocida por los clientes, lo que significa que el certificado o la cadena de certificados utilizados para firmar la app debe estar presente en el almacén de certificados «Entidades de certificación raíz de confianza» del equipo local.



*La opción **Modo de programador** presente en **Configuración***

Una vez terminado el desarrollo de la aplicación, el entorno de desarrollo Visual Studio genera un paquete que va a crear un archivo con extensión .appx o .appxbundle. Se trata de la app firmada con el certificado correspondiente.

1. Instalar una app

Una vez cubiertos los requisitos previos, basta con utilizar el cmdlet **`Add- AppxPackage`** pasándole como parámetro la ruta de acceso al archivo correspondiente a la app. He aquí el parámetro obligatorio para instalar la app:

Parámetro	Descripción
`-Path` `<String>`	Ruta de acceso al archivo .appx o .appxbundle.

Ejemplo: instalación de una app

```
PS C:\Windows\system32> Add-AppxPackage -Path
C:\Temp\MyUWP_1.0.1.0_x86_x64_arm.appxbundle
```

2. Eliminar una app

Para desinstalar una app, hay que conocer el nombre del paquete correspondiente. Para ello, hay que recuperar en primer lugar la información correspondiente a las apps instaladas en Windows. Esta operación se realiza con el cmdlet **Get-AppxPackage**, que devuelve como resultado la lista de apps instaladas (incluidas aquellas preinstaladas por defecto con Windows).

```
PS C:\Users\Admin> Get-AppxPackage | Select-Object
PackageFullName


---------------
Microsoft.BioEnrollment_10.0.14393.0_neutral__cw5n1h2txyewy
Microsoft.AAD.BrokerPlugin_1000.14393.0.0_neutral_neutral_cw5n1h2tx
yewy
Microsoft.Windows.CloudExperienceHost_10.0.14393.0_neutral_neutral_
cw5n1h2txyewy
Microsoft.Windows.ShellExperienceHost_10.0.14393.206_neutral_neutra
l_cw5n1h2txyewy
windows.immersivecontrolpanel_6.2.0.0_neutral_neutral_cw5n1h2txyewy

Microsoft.Windows.Cortana_1.7.0.14393_neutral_neutral_cw5n1h2txyewy

Microsoft.AccountsControl_10.0.14393.206_neutral__cw5n1h2txyewy
Microsoft.LockApp_10.0.14393.0_neutral__cw5n1h2txyewy
Microsoft.MicrosoftEdge_38.14393.0.0_neutral__8wekyb3d8bbwe
Microsoft.PPIProjection_10.0.14393.0_neutral_neutral_cw5n1h2txyewy

Microsoft.Windows.Apprep.ChxApp_1000.14393.0.0_neutral_neutral_cw5n
1h2txyewy
Microsoft.Windows.AssignedAccessLockApp_1000.14393.0.0_neutral_neut
ral_cw5n1h2txyewy
Microsoft.Windows.ContentDeliveryManager_10.0.14393.0_neutral_neutr
al_cw5n1h2txyewy
Microsoft.Windows.ParentalControls_1000.14393.0.0_neutral_neutral_c
w5n1h2txyewy
Microsoft.Windows.SecondaryTileExperience_10.0.0.0_neutral__cw5n1h2
txyewy
Microsoft.Windows.SecureAssessmentBrowser_10.0.14393.0_neutral_neut
ral_cw5n1h2txyewy
Microsoft.XboxGameCallableUI_1000.14393.0.0_neutral_neutral_cw5n1h2
txyewy
Windows.ContactSupport_10.0.14393.0_neutral_neutral_cw5n1h2txyewy
Windows.MiracastView_6.3.0.0_neutral_neutral_cw5n1h2txyewy
Windows.PrintDialog_6.2.0.0_neutral_neutral_cw5n1h2txyewy
Microsoft.VCLibs.140.00_14.0.24123.0_x64__8wekyb3d8bbwe
Microsoft.VCLibs.140.00_14.0.24123.0_x86__8wekyb3d8bbwe
Microsoft.NET.Native.Framework.1.3_1.3.24201.0_x86__8wekyb3d8bbwe
Microsoft.NET.Native.Framework.1.3_1.3.24201.0_x64__8wekyb3d8bbwe
Microsoft.NET.Native.Runtime.1.3_1.3.23901.0_x86__8wekyb3d8bbwe
Microsoft.NET.Native.Runtime.1.3_1.3.23901.0_x64__8wekyb3d8bbwe
Microsoft.Getstarted_4.1.15.0_x64__8wekyb3d8bbwe
Microsoft.WindowsAlarms_10.1609.2843.0_x64__8wekyb3d8bbwe
Microsoft.WindowsSoundRecorder_10.1608.2211.0_x64__8wekyb3d8bbwe
Microsoft.WindowsPhone_10.1609.2561.0_x64__8wekyb3d8bbwe
Microsoft.People_10.0.11902.0_x64__8wekyb3d8bbwe
Microsoft.Office.OneNote_17.7466.57691.0_x64__8wekyb3d8bbwe
Microsoft.MicrosoftOfficeHub_17.7420.23751.0_x64__8wekyb3d8bbwe
Microsoft.Messaging_3.19.1001.0_x86__8wekyb3d8bbwe
Microsoft.Appconnector_1.3.3.0_neutral__8wekyb3d8bbwe
```

```
Microsoft.XboxIdentityProvider_11.19.19003.0_x64__8wekyb3d8bbwe
Microsoft.DesktopAppInstalar_1.0.2181.0_x64__8wekyb3d8bbwe
Microsoft.WindowsCalculator_10.1608.2213.0_x64__8wekyb3d8bbwe
Microsoft.OneConnect_1.1607.6.0_x64__8wekyb3d8bbwe
Microsoft.XboxApp_19.22.6017.0_x64__8wekyb3d8bbwe
Microsoft.Windows.Photos_16.722.10060.0_x64__8wekyb3d8bbwe
Microsoft.WindowsCamera_2016.816.20.0_x64__8wekyb3d8bbwe
Microsoft.NET.Native.Runtime.1.4_1.4.24201.0_x86__8wekyb3d8bbwe
Microsoft.NET.Native.Runtime.1.4_1.4.24201.0_x64__8wekyb3d8bbwe
Microsoft.MicrosoftStickyNotes_1.1.41.0_x64__8wekyb3d8bbwe
Microsoft.WindowsFeedbackHub_1.1608.2441.0_x64__8wekyb3d8bbwe
Microsoft.WindowsStore_11609.1001.29.0_x64__8wekyb3d8bbwe
Microsoft.ZuneVideo_10.16092.10311.0_x64__8wekyb3d8bbwe
microsoft.windowscommunicationsapps_17.7369.40797.0_x64__8wekyb3d8b
bwe
Microsoft.StorePurchaseApp_11608.1000.2431.0_x64__8wekyb3d8bbwe
Microsoft.WindowsMaps_5.1609.2651.0_x64__8wekyb3d8bbwe
Microsoft.BingNews_4.16.18.0_x86__8wekyb3d8bbwe
Microsoft.3DBuilder_11.1.9.0_x64__8wekyb3d8bbwe
Microsoft.BingFinance_4.16.19.0_x86__8wekyb3d8bbwe
Microsoft.SkypeApp_11.8.197.0_x64__kzf8qxf38zg5c
Microsoft.Advertising.Xaml_10.1609.2.0_x64__8wekyb3d8bbwe
Microsoft.Advertising.Xaml_10.1609.2.0_x86__8wekyb3d8bbwe
Microsoft.MicrosoftSolitaireCollection_3.12.8312.0_x64__8wekyb3d8bb
we
Microsoft.BingSports_4.16.17.0_x86__8wekyb3d8bbwe
Microsoft.BingWeather_4.16.15.0_x86__8wekyb3d8bbwe
Microsoft.ZuneMusic_10.16092.10311.0_x64__8wekyb3d8bbwe
MyUWP.App_1.0.1.0_x64__pyde9gwt8sqkr
```

Una vez encontrado el nombre del paquete correspondiente a la app que desea desinstalar, basta con indicárselo al parámetro **-Name** del cmdlet **Remove-AppxPackage**. A continuación se desinstala la app.

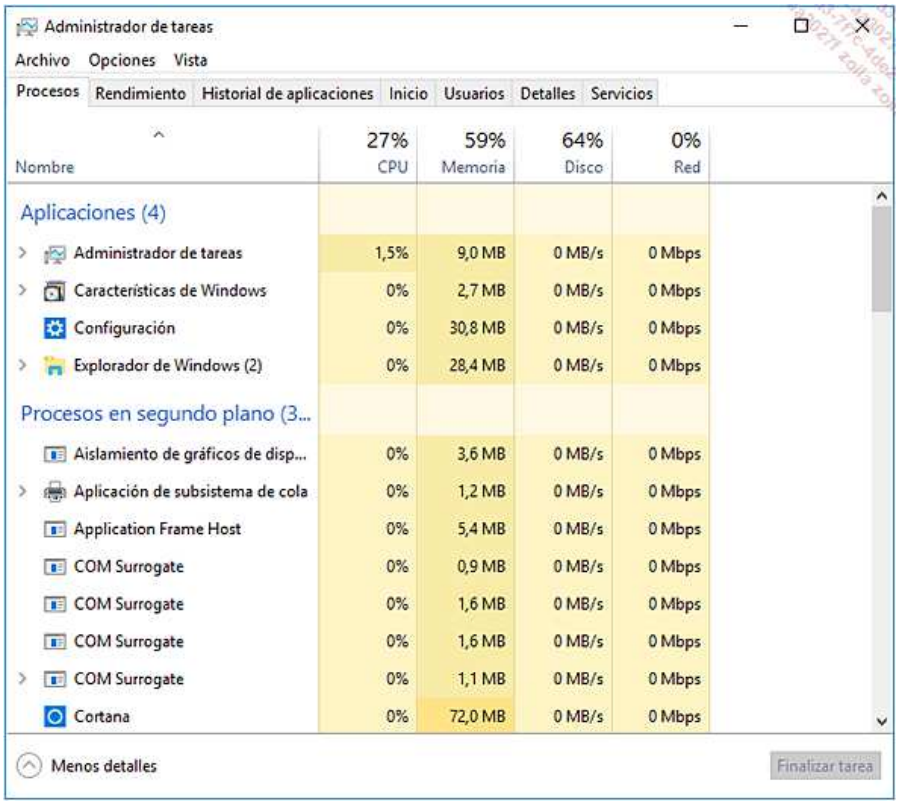
Parámetro	Descripción
-Package <String>	Nombre del paquete correspondiente a la app.

```
PS C:\Windows\system32> Remove-AppxPackage -Package
MyUWP.App_1.0.1.0_x64__pyde9gwt8sqkr
```

Los procesos

La gestión de procesos en los puestos de trabajo es una tarea importante. Se trata de conocer, en todo momento, el conjunto de programas que se ejecutan en Windows y los recursos de hardware (procesador y memoria) que consumen.

Puede realizar las mismas acciones que en el Administrador de tareas de Windows: finalizar o ejecutar nuevos procesos y también recuperar información sobre ellos.



El Administrador de tareas

1. Obtener la lista de procesos

El cmdlet que permite recuperar el conjunto de procesos activos en el puesto de trabajo es **`Get-Process`**. He aquí algunos parámetros que pueden utilizarse con este cmdlet:

Parámetro	Descripción
-FileVersionInfo	Obtiene información acerca de la versión del archivo del programa que se ejecuta en el proceso.
-Id <Int32[]>	Especifica uno o varios procesos por su número de identificador de proceso (PID).
-Name <String[]>	Especifica uno o varios procesos por su nombre.

Ejemplo: obtener la lista de todos los procesos activos en el puesto local

```
PS C:\Windows\system32> Get-Process

Handles   NPM(K)      PM(K)          WS(K)        CPU(s)        Id   SI
ProcessName
-------  ------     -----       -----      ------      --  -- ---------
--
    230      14     7348       13524        0,11   5396   0 audiodg
     225      15       4820         17952          0,08    1564    2
backgroundTaskHost
    168      12     3372       14240        0,16   5588   2 conhost
    288      12     1364        3944        0,23    384   0 csrss
    125       9     1384        3984        0,11    468   1 csrss
    232      12     1384        4648        0,44   1396   2 csrss
    328      17    11684       28612        0,16    848   1 dwm
    374      23    12072       40428        0,17   1504   2 dwm
    1633      63     23116         67764          1,72    3828    2
explorer
      0       0        0           4                0   0 Idle
    594      32    13040       49868        0,23    840   1 LogonUI
    938      20     4632       12892        0,72    592   0 lsass
      0       0         0           8        0,00   2336   0 Memory
Compression
     317      14       3004         13104          0,11    4128    2
MSASCuiL
    646      58   109176     106308        6,73   2292   0 MsMpEng
    278     122    12024        7440        0,47   3064   0 NisSrv
     365      25       5064         18652          0,09    3968    2
OneDrive
     596      29     58000         66404          0,70    5580    2
powershell
    284      12     2408       10560        0,05   3316   2 rdpclip
     476      26       8616         29892          0,27    3704    2
RuntimeBroker
     535      29     12616         11068          0,19    4056    0
SearchIndexer
     935      57     38912         85060          0,66    4500    2
SearchUI
     305      12       3852          7028          0,41     584    0
services
     808      34     21124         60436          0,89    4252    2
ShellExperienceHost
    415      16     4428       18880        0,56   3344   2 sihost
    172      12     2100        9620        0,00   3132   0 slui
```

```
    157      11     2044       9424       0,05   3228   2 slui
    129       9     1572       8052       0,05   4068   0 slui
     167      13     8212      13700       0,06   3080   2
smartscreen
     54       3      424       1212       0,11    296   0 smss
    526      25     7296      18528       0,33   1536   0 spoolsv
     189      14     2320      10176       0,03   2792   0
SppExtComObj
    231      10     6216      16248       1,53   2824   0 sppsvc
    256      14     2988       9444       0,02    652   0 svchost
    799      28     9944      25236       0,86    676   0 svchost
[...]
```

Para comprender el contenido de las distintas columnas devueltas por el comando **Get-Process**, he aquí una tabla resumen de cada una de las propiedades:

Información	Explicación
Handles	Número de descriptores abiertos por el proceso.
NPM(K)	Memoria no paginada utilizada por el proceso (tamaño en kilobytes).
PM(K)	Memoria paginada utilizada por el proceso (tamaño en kilobytes).
WS(K)	Rango de trabajo del proceso (tamaño en kilobytes).
CPU(s)	Tiempo de procesador utilizado por el proceso en el conjunto de procesadores (en segundos).
Id	Número de identificador del proceso (PID).
SI	Número de SessionID. Se utiliza para identificar el propietario del proceso. Cada usuario autenticado en la máquina recibe un SessionID propio.
ProcessName	Nombre del proceso.

Para indicar un proceso puede, con ayuda del parámetro **-Name**, escribir el nombre de los archivos ejecutables. Esto tendrá como efecto mostrar únicamente los procesos correspondientes.

Por ejemplo, para saber si las aplicaciones Internet Explorer, Windows PowerShell y Word se ejecutan actualmente en su puesto de trabajo, basta con ejecutar el siguiente comando:

```
PS          C:\Windows\system32>          Get-Process          -Name
MicrosoftEdge*,powershell,winword
```

```
Handles  NPM(K)      PM(K)          WS(K)        CPU(s)        Id   SI
ProcessName
-------  ------     -----       -----      ------      --  -- ---------
--
   1016        66      22324         69920          1,09    3420    2
MicrosoftEdge
    805        68      31964         66956          1,06    5156    2
MicrosoftEdgeCP
    780        69      39492         94156          2,09    5876    2
MicrosoftEdgeCP
    611        28      69376         73996          1,03    5580    2
powershell
    383      56    53120       99800        8,86   5040   2 WINWORD
```

Si no se encuentra ningún proceso entre las tres aplicaciones indicadas, se obtiene un error.

Es posible hacer lo mismo con el parámetro **-Id** e indicar uno o varios números de identificador de proceso:

```
PS C:\Windows\system32> Get-Process -Id 5580

Handles  NPM(K)      PM(K)          WS(K)        CPU(s)        Id   SI
ProcessName
-------  ------     -----       -----      ------      --  -- ---------
--
    591        28      69576         74624          1,13    5580    2
powershell
```

Con ayuda del parámetro **-FileVersionInfo**, es posible recuperar el número de versión del archivo que ejecuta el proceso. He aquí el resultado con el proceso Internet Explorer, Windows PowerShell y Word:

```
PS          C:\Windows\system32>          Get-Process          -Name
MicrosoftEdge*,powershell,winword
-FileVersionInfo

ProductVersion   FileVersion      FileName
--------------   -----------      --------
11.00.14393.321                                    11.00.14393.3...
C:\Windows\SystemApps\Microsoft.MicrosoftEd...
11.00.14393.82                                     11.00.14393.8...
C:\Windows\SystemApps\Microsoft.MicrosoftEd...
11.00.14393.82                                     11.00.14393.8...
C:\Windows\SystemApps\Microsoft.MicrosoftEd...
10.0.14393.0                                  10.0.14393.0       ...
C:\Windows\System32\WindowsPowerShell\v1.0\...
16.0.4266.1003    16.0.4266.100... C:\Program Files (x86)\Microsoft
Office\Roo...
```

De este modo, puede conocer rápidamente la versión de las aplicaciones que se ejecutan en el puesto de trabajo. Pero la información disponible sobre el proceso no termina aquí. En

efecto, es posible obtener mucha más información sobre cada proceso. La lista es muy extensa, y no está completamente detallada, pero conviene saber que existe, pues es posible encontrar información muy valiosa. Está accesible mediante el siguiente comando:

```
PS C:\Windows\system32> Get-Process -Name winword | Format-List
*


Name                  : WINWORD
Id                    : 4468
PriorityClass         : Normal
FileVersion           : 15.0.4569.1504
HandleCount           : 997
WorkingSet            : 87470080
PagedMemorySize       : 59060224
PrivateMemorySize     : 59060224
VirtualMemorySize     : 520331264
TotalProcessorTime    : 00:00:02.7500000
SI                    : 2
Handles               : 997
VM                    : 520331264
WS                    : 87470080
PM                    : 59060224
NPM                   : 46080
Path                                  : C:\Program Files\Microsoft
Office\Office15\WINWORD.EXE
Company               : Microsoft Corporation
CPU                   : 2,75
ProductVersion        : 15.0.4569.1504
Description           : Microsoft Word
Product               : Microsoft Office 2013
__NounName            : Process
BasePriority          : 8
ExitCode              :
HasExited             : False
ExitTime              :
Handle                : 2448
SafeHandle                                                          :
Microsoft.Win32.SafeHandles.SafeProcessHandle
MachineName           : .
MainWindowHandle      : 262780
MainWindowTitle       : Document1 - Word
[...]
```

He aquí una tabla resumen de las propiedades más importantes:

Propiedad	Explicación
Name	Nombre del proceso.
Path	Ruta de acceso al ejecutable del proceso.
Company	Nombre del fabricante de la aplicación.

Propiedad	Explicación
`FileVersion`	Versión del archivo ejecutable.
`ProductVersion`	Versión del producto (aplicación).
`Descripción`	Descripción de la aplicación.
`Product`	Nombre del producto o de la suite de aplicaciones.
`Id`	Número de identificador del proceso (PID).
`MainWindowTitle`	Descripción presente en la barra de título de la aplicación.
`ProcessName`	Nombre del proceso.
`Responding`	Indica, mediante los valores `True` o `False`, si el proceso responde o no.
`StartTime`	Fecha y hora de ejecución del proceso.

Session ID

Como ha podido ver, el comando de PowerShell **`Get-Process`** devuelve el conjunto de procesos que se ejecutan en la máquina en un instante T, y esto incluye los procesos de todos los usuarios eventualmente autenticados en la máquina, de manera local o remota.

Es posible recuperar la lista de SessionID en curso mediante el siguiente comando: **`query session`**. El resultado devuelve una lista de las sesiones en curso con el número de SessionID correspondiente, así como el nombre del usuario autenticado.

```
PS C:\Users\Julien.Musy> query session
NOMBRE DE SESIÓN   NOMBRE DE USUARIO          ID ESTADO    TIPO
DISPOSITIVO
 services                                       0  Desc
                   Admin                        1  Desc
>console           Julien.Musy                  2  Activo
 31c5ce94259d4...                           65536  Escuchar
```

El SessionID 0 está reservado a Windows: desde esta sesión se ejecutan los principales servicios del sistema, como por ejemplo smss.exe (Session Manager Subsystem). Este último, ejecutado durante el arranque del sistema operativo, permite crear variables de entorno, así como arrancar la gestión de la memoria, el núcleo del subsistema Win32, el proceso Winlogon.exe, etc.

A partir de 1, el SessionID se corresponde con algún usuario autenticado en la máquina. Así, cada uno de los procesos ejecutados en el entorno del usuario o ejecutados explícitamente por el usuario están asociados al SessionID correspondiente.

Para recuperar el conjunto de procesos ejecutados por el usuario Julien.Musy, habrá que filtrar el resultado de la búsqueda de **`Get-Process`** con ayuda de un pipe y especificar el número de SessionID correspondiente a la persona autenticada.

```
PS C:\Users\Admin> Get-Process | Where-Object {$_.SI -eq 2}

Handles   NPM(K)      PM(K)          WS(K)        CPU(s)        Id   SI
ProcessName
-------  ------     -----       -----      ------      --  -- ---------
--
    171       11     3300       12268              7444   2 conhost
    320       12     1344        8660              3936   2 csrss
    178       16     3312       10856              7188   2 dllhost
    447       34    23288       52408              4576   2 dwm
   1676       64    24008       74356                 476    2
explorer
    522       27    17240       51812              7692   2 LogonUI
    318       14     3012       11548                 7632    2
MSASCuiL
    350       23     4620       16744                 8068    2
OneDrive
    558       28    52832       50608                 7432    2
powershell
    263       11     2256       10496              5296   2 rdpclip
    549       32    14556       39520                  260    2
RuntimeBroker
   1071       72    69960      108700                 3136    2
SearchUI
    928       40    25256       45708                 1720    2
ShellExperienceHost
    472       17     5812       22548              4928   2 sihost
    550       34    10804       35496              2440   2 svchost
    373       30     6580       18372                 2924    2
taskhostw
    191       14     2712       12544                 7204    2
taskhostw
    196        9     1948        7172              5472   2 winlogon
```

El conjunto de procesos que se obtiene se corresponde, así, con mi sesión de usuario.

2. Detener un proceso

Para detener un proceso con Windows PowerShell, existe el cmdlet **`Stop-Process`**.

Parámetro	Descripción

Parámetro	Descripción
-Force	Detiene el proceso identificado sin pedir una confirmación de la operación.
-Id <Int32[]>	Especifica uno o varios procesos por su número identificador del proceso (PID).
-Name <String[]>	Especifica uno o varios procesos por su nombre.

Ejemplo: detener todos los procesos con el mismo nombre

El siguiente comando detiene todos los procesos Microsoft Edge.

```
PS C:\Windows\system32> Stop-Process -Name MicrosoftEdge
```

Durante la ejecución de un script, imaginemos que se ejecuta un **Stop- Process** y que no queremos seguir con los comandos que vienen a continuación hasta que las aplicaciones especificadas se hayan detenido completamente. En este caso, interviene el cmdlet **Wait-Process**. Este comando permite poner en «pausa» la ejecución del script mientras los procesos definidos no se encuentren completamente detenidos.

Ejemplo 1:

Uso de **Wait-Process** con una o varias instancias de la misma aplicación.

```
PS C:\> $wordId = (Get-Process WINWORD).id
PS C:\> Stop-Process -Id $wordId
PS C:\> Wait-Process -Id $wordId
PS C:\> # Todos los procesos WINWORD están detenidos
definitivamente
```

Ejemplo 2:

Otro ejemplo con varias aplicaciones diferentes, alguna de ellas con varias instancias de la misma aplicación.

```
PS C:\> Stop-Process -Name outlook, winword, MicrosoftEdge
PS C:\> Wait-Process -Name outlook, winword, MicrosoftEdge -Timeout
30
```

En este ejemplo, si al cabo de 30 segundos (parámetro **-Timeout**) no se han cerrado las aplicaciones Outlook, Word y Microsoft Edge, se muestra un mensaje que indica que una o varias aplicaciones no se han cerrado en el tiempo definido. La

consola Windows PowerShell toma el control, lo que le permite introducir nuevos comandos (en el caso de un script, se retoma la ejecución de los comandos).

El uso de **Wait-Process** sin el parámetro **-Timeout** indica que no existe un tiempo de expiración. Por consiguiente, si un programa no termina por algún motivo indeterminado, entonces la ejecución del script se verá bloqueada por completo. Puede forzar a la consola de Windows PowerShell a retomar el control utilizando la combinación de teclas [Ctrl] C.

3. Arrancar un proceso

Ahora que hemos visto cómo detener procesos, veamos cómo arrancarlos. Para ejecutar un nuevo proceso, es necesario indicar como parámetro el ejecutable del programa que se va a iniciar, con el cmdlet **Start-Process**. He aquí los parámetros que pueden utilizarse con este comando:

Parámetro	Descripción
-ArgumentList <String>	Especifica los parámetros o los valores de parámetros que se van a utilizar durante el arranque del proceso.
-FilePath <String>	Especifica la ruta de acceso al archivo que se va a ejecutar.
-Wait	Espera a que finalice la ejecución del proceso especificado antes de aceptar una nueva entrada (o de proseguir con la ejecución del script).

Ejemplo 1: arrancar un proceso (Word)

```
PS C:\Windows\system32> Start-Process -FilePath 'C:\Program Files
\Microsoft Office\Office15\WINWORD.EXE'
```

Ejemplo 2: arrancar un proceso (Calculadora)

```
PS C:\Windows\system32> Start-Process calc
```

En el caso anterior, no se especifica la ruta completa del ejecutable (calc.exe), pues está presente en la variable de

entorno PATH de Windows (C:\Windows\System32).

Start-Process también permite pasar parámetros al ejecutable que se lanzará. Esto se realiza con el parámetro **-ArgumentList**.

Ejemplo 3: arrancar un proceso con un parámetro

```
PS C:\> Start-Process iexplore.exe -ArgumentList
http://www.ediciones-eni.com.sabidi.urv.cat/
```

Este comando permite ejecutar Internet Explorer abriendo directamente el sitio web de Ediciones ENI. Puede especificar, gracias al parámetro **-Argument List**, aquellas opciones de ejecución que el ejecutable sea capaz de interpretar.

Del mismo modo que con **Stop-Process** y **Wait-Process**, puede que tenga que esperar a que termine la ejecución del proceso (o a que se cierre la aplicación) antes de proseguir con los siguientes comandos. En ese caso, hay que especificar el parámetro **-Wait**:

```
PS C:\> Start-Process iexplore.exe -Wait
```

En este caso concreto, Windows PowerShell no le devolverá el control hasta que el proceso Internet Explorer esté en ejecución.

4. Arrancar una app

Si bien las apps se corresponden con ejecutables que podemos encontrar en la carpeta C:\Windows\SystemApps\, estas no pueden ejecutarse desde el archivo .exe correspondiente.

Para ejecutar una app, hay que utilizar el protocolo asociado. Por ejemplo, para ejecutar la aplicación Sugerencias (introducción a Windows 10) desde una consola de Windows PowerShell, hay que utilizar el siguiente comando:

```
PS C:\> Start-Process MS-Get-Started://
```

He aquí una tabla resumen de las apps que pueden ejecutarse de esta manera:

Nombre de la app	Protocolo asociado
Microsoft Edge	Microsoft-Edge:

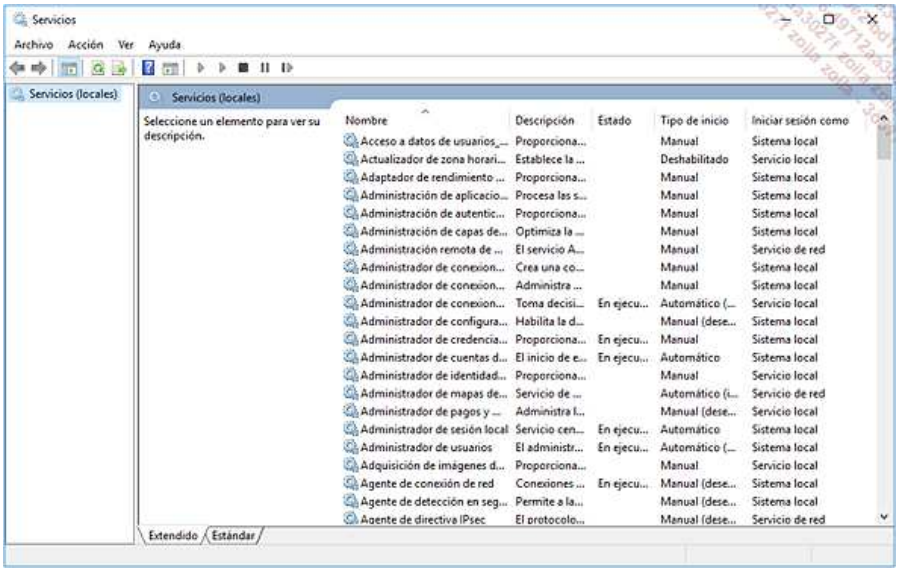
Nombre de la app	Protocolo asociado
Centro de actividades	MS-ActionCenter://
Alarmas y reloj	MS-Clock://
Windows Store	MS-Windows-Store://
Calculadora	Calculator://
Sugerencias	MS-Get-Started://
Mapas	BingMaps://

En el caso de Microsoft Edge, siempre es posible abrir la aplicación especificando la página de inicio correspondiente. He aquí un ejemplo:

```
PS C:\> Start-Process Microsoft-Edge:http://www.ediciones-eni.com.sabidi.urv.cat/
```

Los servicios

Los servicios de Windows son programas que, en función de su modo de arranque, se ejecutan permanentemente, o bien arrancan solo en caso necesario. También es posible desactivar estos servicios, y en ese caso no arrancar jamás. Cada servicio ejecuta, en realidad, uno o varios procesos visibles en el Administrador de tareas.



Lista de servicios

El administrador de servicios de Windows proporciona cuatro modos de arranque que puede configurar para cada servicio:

Tipo de inicio	Descripción
Manual	El arranque del servicio está autorizado, y lo arranca Windows automáticamente cuando es necesario.
Automático	El servicio arranca automáticamente con el inicio de Windows.

Tipo de inicio	Descripción
Automático (inicio retrasado)	El servicio arranca automáticamente, pero con cierto retraso respecto al inicio de la sesión.
Deshabilitado	El servicio no arrancará en ningún caso.

1. Manipulaciones básicas

Con Windows PowerShell, la administración de los servicios y los procesos se realiza de forma similar. Los cmdlets utilizados son:

- **`Get-Service`**: recupera el estado de uno o varios servicios.
- **`Start-Service`**: arranca uno o varios servicios.
- **`Restart-Service`**: reinicia uno o varios servicios.
- **`Stop-Service`**: detiene uno o varios servicios.

a. Obtener información acerca de los servicios

Para comenzar, el comando **`Get-Service`** permite recuperar el estado de cada servicio presente en Windows. He aquí una presentación de los parámetros utilizados para este cmdlet:

Parámetro	Descripción
`-DependentServices`	Obtiene únicamente los servicios que dependen del servicio especificado.
`-DisplayName` `<String[]>`	Especifica los nombres para mostrar de los servicios que hay que recuperar.
`-Name` `<String[]>`	Especifica los nombres de los servicios que hay que recuperar.
`-RequiredServices`	Obtienen únicamente los servicios que este servicio requiere.

Ejemplo: obtener la lista de todos los servicios y su estado en el puesto local

```
PS C:\Windows\system32> Get-Service
```

```
Status   Name               DisplayName
------   ----               -----------
Stopped  AJRouter           Servicio de enrutamiento AllJoyn
Stopped  ALG                Servicio de la puerta de enlace...
Stopped  AppIDSvc           Identidad de la aplicación
Running  Appinfo            Información de la aplicación
Stopped  AppMgmt            Administración de aplicaciones
Stopped  AppReadiness       Preparación de aplicaciones
Stopped  AppVClient         Microsoft App-V Client
Stopped   AppXSvc                 Servicio de implementación AppX
(AppXSVC)
Running  AudioEndpointBu... Generador de puntos de acceso...
Running  Audiosrv           Audio de Windows
Stopped   AxInstSV              Programa de instalación de ActiveX
(A...
Stopped   BDESVC                 Servicio de cifrado de la unidad
B...
[...]
```

La columna `Status` permite saber si el servicio está arrancado (`Running`) o no (`Stopped`). Se trata de una propiedad de cada servicio. Por consiguiente, si desea conocer el conjunto de servicios iniciados, escriba:

```
PS C:\Windows\system32> Get-Service | Where-Object {$_.Status -
eq
"Running"}

Status   Name               DisplayName
------   ----               -----------
Running  Appinfo            Información de aplicaciones
Running  AudioEndpointBu... Generador de puntos de acceso...
Running  Audiosrv           Audio de Windows
Running  BFE                Motor de filtrado de base
[...]
```

Para obtener información complementaria, también puede enumerar el conjunto de propiedades de un servicio en particular:

```
PS C:\> Get-Service -Name WSearch | Format-List *


Name                : wsearch
RequiredServices    : {RPCSS}
CanPauseAndContinue: False
CanShutdown         : True
CanStop             : True
DisplayName         : Windows Search
DependentServices   : {workfolderssvc, WMPNetworkSvc}
MachineName         : .
ServiceName         : wsearch
ServicesDependedOn : {RPCSS}
ServiceHandle       :
Status              : Running
ServiceType         : Win32OwnProcess
Site                :
Container           :
```

Observe la presencia de dos propiedades importantes: `RequiredServices` y `DependentService`. En efecto, la mayoría de los servicios tienen dependencias entre sí. Los servicios requeridos (`RequiredServices`) enumerados aquí son los servicios necesarios para permitir el arranque del servicio. En el ejemplo, el servicio RPCSS (Llamada a procedimiento remoto (RPC)) debe estar iniciado para poder arrancar el servicio Windows Search. Es posible obtener esta información directamente con el parámetro **-RequiredServices** de **Get-Service**:

```
PS C:\> Get-Service WSearch -RequiredServices

Status   Name               DisplayName
------   ----               -----------
Running  RPCSS              Llamada a procedimiento remoto (RPC)
```

El servicio o los servicios que dependen (`DependentServices`) del servicio especificado requieren que este último esté iniciado para poder funcionar. Así, los servicios workfolderssvc (Carpetas de trabajo) y WMPNetworkSvc (Servicio de uso compartido de red del Reproductor de Windows Media) dependen de Windows Search para poder funcionar.

Esta información puede obtenerse con el parámetro **-DependentServices**:

```
PS C:\> Get-Service WSearch -DependentServices

Status   Name            DisplayName
------   ----            -----------
Stopped  workfolderssvc  Carpetas de trabajo
Running   WMPNetworkSvc    Servicio de uso compartido de red del
Reproductor de W...
```

En el caso anterior, si detiene el servicio Windows Search, el servicio WMPNetworkSvc también se detiene.

b. Arrancar un servicio

Start-Service permite arrancar servicios. He aquí el conjunto de parámetros:

Parámetro	Descripción
-DisplayName `<String[]>`	Especifica los nombres para mostrar de los servicios que hay que arrancar.

Parámetro	Descripción
-Name <String[]>	Especifica los nombres de los servicios que hay que arrancar.

Ejemplo 1: arrancar un servicio con su nombre

```
PS C:\> Start-Service -Name WinRM
```

Ejemplo 2: arrancar un servicio con su nombre para mostrar

```
PS C:\> Start-Service -DisplayName "Administración remota de Windows
(Administración WSM)"
```

Cuando se utiliza **Start-Service**, si se requieren servicios para el correcto funcionamiento del servicio especificado, estos se inician automáticamente.

Sin embargo, si el servicio especificado o si alguno de los servicios requeridos (RequiredServices) tienen su tipo de inicio deshabilitado, se obtiene un error.

```
PS C:\Windows\system32> Start-Service workfolderssvc
Start-Service: No se puede iniciar el servicio 'Carpetas de
trabajo (workfolderssvc)' debido al error siguiente:
No se puede iniciar el servicio workfolderssvc en el equipo '.'.
```

Hay que asegurarse de que estos servicios tienen un tipo de inicio en modo manual o automático.

Esta situación también puede deberse a falta de permisos. Asegúrese de haber abierto la consola de Windows PowerShell como administrador.

c. Reiniciar un servicio

Restart-Service permite reiniciar servicios. Es posible especificar varios servicios en una única línea de comandos (esto es válido también con los cmdlets **Get-Service**, **Start-Service** y **Stop-Service**).

Parámetro	Descripción
-DisplayName <String[]>	Especifica los nombres para mostrar de los servicios que hay que reiniciar.

Parámetro	Descripción
-Force	Permite al applet de comando reiniciar un servicio incluso aunque tenga servicios dependientes.
-Name <String[]>	Especifica los nombres de los servicios que hay que reiniciar.

Ejemplo: reinicio de dos servicios

```
PS C:\Windows\system32> Restart-Service Spooler,WSearch -Force
```

d. Detener un servicio

Por último, **Stop-Service** detiene el funcionamiento de los servicios.

Parámetro	Descripción
-DisplayName <String[]>	Especifica los nombres para mostrar de los servicios que hay que detener.
-Force	Permite al applet de comando detener un servicio incluso aunque tenga servicios dependientes.
-Name <String[]>	Especifica los nombres de los servicios que hay que detener.

Ejemplo: detener el funcionamiento de un servicio

```
PS C:\Windows\system32> Stop-Service WSearch
```

Si hay arrancado algún servicio dependiente (DependentServices) del servicio especificado, se obtiene un mensaje de error. Debe incluir el parámetro **-Force** con el valor **Stop-Service** para forzar la detención de todos los servicios dependientes.

Ejemplo: forzar la detención de todos los servicios dependientes

```
PS C:\Windows\system32> Get-Service WSearch

Status   Name               DisplayName
------   ----               -----------
Running  WSearch            Windows Search
```

```
PS C:\Windows\system32> Get-Service WSearch -DependentServices

Status   Name               DisplayName
------   ----               -----------
Stopped  workfolderssvc     Carpetas de trabajo
Stopped  WMPNetworkSvc      Servicio de uso compartido de red del
R...


PS C:\Windows\system32> Start-Service WMPNetworkSvc
PS C:\Windows\system32> Stop-Service WSearch
stop-service: No se puede detener el servicio 'Windows Search
(wsearch)' porque tiene servicios dependientes. Sólo se puede
detener si está establecido la marca Force.
```

Especificando el parámetro **`-Force`**, se obtiene:

```
PS C:\Windows\system32> Get-Service workfolderssvc

Status   Name               DisplayName
------   ----               -----------
Running  workfolderssvc     Carpetas de trabajo


PS C:\Windows\system32> Stop-Service WSearch -Force
PS C:\Windows\system32> Get-Service workfolderssvc

Status   Name               DisplayName
------   ----               -----------
Stopped  workfolderssvc     Carpetas de trabajo
```

2. Operaciones avanzadas

Los servicios de Windows no se limitan a simples arranques, detenciones y reinicios. También es posible crear nuevos servicios para, por ejemplo, lanzar un ejecutable durante el arranque de Windows, incluso antes de iniciar una sesión de usuario.

Para ello, debe utilizar los siguientes cmdlets:

- **`New-Service`**: crear un nuevo servicio.
- **`Set-Service`**: editar los parámetros de un servicio.

a. Crear un nuevo servicio

Para crear un nuevo servicio, utilice el cmdlet **`New-Service`**. Habrá que pasarle como parámetro un nombre y la ruta de acceso al archivo binario.

Parámetro	Descripción

Parámetro	Descripción
-BinaryPathName <String>	Especifica la ruta de acceso al archivo binario para este servicio.
-Credential <PSCredential>	Indica la cuenta con la que debe ejecutarse el servicio.
-Description <String>	Indica la descripción del servicio que se va a crear.
-DisplayName <String>	Especifica el nombre para mostrar del servicio que se va a crear.
-Name <String>	Especifica el nombre del servicio que se va a crear.
-StartupType <ServiceStartNode>	Define el tipo de arranque del servicio. Si no se especifica, entonces se configura en modo automático.

Recordemos los cuatro tipos de arranque posibles para un servicio, y los valores que hay que introducir para el parámetro **-StartupType**:

Tipo de arranque	ServiceStartMode
Manual	Manual
Automático	Automatic
Automático (inicio retrasado)	N/A
Deshabilitado	Disabled

Ejemplo 1: creación de un nuevo servicio

```
PS C:\> New-Service -Name "Servicio Test" -Description
"Descripción del servicio de test" -BinaryPathName
C:\Temp\myService.exe
```

La creación del servicio se vuelve visible en la consola de servicios (services.msc). El servicio se ejecutará, por defecto, con

la cuenta de sistema local. Esto quiere decir que el programa que se ejecuta posee todos los permisos en la máquina sobre la que corre.

Ejemplo 2: creación de un nuevo servicio con una cuenta específica

Puede que el programa necesite ejecutarse con alguna otra cuenta, pues requiera comunicarse con otras máquinas remotas, por ejemplo. Para ello, habrá que proporcionar las credenciales de la cuenta con la que se va a ejecutar el servicio.

```
PS C:\Windows\system32> $contraseña = ConvertTo-SecureString
"P@$$w0rd" -AsPlainText -Force
PS C:\Windows\system32> $usuario = New-Object
System.Management.Automation.PSCredential("DOMAIN\User1",
$contraseña)
PS C:\Windows\system32> $binario = "C:\Temp\miPrograma.exe"
PS C:\Windows\system32> New-Service -Name "Servicio Test 2"
-BinaryPathName $binario -Startuptype Automatic -Credential
$usuario
```

b. Modificar un servicio existente

En caso de error durante la creación de un servicio, o si resulta necesario modificar algún parámetro en el servicio existente, se utiliza **`Set-Service`**. Este cmdlet permite modificar la configuración de un servicio. Los parámetros que pueden modificarse con **`Set-Service`** son: el nombre para mostrar, la descripción, el tipo de inicio, así como el estado del servicio. El parámetro **`-Name`** es obligatorio para precisar sobre qué servicio se lleva a cabo la acción.

Parámetro	Descripción
`-Description` <String>	Cambia la descripción del servicio.
`-DisplayName` <String>	Cambia el nombre para mostrar del servicio.
`-Name` <String>	Especifica el nombre del servicio que se va a modificar.
`-StartupType` <ServiceStartMode>	Define el tipo de inicio del servicio.

Parámetro	Descripción
-Status <String>	Arranca, detiene o pone en pausa el servicio.

Ejemplo: modificar el tipo de inicio de un servicio

```
PS C:\> Set-Service -Name WSearch -StartupType Disabled -Status
Stopped
```

En el ejemplo anterior, se detiene el servicio Windows Search, y su tipo de inicio pasa a deshabilitado.

c. Eliminar un servicio

Hemos visto cómo crear un nuevo servicio, pero no cómo eliminarlo. No hay ningún cmdlet que permita eliminar un servicio; sin embargo, existen dos maneras de realizarlo. La primera consiste en hacerlo por consola de comandos mediante el parámetro `sc delete`. La segunda solución se basa en un comando WMI (clase Win32_Service).

Ejemplo: eliminar un servicio utilizando WMI

```
PS C:\Windows\system32> $service = Get-WmiObject -Class
Win32_Service -Filter "Name='nombre_del_servicio'"
PS C:\Windows\system32> $service.delete()
```

El equipo y el sistema operativo

En esta sección veremos los cmdlets que permiten interactuar con el equipo y el sistema operativo Windows. He aquí un resumen:

- **Add-Computer**: permite unir el equipo a un dominio o grupo de trabajo.
- **Remove-Computer**: retira el equipo local de su dominio.
- **Rename-Computer**: renombra el equipo.
- **Restart-Computer**: reinicia el equipo.
- **Reset-ComputerPassword**: reinicia el canal seguro entre el equipo y su dominio.
- **Stop-Computer**: detiene el equipo.
- **Test-ComputerSecureChannel**: comprueba el canal seguro. También permite repararlo en caso necesario.

1. Renombrar el equipo

El cmdlet que se estudia aquí permite renombrar el equipo antes, por ejemplo, de integrarlo en un dominio Active Directory. **Rename-Computer** se utiliza fácilmente con los siguientes parámetros:

Parámetro	Descripción
-ComputerName <String>	Especifica el equipo (local o remoto).
-Confirm	Solicita una confirmación antes de ejecutar el comando.
-DomainCredential <PSCredential>	Indica la cuenta que posee permisos para conectarse al dominio.
-LocalCredential <PSCredential>	Indica la cuenta que posee permisos para conectarse al equipo indicado mediante el parámetro **-ComputerName**.

Parámetro	Descripción
-NewName <String>	Designa el nuevo nombre que se asignará al equipo. El nombre puede contener únicamente caracteres alfanuméricos (A-Z o 0-9) y el guion (-).
-Protocol <String>	Especifica el protocolo utilizado para renombrar el equipo remoto. Los posibles valores son DCOM y WSMan. Por defecto, se utiliza el protocolo DCOM.
-Restart	Se realizará un reinicio automático del equipo tras ejecutar el comando. Se requiere un reinicio para que el cambio de nombre tenga efecto.

Ejemplo 1: renombrar un equipo local (no unido a un dominio Active Directory)

```
PS C:\Windows\system32> Rename-Computer -NewName "PC-12345" -
Restart
```

En este ejemplo, se asigna el nombre PC-12345 al equipo, y este se reiniciará automáticamente para que el cambio tenga efecto. Por defecto, este comando se ejecuta con la cuenta del usuario en curso. Si este usuario no posee permisos de administrador en la máquina, debe utilizar el parámetro **-LocalCredential** especificando una cuenta que posea estos permisos.

Ejemplo 2: renombrar un equipo local (unido a un dominio Active Directory)

El siguiente ejemplo muestra cómo renombrar un equipo que ya está presente en un dominio Active Directory. Para ello, hay que utilizar el parámetro **-DomainCredential**, especificando una cuenta que posea los permisos necesarios para agregar equipos al directorio Active Directory.

```
PS C:\Windows\system32> $cred = Get-Credential
PS C:\Windows\system32> Rename-Computer -NewName "PC-12345"
-DomainCredential $cred
ADVERTENCIA: Los cambios se tendrán en cuenta tras
reiniciar el equipo W10.
```

Tras reiniciar el equipo, este seguirá estando unido al dominio Active Directory, pero con su nuevo nombre.

Ejemplo 3: renombrar un equipo (no unido a un dominio Active Directory) de manera remota

Es posible renombrar un equipo remoto, esté presente o no en un dominio Active Directory. Para ello, se utiliza el parámetro **-LocalCredential** indicándole las credenciales de conexión de una cuenta local que posea permisos de administración en la máquina.

```
PS C:\Windows\system32> $localCred = Get-Credential
PS C:\Windows\system32> Rename-Computer -ComputerName "PC-12345"
-LocalCredential $localCred -NewName "W10"
ADVERTENCIA: Los cambios se tendrán en cuenta tras
reiniciar el equipo PC-12345.
```

Debido a que no se ha especificado el parámetro **-Protocol**, el protocolo utilizado será DCOM. Asegúrese de que la cuenta local utilizada en el equipo remoto posee los permisos necesarios para ejecutar consultas WMI remotas. Tenga precaución, también, de abrir los flujos en el firewall para enviar las peticiones WMI y que el control de la cuenta de usuario (UAC) remoto esté deshabilitado (propiedad que se encuentra en el registro LocalAccountTokenFilterPolicy de tipo DWORD con el valor 1, que debe ubicarse en la clave HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\ CurrentVersion\Policies\System).

Ejemplo 4: renombrar un equipo (unido a un dominio Active Directory) de manera remota

Lo mismo que en el ejemplo anterior, pero con un puesto que ya existe en un dominio. Además de utilizar el parámetro **-LocalCredential**, hay que utilizar el parámetro **-DomainCredential** pasándole las credenciales de conexión de una cuenta de dominio que posea los permisos necesarios para agregar equipos en el directorio Active Directory.

```
PS C:\Windows\system32> $localCred = Get-Credential
PS C:\Windows\system32> $domainCred = Get-Credential
PS C:\Windows\system32> Rename-Computer -ComputerName "W10"
-LocalCredential $localCred -DomainCredential $domainCred -NewName

"PC-12345" -Protocole WSMan -Restart
```

Como se ha indicado el parámetro **-Restart**, el puesto se reiniciará automáticamente tras la ejecución del comando. A su vez, el uso del parámetro **-Protocol** con el valor `WSMan` indica que la consulta enviada utilizará el protocolo de comunicación WS-Management (consulte la sección Administración remota con Windows PowerShell del capítulo Administración remota de puestos de trabajo, para obtener más información).

2. Reiniciar el equipo

Para reiniciar un puesto de trabajo, existe el cmdlet **Restart-Computer**. Permite reiniciar uno o varios puestos de trabajo al mismo tiempo, pues es posible, con este cmdlet, lanzar reinicios sobre equipos remotos.

Parámetro	Descripción
-ComputerName `<String[]>`	Especifica un equipo local o varios equipos remotos).
-Credential `<PSCredential>`	Indica la cuenta con la que se ejecutará el comando.
-Force	Fuerza el reinicio inmediato de los equipos.
-Protocol `<String>`	Especifica el protocolo utilizado para reiniciar el equipo remoto. Los posibles valores son `DCOM` y `WSMan`. Por defecto, el protocolo utilizado es DCOM.
-Timeout `<Int32>`	Especifica un tiempo de espera antes de que la consola de comandos retome el control, tanto si los puestos de trabajo han terminado de reiniciar como si no. Este parámetro se utiliza junto con el parámetro **-Wait**.

Parámetro	Descripción
-Wait	Pone en espera la consola de comandos hasta que termine el reinicio del equipo o de los equipos de trabajo. Preste atención, no existe un límite en el tiempo de espera por defecto. Para fijar uno, hay que utilizar el parámetro **-TimeOut**.

Ejemplo 1: reiniciar el sistema operativo del equipo local

```
PS C:\Windows\system32> Restart-Computer
```

Ejemplo 2: reiniciar varios sistemas operativos (local y remotos)

```
PS C:\Windows\system32> Restart-Computer -ComputerName maquina1,
maquina2, servidor1
```

Sin embargo, en un entorno empresarial, y también por motivos de seguridad, es necesaria una autenticación previa para poder ejecutar este comando. Para ello, debe crear previamente un objeto de credenciales y, a continuación, especificarlo en el comando **Restart-Computer** mediante el parámetro **-Credential**:

```
PS C:\Users\Administrator> $cred = Get-Credential
PS  C:\Users\Administrator>  Restart-Computer  -ComputerName  maquina1

-Credential $cred
```

El uso del parámetro **-Force** permite forzar el reinicio de la máquina en caso de que exista alguna sesión abierta en el puesto de trabajo remoto.

Como no se ha especificado el parámetro **-Protocol**, el comando va a enviar las peticiones de reinicio a los equipos remotos mediante WMI. Asegúrese por tanto de disponer de los permisos necesarios.

Ejemplo 3: forzar el reinicio de un sistema operativo

```
PS C:\Users\Administrator> Restart-Computer -ComputerName
192.168.1.14 -Credential $cred
restart-computer: Failed to restart the computer 192.168.1.14
with the following error message: The system shutdown cannot be
initiated because there are other users logged on to the computer.
```

Con el parámetro **-Force**, el reinicio de la máquina remota tiene lugar:

```
PS C:\Users\Administrator> Restart-Computer -ComputerName
192.168.1.14 -Credential $cred -Force
```

Ejemplo 4: reiniciar un puesto remoto con el protocolo WSMan

```
PS C:\Users\Administrator> Restart-Computer -ComputerName
192.168.1.14 -Credential $cred -Protocol WSMan -Force -Wait
```

En este ejemplo, la petición enviada al equipo remoto utilizará el protocolo WS-Management. El comando se quedará en espera (parámetro **-Wait**) hasta que el puesto de trabajo remoto haya terminado de reiniciar y el servicio WinRM esté de nuevo accesible. Asegúrese de que el servicio esté configurado en modo de inicio automático. Preste atención al modo de inicio automático retrasado, que puede tomar cierto tiempo y, por tanto, bloquear el resto de las instrucciones de un script.

Ejemplo 5: reiniciar un puesto remoto especificando un tiempo de espera

```
PS C:\Users\Administrator> Restart-Computer -ComputerName
192.168.1.14 -Credential $cred -Protocol WSMan -Force -Wait
-Timeout 30
```

Lo mismo que con el ejemplo anterior, pero con una pausa de 30 segundos. Más allá de este límite, Windows PowerShell le devolverá el control, pero especificando el puesto o los puestos de trabajo que no hayan terminado su reinicio.

```
Restart-Computer: Failed to restart the computer W10 with
the following error message: The computer did not finish
restarting within the specified time-out period.
```

3. Detener el equipo

El cmdlet **Stop-Computer** permite, como su nombre indica, detener el equipo. Así, una vez ejecutado, el sistema operativo cierra las sesiones de los usuarios en curso y a continuación se detiene: el equipo queda apagado.

He aquí un resumen de los parámetros de **Stop-Computer**:

Parámetro	Descripción

Parámetro	Descripción
-ComputerName <String[]>	Especifica un equipo local o varios equipos remotos.
-Credential <PSCredential>	Indica la cuenta con la que debe ejecutarse el comando.
-Force	Fuerza la parada inmediata de los equipos.
-Protocol <String>	Especifica el protocolo utilizado para detener el equipo remoto. Los posibles valores son DCOM y WSMan. Por defecto, se utiliza el protocolo DCOM.

Ejemplo 1: detener el equipo local

```
PS C:\Windows\system32> Stop-Computer
```

Ejemplo 2: detener un equipo remoto utilizando el protocolo WMI

```
PS C:\Windows\system32> $cred = Get-Credential
PS C:\Windows\system32> Stop-Computer -Computer W10 -Credential
$cred -Force
```

El uso del parámetro **-Force** permite forzar la parada de la máquina correspondiente. Así, si existen sesiones de usuario abiertas en el equipo, se cerrarán automáticamente (tenga precaución, no se guardará el trabajo en curso). Si se omite el parámetro **-Force** y existe al menos una sesión de usuario abierta, la petición no se ejecutará y se obtendrá un mensaje de error:

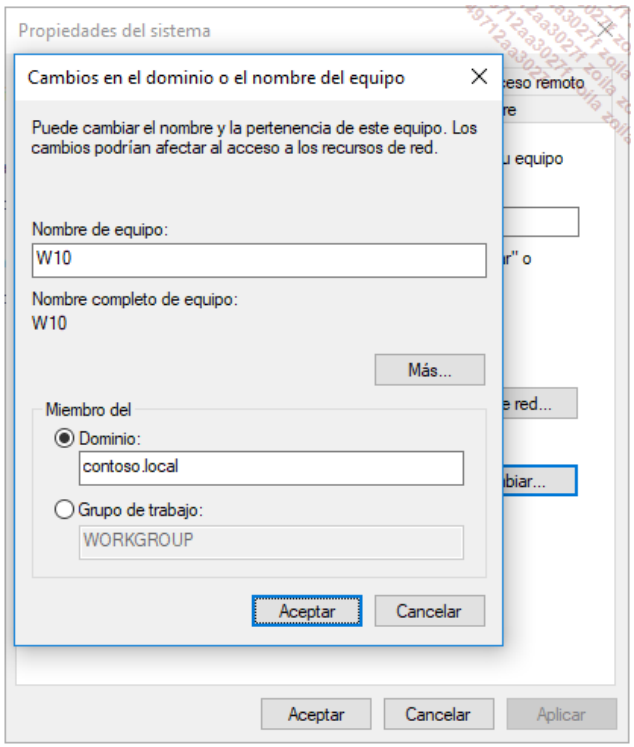
```
Stop-Computer: Este comando no se puede ejecutar en el equipo
de destino ('W10') debido al siguiente error: No se puede
iniciar el cierre del sistema porque otros usuarios iniciaron
sesión en el equipo.
```

Ejemplo 3: detener varios equipos remotos utilizando el protocolo WS-Management

```
PS C:\Windows\system32> $cred = Get-Credential
PS C:\Windows\system32> Stop-Computer -Computer W10,LAB-WS01
-Credential $cred -Protocol WSMan -Force
```

4. Unirse a un dominio

Es posible unir el puesto de trabajo a un dominio administrado por Windows Server con el rol de controlador de dominio (*Domain Controller* - DC) a través de PowerShell. En Windows, el equivalente se haría desde la ventana **Propiedades del sistema**, en la pestaña **Nombre del equipo**.



Modificación del nombre o del dominio del equipo

Incluso en un script para automatizar esta tarea, el cmdlet **`Add-Computer`** permite agregar el equipo a un dominio determinado. Por supuesto, para realizar esta manipulación es necesario disponer de una cuenta de dominio dotada de los permisos necesarios para ello.

Parámetro	Descripción
`-ComputerName` `<String[]>`	Especifica un equipo local o varios equipos remotos.
`-Credential` `<PSCredential>`	Especifica una cuenta de usuario que tiene permisos para unir equipos a un nuevo dominio.
`-DomainName` `<String>`	Especifica el dominio actual al que se une el equipo.

Parámetro	Descripción
-LocalCredential <PSCredential>	Indica la cuenta que posee permisos para conectarse al equipo especificado a través del parámetro **-ComputerName**.
-OUPath <String>	Especifica la unidad organizativa donde se registrará la cuenta equipo.
-Restart	Reinicia el equipo que se ha agregado al dominio o al grupo de trabajo.
-Server <String>	Especifica el nombre de un controlador de dominio que agrega el equipo al dominio.

Ejemplo: agregar el equipo local en un dominio

En primer lugar, identifique una cuenta de dominio que posea los permisos necesarios para agregar puestos de trabajo a un dominio:

```
PS C:\Windows\system32> $credential = New-Object
System.Management.Automation.PSCredential("CONTOSO\Administrador",

(ConvertTo-SecureString "P@$$w0rd" -AsPlainText -Force))
```

A continuación, solo queda por agregar el equipo al dominio:

```
PS C:\Windows\system32> Add-Computer -DomainName "CONTOSO.LOCAL"
-Credential $credential
ADVERTENCIA: Los cambios serán efectivos tras reiniciar el equipo
W10.
```

Debe reiniciar el equipo para que tengan efecto las modificaciones. Para ello, existen dos posibilidades:

- Reinicie el puesto gracias al cmdlet **Restart-Computer** que hemos visto anteriormente.
- Especifique el parámetro **-Restart** al comando **Add-Computer**. Con esto, el equipo se reinicia automáticamente si el comando tiene éxito.

Existen otros parámetros de **Add-Computer**, en particular **-OUPath**, que permite especificar en qué unidad organizativa (*Organizational Unit* - OU) se agregará el equipo dentro de Active Directory.

Ejemplo: agregar el equipo en un dominio con -OUPath

```
PS C:\Windows\system32> Add-Computer -DomainName "CONTOSO.LOCAL"
-Credential $credential -OUPath "OU=España,DC=CONTOSO,DC=LOCAL"
```

He aquí otro parámetro, **-Server**. Este parámetro permite especificar el nombre del servidor encargado de agregar el puesto de trabajo al dominio. Hay que introducir la información correspondiente con el formato `Nombre_Dominio\Nombre_Servidor`.

Ejemplo: agregar el equipo en un dominio especificando un DC

```
PS C:\Windows\system32> Add-Computer -DomainName "CONTOSO.LOCAL"
-Credential $credential -Server CONTOSO\server01
```

En último lugar, es posible agregar un puesto de trabajo remoto en un dominio. Para ello, se utilizan los parámetros **-ComputerName** para designar el equipo o los equipos remotos, y **-LocalCredential** para especificar la cuenta que posee las credenciales de conexión para conectarse a ellos. Preste atención, el protocolo utilizado es WMI; asegúrese por tanto de que la cuenta local utilizada posee los permisos necesarios en el puesto o los puestos remotos y que los firewalls no bloquean las peticiones.

Ejemplo: agregar un equipo remoto en un dominio

```
PS C:\Windows\system32> $localCred = New-Object
System.Management.Automation.PSCredential("AdminW10",
(ConvertTo-SecureString "LocalP@$$w0rd" -AsPlainText -Force))
PS C:\Windows\system32> $credential = New-Object
System.Management.Automation.PSCredential("CONTOSO\Administrador",

(ConvertTo-SecureString "P@$$w0rd" -AsPlainText -Force))
PS C:\Windows\system32> Add-Computer -ComputerName W10 -
LocalCredential
$localCred -DomainName "CONTOSO.LOCAL" -Credential $credential -
Restart
```

5. Sacar del dominio

El cmdlet **Remove-Computer** se utiliza específicamente para sacar a los puestos de trabajo de un dominio. Es obligatorio reiniciar el puesto de trabajo tras ejecutar el cmdlet para aplicar los cambios. Una vez reiniciado el equipo, este se encuentra en un grupo de trabajo que se habrá designado mediante el parámetro **-WorkgroupName** (si no se ha especificado nada, el grupo de trabajo será WORKGROUP).

Antes de utilizar **Remove-Computer**, asegúrese de disponer de las credenciales de una cuenta de Administrador local de la máquina, pues, una vez reiniciada, no podrá utilizar la cuenta de dominio para iniciar una sesión en el puesto de trabajo.

Además de que **Remove-Computer** saca al puesto del dominio en el que está, la cuenta de equipo registrada en el directorio Active Directory se deshabilita igualmente, lo que no ocurre si realizamos esta misma acción a través de la interfaz gráfica de Windows.

He aquí los principales parámetros de **Remove-Computer**:

Parámetro	Descripción
-ComputerName <String[]>	Especifica un equipo local o varios equipos remotos.
-Force	Elimina la petición de confirmación del usuario.
-LocalCredential <PSCredential>	Indica la cuenta que posee permisos para conectarse al equipo especificado mediante el parámetro **-ComputerName**.
-Restart	Reinicia el equipo que se ha sacado del dominio.
-UnjoinDomainCredential <PSCredential>	Especifica una cuenta de usuario que tiene permisos para sacar al equipo del dominio.
-WorkgroupName <String>	Especifica el grupo de trabajo al que se unirá el equipo. Si no se indica, el valor por defecto es WORKGROUP.

Ejemplo: sacar un equipo del dominio

```
PS C:\Windows\system32> $credential = New-Object
System.Management.Automation.PSCredential("CONTOSO\Administrador",

(ConvertTo-SecureString "P@$$w0rd" -AsPlainText -Force))
PS C:\Windows\system32> Remove-Computer -WorkgroupName CTS-WG
-UnjoinDomainCredential $credential -Force -Restart
```

Sin petición de confirmación, y tras el reinicio automático con la validación del comando, el equipo se encuentra en el grupo de trabajo CTS-WG.

Ejemplo: sacar un equipo remoto del dominio

```
PS C:\Windows\system32> $localCred = New-Object
System.Management.Automation.PSCredential("W10\Admin",
(ConvertTo-SecureString "LocalP@$$w0rd" -AsPlainText -Force))
PS C:\Windows\system32> $domainCred = New-Object
System.Management.Automation.PSCredential("CONTOSO\Administrador",

(ConvertTo-SecureString "P@$$w0rd" -AsPlainText -Force))
PS C:\Windows\system32> Remove-Computer -ComputerName W10
-UnjoinDomainCredential $domainCred -LocalCredential $localCred
-WorkgroupName "CTS-WG" -Force -Restart
```

Para el parámetro **-LocalCredential**, tiene la posibilidad de informar las credenciales de una cuenta de dominio que posea los permisos de conexión necesarios. Como con **Add-Computer**, el protocolo utilizado para enviar la petición hacia el equipo remoto es WMI.

6. Cambiar de dominio o de grupo de trabajo

También es posible pasar directamente a otro dominio un equipo que ya pertenezca a un dominio, de igual modo que es posible pasar directamente de un grupo de trabajo a otro.

Para realizar estas acciones, se utiliza nuevamente el cmdlet **Add-Computer**, pero complementado con algunos parámetros suplementarios:

Parámetro	Descripción
-ComputerName <String[]>	Especifica un equipo local o varios equipos remotos.
-Credential <PSCredential>	Especifica una cuenta de usuario que posee permisos para unir equipos al nuevo dominio.
-DomainName <String>	Especifica el dominio al que se une el equipo.

Parámetro	Descripción
-LocalCredential <PSCredential>	Indica la cuenta que posee permisos para conectarse a los equipos especificados mediante el parámetro **-Computer Name**.
-NewName <String>	Especifica un nuevo nombre que se asignará al equipo y con el que se registrará en el dominio. Este parámetro es válido únicamente si solo se indica un puesto de trabajo en el comando.
-OUPath <String>	Especifica la unidad organizativa donde se registrará la cuenta de equipo.
-Restart	Reinicia el equipo que se ha agregado al dominio o al grupo de trabajo.
-Server <String>	Especifica el nombre de un controlador de dominio que agrega el equipo al dominio.
-UnjoinDomainCredential <PSCredential>	Especifica una cuenta de usuario que tiene permisos para sacar al equipo del dominio.
-WorkgroupName <String>	Especifica el grupo de trabajo al que se agrega el equipo. Si no se indica nada, el valor por defecto es WORKGROUP.

Así, si un puesto de trabajo ya se encuentra en un dominio y hay que desplazarlo a otro dominio, las credenciales para sacar al equipo del dominio de origen deberán indicarse mediante el parámetro **-UnjoinDomainCredential**, y las credenciales de una cuenta del dominio de destino que posea los permisos necesarios para unir equipos al dominio deberán indicarse mediante el parámetro **-Credential**.

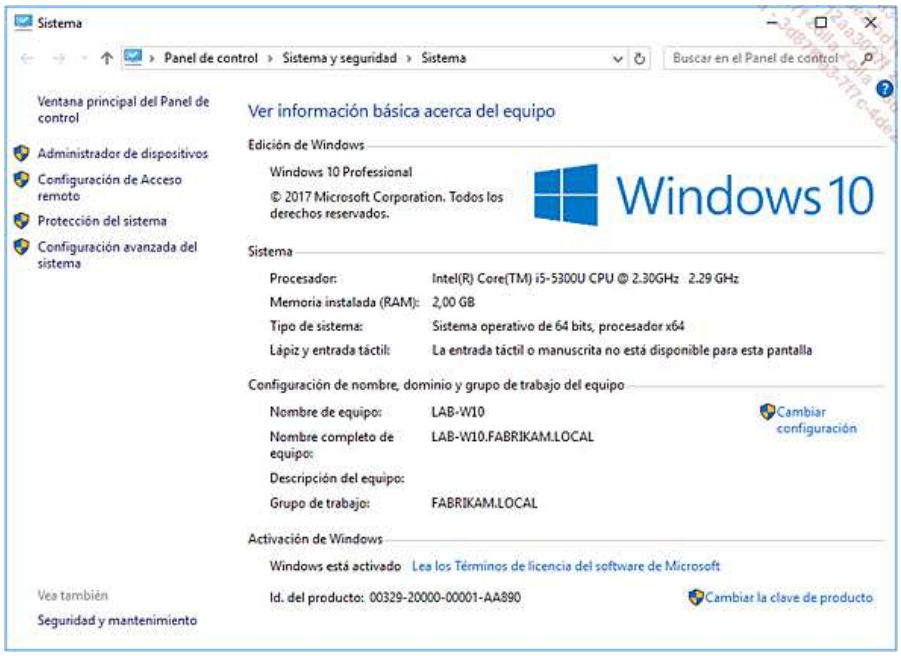
Ejemplo: cambiar de dominio en un equipo local

```
PS C:\Windows\system32> $unjoinCred = New-Object
System.Management.Automation.PSCredential("CONTOSO\Administrador",

(ConvertTo-SecureString "P@$$w0rd" -AsPlainText -Force))
PS C:\Windows\system32> $domainCred = New-Object
System.Management.Automation.PSCredential("FABRIKAM\Administrador",

(ConvertTo-SecureString "Pr|nc3$$" -AsPlainText -Force))
PS C:\Windows\system32> Add-Computer -DomainName "FABRIKAM.LOCAL"

-UnjoinDomainCredential $unjoinCred -Credential $domainCred -Force
-Restart
```

Ejemplo: cambiar de dominio y renombrar un equipo remoto

```
PS C:\Windows\system32> $localCred = New-Object
System.Management.Automation.PSCredential("W10\Admin",
(ConvertTo-SecureString "localP@ss" -AsPlainText -Force))
PS C:\Windows\system32> $unjoinCred = New-Object
System.Management.Automation.PSCredential("CONTOSO\Administrador",

(ConvertTo-SecureString "P@$$w0rd" -AsPlainText -Force))
PS C:\Windows\system32> $domainCred = New-Object
System.Management.Automation.PSCredential("FABRIKAM\Administrador",

(ConvertTo-SecureString "Pr|nc3$$" -AsPlainText -Force))
PS C:\Windows\system32> Add-Computer -ComputerName W10 -
localCredential
$localCred -DomainName "FABRIKAM.LOCAL" -UnjoinDomainCredential
$unjoinCred -Credential $domainCred -NewName "LAB-W10" -Force -
Restart
```

Tras el reinicio, el equipo se encuentra en el dominio FABRIKAM.LOCAL con el nombre LAB-W10. Sin embargo, antes de llevar a cabo esta operación, asegúrese de que el puesto de trabajo pueda comunicarse correctamente (configuración de red y DNS) con el nuevo dominio al que se va a unir.



El equipo se encuentra ahora en el dominio FABRIKAM.LOCAL con el nombre LAB-W10

Ejemplo: mover un equipo de un grupo de trabajo a otro

```
PS C:\Windows\system32> Add-Computer -WorkgroupName "LAB-1"
ADVERTENCIA: Los cambios serán efectivos tras reiniciar
el equipo W10.
```

7. Reiniciar el canal seguro

Cuando se une un puesto de trabajo a un dominio Active Directory, se crea una cuenta de equipo. Como con los usuarios, esta cuenta está formada por un nombre de equipo (sAMAccountName) y una contraseña. Estas credenciales permiten establecer una relación de confianza entre el controlador de dominio y el puesto de trabajo. Así, el servicio NetLogon, por ejemplo, utiliza esta información para conectarse al dominio, lo que establece un canal seguro con el controlador de dominio. La contraseña se modifica automáticamente cada 30 días.

En ciertas situaciones, la relación de confianza entre ambas máquinas puede romperse (por ejemplo, tras una restauración del puesto de trabajo que tuviera, en ese momento, una clave antigua). Es aquí donde interviene el reinicio del canal seguro.

Para restablecerlo, se utiliza el cmdlet **`Reset-ComputerPassword`**. He aquí los parámetros:

Parámetro	Descripción
-Credential <PSCredential>	Especifica una cuenta de usuario del dominio que tiene permisos para realizar esta operación. Si no se especifica, se utilizará el usuario en curso.
-Server <String>	Especifica el nombre de un controlador de dominio que se utilizará para reinicializar la contraseña de la cuenta de equipo.

Ejemplo: reinicializar la contraseña de la cuenta de equipo

```
PS C:\Windows\system32> $cred = New-Object
System.Management.Automation.PSCredential("CONTOSO\Administrador",

(ConvertTo-SecureString "P@$$w0rd" -AsPlainText -Force))
PS C:\Windows\system32> Reset-ComputerMachinePassword -Credential
$cred
```

8. Probar y reparar el canal seguro

Existe un cmdlet dedicado que permite comprobar el canal seguro entre el puesto de trabajo y el dominio en el que está presente. Este cmdlet es el siguiente: **Test-ComputerSecureChannel**.

Además de su función para realizar la comprobación, este cmdlet ofrece también la posibilidad de reparar el canal seguro si la comprobación falla, mediante el parámetro **-Repair**. Este último va a reconstruir el canal seguro, eliminando previamente el que ya existía. Preste atención a no confundirlo con el reinicio que acabamos de ver anteriormente: aquí no se reinicializa la contraseña.

He aquí el conjunto de parámetros:

Parámetro	Descripción
-Credential <PSCredential>	Especifica una cuenta de usuario del dominio que tiene permisos para realizar esta operación. Si no se especifica, se utilizará la cuenta del usuario en curso.

Parámetro	Descripción
-Repair	Repara el canal seguro si este se comprueba defectuoso.
-Server <String>	Especifica el nombre de un controlador de dominio que se utilizará para reinicializar la contraseña de la cuenta de equipo.

Ejemplo: comprobar el canal seguro

```
PS C:\Windows\system32> Test-ComputerSecureChannel
True
```

Ejemplo: comprobar y reparar el canal seguro

```
PS C:\Windows\system32> Test-ComputerSecureChannel
False
PS C:\Windows\system32> $cred = New-Object
System.Management.Automation.PSCredential("CONTOSO\Administrador",

(ConvertTo-SecureString "P@$$w0rd" -AsPlainText -Force))
PS   C:\Windows\system32>   Test-ComputerSecureChannel   -Repair   -
Credential
$cred
True
```

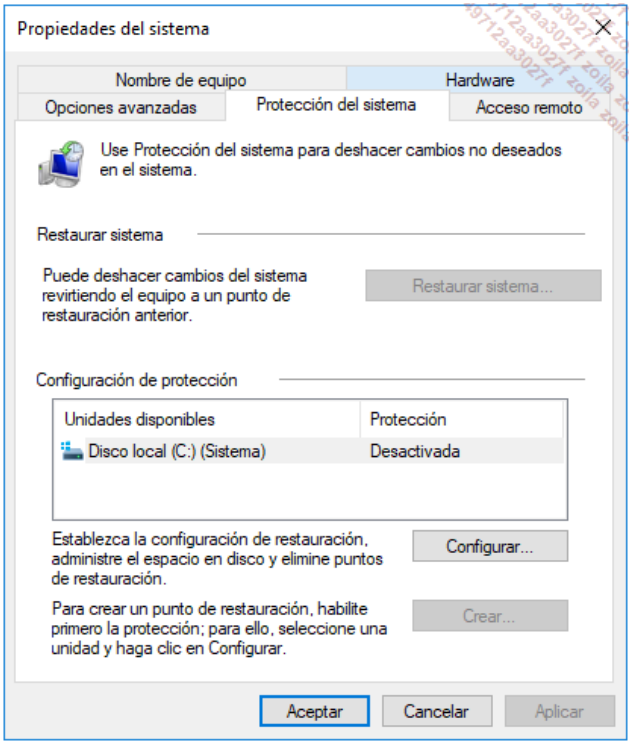
La protección del sistema

Windows posee una característica nativa de protección del sistema una vez se encuentra habilitada, cuyo objetivo es crear puntos de restauración de manera regular, y también tras un cambio a nivel del sistema operativo: instalación de controladores, actualizaciones, etc. Es posible crear un punto de restauración manualmente a través de la interfaz gráfica de Windows, y también mediante Windows PowerShell.

Los cmdlets de Windows Powershell le ofrecen la posibilidad de administrar esta característica. He aquí los comandos:

- **`Checkpoint-Computer`**: permite crear un punto de restauración.
- **`Disable-ComputerRestore`**: desactiva la característica de protección del sistema.
- **`Enable-ComputerRestore`**: activa la característica de protección del sistema.
- **`Get-ComputerRestorePoint`**: recupera la lista de puntos de restauración guardados en el puesto de trabajo.
- **`Restore-Computer`**: permite ejecutar una restauración del sistema.

Para acceder a las opciones de protección del sistema, haga clic con el botón derecho en el botón **Inicio** y seleccione **Sistema** en la lista de accesos directos. En la ventana que se abre, haga clic en el vínculo **Configuración avanzada del sistema** y a continuación, en la nueva ventana, haga clic en la pestaña **Protección del sistema**.



Ventana de configuración de la característica Protección del sistema

1. Activar la protección del sistema

`Enable-ComputerRestore` permite activar la protección del sistema. Este cmdlet se utiliza de una manera muy simple, especificando sobre qué unidad la desea activar.

Parámetro	Descripción
`-Drive` `<String[]>`	Indica la unidad o las unidades de tipo sistema de archivos. Las unidades de red no están permitidas.

Ejemplo: activación de la protección del sistema en la unidad C:

```
PS C:\Windows\system32> Enable-ComputerRestore -Drive C:
```

La activación de esta característica de Windows debe hacerse obligatoriamente en la unidad donde está instalado el sistema operativo. Tiene la posibilidad de activar la protección del sistema en otras unidades con objeto de aumentar el tamaño máximo disponible para guardar los puntos de restauración.

Así, si esta característica debe estar activa sobre la segunda unidad, debe informar obligatoriamente la unidad de sistema en el parámetro **-Drive**:

```
PS C:\Windows\system32> Enable-ComputerRestore -Drive D:
Enable-ComputerRestore: Include System Drive in the list of
Drives.

PS C:\Windows\system32> Enable-ComputerRestore -Drive C:,D:
```

Una vez realizada la activación, configure el espacio en disco máximo autorizado en cada una de las unidades. En efecto, cada punto de restauración creado ocupa espacio en disco (algunos megabytes como mínimo). En caso de superar el tamaño máximo autorizado, se eliminan los puntos de restauración más antiguos.

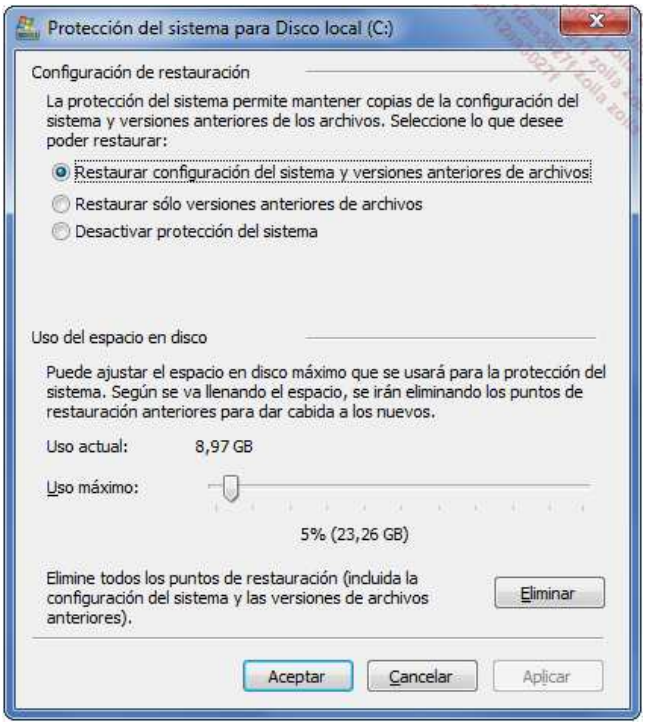
Windows 7

En Windows 7, la protección del sistema es ligeramente diferente, e incluye en realidad dos características que se han separado en las versiones posteriores de Windows:

- La protección del sistema, con los puntos de restauración y la posibilidad de restaurar el sistema a una fecha anterior.
- El historial de archivos, que ofrece la posibilidad de restaurar versiones previas de los archivos.

Así, utilizando el cmdlet **Enable-ComputerRestore** en Windows 7, activará a su vez la característica de historial de archivos; la captura de distintas versiones de los archivos se realiza en el momento de la creación de un punto de restauración.

Cabe destacar también que, por defecto, Windows 7 realiza la creación de un punto de restauración automáticamente con cada inicio del sistema, así como todos los días a medianoche.



Activación de la protección del sistema en Windows 7

2. Desactivar la protección del sistema

De manera inversa a **`Enable-ComputerRestore`**, **`Disable-ComputerRestore`** permite desactivar la protección del sistema en las unidades especificadas mediante el comando PowerShell.

Parámetro	Descripción
`-Drive` `<String[]>`	Indica la unidad o las unidades de tipo sistema de archivos.

Ejemplo: desactivación de la protección del sistema en la unidad C:

```
PS C:\Windows\system32> Disable-ComputerRestore -Drive C:
```

3. Explotar los puntos de restauración

Acabamos de ver cómo activar y desactivar la característica de protección del sistema dentro de Windows. Los siguientes cmdlets nos van a permitir obtener el conjunto de puntos de restauración ya creados, y también crear nuevos. Por último, una vez seleccionado el punto de restauración, un cmdlet específico nos permitirá ejecutar el procedimiento de restauración.

a. Obtener los puntos de restauración

Get-ComputerRestorePoint permite conocer el conjunto de puntos de restauración creados. El resultado del comando proporciona los detalles para cada uno de los puntos de restauración guardados en el puesto de trabajo. Si no devuelve ningún resultado es porque todavía no se ha creado ningún punto de restauración.

Parámetro	Descripción
-LastStatus	Obtiene el estado de la última operación de restauración del sistema.
-RestorePoint <Int32[]>	Obtiene los puntos de restauración con los números de secuencia especificados.

Ejemplo: obtener la lista de puntos de restauración creados

```
PS C:\Windows\system32> Get-ComputerRestorePoint | Select-Object
CreationTime,Description,SequenceNumber

CreationTime                Description
SequenceNumber
------------            -----------                         -----------
---
08/01/2014 22:47:08     Instalación Philips SPC 900NC P...
 45
28/01/2014 13:42:54     Windows Update
 46
28/01/2014 14:38:56     Windows Update
 47
28/01/2014 15:14:32     Windows Update
 48
31/01/2014 14:07:52     Installed Business Everywhere
49
01/02/2014 18:41:43     Punto de restauración 01/02
50
```

He aquí un resumen de la información que devuelve **Get-ComputerRestorePoint**:

Tipo	Descripción
CreationTime	Fecha y hora de creación del punto de restauración.

Tipo	Descripción
Description	Describe el punto de restauración y permite identificar rápidamente por qué motivo se ha creado. Se trata de una información que se introduce automáticamente cuando se crea un punto de restauración por el sistema (instalación de una aplicación, de un controlador o de actualizaciones de Windows Update). En el caso de un punto de restauración creado manualmente, puede introducir la descripción que desee.
SequenceNumber	Número de secuencia asociado al punto de restauración. Es necesario para lanzar la restauración con el cmdlet **`Restore-Computer`**. Este número de secuencia se incrementa automáticamente cada vez que se crea un punto de restauración.
EventType	Tipo de evento: BEGIN_SYSTEM_CHANGE.
RestorePointType	Tipo de punto de restauración.

`Get-ComputerRestorePoint` también permite conocer el estado del último intento de restauración del puesto de trabajo. Para ello, hay que especificar el parámetro **`-LastStatus`**.

Ejemplo: conocer el estado de la última restauración del sistema realizada

```
PS C:\Windows\system32> Get-ComputerRestorePoint -LastStatus
Error en el último intento de restauración del equipo.
```

b. Crear un punto de restauración

El cmdlet **`Checkpoint-Computer`** permite crear un punto de restauración del sistema en el puesto de trabajo. Observe que por defecto, y a partir de Windows 8, solo es posible crear un punto de restauración de esta manera una vez por día (cada 1440 minutos). Es posible modificar este valor agregando el siguiente elemento en el registro:

- Clave del registro: HKLM\SOFTWARE\Microsoft\Windows NT\Current Version\SystemRestore

- Nombre del valor: SystemRestorePointCreationFrequency
- Tipo de valor: DWORD
- Dato del valor: *insertar aquí la cifra deseada*

El valor del dato indica el intervalo (en minutos) requerido entre dos operaciones de creación de puntos restauración.

Así, para modificar el intervalo de creación de puntos de restauración a 12 h mediante un comando PowerShell:

```
Set-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\SystemRestore'                                    -Name
SystemRestorePointCreationFrequency
-Type DWORD -Value 1220
```

He aquí los parámetros necesarios para utilizar **`Checkpoint-Computer`**:

Parámetro	Descripción
`-Description` `<String>`	Agrega una descripción que se asociará al punto de restauración creado.

Ejemplo: crear un punto de restauración

```
PS     C:\WINDOWS\system32>     Checkpoint-Computer     -Description
'Checkpoint PS'
PS C:\WINDOWS\system32> Get-ComputerRestorePoint -RestorePoint
(Get-ComputerRestorePoint).SequenceNumber[-1] | Select-Object
CreationTime,Description,SequenceNumber

CreationTime                  Description
   SequenceNumber
------------           -----------                      -----------
---
29/12/2016  17:02:48       CheckPoint  PS
               51
```

Durante la creación del punto de restauración, un indicador muestra en la parte superior de la consola Windows PowerShell el estado del avance de la operación.

c. Restaurar el sistema

Ahora que sabemos cómo recuperar el número de secuencia de los puntos de restauración, puede usarlos. Para ello se utiliza el cmdlet **`Restore-Computer`**.

Parámetro	Descripción

Parámetro	Descripción
-Confirm	Permite pedir una confirmación antes de ejecutar el comando.
-RestorePoint <Int32>	Especifica el número de secuencia del punto de restauración.

Ejemplo: lanzar la restauración del sistema

Para restaurar un punto de restauración del sistema, debe utilizar el número de secuencia del punto de restauración que desea aplicar.

```
PS C:\Windows\system32> Restore-Computer -RestorePoint 45
```

Preste atención, el hecho de lanzar el comando **Restore-Computer** cierra la sesión en curso y reinicia el equipo. Tenga la precaución de guardar todos sus datos antes.

La restauración del sistema comienza, y el sistema operativo restaura el conjunto de archivos y parámetros de Windows contenidos en el punto de restauración especificado. Una vez terminada la operación, el equipo se reinicia automáticamente. A continuación, como hemos visto antes, puede, gracias a **Get-ComputerRestorePoint**, verificar si el procedimiento de restauración se ha desarrollado correctamente:

```
PS C:\Windows\system32> Get-ComputerRestorePoint -LastStatus
El equipo se ha restaurado en el punto de restauración especificado.
```

Ejemplo: lanzar la restauración del sistema con el último punto de restauración creado

Para lanzar de una manera más simple la restauración del sistema con el último punto de restauración creado, bastará con invocar al cmdlet **Get-ComputerRestorePoint** y recuperar el último número de secuencia.

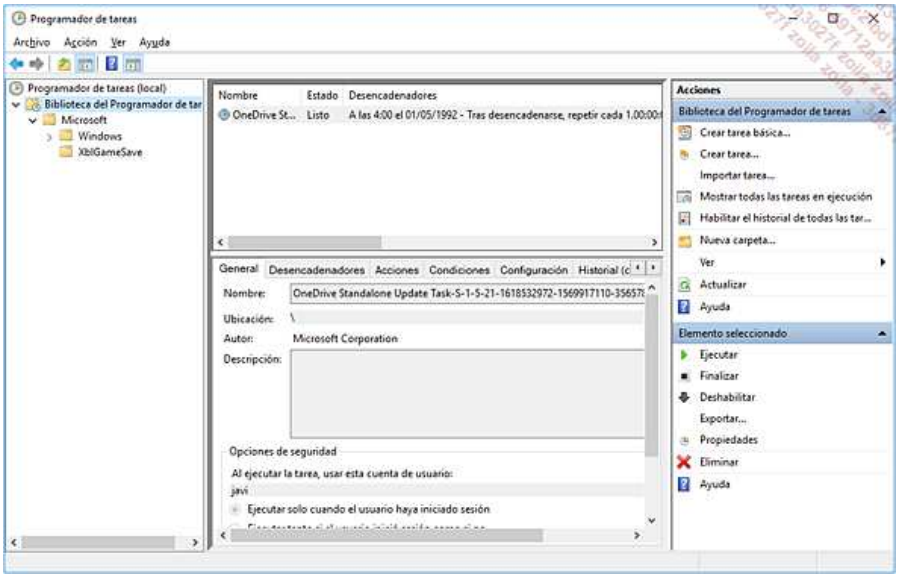
```
PS C:\Windows\system32> Restore-Computer -RestorePoint
(Get-ComputerRestorePoint).SequenceNumber[-1]
```

Las tareas planificadas

Es posible administrar tareas planificadas desde Windows 8. Si desea administrar esta característica en Windows 7, utilice el ejecutable schtasks.exe.

Veremos en esta sección la administración de las tareas planificadas con Windows PowerShell. Desde su creación hasta su configuración, es posible automatizar el conjunto de tareas por línea de comandos PowerShell.

El **Programador de tareas** , accesible desde las **Herramientas administrativas** del **Panel de control**, permite lanzar programas o scripts a horas concretas o tras un evento determinado. Por ejemplo, puede lanzar un ejecutable todas las tardes a las 23 horas, o bien un script de PowerShell tras la aparición de un evento determinado en el registro de eventos de Windows.



El Programador de tareas

He aquí los cmdlets que vamos a describir para la administración de las tareas planificadas:

- **`Get-ScheduledTask`**: obtener información acerca de las tareas planificadas existentes.
- **`New-ScheduledTask`**: crear una tarea planificada.

- **New-ScheduledTaskAction**: crear un nuevo objeto de tipo Scheduled-TaskAction.
- **New-ScheduledTaskTrigger**: crear un nuevo objeto de tipo ScheduledTaskTrigger.
- **New-ScheduledTaskSettingsSet**: crear un nuevo objeto de tipo ScheduledTaskSettingsSet.
- **Register-ScheduledTask**: registrar una tarea planificada.
- **Set-ScheduledTask**: modificar una tarea planificada.
- **Enable-ScheduledTask**: activar una tarea planificada.
- **Disable-ScheduledTask**: desactivar una tarea planificada.
- **Start-ScheduledTask**: arrancar una tarea planificada.
- **Stop-ScheduledTask**: detener una tarea planificada.
- **Export-ScheduledTask**: exportar tareas planificadas.
- **Unregister-ScheduledTask**: eliminar del registro una tarea planificada.

1. Visualizar las tareas planificadas

Antes de ver cómo crear nuevas tareas planificadas, veamos cómo recuperar las existentes. El cmdlet **Get-ScheduledTask** sirve para recuperar la lista de tareas planificadas, así como su ubicación.

Parámetro	Descripción
-TaskName <String[]>	Indica el nombre de una o varias tareas planificadas.
-TaskPath <String[]>	Indica la ruta de acceso a la carpeta de la tarea planificada en el Programador de tareas. «****» representa la carpeta raíz.

Ejemplo: mostrar el conjunto de tareas planificadas registradas

```
PS C:\Windows\system32> Get-ScheduledTask | Select-Object
TaskName,State

TaskName
 State
--------
```

```
 -----
OneDrive Standalone Update Task v2
 Ready
.NET Framework NGEN v4.0.30319
 Ready
.NET Framework NGEN v4.0.30319 64
Ready
.NET Framework NGEN v4.0.30319 64 Critical
Disabled
.NET Framework NGEN v4.0.30319 Critical
 Disabled
AD RMS Rights Policy Template Management (Automated)
Disabled
D RMS Rights Policy Template Management (Manual)
Ready
EDP Policy Manager
 Ready
PolicyConverter
 Disabled
SmartScreenSpecific
Ready
VerifiedPublisherCertStoreCheck
 Disabled
Microsoft Compatibility Appraiser
Ready
ProgramDataUpdater
 Ready
tartupAppTask
 Ready
appuriverifierdaily
Ready
appuriverifierinstall
Ready
CleanupTemporaryState
Ready
[...]
```

Para conocer todas las propiedades de una tarea planificada, puede pasarle el resultado del comando **Get-SchedultedTask** a **Format-List**:

```
PS       C:\Windows\system32>       Get-ScheduledTask       -TaskName
"ScheduledDefrag" | fl *


State                 : Ready
Actions               : {MSFT_TaskExecAction}
Author                : Microsoft Corporation
Date                  :
Description                : Esta tarea optimiza las unidades de
almacenamiento
                       locales.
Documentation         :
Principal             : MSFT_TaskPrincipal2
SecurityDescriptor     :  D:AI(A;;FA;;;BA)(A;;FA;;;SY)(A;;FRFX;;;LS)
(A;;FR;;;AU)
Settings              : MSFT_TaskSettings3
Source                : Microsoft Corporation
TaskName              : ScheduledDefrag
TaskPath              : \Microsoft\Windows\Defrag\
Triggers              :
```

```
URI                   : Microsoft\Windows\Defrag\ScheduledDefrag
Version               :
PSComputerName        :
CimClass              : Root/Microsoft/Windows/TaskScheduler:
                          MSFT_ScheduledTask
CimInstanceProperties: {Actions, Author, Date, Description...}
CimSystemProperties                                                    :
Microsoft.Management.Infrastructure.CimSystemProperties
```

De este modo, es posible recuperar la información correspondiente a la tarea planificada seleccionada. El estado de la tarea, la descripción, las acciones y los desencadenadores son los parámetros más importantes que constituyen una tarea planificada.

Ejemplo: recuperar la acción de una tarea planificada

He aquí cómo recuperar la acción realizada al lanzarse la tarea SheduledDefrag.

```
PS C:\Windows\system32> $tpDefrag = Get-ScheduledTask -TaskName
"ScheduledDefrag"
PS C:\Windows\system32> $tpDefrag.Actions


Id               :
Argumentos        : -c -h -o -$
Execute          : %windir%\system32\defrag.exe
WorkingDirectory:
PSComputerName   :
```

La propiedad `Execute` indica el programa que se ejecuta, y la propiedad `Argumentos` indica los argumentos que se pasan al archivo binario.

2. Crear una tarea planificada

Para crear una tarea planificada, no basta con un único cmdlet. Antes de utilizar **`New-ScheduledTask`**, previamente hay que crear el conjunto de objetos que la componen.

He aquí una tabla que describe la información que se requiere para crear una tarea planificada:

Tipo de dato	Descripción	Cmdlet necesario
Acción	Se trata de un programa que se ejecutará, con sus posibles parámetros.	**`New-ScheduledTaskAction`**

Tipo de dato	Descripción	Cmdlet necesario
Desencadenador (Trigger)	Se trata del momento en que el evento desencadenará la ejecución de la tarea planificada.	**New-ScheduledTaskTrigger**
Cuenta de usuario	Se trata de la cuenta de usuario con la que se ejecutará el programa.	Ninguno. Cadena de caracteres.
Parámetros diversos	Se trata de definir otros parámetros que influyen en el comportamiento de la tarea.	**New-ScheduledTaskSettingsSet**

Una vez creados todos los objetos y definidos en variables, puede utilizarse el cmdlet **New-ScheduledTask** para crear la tarea planificada. He aquí un ejemplo, paso a paso.

Paso 1: creación de la acción

Utilice el cmdlet **New-ScheduledTaskAction**. He aquí una presentación de los parámetros:

Parámetro	Descripción
-Argument <String>	Especifica los argumentos para la línea de comandos.
-Execute <String>	Indica la ruta de acceso al archivo ejecutable.

```
PS C:\Windows\system32> $action = New-ScheduledTaskAction
-Execute "PowerShell.exe" -Argument
"C:\Temp\myPowerShellScript.ps1"
```

Paso 2: creación del evento

Utilice el cmdlet **New-ScheduledTaskTrigger**. He aquí una presentación de los parámetros:

Parámetro	Descripción
-At <DateTime>	Especifica una fecha y una hora para el desencadenador de la tarea planificada.
-AtLogOn	Indica que el desencadenador arranca la tarea planificada cuando se inicia una sesión de usuario.
-AtStartup	Indica que el desencadenador arranca la tarea planificada cuando arranca el sistema.
-Daily	Indica que el desencadenador arranca la tarea planificada todos los días de manera recurrente.
-DaysInterval <Int32>	Número de días de intervalo entre dos ocurrencias de la tarea planificada.
-DayOfWeek <DayOfWeek[]>	Designa los días en los que el Programador de tareas ejecutará la tarea planificada. El valor por defecto es: {DayOfWeek.Monday, DayOfWeek.Tuesday, DayOfWeek.Wednesday, DayOfWeek.Thursday, DayOfWeek.Friday, DayOfWeek.Saturday, DayOfWeek.Sunday}.
-Once	Arranca la tarea planificada una única vez, a la fecha y hora definidas mediante el parámetro **-At**.
-Weekly	Indica que el desencadenador arranca la tarea planificada todas las semanas de forma recurrente.
-WeeksInterval <Int32>	Número de semanas de intervalo entre dos ocurrencias de la tarea planificada.

```
PS C:\Windows\system32> $trigger = New-ScheduledTaskTrigger
-Daily -At 6am
```

Paso 3: creación de un objeto que contiene todos los parámetros de la tarea planificada

Utilice el cmdlet **New-ScheduledTaskSettingsSet**. El comando puede ejecutarse sin ningún parámetro; no obstante, he aquí un resumen de qué se puede definir para la tarea planificada:

Parámetro	Descripción
-AllowStartIfOnBatteries	Autoriza al Programador de tareas a arrancar la tarea planificada si el equipo se encuentra trabajando con batería.
-Disable	Indica que la tarea planificada está desactivada.
-DontStopIfGoingOnBatteries	Indica que la tarea planificada no se detiene si el equipo pasa a trabajar con batería.
-DontStopOnIdleEnd	Indica que el Programador de tareas no terminará la ejecución de la tarea planificada si el equipo sale de la inactividad antes de que esta termine.
-Hidden	Indica que la tarea planificada no es visible en el Programador de tareas en modo gráfico.

```
PS C:\Windows\system32> $settings = New-ScheduledTaskSettingsSet
```

Paso 4: creación de la tarea planificada

Utilice el cmdlet **New-ScheduledTask**. He aquí el detalle de algunos parámetros:

Parámetro	Descripción

Parámetro	Descripción
-Action <CimInstance[]>	Especifica un array con una o varias acciones para ejecutar por la tarea planificada.
-Description <String>	Indica una descripción de la tarea planificada.
-Settings <CimInstance>	Especifica la configuración y los parámetros de la tarea planificada.
-Trigger <CimInstance[]>	Especifica un array de uno o varios desencadenadores que provocarán el arranque de la tarea planificada.

```
PS C:\Windows\system32> $inputObject = New-ScheduledTask -Action
$action -Trigger $trigger -Settings $settings
```

Paso 5: registro de la tarea planificada en el equipo local

Último paso: el objeto creado por **New-ScheduledTask** debe registrarse a continuación. Este es el rol de **Register-ScheduledTask**. He aquí una presentación de los parámetros para este cmdlet:

Parámetro	Descripción
-InputObject <CimInstance>	Especifica la entrada para este comando (objeto que contiene la definición de una tarea planificada).
-Password <String>	Indica la contraseña de la cuenta de usuario designada por **-User**.
-TaskName <String>	Especifica el nombre de la tarea planificada.
-TaskPath <String>	Indica la ruta de acceso a la carpeta de la tarea planificada en el Programador de tareas. «****» representa la carpeta raíz.
-User <String>	Especifica la cuenta de usuario con la que se ejecutará la tarea planificada.

```
PS C:\Windows\system32> Register-ScheduledTask -TaskName
"Ejecución de un script PowerShell" -InputObject $inputObject
```

Por defecto, la tarea planificada se crea en la raíz del árbol del Programador de tareas. Utilizando el parámetro **-TaskPath**, tiene la posibilidad de registrarla en alguna otra carpeta.

En el ejemplo anterior, cuando el cmdlet **Register-ScheduledTask** registra la tarea planificada, esta se ejecutará con la cuenta del usuario con que se ha ejecutado el comando.

Para definir otra cuenta de usuario, hay que especificarla durante la ejecución de **Register-ScheduledTask**:

```
PS C:\Windows\system32> $cuenta = "DOMINIO\usuario"
PS C:\Windows\system32> $password = "contraseña"
PS C:\Windows\system32> Register-ScheduledTask -TaskName
"Nombre de la tarea planificada" -InputObject $inputObject -User
$cuenta
-Password $password
```

3. Modificar una tarea planificada

Veremos a continuación cómo modificar o actualizar una tarea planificada existente. Los elementos que cambian con mayor frecuencia son la acción, el desencadenador y la cuenta de usuario con la que se va a ejecutar la tarea. Cada modificación se detallará y vendrá acompañada de un ejemplo. El cmdlet encargado de actualizar una tarea planificada es **Set-ScheduledTask**. La mayoría de los parámetros ya se han abordado y se repiten aquí:

Parámetro	Descripción
-Action <CimInstance[]>	Especifica un array con una o varias acciones para ejecutar por la tarea planificada.
-Password <String>	Indica la contraseña de la cuenta de usuario designada por **-User**.
-Settings <CimInstance>	Especifica la configuración y los parámetros de la tarea planificada.
-TaskName <String>	Especifica el nombre de la tarea planificada.

Parámetro	Descripción
-TaskPath <String>	Indica la ruta de acceso a la carpeta de la tarea planificada en el Programador de tareas. «\» representa la carpeta raíz.
-Trigger <CimInstance[]>	Especifica un array con uno o varios desencadenadores que provocarán el arranque de la tarea planificada.
-User <String>	Especifica una cuenta de usuario con la que se ejecutará la tarea planificada.

Modificar la acción

Como con la creación de una nueva tarea planificada, hay que crear un objeto que contiene la nueva acción que se va a actualizar. En el ejemplo, se utiliza **New-ScheduledTaskAction** para definir la nueva acción y se registra en una variable:

```
PS C:\Windows\system32> $newAction = New-ScheduledTaskAction
-Execute myProgram.exe
```

A continuación, con **Set-ScheduledTask**, puede actualizar la tarea planificada con la nueva acción:

```
PS C:\Windows\system32> Set-ScheduledTask -TaskName T1 -Action
$newAction
```

El uso de **Set-ScheduledTask** va a borrar el conjunto de acciones que existen hasta ahora en la tarea planificada. Si se deben registrar varias acciones, entonces hay que recrear el conjunto mediante el cmdlet **New-Scheduled TaskAction** antes de utilizar **Set-ScheduledTask** (veremos un ejemplo en el siguiente párrafo).

En el caso de tener que ejecutar varios programas o scripts, será preciso crear tantas acciones como sea necesario, y aplicar la configuración a continuación:

```
PS C:\Windows\system32> $newAction = New-ScheduledTaskAction
-Execute myProgram.exe
PS C:\Windows\system32> $newAction2 = New-ScheduledTaskAction
-Execute myScript.ps1
```

```
PS C:\Windows\system32> Set-ScheduledTask -TaskName T1 -Action
$newAction,$newAction2
```

En caso de que la tarea planificada que desea actualizar se ejecute con una cuenta de usuario diferente a la del usuario autenticado, debe especificar de nuevo la cuenta y su contraseña:

```
PS C:\Windows\system32> $newAction = New-ScheduledTaskAction
-Execute myProgram.exe
PS C:\Windows\system32> Set-ScheduledTask -TaskName T1 -Action
$newAction -User "Contoso\Julien" -Password "P@$$w0rd"
```

Modificar el desencadenador

La modificación del desencadenador es comparable a lo que hemos visto hasta ahora. Debe crearse un nuevo objeto mediante el cmdlet **`New-ScheduledTaskTrigger`** y, a continuación, definirlo en una tarea planificada existente:

```
PS C:\Windows\system32> $newTrigger = New-ScheduledTaskTrigger
-At 10pm -DaysOfWeek 02 -Weekly
PS C:\Windows\system32> Set-ScheduledTask -TaskName T1 -Trigger
$newTrigger
```

Este comando reemplaza todos los desencadenadores presentes en la tarea planificada. En lo sucesivo, esta se ejecutará todos los martes de cada semana a las 22 horas.

Es posible definir varios desencadenadores. En el siguiente ejemplo, se crean dos nuevos desencadenadores y, a continuación, se especifican en la tarea planificada:

```
PS C:\Windows\system32> $trigger1 = New-ScheduledTaskTrigger
-At 10pm -DaysOfWeek 02 -Weekly
PS C:\Windows\system32> $trigger2 = New-ScheduledTaskTrigger
-At 8am -Daily
PS C:\Windows\system32> Set-ScheduledTask -TaskName T1 -Trigger
$trigger1,$trigger2
```

Modificar la cuenta de usuario

Es posible modificar la cuenta de usuario con la que se va a ejecutar la tarea planificada, por ejemplo tras un cambio en la contraseña. Para ello, basta simplemente con redefinir la tarea planificada con las nuevas credenciales (usuario y contraseña):

```
PS C:\Windows\system32> $task = Get-ScheduledTask -TaskName T1
PS C:\Windows\system32> Set-ScheduledTask -InputObject $task
-User "Contoso2\Admin" -Password "newP@$$w0rd"
```

4. Activar/desactivar una tarea planificada

Es posible desactivar una tarea planificada en cualquier momento, lo que provocará que esta quede totalmente inerte. Con ello, la tarea quedará registrada y seguirá presente en el puesto de trabajo, pero no se ejecutará hasta que no se vuelva a activar. Del mismo modo, no es posible ejecutar manualmente una tarea desactivada a través de la interfaz gráfica de Windows.

Para activar y desactivar una tarea planificada, debe utilizar respectivamente los cmdlets **`Enable-ScheduledTask`** y **`Disable-ScheduledTask`**, a los que hay que especificar el nombre de la tarea planificada.

Parámetro	Descripción
`-TaskName` `<String>`	Especifica el nombre de la tarea planificada.
`-TaskPath` `<String>`	Indica la ruta de acceso a la carpeta de la tarea planificada en el Programador de tareas. «****» representa la carpeta raíz.

Ejemplo 1: desactivar una tarea planificada

```
PS C:\Windows\system32> Disable-ScheduledTask -TaskName T1
```

Ejemplo 2: activar una tarea planificada

```
PS C:\Windows\system32> Enable-ScheduledTask -TaskName T1
```

Ejemplo 3: activar varias tareas planificadas

Si deben activarse o desactivarse varias tareas planificadas, es posible utilizar un pipe que transfiera el resultado del cmdlet **`Get-ScheduledTask`** a **`Enable-ScheduledTask`**.

```
PS C:\Windows\system32> Get-ScheduledTask -TaskPath "\Mis
tareas\"
| Enable-ScheduledTask
```

El comando anterior reactiva todas las tareas planificadas presentes en la carpeta \Mis tareas de la Biblioteca del Programador de tareas.

Información adicional relativa al parámetro **`-TaskPath`**: el carácter «****» designa la raíz, es decir, la carpeta Biblioteca del

Programador de tareas.

5. Arrancar/detener una tarea planificada

Por último, si no desea esperar al siguiente inicio automático de la tarea planificada, siempre puede lanzarla y detenerla manualmente. Para ello, se utilizan respectivamente los cmdlets **`Start-ScheduledTask`** y **`Stop-ScheduledTask`**.

Parámetro	Descripción
`-TaskName` `<String>`	Especifica el nombre de la tarea planificada.
`-TaskPath` `<String>`	Indica la ruta de acceso a la carpeta de la tarea planificada en el Programador de tareas. «****» representa la carpeta raíz.

Ejemplo 1: lanzar la ejecución de varias tareas, indicando una carpeta

```
PS C:\Windows\system32> Get-ScheduledTask -TaskPath
"\myPersonalTasks\" | Start-ScheduledTask
```

Ejemplo 2: lanzar la ejecución de varias tareas, indicando los nombres

```
PS C:\Windows\system32> Get-ScheduledTask -TaskName "Mi tarea
1","Mi tarea 2" | Start-ScheduledTask
```

Ejemplo 3: detener una tarea en ejecución

```
PS C:\Windows\system32> Stop-ScheduledTask -TaskName T1
```

6. Exportar una tarea planificada

`Export-ScheduledTask` permite exportar tareas planificadas. Este tipo de operación es interesante por dos motivos:

- Para realizar una copia de seguridad.
- Para crear nuevas tareas a partir de la plantilla exportada.

Sin embargo, el cmdlet **`Export-ScheduledTask`** no crea un archivo, ni tampoco un objeto que pueda almacenarse en una variable. Este comando devuelve únicamente una cadena de

caracteres, en formato XML, de modo que habrá que escribir el flujo generado en un archivo .xml. He aquí un resumen de los parámetros:

Parámetro	Descripción
-TaskName <String>	Especifica el nombre de la tarea planificada.
-TaskPath <String>	Indica la ruta de acceso a la carpeta de la tarea planificada en el Programador de tareas. «\» representa la carpeta raíz.

Ejemplo: exportar una tarea planificada en un archivo XML

```
PS C:\Windows\system32> Get-ScheduledTask -TaskName T1 |
Export-ScheduledTask | Out-File -FilePath C:\Temp\T1.xml
```

7. Importar una tarea planificada

Veamos cómo importar tareas planificadas. Esta operación se realiza directamente con el cmdlet **Register-ScheduledTask** que hemos visto en la creación de tareas planificadas. Hay que utilizar el parámetro **-XML** <String>, al que se le pasará la información necesaria.

Parámetro	Descripción
-Password <String>	Indica la contraseña de la cuenta de usuario designada por **-User**.
-TaskName <String>	Especifica el nombre de la tarea planificada.
-TaskPath <String>	Indica la ruta de acceso a la carpeta de la tarea planificada en el Programador de tareas. «\» representa la carpeta raíz.
-User <String>	Especifica una cuenta de usuario con la que se ejecutará la tarea planificada.
-XML <String>	Especifica la cadena de caracteres en formato XML que contiene una definición de la tarea planificada.

Precaución: el parámetro **-Xml** espera como variable una cadena de caracteres, y no un archivo, de modo que resulta

imprescindible prestar atención al tipo de dato que se le pasará a este parámetro.

```
PS C:\Windows\system32> Register-ScheduledTask -Xml (Get-Content
C:\Temp\T1.xml | Out-String) -TaskName "newT1"
```

Durante la importación, siempre es posible agregar los parámetros **-User** y **-Password** para definir las credenciales de la cuenta con la que se ejecutará la tarea. Además, el parámetro **-Force** puede ser necesario para borrar cualquier tarea existente que tenga el mismo nombre. Por último, precisaremos que la tarea se crea en la raíz de la Biblioteca del Programador de tareas. Para registrar la tarea importada en otra carpeta, está disponible el parámetro **-TaskPath**.

8. Eliminar del registro una tarea planificada

Como hemos visto en la creación o la importación de una tarea planificada, hay que registrar antes obligatoriamente una tarea planificada para que esta aparezca en el Programador de tareas. La operación inversa, la eliminación del registro (y, por consiguiente, su borrado) de una tarea planificada puede llevarse a cabo con el cmdlet **Unregister-ScheduledTask**. He aquí una presentación de los parámetros:

Parámetro	Descripción
-InputObject <CimInstance[]>	Especifica la entrada para este comando (objeto que contiene la definición de una tarea planificada).
-TaskName <String[]>	Indica el nombre de una o varias tareas planificadas.
-TaskPath <String[]>	Indica la ruta de acceso a la carpeta de la tarea planificada en el Programador de tareas. «\» representa la carpeta raíz.

Ejemplo 1: eliminar del registro una tarea planificada

```
PS C:\Windows\system32> Unregister-ScheduledTask -TaskName T1
```

```
Confirmar
¿Está seguro de que desea realizar esta acción?
Realizando la operación 'Delete' en el destino '\Test T1'.
[S] Sí  [O] Sí a todo  [N] No  [T] No a todo  [U]
Suspender  [?] Ayuda (el valor predeterminado es 'S'): S
PS C:\Windows\system32> # La tarea planificada se ha eliminado
```

Ejemplo 2: eliminar del registro una tarea planificada con -InputObject

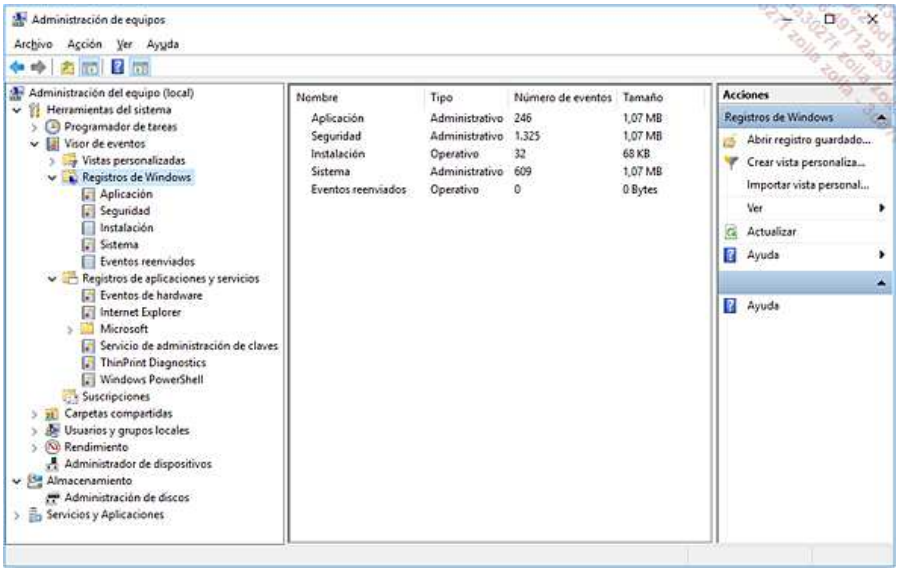
```
PS C:\Windows\system32> $inputObject = Get-ScheduledTask -TaskName
T1
PS C:\Windows\system32> Unregister-ScheduledTask -InputObject
$inputObject -Confirm:$false
```

- Puede omitir la petición de confirmación utilizando el parámetro **-Confirm: $false**.

Los registros de eventos

Cada ocurrencia de un evento (acción, error de un programa o de un servicio, inicio de sesión, etc.) se recoge en los registros de Windows. Estos son el equivalente de los logs para el sistema operativo. Así, la información contenida permite resolver problemas en Windows y también en otros programas instalados en el equipo.

Es posible consultar los registros a través de la interfaz gráfica que proporciona el Visor de eventos. Windows PowerShell posee dos cmdlets para consultar los registros de eventos: **`Get-EventLog`** y **`Get-WinEvent`**. El primer cmdlet permite consultar únicamente los registros de eventos clásicos (Aplicación, Sistema, etc.), el segundo ofrece un acceso al conjunto de registros de eventos generados con la tecnología Windows Event Log (Crimson), introducida con Windows Vista.



El Visor de eventos y los Registros de Windows

Listado de los registros accesibles con Get-EventLog

```
PS C:\Windows\system32> Get-EventLog -LogName *

  Max(K) Retain OverflowAction       Entries Log
  ------ ------ --------------       ------- ---
  20 480      0 OverwriteAsNeeded      10 571 Application
  20 480      0 OverwriteAsNeeded           0 HardwareEvents
```

```
     512      7 OverwriteOlder              0 Internet Explorer
   20  480         0  OverwriteAsNeeded               0  Key  Management
Service
     128      0 OverwriteAsNeeded           0 Oalerts
  20 480      0 OverwriteAsNeeded      30 455 Security
  20 480      0 OverwriteAsNeeded       4 172 System
  15 360      0 OverwriteAsNeeded         800 Windows PowerShell
```

Listado de los registros accesibles con Get-WinEvent

```
PS C:\Windows\system32> Get-WinEvent -ListLog *

LogMode   MaximumSizeInBytes RecordCount LogName
-------   ------------------ ----------- -------
Circular            20971520       10571 Application
Circular            20971520           0 HardwareEvents
Circular             1052672           0 Internet Explorer
Circular            20971520           0 Key Management Service
Circular             1052672           0 OAlerts
Circular            20971520       30455 Security
Circular            20971520        4172 System
Circular            15728640         800 Windows PowerShell
Circular            20971520             ForwardedEvents
Circular                  10485760                0  Microsoft-AppV-
Client/Admin
Circular                  10485760                0  Microsoft-AppV-
Client/Operational
Circular                  10485760                0  Microsoft-AppV-
Client/Virtual Applic...
Circular             1052672         418 Microsoft-Client-Licensing-
Platform/...
Circular                   1052672                    Microsoft-IIS-
Configuration/Administ...
Circular                   1052672                    Microsoft-IIS-
Configuration/Operational
Circular                   1052672                    Microsoft-IIS-
Logging/Logs
Circular              1052672            0 Microsoft-User Experience
Virtualiza...
Circular              1052672            0 Microsoft-User Experience
Virtualiza...
Circular              1052672            0 Microsoft-User Experience
Virtualiza...
Circular              1052672            0 Microsoft-User Experience
Virtualiza...
Circular                  1052672              92  Microsoft-Windows-
AAD/Operational
Circular             1052672           0 Microsoft-Windows-All-User-
Install-A...
[...]
```

Como administrador del sistema, la actividad principal sobre los registros de eventos será auditar los fallos y filtrar los distintos registros de seguridad, y también administrarlos. Los distintos cmdlets de nombre ***-EventLog** están mejor adaptados para este uso y se detallarán a continuación.

He aquí la lista de cmdlets estudiados y un breve resumen de su función:

- **Get-EventLog**: obtener la lista de los registros de eventos, el conjunto de eventos y su información.
- **Limit-EventLog**: administrar el tamaño y el ciclo de vida de los registros de eventos.
- **Clear-EventLog**: borrar el contenido de un registro de eventos.
- **New-EventLog**: crear un nuevo registro de eventos.
- **Write-EventLog**: escribir una entrada en un registro de eventos.
- **Remove-EventLog**: eliminar un registro de eventos.

1. Obtener información acerca de los eventos

El cmdlet **Get-EventLog** permite recuperar o consultar los eventos contenidos en un registro de eventos. Veremos, a través de los distintos parámetros, cómo obtener un resultado filtrado. Pero, en primer lugar, he aquí los parámetros utilizados:

Parámetro	Descripción
-After <DateTime>	Obtiene únicamente los eventos que se producen tras la fecha y hora especificadas.
-Before <DateTime>	Obtiene únicamente los eventos que se producen antes de la fecha y hora especificadas.
-EntryType <String[]>	Obtiene únicamente los eventos con el tipo de entrada especificado.
-Index <Int32[]>	Obtiene únicamente los eventos con los valores de índice especificados.
-InstanceId <Int64[]>	Obtiene únicamente los eventos con los ID de instancia especificados.
-LogName <String>	Especifica el nombre del registro de eventos.
-Message <String>	Obtiene los eventos que incluyen la cadena especificada en sus mensajes.

Parámetro	Descripción
-Newest <Int32>	Especifica el número máximo de eventos recuperados.
-Source <String[]>	Obtiene los eventos que se han escrito en el registro desde los orígenes especificados.
-UserName <String[]>	Obtiene únicamente los eventos que tienen asociados los nombres de usuario especificados.

Ejemplo: consultar el registro de eventos Sistema

```
PS C:\Users\Admin> Get-EventLog -LogName System | Select-Object
Index,TimeGenerated,EntryType,Message

Index TimeGenerated         EntryType Message
----- -------------         --------- -------
48681  04/01/2017  17:01:33  Information  El  explorador  forzó  una
elección en la ...
48680 04/01/2017 15:58:01 Information Instalación correcta: Windows
instaló...
48679 04/01/2017 15:57:49 Information Instalación iniciada: Windows
ha comenz...
48678  04/01/2017  15:57:49  Information  Windows  Update  comenzó  a
descargar una...
48677  04/01/2017  14:26:54  Information  La  descripción  del  ID  de
evento '16'...
48676 04/01/2017 12:00:00 Information El tiempo límite del sistema
es de 15683...
48675 04/01/2017 11:59:59 Information El tiempo límite del sistema
es de 15683...
48674 04/01/2017 09:50:30 Information Instalación correcta: Windows
instaló...
48673 04/01/2017 09:50:30 Information Instalación iniciada: Windows
ha comenz...
48672  04/01/2017  09:50:24  Information  La  descripción  del  ID  de
evento '16'...
48671  04/01/2017  09:50:19  Information  El  servicio  de  hora  está
sincronizando...
48670  04/01/2017  09:50:10  Information  Windows  Update  comenzó  a
descargar una...
48669 04/01/2017 09:50:03        Error El servicio de hora detectó
que la hora...
48668  04/01/2017  09:49:56  Information  La  descripción  del  ID  de
evento '16'...
48667  04/01/2017  09:49:53  Information  La  descripción  del  ID  de
evento '16'...
48666  04/01/2017  09:49:53  Information  La  descripción  del  ID  de
evento '16'...
48665  04/01/2017  09:49:50  Information  La  descripción  del  ID  de
evento '16'...
48664  04/01/2017  09:49:47  Information  El  proveedor  de  tiempo
NtpClient está...
[...]
```

Puede ejecutar el comando anterior sin indicar el cmdlet **`Select-Object`**. Así, se mostrarán todos los elementos descritos en la tabla anterior.

Como puede observar, la lista puede ser muy extensa. Conviene, por tanto, filtrar el resultado para ajustar la búsqueda.

Durante la ejecución del comando **`Get-Eventlog`**, este devuelve seis columnas, por las que es posible filtrar. En primer lugar, he aquí una explicación de estas columnas:

Columna	Parámetro utilizado para el filtrado	Descripción
Index	**`-Index`** `<Int32[]>`	El índice es, simplemente, un número asociado al evento que se incrementa automáticamente con cada nueva entrada.
TimeGenerated	**`-After`** `<DateTime>` **`-Before`** `<DateTime>`	Fecha y hora a las que se ha generado el evento.
EntryType	**`-EntryType`** `<String[]>`	Define la criticidad del evento.
Source	**`-Source`** `<String[]>`	Indica el origen que ha generado el evento, por ejemplo: Disk, Kernel-Power, GroupPolicy, WindowsUpdateClient, etc.
InstanceID	**`-InstanceId`** `<Int64[]>`	Identifica de manera unívoca una entrada de evento para un origen de evento concreto.

Columna	Parámetro utilizado para el filtrado	Descripción
Message	**-Message** <String>	Mensaje asociado al evento. Ofrece información o detalles acerca del error que se ha producido.

Selección en función del número de índice

Para recuperar más información acerca de un evento concreto, hay que indicar el número de índice del evento, mediante el parámetro **-Index**:

```
PS C:\Windows\system32> Get-EventLog -LogName System -Index 48378
|
Select-Object Index,TimeGenerated,EntryType,Message

Index TimeGenerated          EntryType   Message
----- -------------          ---------   -------
48378 03/01/2017 18:11:44   Information    Microsoft (R) Windows (R)
10.00. 1439...


PS C:\Windows\system32> Get-EventLog -LogName System -Index 48378 |
fl *


EventID            : 6009
MachineName        : W10.CONTOSO.LOCAL
Data               : {}
Index              : 48378
Category           : (0)
CategoryNumber     : 0
EntryType          : Information
Message                : Microsoft (R) Windows (R) 10.00. 14393
Multiprocessor Free.
Source             : EventLog
ReplacementStrings: {10.00., 14393, , Multiprocessor Free...}
InstanceId         : 2147489657
TimeGenerated      : 03/01/2017 18:11:44
TimeWritten        : 03/01/2017 18:11:44
UserName           :
Site               :
Container          :
```

Filtrado en función de la fecha

Es posible filtrar el registro de eventos en función de los datos devueltos. Veremos, en primer lugar, cómo filtrar por fecha, en particular antes de una fecha concreta (con el parámetro **-Before**):

```
PS C:\Windows\system32> $date = Get-Date -Date 01/05/16
PS C:\Winsows\system32> Get-EventLog -LogName System -Before $date
|
Select-Object Index,TimeGenerated,EntryType,Message

Index TimeGenerated           EntryType  Message
----- -------------           ---------  -------
   699 30/04/2016 12:01:00     Information   El tiempo límite del
sistema es de...
  698 30/04/2016 02:16:44          Error  La descripción del ID de
evento...
  697 30/04/2016 02:16:40          Error  La descripción del ID de
evento...
   696 30/04/2016 02:16:39           Warning   NtpClient no pudo
establecer un...
   695 30/04/2016 02:16:38           Warning   NtpClient no pudo
establecer un...
  694 30/04/2016 00:38:09          Warning   No se pudo conectar:
Windows ...
   693 30/04/2016 12:00:12     Information   El tiempo límite del
sistema es de...
[...]
```

Hacemos lo mismo, pero esta vez después de la fecha mencionada (con el parámetro **-After**):

```
PS C:\Windows\system32> $date = Get-Date -Date 01/05/16 -Hour 18
PS C:\Windows\system32> Get-EventLog -LogName System -After $date |

Select-Object Index,TimeGenerated,EntryType,Message

Index TimeGenerated           EntryType   Message
----- -------------         - ---------   -------
 2031 19/05/2016 18:15:11    Information    La descripción del ID de
evento...
 2030 19/05/2016 18:13:07    Information    El servicio de hora está
sincroniza...
 2029 19/05/2016 18:13:05        Warning    La resolución del nombre
sls.update...
 2028 19/05/2016 18:13:05    Information    El proveedor de tiempo
NtpClient...
 2027 19/05/2016 18:13:05    Information    El proveedor de tiempo
NtpClient está...
 2026 19/05/2016 18:13:04    Information    La descripción del ID de
evento...
[...]
```

Los resultados de los eventos devueltos por el comando **Get-EventLog** se muestran del más reciente al más antiguo.

También es posible combinar ambos parámetros, **-Before** y **-After** para definir un rango de búsqueda más restringido:

```
PS C:\Windows\system32> $dateAfter = Get-Date -Date 19/05/16 -Hour
18
-Minute 11 -Second 40
PS C:\Windows\system32> $dateBefore = Get-Date -Date 19/05/16 -Hour
```

```
18
-Minute 11 -Second 45
PS  C:\Windows\system32>  Get-EventLog  -LogName  System  -Before
$dateBefore
-After            $dateAfter            |            Select-Object
Index,TimeGenerated,EntryType,Message

Index  TimeGenerated          EntryType   Message
-----  -------------          ---------   -------
 2020   19/05/2016 18:11:47   Information    Filtro de sistema de
archivos...
 1996   19/05/2016 18:11:45   Information    El tiempo límite del
sistema es de...
 1995   19/05/2016 18:11:44   Information   Se inició el servicio de
registro...
 1994   19/05/2016 18:11:44   Information   Microsoft (R) Windows (R)
10.00...
```

FIltrado en función del nivel de criticidad

Gracias al parámetro **-EntryType**, puede filtrar los eventos en función de su nivel de criticidad. Los posibles valores para este parámetro son: `Error`, `Warning`, `Information`, `FailureAudit` y `SuccessAudit`. Por ejemplo:

```
PS  C:\Windows\system32>  Get-EventLog  -LogName  System  -EntryType
Warning
-Newest 4 | Select-Object Index,TimeGenerated,EntryType,Message

Index  TimeGenerated        EntryType  Message
-----  -------------        ---------  -------
 2029   19/05/2016 18:13:05     Warning   La resolución del nombre
sls.update.m...
 1989   16/05/2016 10:17:54     Warning   La descripción del ID de
evento...
 1985   16/05/2016 07:22:25     Warning   La resolución del nombre
settings-win...
 1948   15/05/2016 01:00:48    Warning  NtpClient no pudo establecer
un...
```

El parámetro **-Newest** seguido de una cifra limita simplemente el resultado del comando a los números de eventos más recientes. Así, **-Newest 4** muestra únicamente los cuatro resultados más recientes.

Filtrado en función del origen

También es posible filtrar en función del origen del evento. Este filtro utiliza el parámetro **-Source** seguido del nombre del origen. En el siguiente ejemplo, el origen `Microsoft-Windows-GroupPolicy` representa el conjunto de eventos relacionados con los GPO:

```
PS C:\Windows\system32> Get-EventLog -LogName System -Source
Microsoft-
Windows-GroupPolicy -Newest 4 | Select-Object
TimeGenerated,Source,Message

TimeGenerated       Source                          Message
-------------       ------                          -------
31/10/2016 15:53:47 Microsoft-Windows-GroupPolicy Los parámetros de
la...
31/10/2016 15:53:47 Microsoft-Windows-GroupPolicy Error de proceso
...
31/10/2016 14:43:15 Microsoft-Windows-GroupPolicy Los parámetros de
st...
31/10/2016 14:43:15 Microsoft-Windows-GroupPolicy Error de proceso
...
```

Filtrado en función del InstanceID

Observe la diferencia entre las propiedades `InstanceID` y `EventID`: los números son diferentes. Actualmente, el uso de `EventID` es despreciable. Es preferible utilizar el `InstanceID` para realizar búsquedas en los registros de eventos. Además, este dato está indexado, a diferencia del `EventID`.

Así, para conocer los arranques de Windows en un puesto de trabajo, con un `InstanceID` correspondiente al número 2147489657, escriba:

```
PS C:\Windows\system32> Get-EventLog -LogName System -InstanceId
2147489657 | Select-Object TimeGenerated,InstanceID,Message

TimeGenerated        InstanceID Message
-------------        ---------- -------
03/01/2017 18:11:44   2147489657 Microsoft (R) Windows (R) 10.00.
14393...
02/01/2017 16:34:55   2147489657 Microsoft (R) Windows (R) 10.00.
14393...
02/01/2017 13:46:39   2147489657 Microsoft (R) Windows (R) 10.00.
14393...
02/01/2017 13:40:28   2147489657 Microsoft (R) Windows (R) 10.00.
14393...
[...]
```

Filtrado en función del mensaje registrado en el evento

Es posible realizar otro filtrado en función del mensaje vinculado al evento. Para ello, especifique el parámetro **-Message**:

```
PS C:\Windows\system32> Get-EventLog -LogName System -EntryType
Warning
-Newest 4 -Message "*conexión*" | Select-Object
Index,TimeGenerated,EntryType,Message
```

```
Index TimeGenerated        EntryType Message
----- -------------        --------- -------
 1155 13/05/2016 15:06:28      Warning El Sistema de seguridad no
pu...
  870 08/05/2016 11:03:34     Warning NtpClient no pudo establecer
un...
  762 06/05/2016 06:38:25     Warning NtpClient no pudo establecer
un...
  752 04/05/2016 04:38:38     Warning NtpClient no pudo establecer
un...
```

El resultado del comando devuelve todos los eventos que contengan la cadena de caracteres «conexión» en su mensaje.

Filtrado en función del usuario

Por último, es posible filtrar por usuario con el parámetro **-UserName** seguido de la cuenta de usuario para la que se desea realizar la búsqueda. Así, las ocurrencias correspondientes a las sesiones del usuario quedan registradas y son fáciles de consultar:

```
PS C:\Windows\system32> Get-EventLog -LogName System -UserName
CONTOSO\Julien -Newest 4 | Select-Object
Index,TimeGenerated,EntryType,Message

Index TimeGenerated          EntryType  Message
----- -------------          ---------  -------
 1008  10/05/2016  18:46:47   Information   Los  parámetros  de  la
directiva ...
 1005  10/05/2016  17:16:47   Information   Los  parámetros  de  la
directiva ...
 1001  10/05/2016  15:46:47   Information   Los  parámetros  de  la
directiva ...
  999  10/05/2016  14:16:47   Information   Los  parámetros  de  la
directiva ...
```

2. Administrar los registros de eventos

Limit-EventLog permite administrar los registros de eventos. En medios seguros, se desea, por ejemplo, que los logs se conserven durante cierto tiempo. Es aquí donde interviene el cmdlet **Limit-EventLog**. Puede aumentar el tamaño de los registros de eventos, el período de retención, y definir qué acción realizar cuando un registro alcance su tamaño máximo.

Parámetro	Descripción
-LogName <String[]>	Especifica uno o varios registros de eventos.

Parámetro	Descripción
-MaximumSize <Int64>	Especifica el tamaño máximo de un registro de eventos (en bytes).
-OverflowAction <OverflowAction>	Especifica la acción prevista cuando un registro de eventos alcanza su tamaño máximo.
-RetentionDays <Int32>	Especifica el número mínimo de días durante los cuales un evento debe permanecer en el registro de eventos.

Ejemplo: definir el tamaño máximo de un registro de eventos

Para definir el tamaño máximo acordado a un registro de eventos, hay que utilizar el parámetro **-MaximumSize** y definir el tamaño máximo en bytes:

```
PS C:\Windows\system32> Limit-EventLog -LogName System
-MaximumSize 40MB
```

El tamaño definido mediante **-MaximumSize** debe estar comprendido entre 64 KB y 4 GB. En el caso anterior, el tamaño del registro Sistema está limitado a 40 MB.

Hay que utilizar el parámetro **-RetentionDays** para definir el período de retención. Esto garantiza que los eventos se conservarán durante un número mínimo de días. Sin embargo, para activar esta función, es preciso que la acción por defecto cuando se alcanza el tamaño máximo por el registro de eventos esté definida a `OverwriteOlder`. En caso contrario, obtendrá el siguiente mensaje:

```
PS C:\Windows\system32> Limit-EventLog -LogName System
-RetentionDays 7
Limit-EventLog: El período de retención es válido únicamente si
la acción asociada a la superación de la capacidad posee el valor
«OverwriteOlder». Edítela y vuelva a intentarlo.
```

De modo que hay que configurar la acción por defecto en caso de superar el tamaño máximo, con el parámetro **-OverflowAction**.

Ejemplo: definir un período de retención para un registro de eventos

```
PS C:\Windows\system32> Limit-EventLog -LogName System
-RetentionDays 7 -OverflowAction OverwriteOlder
```

Para definir un período mínimo de retención en un registro de eventos, es obligatorio que este posea la directiva `OverwriteOlder`. Esta directiva define la acción que se ha de emprender cuando el registro de eventos alcanza su tamaño máximo (consulte la siguiente tabla).

Para explicar el rol del parámetro **`-OverflowAction`** y de los valores que puede tomar, he aquí una tabla resumen:

Valor	Descripción
`DoNotOverwrite`	Los elementos registrados en el registro de eventos se conservan. En caso de alcanzar el tamaño máximo, los nuevos eventos no se registran.
`OverwriteAsNeeded`	Cada nueva entrada del registro de eventos borra la entrada más antigua una vez alcanzado el tamaño máximo.
`OverwriteOlder`	Las nuevas entradas en el registro de eventos borran los eventos anteriores al número de días de retención especificado. En caso de que no haya eventos más antiguos que el número de días de retención definido, se abandonan los nuevos registros.

Por último, para cambiar el valor de **`-OverflowAction`** al modo `DoNotOverwrite` o `OverwriteAsNeeded`, es muy sencillo:

```
PS C:\Windows\system32> Limit-EventLog -LogName System
-OverflowAction DoNotOverwrite
PS C:\Windows\system32> Limit-EventLog -LogName System
-OverflowAction OverwriteAsNeeded
```

Por supuesto, esta acción tiene como consecuencia la eliminación del período de retención de los eventos.

3. Borrar el contenido de un registro de eventos

`Clear-EventLog` permite eliminar el conjunto de eventos contenidos en un registro de eventos. Así, el cmdlet se utiliza especificando simplemente el nombre del registro de eventos sobre el que se desea llevar a cabo la acción.

Parámetro	Descripción
`-LogName` `<String[]>`	Especifica los registros de eventos.

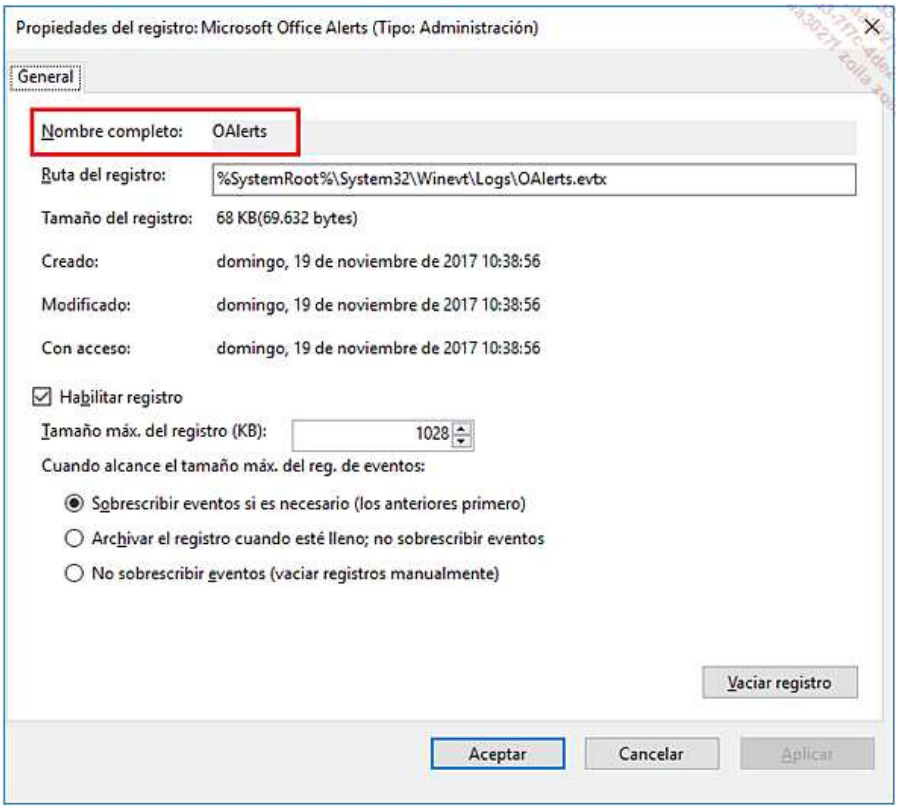
Ejemplo: borrar todos los eventos de un registro de eventos

```
C:\Windows\system32> Clear-EventLog -LogName System
```

Recuerde que la lista de nombres de los registros de eventos puede recuperarse mediante el siguiente comando:

```
PS C:\Windows\system32> Get-EventLog -LogName *
```

También es posible recuperar el nombre a través de la interfaz gráfica, en las propiedades de un registro de eventos. He aquí un ejemplo con el registro de alertas de Microsoft Office: se trata de OAlerts.



Propiedades de un registro de eventos

4. Escribir un evento en un registro de eventos

`Write-EventLog` permite escribir un evento en el registro de eventos seleccionado. De este modo, es posible incluir en un script comandos que registran eventos de acuerdo con su desarrollo, y así comprobar que todo finaliza correctamente.

He aquí un resumen de varios parámetros que pueden utilizarse con **`Write-EventLog`**:

Parámetro	Descripción

Parámetro	Descripción
-EntryType <EventLogEntryType>	Especifica el tipo de criticidad del evento, a saber: Error, Warning, Information, SuccessAudit y FailureAudit.
-EventID <Int32>	Especifica un número de identificador, que se asignará al evento.
-LogName <String>	Nombre del registro donde se escribirá el nuevo evento.
-Message <String>	Mensaje descriptivo del evento.
-Source <String>	El origen del evento. Se trata, en general, de la aplicación que escribe el nuevo evento.

Sin embargo, antes de empezar a escribir nuevos eventos, hay que crear un origen: **New-EventLog** permite hacer esto.

```
PS C:\Windows\system32> New-EventLog -LogName Application -Source
"My PowerShell events"
```

Si el registro definido mediante el parámetro **-LogName** no existe, entonces **New-EventLog** crea un nuevo registro de eventos. El origen se asociará también al registro de eventos que se haya indicado en el comando. No será posible escribir en otro registro.

A continuación, es posible escribir nuevas entradas en el registro de eventos, especificando el registro con el que se ha creado el origen:

```
PS C:\Windows\system32> Write-EventLog -LogName Application
-Source "My PowerShell events" -EntryType Information -EventId 1
-Message "¡Mi evento escrito desde Windows PowerShell!"
```

Como puede observar en el registro de eventos Application, se ha registrado una nueva entrada:

Evento escrito desde Windows PowerShell

Y como muestra la captura de pantalla, el evento tiene como origen «My PowerShell events». No es necesario volver a crear el origen con cada ejecución del script: una vez creado el origen, queda registrado en Windows, salvo si se elimina con el cmdlet **`Remove-EventLog`**.

Resulta, así, fácil escribir nuevos eventos en un registro. Esto adquiere todo el sentido con scripts que se ejecutan de manera remota o que se ejecutan periódicamente: esto le permite guardar una traza de la ejecución de sus scripts en los puestos de trabajo que administre para comprobar si se han ejecutado correctamente y sin errores.

Por último, terminaremos esta sección sobre los registros de eventos con **`Remove-EventLog`**. Este cmdlet puede o bien anular la inscripción de un origen de eventos, o bien eliminar un registro de eventos (eliminando a su vez el conjunto de orígenes que estuvieran asociados a este registro de eventos).

Ejemplo: anular la inscripción de un origen de eventos

Para anular la inscripción de un origen de eventos, basta con indicar su nombre mediante el parámetro **`-Source`**:

```
PS C:\Windows\system32> Remove-EventLog -Source "My PowerShell
events"
```

- Una vez eliminado el origen, ya no es posible escribir en el registro de eventos, aunque este sigue estando presente y puede consultarse.

Ejemplo: eliminar un registro de eventos

Por último, para eliminar un registro de eventos, hay que escribir el nombre del registro de eventos mediante el parámetro **-LogName**:

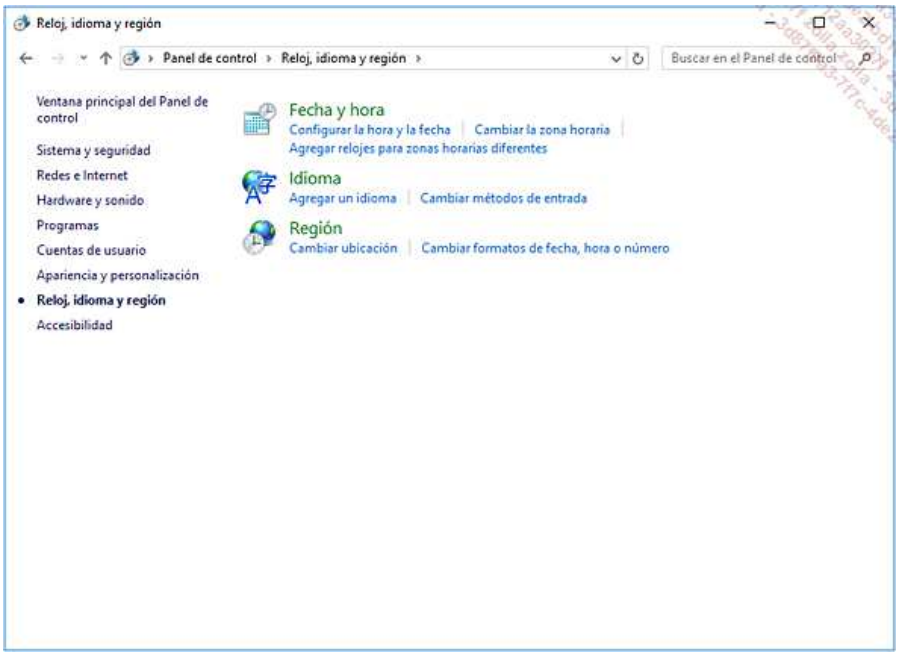
```
PS C:\Windows\system32> Remove-EventLog -LogName myAppliLog
```

Los parámetros regionales y de idioma

Es posible administrar los parámetros regionales y de idioma a partir de Windows 8. Con Windows 7, solo hay disponibles los siguientes cmdlets: **`Get-Culture`** y **`Get-UICulture`**.

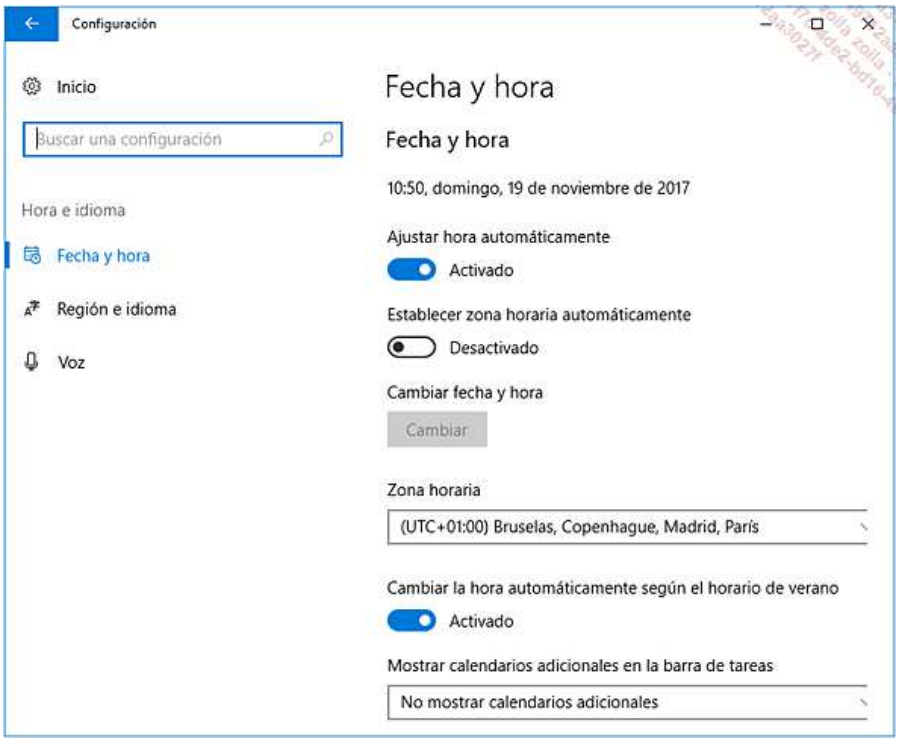
Los parámetros regionales y de idioma no afectan únicamente a la interfaz gráfica de Windows. La ubicación, el código del teclado, los formatos de representación de las fechas, etc., son solo una pequeña parte de los parámetros regionales y de idioma configurados en Windows.

Vamos a explicar el rol de cada uno de estos parámetros.



*Categoría **Reloj, idioma y región** desde el Panel de control*

La aplicación **Configuración** de Windows 10 incluye también una página dedicada a la gestión de los parámetros regionales. Puede abrirla haciendo clic en el icono **Hora e idioma**.



*Categoría **Hora e idioma** en la aplicación **Configuración** de Windows 10*

He aquí una lista de los cmdlets que estudiaremos en esta sección:

- **`Get-Culture`**: obtener los parámetros regionales.
- **`Get-UICulture`**: obtener el idioma de la interfaz gráfica de Windows.
- **`Set-Culture`**: modificar los parámetros regionales.
- **`Get-WinUserLanguageList`**: recuperar las preferencias de idioma.
- **`Set-WinUserLanguageList`**: modificar las preferencias de idioma.
- **`New-WinUserLanguageList`**: instanciar un nuevo idioma.
- **`Get-WinHomeLocation`**: obtener la ubicación configurada.
- **`Set-WinHomeLocation`**: modificar la ubicación.

1. Obtener los parámetros regionales

Get-Culture permite recuperar la información correspondiente a los parámetros regionales configurados en Windows. Esto incluye, entre otros, el idioma del dispositivo de entrada (teclado), el formato de representación de las fechas, números y divisas.

Este cmdlet se utiliza de una manera muy simple, pues no es preciso indicar ningún parámetro:

```
PS C:\Windows\system32> Get-Culture

LCID             Name             DisplayName
----             ----             -----------
1036             es-ES            Español (España)
```

Para obtener una lista más detallada de las propiedades de cultura actualmente definidas, escriba:

```
PS C:\Windows\system32> Get-Culture | Format-List -Property *


Parent                         : es
LCID                           : 3082
KeyboardLayoutId               : 3082
Name                           : es-ES
IetfLanguageTag                : es-ES
DisplayName                    : Español (España)
NativeName                     : español (España)
EnglishName                    : Spanish (España)
TwoLetterISOLanguageName       : es
ThreeLetterISOLanguageName     : spa
ThreeLetterWindowsLanguageName: ESN
CompareInfo                    : CompareInfo - es-ES
TextInfo                       : TextInfo - es-ES
IsNeutralCulture               : False
CultureTypes                                  :  SpecificCultures,
InstalledWin32Cultures,
                                 FrameworkCultures
NumberFormat                                                      :
System.Globalization.NumberFormatInfo
DateTimeFormat                                                    :
System.Globalization.DateTimeFormatInfo
Calendar                                                          :
System.Globalization.GregorianCalendar
OptionalCalendars                                                 :
{System.Globalization.GregorianCalendar}
UseUserOverride                : True
IsReadOnly                     : False
```

En el ejemplo anterior, los parámetros regionales del puesto de trabajo están configurados en es-ES (se trata de un identificador de cultura), que se corresponde literalmente a Español (España). Un identificador de cultura está compuesto como mínimo de un identificador de idioma y un identificador de región. Las propiedades que se han de consultar son `Name` y `DisplayName`.

2. Obtener el idioma de la interfaz gráfica

Get-UICulture devuelve el idioma de la interfaz de usuario de Windows. Desde Windows Vista, y según la edición de Windows, es posible instalar paquetes de idioma. Así, un sistema Windows puede tener una interfaz de usuario en idioma inglés para un perfil de usuario determinado y, para otro, una interfaz en idioma español.

Este cmdlet devuelve como resultado el idioma utilizado por el perfil de usuario de la cuenta en curso durante la ejecución del comando, y no hace falta especificar ningún parámetro.

Ejemplo 1: para una interfaz gráfica en idioma inglés

```
PS C:\Windows\system32> Get-UICulture

LCID             Name             DisplayName
----             ----             -----------
1033             en-US            English (United States)
```

Ejemplo 2: para una interfaz gráfica en idioma español

```
PS C:\Windows\system32> Get-UICulture

LCID             Name             DisplayName
----             ----             -----------
1036             es-ES            Español (España)
```

3. Modificar los parámetros regionales

El cmdlet **Set-Culture** permite definir un nuevo juego de parámetros regionales para la sesión de usuario en curso. Esto tiene como resultado modificar los siguientes elementos:

- Fecha y hora (formato de representación de la fecha y de la hora, definición del primer día de la semana).
- Números (formato de representación de los números y sistema de unidad de medida).
- Símbolo monetario (divisa monetaria del país).

A partir de Windows 8, región e idioma se han separado. La parte de la región gestiona únicamente los puntos descritos anteriormente, mientras que la parte del idioma se ocupa del idioma de la interfaz gráfica, así como del idioma del dispositivo de entrada (teclado).

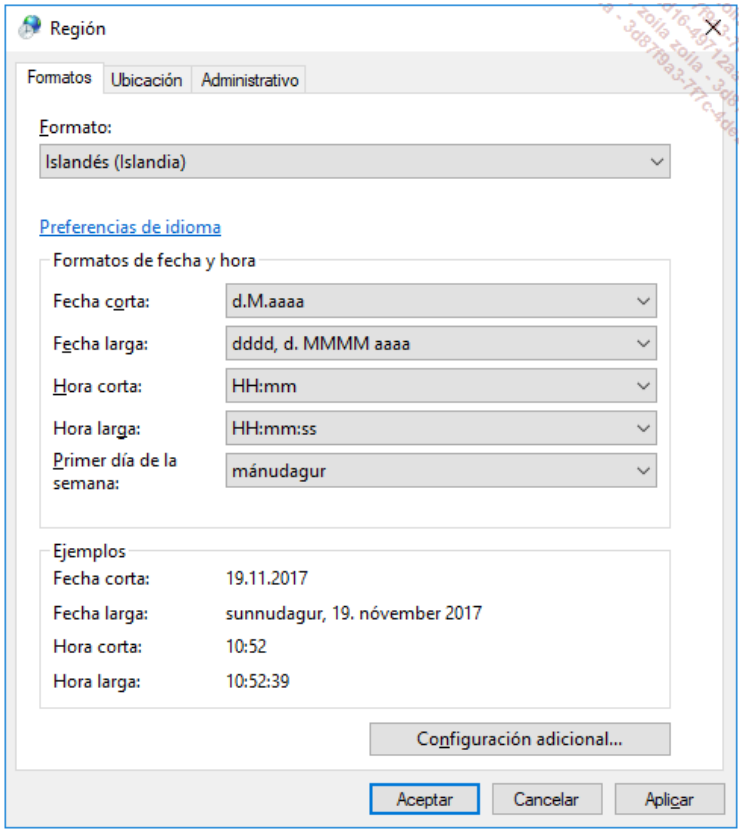
El cmdlet se ejecuta de una manera muy sencilla, y los cambios tienen efecto inmediatamente. Basta con conocer el identificador de cultura (`Culture Info`) asociado al juego de parámetros regionales que se quiere implementar.

Parámetro	Descripción
`-CultureInfo` `<CultureInfo>`	Define el nuevo juego de parámetros regionales.

Ejemplo: cambiar el juego de parámetros regionales en el puesto de trabajo

```
PS C:\Windows\system32> Set-Culture -CultureInfo is-IS
```

Así, el comando anterior define el juego de parámetros regionales (formato) en islandés.



*En la pestaña **Formatos** del cuadro de diálogo **Región**, el formato se define a **Islandés (Islandia)**.*

He aquí una lista no exhaustiva de los formatos disponibles y su identificador de cultura asociado.

Nombre del formato	Identificador de cultura
Búlgaro (Bulgaria)	bg-BG
Croata (Croacia)	hr-HR
Checo (República Checa)	cs-CZ
Danés (Dinamarca)	da-DK
Holandés (Países Bajos)	nl-NL
Inglés (Canadá)	en-CA
Inglés (Estados Unidos)	en-US
Inglés (Gran Bretaña)	en-GB
Estonio (Estonia)	et-EE
Finés (Finlandia)	fi-FI
Francés (Bélgica)	fr-BE
Francés (Canadá)	fr-CA
Francés (Francia)	fr-FR
Francés (Suiza)	fr-CH
Alemán (Alemania)	de-DE
Alemán (Suiza)	de-CH
Griego (Grecia)	el-GR
Húngaro (Hungría)	hu-HU
Islandés (Islandia)	is-IS
Italiano (Italia)	it-IT
Japonés (Japón)	ja-JP
Coreano (Corea)	ko-KR
Letón (Letonia)	lv-LV
Lituano (Lituania)	lt-LT
Noruego (Noruega)	nb-NO
Polonés (Polonia)	pl-PL
Portugués (Brasil)	pt-BR
Portugués (Portugal)	pt-PT
Rumano (Rumanía)	ro-RO
Ruso (Rusia)	ru-RU
Eslovaco (Eslovaquia)	sk-SK

Nombre del formato	Identificador de cultura
Esloveno (Eslovenia)	sl-SI
Español (España)	es-ES
Sueco (Suecia)	sv-SE

4. Idioma

Las preferencias de idioma en Windows juegan varios roles importantes en todo el entorno de Windows y en la interfaz con el usuario. En efecto, las preferencias de idioma incluyen la información correspondiente al método de entrada (código del teclado), el corrector ortográfico, el texto predictivo y la escritura a mano (con una pluma).

a. Obtener las preferencias de idioma

`Get-WinUserLanguageList` permite recuperar las preferencias de idioma configuradas en la cuenta del usuario en curso de Windows. Cada idioma presente en la lista de preferencias de idioma posee la distinta información descrita en el párrafo anterior. El cmdlet se utiliza sin ningún parámetro.

```
PS C:\Windows\system32> Get-WinUserLanguageList


LanguageTag     : es-ES
Autonym         : español (España)
EnglishName     : Spanish
LocalizedName   : Español (España)
ScriptName      : Script latin
InputMethodTips: {040C:0000040C}
Spellchecking   : True
Handwriting     : False
```

b. Modificar las preferencias de idioma

`Set-WinUserLanguageList` permite cambiar los parámetros lingüísticos por defecto en la sesión del usuario donde se ejecuta el comando.

Parámetro	Descripción
`-LanguageList` `<List<WinUserLanguage>>`	Define las preferencias de idioma.

Si se hubieran configurado varios idiomas en la lista de preferencias de idioma, todos se pierden y se reemplazan por

los definidos mediante **Set-WinUserLanguageList**.

Ejemplo 1: definir un idioma en la lista de preferencias de idioma

```
PS C:\Windows\system32> Set-WinUserLanguageList es-ES

Confirmar
¿Desea continuar esta operación?
[S]  Sí    [N]  No    [U]  Suspender    [?]  Ayuda  (el  valor
predeterminado
es 'S'): s
PS C:\Windows\system32> Get-WinUserLanguageList


LanguageTag     : es-ES
Autonym         : español (España)
EnglishName     : Spanish
LocalizedName   : Español (España)
ScriptName      : Script latin
InputMethodTips: {040C:0000040C}
Spellchecking   : True
Handwriting     : False
```

Puede consultar la tabla presente en la sección Modificar los parámetros regionales para conocer el identificador de cultura que se puede utilizar.

Ejemplo 2: definir varios idiomas en la lista de preferencias de idioma

Es posible definir varios idiomas en la lista de preferencias de idioma durante la ejecución del comando.

```
PS C:\Windows\system32> Set-WinUserLanguageList es-ES, is-IS

Confirmar
¿Desea continuar esta operación?
[S] Sí  [N] No  [U] Suspender  [?] Ayuda (el valor predeterminado
es
'S'): s
PS C:\Windows\system32> Get-WinUserLanguageList


LanguageTag     : es-ES
Autonym         : español (España)
EnglishName     : Spanish
LocalizedName   : Español (España)
ScriptName      : Script latin
InputMethodTips: {040C:0000040C}
Spellchecking   : True
Handwriting     : False

LanguageTag     : is
Autonym         : íslenska
```

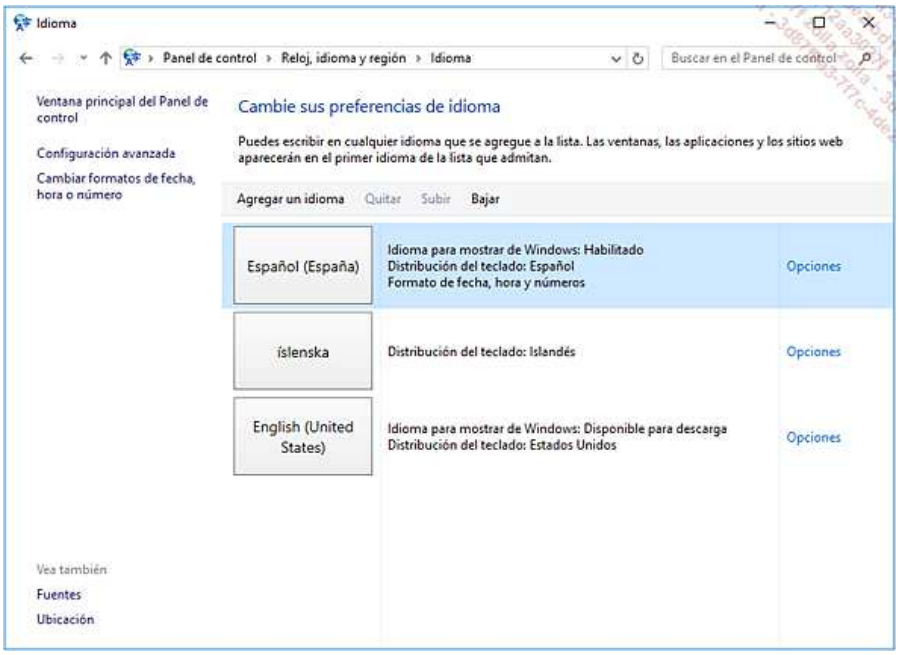
```
EnglishName    : Icelandic
LocalizedName  : Islandés
ScriptName     : Script latin
InputMethodTips: {040F:0000040F}
Spellchecking  : True
Handwriting    : False
```

Ejemplo 3: agregar un idioma a la lista de preferencias de idioma ya existentes

Por último, es posible agregar un idioma suplementario en la lista de preferencias de idioma del usuario en curso de la siguiente manera:

```
PS C:\Windows\system32> $userLL = Get-WinUserLanguageList
PS C:\Windows\system32> $userLL.Add("en-US")
PS C:\Windows\system32> Set-WinUserLanguageList $userLL -Force
```

En esta ocasión, se configuran los tres idiomas y están listos para su uso en la lista de preferencias de idioma de la sesión del usuario.



Lista de idiomas configurados en la lista de preferencias de idioma

Cuando existen varios idiomas en la lista de preferencias de idioma, puede utilizar la combinación de teclas [Alt][Shift] para cambiar de un idioma a otro. Aparece un indicador junto al reloj de la barra de tareas de Windows que le indicará con qué idioma está trabajando.

Sin embargo, puede que no desee agregar un idioma suplementario, sino únicamente cambiar el idioma del método de entrada por el teclado. Puede, por ejemplo, utilizar un sistema operativo Windows en inglés con las preferencias de idioma en inglés, pero con un teclado español (QWERTY).

Las siguientes líneas de comandos toman el idioma configurado en primer lugar en la lista de preferencias de idioma y modifican el método de entrada (teclado) por un teclado español:

```
PS C:\Windows\system32> $userLL = Get-WinUserLanguageList
PS C:\Windows\system32> $userLL[0].InputMethodTips.Clear()
PS                                                  C:\Windows\system32>
$userLL[0].InputMethodTips.Add("0c0a:0000040a")
PS C:\Windows\system32> Set-WinUserLanguageList $userLL -Force
```

La siguiente tabla le muestra el conjunto de códigos de teclado disponibles.

Código de idioma de teclado	Teclado correspondiente
00000402	Búlgaro
0000041a	Croata
00000405	Checo
00000406	Danés
00000413	Holandés (Estándar)
00000813	Holandés (Belga)
00000409	Inglés (Americano)
00000809	Inglés (Reino Unido)
00001009	Inglés (Canadiense)
00001409	Inglés (Nueva Zelanda)
00001809	Inglés (Irlandés)
00000c09	Inglés (Australiano)
0000040b	Finlandés
0000040c	Francés (Estándar)
0000080c	Francés (Belga)
0000100c	Francés (Suizo)
00000c0c	Francés (Canadiense)
00000407	Alemán (Estándar)
00000807	Alemán (Suizo)

Código de idioma de teclado	Teclado correspondiente
00000c07	Alemán (Austríaco)
00000408	Griego
0000040e	Húngaro
0000040f	Islandés
00000410	Italiano (Estándar)
00000810	Italiano (Suizo)
00000414	Noruego (Bokmål)
00000814	Noruego (Nynorsk)
00000415	Polonés
00000816	Portugués (Estándar)
00000416	Portugués (Brasileño)
00000418	Rumano
00000419	Ruso
0000041b	Eslovaco
00000424	Esloveno
0000080a	Español (Mexicano)
0000040a	Español (Tradicional)
00000c0a	Español (Moderno)
0000041d	Sueco
0000041f	Turco

c. Instanciar un nuevo idioma

`New-WinUserLanguageList` permite instanciar un nuevo idioma. Una vez instanciado el objeto desde una plantilla (mediante el identificador de cultura), tiene la posibilidad de modificar los parámetros en función de la configuración deseada. Una vez hecho esto, habrá que aplicarlo con **`Set-WinUserLanguageList`** para agregarlo en la lista de preferencias de idioma del puesto de trabajo.

Parámetro	Descripción
`-Language` `<String>`	Instancia un nuevo idioma.

Ejemplo: instancia dos idiomas y los especifica en la lista de preferencias de idioma

```
PS C:\Windows\system32> $userLL = New-WinUserLanguageList es-ES
PS C:\Windows\system32> $userLL.Add("pt-BR")
PS C:\Windows\system32> Set-WinUserLanguageList $userLL
```

5. Ubicación

a. Obtener la ubicación

`Get-WinHomeLocation` permite obtener la ubicación del puesto de trabajo configurada en Windows. Este parámetro pueden recuperarlo ciertas aplicaciones para aplicar las preferencias de configuración automáticamente (por ejemplo, aplicaciones meteorológicas o periodísticas). El cmdlet se utiliza sin parámetros.

```
PS C:\Users\Admin> Get-WinHomeLocation

GeoId HomeLocation
----- ------------
  217 España
```

b. Configurar la ubicación

Para definir una nueva ubicación del puesto de trabajo, se utiliza el cmdlet **`Set-WinHomeLocation`**. Sin embargo, hay que conocer el identificador GeoID del país correspondiente a la nueva ubicación que desea registrar en Windows.

Parámetro	Descripción
`-GeoId` `<GeoID>`	Especifica la nueva ubicación que se configurará.

Para conocer el código GeoID que debe utilizar, he aquí una lista no exhaustiva de algunos países y su identificador asociado:

Identificador de ubicación geográfica (hexadecimal)	Identificador de ubicación geográfica (decimal)	País
0x23	35	Bulgaria
0x6C	108	Croacia
0x4B	75	República Checa

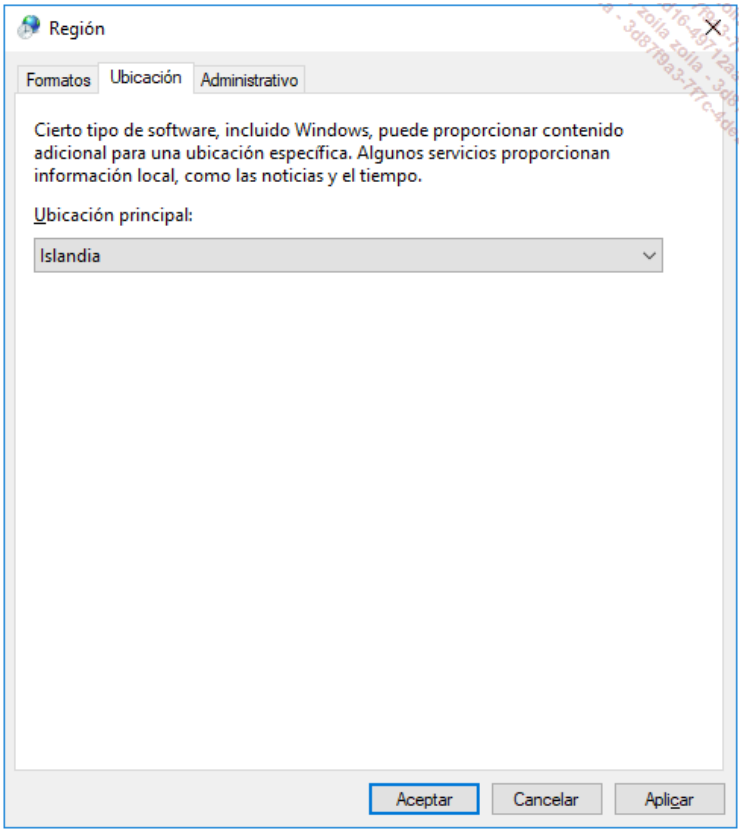
Identificador de ubicación geográfica (hexadecimal)	Identificador de ubicación geográfica (decimal)	País
0x3D	61	Dinamarca
0xB0	176	Países Bajos
0xF4	244	Estados Unidos
0xF2	242	Reino Unido
0x46	70	Estonia
0x4D	77	Finlandia
0x54	84	Francia
0xDF	223	Suiza
0x5E	94	Alemania
0x62	98	Grecia
0x6D	109	Hungría
0x6E	110	Islandia
0x76	118	Italia
0x7A	122	Japón
0x86	134	Corea
0x8C	140	Letonia
0x8D	141	Lituania
0xB1	177	Noruega
0xBF	191	Polonia
0x20	32	Brasil
0xC1	193	Portugal
0xC8	200	Rumanía
0xCB	203	Rusia
0x8F	143	Eslovaquia
0xD4	212	Eslovenia
0xD9	217	España
0xDD	221	Suecia

Puede consultar el conjunto de códigos GeoID en la siguiente dirección: http://msdn.microsoft.com.sabidi.urv.cat/en-us/library/dd374073.aspx

Ejemplo: cambiar la ubicación del sistema

```
PS C:\Windows\system32> Set-WinHomeLocation -GeoId 0x6E
```

El parámetro **`-GeoId`** admite el código hexadecimal o decimal.



*La nueva ubicación se ha definido a **Islandia***

Achivo y compresión de archivos

Una de las novedades aportadas por la versión 5 de Windows PowerShell es la posibilidad de comprimir datos en archivos .zip directamente por línea de comandos. Para ello, hay disponibles dos cmdlets: **`Compress-Archive`** y **`Expand-Archive`**.

La compresión de archivos resulta imprescindible cuando se quiere compartir, enviar o publicar datos a través de la red. La segunda ventaja es, por supuesto, una mejora en el espacio ocupado y, por consiguiente, un ahorro en el ancho de banda de red.

He aquí los cmdlets detallados en esta sección y su rol:

- **`Compress-Archive`**: permite comprimir uno o varios archivos y crear un archivo comprimido (archivo .zip).
- **`Expand-Archive`**: permite descomprimir un archivo comprimido.

1. Compresión

Los diversos parámetros que ofrece el cmdlet **`Compress-Archive`** permiten realizar diversas acciones. La primera es crear un archivo comprimido (.zip) que puede contener uno o varios archivos que se comprimirán o no. La segunda afecta a la actualización de los archivos contenidos en un archivo comprimido; esta última acción ofrece la posibilidad, por ejemplo, de completar un archivo existente, agregando los nuevos archivos.

Cabe destacar, sin embargo, que este cmdlet presenta una limitación: no es posible crear archivos comprimidos de más de 2 GB. Se trata de una limitación a nivel de la API System.IO.Compression.ZipArchive, a la que invoca el cmdlet **`Compress-Archive`**.

He aquí los parámetros disponibles para utilizar este cmdlet:

Parámetro	Descripción

Parámetro	Descripción
-CompressionLevel <String>	Especifica el nivel de compresión que se aplicará durante la creación de un archivo comprimido. Hay tres valores posibles: Fastest, NoCompression, Optimal. Por defecto, si no se especifica este parámetro, el nivel aplicado es Optimal.
-DestinationPath <String>	Indica la ruta de acceso del archivo comprimido que se creará.
-Force	Autoriza el borrado de un archivo comprimido existente.
-Path <String[]>	Indica la ruta o las rutas de acceso a los archivos que se agregarán al archivo comprimido. Está permitido el uso del carácter comodín * (*wildcard*).
-Update	Indica que el archivo comprimido existente se actualizará. Los archivos existentes en el archivo comprimido se reemplazarán por los nuevos si tuvieran el mismo nombre.

He aquí el detalle de los distintos niveles de compresión que pueden utilizarse con el parámetro **-CompressionLevel**:

Nivel de compresión	Descripción
Fastest	Permite un procesamiento rápido de la operación. Tenga precaución, no obstante, pues puede producir un tamaño mayor del archivo comprimido.

Nivel de compresión	Descripción
`NoCompression`	No se realiza ninguna compresión durante la adición de archivos en el archivo comprimido.
`Optimal`	Tasa de compresión optimizada. El tiempo de procesamiento depende del tamaño del archivo que se va a comprimir.

Ejemplo: crear un archivo comprimido con tres archivos

```
PS D:\Proyecto365> Compress-Archive -Path
.\Procedimiento.docx,.\Esquema.vsdx,D:\Archives\Presentacion.pptx

-DestinationPath D:\Temp\DocProyecto.zip
```

Se creará un archivo comprimido llamado DocProyecto.zip en la carpeta especificada, que contendrá los tres archivos indicados. Como no se ha especificado el nivel de compresión, este se define automáticamente a `Optimal`.

Observe a su vez que, durante la fase de compresión, se muestra una barra de progreso en la parte superior de la consola Windows PowerShell: esto le permite obtener una estimación de la duración total de la creación del archivo comprimido.

```
Compress-Archive
            Se    está    creando    el    archivo    comprimido
D:\Temp\DocProyecto.zip...

[ooooooooooooooooooooooooooooooooooooooooooooooooooo
  ]
```

Ejemplo: crear un archivo comprimido con todos los elementos de una carpeta

```
PS D:\Proyecto365> Compress-Archive -Path .\* -DestinationPath
D:\Temp\DocProyecto.zip -Force
```

En este ejemplo, el conjunto de elementos (archivos y carpeta) presentes en la carpeta D:\Proyecto365 se comprime en un archivo comprimido.

El parámetro **`-Force`** permite borrar el archivo DocProyecto.zip si este ya existiera.

Ejemplo: actualizar un archivo comprimido existente

```
PS D:\Proyecto365> Compress-Archive -Path D:\Proyecto365\Tabla.xlsx
-Update -DestinationPath D:\Temp\DocProyecto.zip
```

El archivo Tabla.xlsx se ha agregado al archivo comprimido, que ya contenía otros elementos. En caso de que el archivo Tabla.xlsx ya existiese en el archivo comprimido, este se reemplazaría por la nueva versión.

A su vez, si se especifica el parámetro **-Update** mientras que el destino indicado por el parámetro **-DestinationPath** no existe, entonces se creará un archivo comprimido.

2. Descompresión

La operación inversa, es decir, la descompresión, permite recuperar uno o varios elementos contenidos en un archivo comprimido. Esta operación puede llevarse a cabo con el cmdlet **Expand-Archive**:

Parámetro	Descripción
-DestinationPath <String>	Indica la ruta de acceso de una carpeta donde se descomprimirán los elementos contenidos en el archivo comprimido.
-Force	Autoriza al borrado de los archivos presentes en el destino en caso de conflicto.
-Path <String>	Indica el archivo comprimido que se descomprimirá.

Ejemplo: descomprimir un archivo comprimido

```
PS D:\Proyecto365> Expand-Archive -Path D:\Temp\DocProyecto.zip
-DestinationPath D:\NuevoProyecto
```

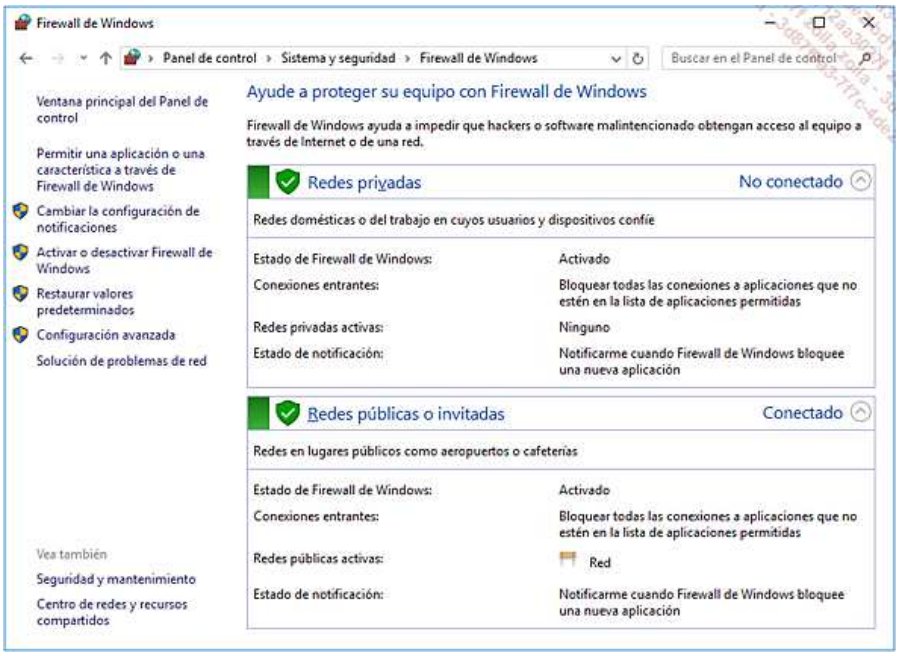
Todo el contenido del archivo comprimido se descomprime en la carpeta D:\NuevoProyecto. No es necesario crear previamente la carpeta de destino antes de ejecutar el comando: esta se creará automáticamente.

En caso de conflicto entre un archivo descomprimido y un archivo con el mismo nombre existente en el destino, se producirá un

error. Para sobrescribirlo, debe utilizarse el parámetro **-Force**, que provocará el borrado de los archivos existentes por los descomprimidos.

La seguridad y el Firewall de Windows

Microsoft ha integrado un firewall con el Service Pack 2 de Windows XP. Si bien era más que bienvenido, rápidamente resultó poco fiable, pues era poco configurable. Desde Windows Vista, el firewall integrado de manera nativa se ha repensado por completo y ahora es posible definir de una forma mucho más específica los parámetros de protección. Veremos cmdlets que nos permitirán interactuar con el firewall.



Página de configuración del Firewall de Windows

He aquí la lista de los cmdlets estudiados:

- **`Get-NetFirewallProfile`**: recuperar la configuración del firewall (perfil).
- **`Set-NetFirewallProfile`**: modificar la configuración del firewall (perfil).
- **`New-NetFirewallRule`**: crear una nueva regla.
- **`Set-NetFirewallRule`**: modificar una regla existente.

1. Activar el Firewall de Windows

El Firewall de Windows está, por supuesto, habilitado por defecto tras una instalación no personalizada de Windows. Sin embargo, según la política de seguridad y las soluciones implementadas en la empresa, es posible desactivar el Firewall de Windows. En caso de tener que reactivarlo, he aquí cómo hacerlo mediante cmdlets de PowerShell.

La activación se lleva a cabo en dos etapas: la primera consiste en asegurarse de que el servicio de Firewall de Windows se ejecuta correctamente. Para ello, utilice **Get-Service**:

```
PS C:\Windows\system32> If ((Get-Service -Name MpsSvc).Status -eq
"Stopped")
>> {
>>     Set-Service -Name MpsSvc -StartupType Automatic
>>     Start-Service -Name MpsSvc
>> }
```

La segunda etapa consiste en activar el Firewall para los distintos perfiles deseados. Para ello, utilice **Set-NetFirewallProfile**. Este cmdlet admite varios parámetros, pero veremos los correspondientes a la activación y desactivación.

Parámetro	Descripción
-AllowInboundRules <GpoBoolean>	Especifica al firewall que bloquee las conexiones entrantes.
-Enabled <GpoBoolean>	Activa o desactiva el Firewall de Windows con seguridad avanzada para el perfil especificado.
-LogAllowed	Indica si los paquetes autorizados deben escribirse en los logs o no.
-LogBlocked	Indica si los paquetes bloqueados deben escribirse en los logs o no.
-LogFileName <String>	Indica la ruta de acceso al archivo para escribir los logs.

Parámetro	Descripción
-LogMaxSizeKilobytes	Especifica el tamaño máximo del archivo de log (en kilobytes).
-Name <String[]>	Especifica uno o varios nombres de perfil de firewall para modificar.
-NotifyOnListen <GpoBoolean>	Autoriza la notificación.

Ejemplo: activación del Firewall de Windows para todos los perfiles

```
PS C:\Windows\system32> Set-NetFirewallProfile -Name * -
Enabled True
```

Existen tres perfiles (tipos de red) en el Firewall de Windows:

- **Public**: este perfil se utiliza en los lugares públicos, como cafés, hoteles, etc. Bloquea por defecto el conjunto de conexiones y la detección de redes.
- **Private**: este perfil se utiliza generalmente cuando se trabaja en una red local privada, como por ejemplo en casa.
- **Domain**: si el puesto de trabajo está asociado a un dominio de Windows Server, entonces el perfil Domain está activo automáticamente. Este perfil se utiliza cuando el equipo está conectado a la red de una empresa.

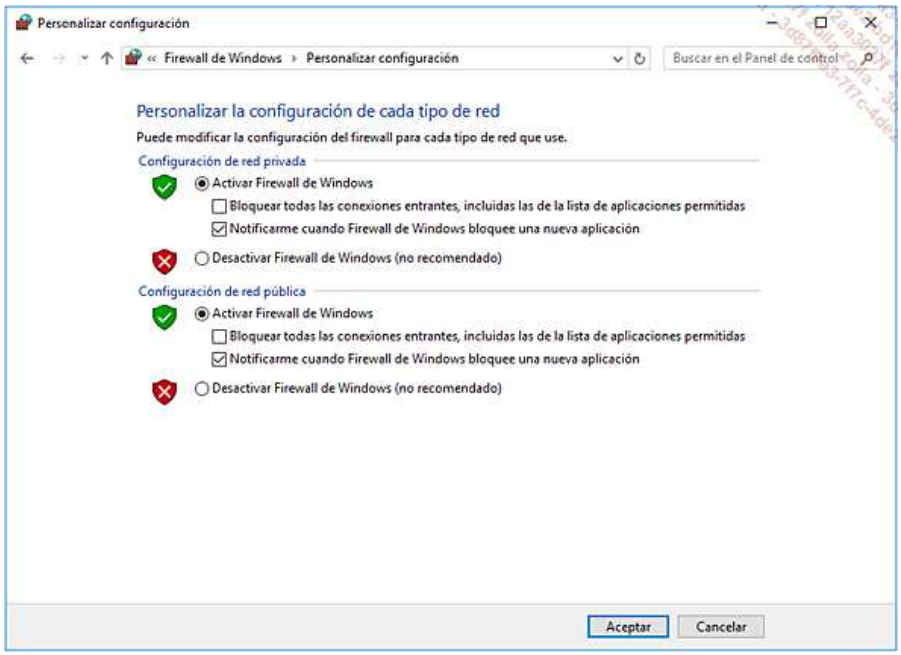
En cada perfil es posible, a su vez, jugar con un parámetro de seguridad en las conexiones entrantes, y un parámetro que gestiona las notificaciones.

El primer parámetro, **-AllowInboundRules**, permite definir qué comportamiento se desea en caso de producirse una conexión entrante. Puede tomar los siguientes valores:

Valor	Descripción
True	Bloquea todas las conexiones entrantes salvo aquellas que estén autorizadas mediante una regla de seguridad específica.
False	Bloquea absolutamente todas las conexiones entrantes.

El segundo parámetro, **-NotifyOnListen**, permite que se le notifique si el Firewall de Windows bloquea una nueva aplicación. Puede tomar los siguientes valores:

Valor	Descripción
`True`	Se muestra una notificación por pantalla en caso de que el Firewall de Windows bloquee una conexión y se le pide decidir si autoriza o no las futuras conexiones.
`False`	No se muestra ningún mensaje si el Firewall de Windows bloquea la conexión de alguna nueva aplicación.



Personalización de los parámetros del Firewall de Windows

Ejemplo: activación del firewall para un perfil con los parámetros

En el siguiente ejemplo, **`Set-NetFirewallProfile`** activa el firewall para el perfil Public, rechaza todas las conexiones entrantes y no se muestra ningún mensaje de advertencia cuando el firewall bloquea la conexión de alguna nueva aplicación.

```
PS C:\Windows\system32> Set-NetFirewallProfile -Name Public
-Enabled True -AllowInboundRules False -NotifyOnListen False
```

2. Desactivar el Firewall de Windows

Para desactivar el Firewall de Windows, basta simplemente con detener el servicio Firewall de Windows, cuyo efecto provoca que se permitan todas las conexiones de red entrantes y salientes del puesto de trabajo:

```
PS C:\Windows\system32> Stop-Service -Name MpsSvc
PS C:\Windows\system32> Set-Service -Name MpsSvc -StartupType
Disabled
```

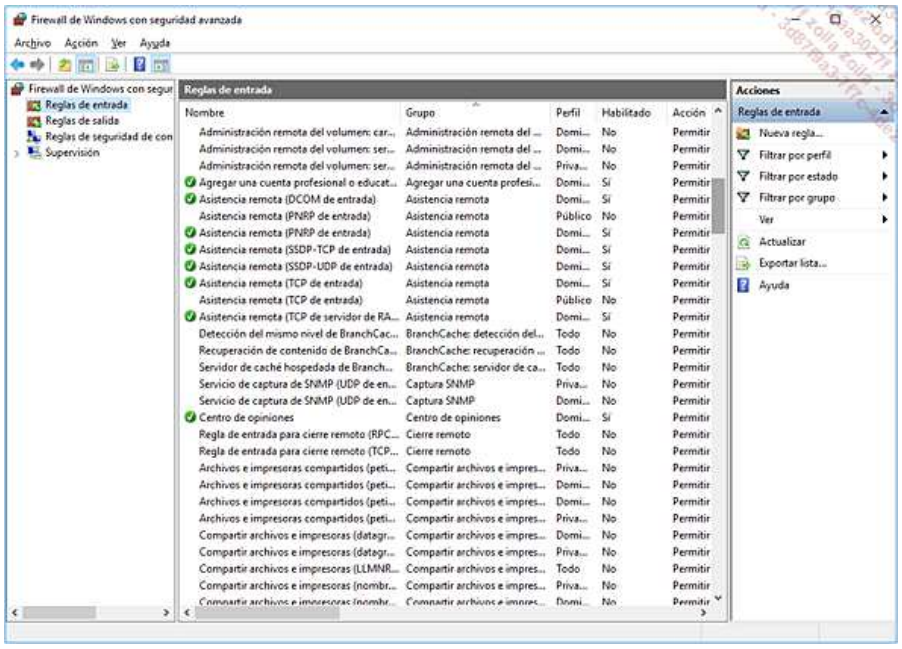
Si simplemente desea desactivar el firewall para un perfil en particular, puede utilizar el cmdlet **`Set-NetFirewallProfile`**:

```
PS C:\Windows\system32> Set-NetFirewallProfile -Name
Public,Private -Enabled False
```

El ejemplo anterior desactiva el Firewall de Windows para los perfiles de conexión Public y Private.

3. Crear una nueva regla de seguridad

Pero la configuración del Firewall de Windows no se limita solamente a activarlo o desactivarlo según los perfiles de red. El modo avanzado del Firewall de Windows permite definir reglas de seguridad en función de las conexiones de red (protocolos, puertos) y también conexiones específicas de algún servicio o aplicación.



Firewall de Windows con seguridad avanzada

Para ello, se utilizan dos cmdlets:

- **`New-NetFirewallRule`**: para crear una nueva regla.

- **`Set-NetFirewallRule`**: para modificar una regla existente.

En primer lugar, he aquí un resumen de los parámetros disponibles para **`New-NetFirewallRule`**:

Parámetro	Descripción
`-Action` `<Action>`	Especifica la acción que debe aplicar el firewall para esta regla. Los posibles valores son: `Allow`, `Block`.
`-Description` `<String>`	Descripción de la regla de firewall.
`-Direction` `<Direction>`	Indica la dirección del tráfico que debe aplicarse para esta regla.
`-DisplayName` `<String>`	Nombre para mostrar de la regla creada.
`-LocalPort` `<String[]>`	Indica uno o varios números de puertos lógicos.
`-Profile` `<Profile>`	Especifica uno o varios nombres de perfil de firewall para modificar.
`-Program` `<String>`	Especifica la ruta de acceso al archivo ejecutable para el que la regla autoriza el tráfico de red.
`-Protocol` `<String>`	Indica el protocolo de comunicación. Posibles valores: `TCP`, `UDP`, etc.

Construcción de una regla para una aplicación

En este ejemplo, se configura una nueva regla de firewall. Su función es bloquear todas las conexiones salientes del programa Internet Explorer. Los valores que se asignan a los parámetros son:

- **`-Program`**: "C:\Program Files\Internet Explorer\iexplore.exe".
- **`-Action`**: Block.

- **-Profile**: Domain, Private.
- **-DisplayName**: "Block IE".
- **-Description**: "Demonstration".
- **-Direction**: Outbound.

En Windows PowerShell, el comando completo es:

```
PS      C:\Windows\system32>      New-NetFirewallRule      -Program
"C:\Program Files\
Internet      Explorer\iexplore.exe"      -Action      Block      -
Profile Domain,Private
-DisplayName  "Block  IE"  -Description  "Demonstration"  -Direction
Outbound


Name                  : {e0afebdb-0b64-4718-8e06-82aeff989e54}
DisplayName           : Block IE
Description           : Demonstration
DisplayGroup          :
Group                 :
Enabled               : True
Profile               : Domain, Private
Platform              : {}
Direction             : Outbound
Action                : Block
EdgeTraversalPolicy   : Block
LooseSourceMapping    : False
LocalOnlyMapping      : False
Owner                 :
PrimaryStatus         : OK
Status                  : Se analizó la regla correctamente desde el
almacén. (65536)
EnforcementStatus     : NotApplicable
PolicyStoreSource     : PersistentStore
PolicyStoreSourceType: Local
```

Construcción de una regla para un puerto

Este ejemplo es parecido al anterior, pero esta vez el bloqueo se realiza sobre un puerto (80), para el protocolo TCP. Para ello, los dos parámetros necesarios para llevar a cabo esta acción se configuran de la siguiente manera:

- **-LocalPort**: 80.
- **-Protocol**: TCP.

He aquí el comando completo con el conjunto de parámetros que permiten crear la regla de firewall:

```
PS C:\Windows\system32> New-NetFirewallRule -LocalPort 80 -Protocol
TCP
-Action Block -Profile Domain,Private -DisplayName "Block 80 TCP"
```

```
-Description "Block http protocol" -Direction Outbound


Name                  : {6ed44aae-a478-4f6a-a2a5-dded0ead305c}
DisplayName           : Block 80 TCP
Description           : Block http protocol
DisplayGroup          :
Group                 :
Enabled               : True
Profile               : Domain, Private
Platform              : {}
Direction             : Outbound
Action                : Block
EdgeTraversalPolicy   : Block
LooseSourceMapping    : False
LocalOnlyMapping      : False
Owner                 :
PrimaryStatus         : OK
Status                  : Se analizó la regla correctamente desde el
almacén. (65536)
EnforcementStatus     : NotApplicable
PolicyStoreSource     : PersistentStore
PolicyStoreSourceType: Local
```

4. Administrar los logs

La gestión de logs es una parte importante dentro de las tareas de todo administrador de sistemas. Permiten realizar un seguimiento de los eventos para la supervisión del sistema y obtener información muy valiosa cuando se produce algún comportamiento erróneo o algún problema técnico.

Por defecto, el Firewall de Windows utiliza un único archivo de log para los tres perfiles: Domain, Private y Public. Se encuentra en la siguiente ubicación: %systemroot%\system32\LogFiles\Firewall\pfirewall.log.

Con PowerShell, como con la interfaz de usuario, puede modificar el conjunto de parámetros de los archivos de log; por ejemplo, separar los archivos de log en función de los perfiles de red, aumentar el tamaño máximo de los archivos de log o editar diversos parámetros de registro.

Veremos en primer lugar, para un perfil concreto, el estado de configuración de los logs. Para ello, basta con utilizar el cmdlet **`Get-NetFirewall Profile`** seguido del nombre del perfil.

Parámetro	Descripción
-Name `<String[]>`	Especifica uno o varios nombres de perfil de firewall para los que se recupera la configuración.

La propiedad `LogFileName` indica la ruta de acceso al archivo de

```
PS C:\Windows\system32> Get-NetFirewallProfile -Name Private


Name                            : Private
Enabled                         : True
DefaultInboundAction            : NotConfigured
DefaultOutboundAction           : NotConfigured
AllowInboundRules               : NotConfigured
AllowLocalFirewallRules         : NotConfigured
AllowLocalIPsecRules            : NotConfigured
AllowUserApps                   : NotConfigured
AllowUserPorts                  : NotConfigured
AllowUnicastResponseToMulticast: NotConfigured
NotifyOnListen                  : True
EnableStealthModeForIPsec       : NotConfigured
LogFileName                     : %systemroot%\system32\LogFiles\
                                   Firewall\pfirewall.log
LogMaxSizeKilobytes             : 4096
LogAllowed                      : False
LogBlocked                      : False
LogIgnored                      : NotConfigured
DisabledInterfaceAliases        : {NotConfigured}
```

log. En el ejemplo anterior, se trata de: %systemroot%\system32\LogFiles\Firewall\ pfirewall.log, que es el archivo de log por defecto del Firewall de Windows.

Para cambiar el archivo de log de un perfil en particular, hay que indicar el parámetro **-LogFileName** de **Set-NetFirewallProfile** y pasarle como valor la ruta de acceso al nuevo archivo de log:

```
PS C:\Windows\system32> Set-NetFirewallProfile -Name Private
-LogFileName %systemdrive%\FWLOG\Private.log
```

Puede definir un archivo de log para cada perfil (tipo de red) del Firewall de Windows.

Para modificar el tamaño máximo del archivo de log, siempre para un perfil concreto, se utiliza el parámetro **-LogMaxSizeKilobytes**, pasándole como valor el tamaño el kilobytes:

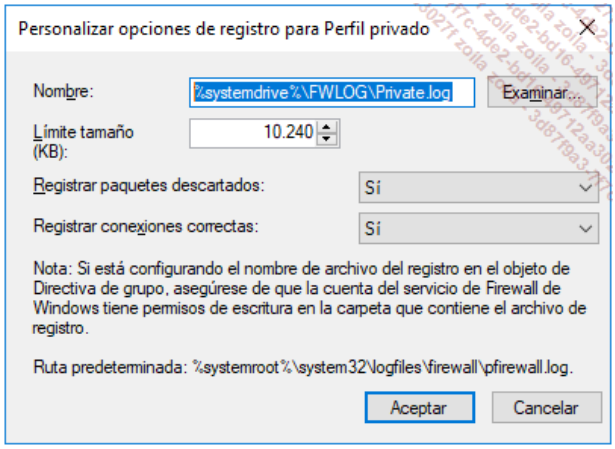
```
PS C:\Windows\system32> Set-NetFirewallProfile -Name Private
-LogMaxSizeKilobytes 10240
```

Por último, para que las conexiones y los paquetes autorizados (**-LogAllowed**) y rechazados (**-LogBlocked**) se registren en el archivo de log, hay que configurar estos parámetros a `True`:

Al final, los parámetros de registro para el perfil privado del

```
PS C:\Windows\system32> Set-NetFirewallProfile -Name Private
-LogAllowed True
PS C:\Windows\system32> Set-NetFirewallProfile -Name Private
-LogBlocked True
```

Firewall de Windows están personalizados, tal y como muestra la siguiente captura de pantalla:



Personalización de los parámetros de registro para el perfil privado

5. Cambiar de perfil de red

Desde Windows Vista, y con el objetivo de reforzar la seguridad de los puestos de trabajo, Microsoft ha implementado la noción de perfil de red. Son tres los perfiles que determinan la manera en la que se comportará el puesto de trabajo a la búsqueda de redes, la compartición de archivos e impresoras, las conexiones del grupo en el hogar, y también con el Firewall de Windows.

Así, cuando conecte un equipo a una nueva red, aparecerá una notificación que le preguntará si desea autorizar a los demás PC y dispositivos de la red a detectar su PC. En función de la respuesta ofrecida, Windows atribuirá uno de los perfiles a la red.

He aquí los cmdlets que permiten manipular los tipos de perfil en las conexiones activas:

- **`Get-NetConnectionProfile`**: muestra el conjunto de conexiones activas en el puesto de trabajo con el tipo de perfil de red asociado.
- **`Set-NetConnectionProfile`**: permite cambiar el tipo de perfil de red en una o varias conexiones activas.

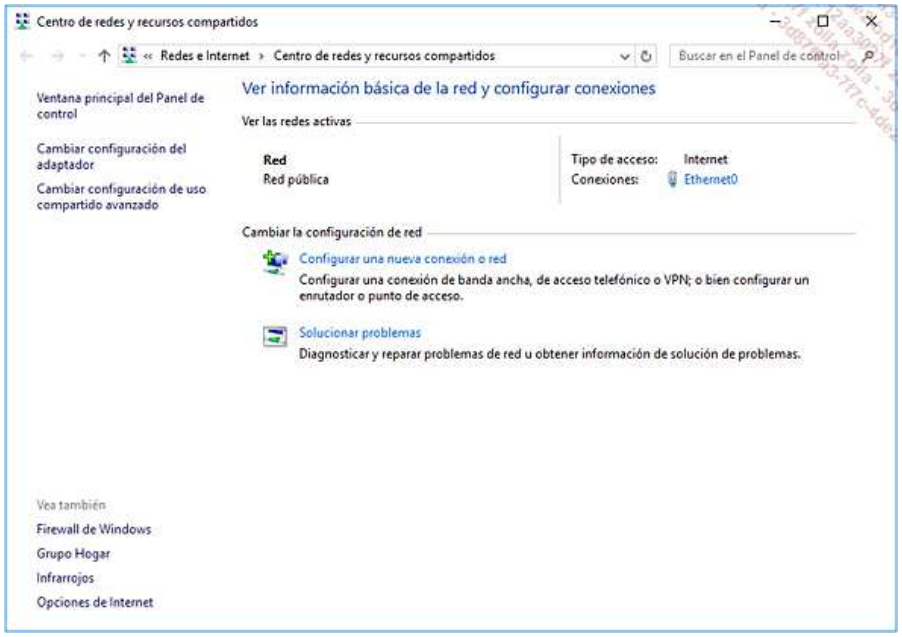
He aquí una explicación de cada uno de los perfiles (configuración por defecto):

Perfil	Descripción
Red pública `Public`	El descubrimiento de red y la compartición de archivos e impresoras están deshabilitados. Así, las carpetas compartidas no están expuestas y el puesto de trabajo no está visible en la red. El firewall bloquea también todos los accesos provenientes de los demás ordenadores. Lugares relacionados: cafés, bibliotecas, aeropuertos, etc.
Red privada `Private`	El descubrimiento de red y la compartición de archivos están habilitados por defecto. Las autorizaciones necesarias a nivel del firewall también están concedidas. Es posible, de este modo, descubrir los demás equipos presentes en la red. Lugares relacionados: casa, oficina.
Dominio `DomainAuthenticated`	Por defecto, es una configuración similar a la de la Red privada. El puesto de trabajo forma parte de un dominio Active Directory y está conectado a su red de empresa. Las reglas de firewall y demás configuraciones son efectivas en el puesto de trabajo. Estas últimas las controla el administrador del sistema.

Si bien es posible reconfigurar completamente cada uno de los perfiles, es necesario poseer los conocimientos necesarios.

Para conocer el perfil activo de la red a la que está conectado actualmente el puesto de trabajo, abra el **Centro de redes y**

recursos compartidos de Windows. En la zona **Ver las redes activas**, se muestra el perfil de red debajo de cada una de las conexiones activas.



Lista de redes activas y perfil asociado desde el ***Centro de redes y recursos compartidos***

El conjunto de conexiones de red a las que se encuentra conectado el puesto de trabajo se catalogan en la siguiente clave del registro: HKLM\SOFTWARE\ Microsoft\WindowsNT\CurrentVersion\NetworkList\Profiles. Cada subclave, en formato GUID, representa una red conocida y registrada en el puesto de trabajo. Así, cuando el equipo se conecta a una red, se comprueba si ya la conoce. En caso negativo, aparece una notificación que le pregunta si el equipo debe ser visible en la red o no.

a. Obtener la lista de conexiones activas y el tipo de perfil asociado

El cmdlet **`Get-NetConnectionProfile`** permite recuperar el conjunto de conexiones activas (propiedad `Name`) en el puesto de trabajo, así como el tipo de perfil asociado (propiedad `NetworkCategory`) a cada una.

Parámetro	Descripción

Parámetro	Descripción
-InterfaceAlias <String[]>	Especifica el alias de la interfaz o las interfaces de red.
-InterfaceIndex <UInt32[]>	Especifica el número de índice de la interfaz o las interfaces de red.
-Name <String[]>	Especifica el nombre de la conexión o las conexiones activas. El nombre contiene, generalmente, un prefijo DNS propio de la conexión.
-NetworkCategory <NetworkCategory[]>	Indica el perfil o los perfiles de red activos para las conexiones actualmente establecidas. Los posibles valores son: Public, Private, DomainAuthenticated.

Ejemplo: mostrar el conjunto de conexiones activas y su perfil asociado

```
PS C:\WINDOWS\system32> Get-NetConnectionProfile


Name             : contoso.local
InterfaceAlias   : Ethernet 2
InterfaceIndex   : 10
NetworkCategory  : DomainAuthenticated
IPv4Connectivity: NoTraffic
IPv6Connectivity: NoTraffic

Name             : fabrikam.com
InterfaceAlias   : Ethernet
InterfaceIndex   : 6
NetworkCategory  : Public
IPv4Connectivity: Internet
IPv6Connectivity: NoTraffic
```

Ejemplo: mostrar únicamente los perfiles de red Public

```
PS C:\WINDOWS\system32> Get-NetConnectionProfile -NetworkCategory
Public


Name             : fabrikam.com
InterfaceAlias   : Ethernet
InterfaceIndex   : 6
NetworkCategory  : Public
```

```
IPv4Connectivity: Internet
IPv6Connectivity: NoTraffic
```

b. Cambiar el tipo de perfil de la conexión activa

Es muy fácil cambiar el tipo de perfil de red de una conexión activa con el cmdlet **`Set-NetConnectionProfile`**. Basta con indicar la conexión sobre la que se realizará la modificación y el tipo de perfil que se configurará para la conexión. He aquí los parámetros:

Parámetro	Descripción
`-InterfaceAlias` `<String[]>`	Especifica el alias de la interfaz o las interfaces de red.
`-InterfaceIndex` `<UInt32[]>`	Especifica el número de índice de la interfaz o las interfaces de red.
`-Name``<String[]>`	Especifica el nombre de la conexión o las conexiones activas. El nombre contiene, generalmente, un prefijo DNS propio de la conexión.
`-NetworkCategory` `<NetworkCategory>`	Indica el perfil o los perfiles de red activos de las conexiones establecidas actualmente. Los posibles valores son: `Public`, `Private`.

Ejemplo: cambiar el tipo de perfil de red a red privada

```
PS C:\WINDOWS\system32> Set-NetConnectionProfile -Name
fabrikam.com
-NetworkCategory Private
```

El tipo de perfil para la conexión activa fabrikam.com pasa a red privada (`Private`).

Como sin duda habrá observado, no es posible utilizar el comando **`Set-ConnectionProfile`** para definir el tipo de red `DomainAuthenticated`. En efecto, este tipo de perfil se define automáticamente en la conexión correspondiente cuando se realiza una autenticación sobre una red empresarial.

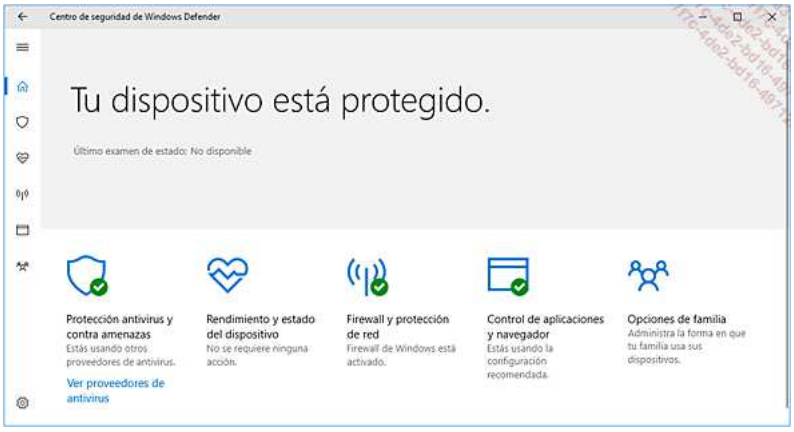
Windows Defender

Windows integra nativamente una protección contra los virus (antivirus), contra el software espía (antispyware) y contra el software malicioso (antimalware): Windows Defender. Activo por defecto una vez instalado el sistema operativo, conviene, en ciertas situaciones, administrarlo. En las empresas que poseen licencias de alguna otra aplicación de protección para los puestos de trabajo o los servidores, conviene desactivar Windows Defender para que no interfiera con una segunda aplicación del mismo tipo.

La administración de Windows Defender se realiza con los siguientes cmdlets, disponibles a partir de Windows 8.1:

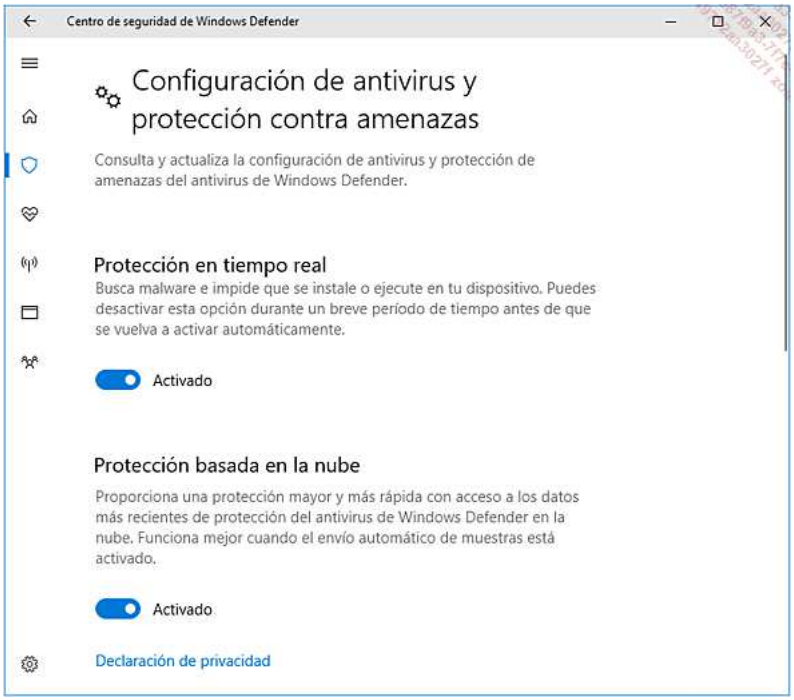
- **`Add-MpPreference`**: permite agregar exclusiones de archivos y carpeta, de tipos de archivo y de procesos.
- **`Get-MpComputerStatus`**: muestra el conjunto de información (estado, definiciones, análisis, etc.) contenida en Windows Defender.
- **`Get-MpPreference`**: muestra el conjunto de parámetros de Windows Defender.
- **`Get-MpThreat`**: devuelve el histórico de software malicioso detectado en el puesto de trabajo.
- **`Get-MpThreatCatalog`**: devuelve el catálogo con las definiciones del software malicioso que puede detectarse con Windows Defender.
- **`Get-MpThreatDetection`**: recupera todas las detecciones (en forma de eventos) de software malicioso durante el análisis de Windows Defender.
- **`Remove-MpPreference`**: elimina las exclusiones o reinicia las acciones por defecto.
- **`Remove-MpThreat`**: permite eliminar todas las amenazas activas detectadas con Windows Defender.
- **`Set-MpPreference`**: permite modificar los parámetros de Windows Defender.
- **`Start-MpScan`**: permite ejecutar análisis de tipo rápido, completo o personalizado.
- **`Start-MpWDOScan`**: permite ejecutar Windows Defender Offline.
- **`Update-MpSignature`**: actualiza las definiciones del antimalware y del antivirus. Se requiere una conexión a Internet en función del origen de las actualizaciones.

En Windows 10, el acceso a Windows Defender se realiza desde la aplicación **Configuración**, sección **Actualización y seguridad**: seleccione la opción **Windows Defender** y haga clic en el botón **Abrir el Centro de seguridad de Windows Defender**.



Windows Defender

Desde esta página puede configurar las distintas opciones de la aplicación y, así, consultar las versiones de las definiciones del antivirus y del antimalware.



Configuración de Windows Defender

1. Recuperar la información de Windows Defender

El cmdlet **Get-MpComputerStatus** devuelve un conjunto detallado de la información de Windows Defender. Así, se obtiene la información correspondiente a la versión del componente (versión del motor y también definiciones de virus, etc.), el estado de las características (activas o no) y diversas informaciones sobre los horarios (fecha y hora de última actualización, de inicio y fin del último escaneo, etc.).

Get-MpComputerStatus se utiliza de una manera muy sencilla, sin necesidad de especificar ningún parámetro. He aquí un ejemplo del resultado devuelto por el cmdlet:

```
PS C:\WINDOWS\system32> Get-MpComputerStatus


AMEngineVersion                 : 1.1.13202.0
AMProductVersion                : 4.10.14393.0
AMServiceEnabled                : True
AMServiceVersion                : 4.10.14393.0
AntispywareEnabled              : True
AntispywareSignatureAge         : 23
AntispywareSignatureLastUpdated: 16/11/2016 09:20:50
AntispywareSignatureVersion     : 1.231.2087.0
AntivirusEnabled                : True
AntivirusSignatureAge           : 23
AntivirusSignatureLastUpdated   : 16/11/2016 09:20:51
AntivirusSignatureVersion       : 1.231.2087.0
BehaviorMonitorEnabled          : True
ComputerID                                : 47337C95-D959-4294-8458-
62C9221328EB
ComputerState                   : 0
FullScanAge                     : 4294967295
FullScanEndTime                 :
FullScanStartTime               :
IoavProtectionEnabled           : True
LastFullScanSource              : 0
LastQuickScanSource             : 2
NISEnabled                      : True
NISEngineVersion                : 2.1.12706.0
NISSignatureAge                 : 0
NISSignatureLastUpdated         : 09/12/2016 19:42:00
NISSignatureVersion             : 116.65.0.0
OnAccessProtectionEnabled       : True
QuickScanAge                    : 1
QuickScanEndTime                : 08/12/2016 19:04:03
QuickScanStartTime              : 08/12/2016 19:02:08
RealTimeProtectionEnabled       : True
RealTimeScanDirection           : 0
PSComputerName                  :
```

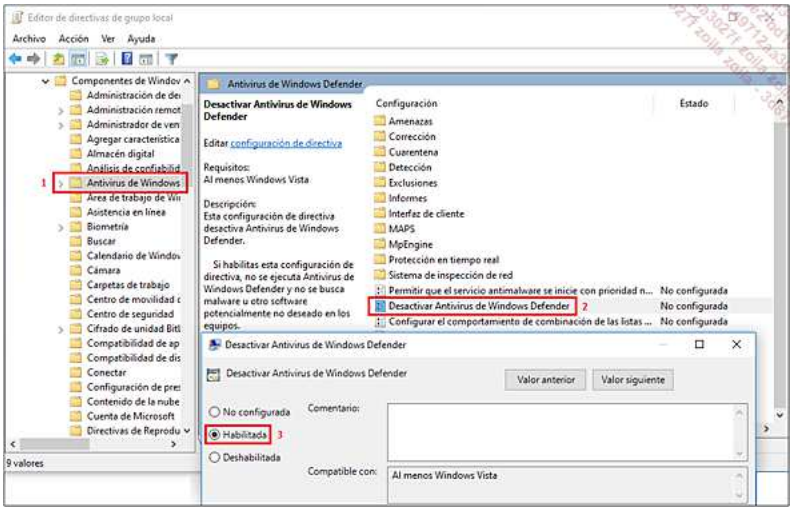
He aquí un resumen de la información más importante, clasificada por categoría.

- Información relativa a la protección contra el software malicioso (Windows Defender):
  - **AMProductVersion**: versión del cliente antimalware.

- **AMServiceEnabled**: indica el estado del servicio del cliente antimalware.

- Información relativa a la protección contra el software espía:
  - **AntispywareEnabled**: indica si el antispyware está activo o no.
  - **AntispywareSignatureAge**: indica, en número de días, el tiempo de la versión de la definición del antispyware.
  - **AntispywareSignatureLastUpdated**: indica la fecha de la última actualización de la definición del antispyware.
  - **AntispywareSignatureVersion**: versión de la definición del antispyware.
- Información relativa a la protección contra los virus:
  - **AntivirusEnabled**: indica si el antivirus está activo o no.
  - **AntivirusSignatureAge**: indica, en número de días, el tiempo de la versión de la definición del antivirus.
  - **AntivirusLastUpdated**: indica la fecha de la última actualización de la definición del antivirus.
  - **AntivirusSignatureVersion**: versión de la definición del antivirus.
- Información relativa a los análisis:
  - **FullScanAge**: indica el tiempo (en días) desde el último análisis completo.
  - **FullScanEndTime**: fecha y hora de fin del último análisis completo. Vacío si este tipo de análisis no se ha realizado nunca.
  - **FullScanStartTime**: fecha y hora de inicio del último análisis completo. Vacío si este tipo de análisis no se ha realizado nunca.
  - **QuickScanAge**: indica el tiempo (en días) desde el último análisis rápido.
  - **QuickScanEndTime**: fecha y hora de fin del último análisis rápido. Vacío si este tipo de análisis no se ha realizado nunca.
  - **QuickScanStartTime**: fecha y hora de inicio del último análisis rápido. Vacío si este tipo de análisis no se ha realizado nunca.
  - **RealTimeProtectionEnabled**: indica si la protección en tiempo real está activa o no.

A título informativo, el servicio de Windows Defender no puede detenerse desde la consola Servicios de Windows. Para detener Windows Defender, hay que utilizar un objeto de directiva de grupo (GPO) o local. Este parámetro se configura en **Configuración del equipo** - **Plantillas administrativas** - **Componentes de Windows** - **Antivirus de Windows Defender**. Abra el parámetro **Desactivar Antivirus de Windows Defender** y haga clic en la opción **Habilitada**. Valide la modificación haciendo clic en **Aceptar**: el servicio de Windows Defender se detendrá, y no será posible volver a ejecutarlo mientras la GPO esté activa.



Desactivar Windows Defender desde el Editor de directivas de grupo

Si Windows Defender está desactivado, el cmdlet **`Get-MpComputerStatus`** devuelve un mensaje de error indicando que esta operación no puede ejecutarse.

Finalmente, un último aspecto importante, relativo a las versiones de las definiciones del antimalware y del antivirus: se actualizan más o menos una vez al día. De modo que puede, gracias al resultado devuelto por **`Get-MpComputerStatus`**, comprobar rápidamente si las definiciones están actualizadas o no. También puede conocer el número de versión de las definiciones desplegadas por Microsoft desde el sitio Malware Protection Center en la siguiente página: https://www.microsoft.com/en-us/security/portal/definitions/adl.aspx

2. Actualizar las definiciones del antimalware y del antivirus

El cmdlet **`Update-MpSignature`** permite actualizar las definiciones del antimalware y del antivirus. Esto permite garantizar que se posee la última versión de las definiciones y

asegurarse de disponer de un buen nivel de protección contra este software malicioso.

Parámetro	Descripción
-UpdateSource <UpdateSource>	Indica el origen para la actualización de las definiciones. Los posibles valores son InternalDefinitionUpdateServer, MicrosoftUpdate Server, MMPC, FileShares. Por defecto, cuando no se especifica este parámetro, el cmdlet descargará las actualizaciones de los servidores de Microsoft Update Server y Microsoft Malware Protection Center (MMPC).

Ejemplo: actualizar las definiciones

```
PS C:\Windows\system32> Update-MpSignature
```

Al no especificar el parámetro **-UpdateSource**, el cmdlet utilizará los orígenes por defecto para recuperar las últimas actualizaciones de definiciones del antimalware y del antivirus.

Ejemplo: actualizar las definiciones especificando el origen de las actualizaciones

```
PS C:\Windows\system32> Update-MpSignature -UpdateSource
InternalDefinitionUpdateServer
```

El valor InternalDefinitionUpdateServer que se pasa al parámetro **-UpdateSource** indica que el cmdlet contactará con un servidor Windows Server Update Services (WSUS) para comprobar si hay actualizaciones disponibles.

3. Administrar los análisis de Windows Defender

La gestión de los análisis de Windows Defender se realiza mediante el cmdlet **Start-MpScan**. Este cmdlet permite lanzar análisis, ya sean de tipo rápido, completo o personalizado:

- Rápido: comprueba únicamente las ubicaciones del equipo más susceptibles de contener software malicioso.
- Completo: escanea el conjunto de archivos presentes en el equipo (incluyendo los medios extraíbles). Este tipo de análisis es mucho más largo.
- Personalizado: permite seleccionar la unidad o las unidades y carpetas que se van a analizar. Se analizarán todos los archivos incluidos en la selección.

El cmdlet **Start-MpScan** permite lanzar análisis en función de las opciones descritas a continuación. He aquí un resumen de los parámetros:

Parámetro	Descripción
-ScanPath <String>	Designa una ruta de acceso a un archivo o carpeta que se analizará. El formato UNC también es válido. Este parámetro debe utilizarse en combinación con un análisis de tipo personalizado (CustomScan).
-ScanType <ScanType>	Especifica el tipo de escaneo que se ejecutará. Los posibles valores son QuickScan, FullScan, CustomScan.

Por defecto, Windows Defender eliminará los programas maliciosos que encuentre durante el análisis. Si se encuentra un software potencialmente peligroso, se nos advertirá con una notificación en la que le pedirá que compruebe y tome las acciones necesarias. Abriendo Windows Defender, podrá escoger entre hacer clic en **Limpiar PC**, que eliminará las amenazas encontradas, o bien hacer clic en **Mostrar detalles**, que abre un cuadro de diálogo donde tendrá la posibilidad de escoger entre las distintas acciones que pueden emprenderse para cada una de las amenazas detectadas: su eliminación (opción **Eliminar**), su puesta en cuarentena (opción **Cuarentena**), o incluso la autorización, por ejemplo en el caso de que se haya detectado un falso positivo (opción **Autorizar**).

También es posible gestionar las amenazas a través de Windows PowerShell, mediante los cmdlets **Get-MpThreat** y **Remove-MpThreat**.

Ejemplo: lanzar un análisis rápido

```
PS C:\Windows\system32> Start-MpScan
```

Cuando no se indica ningún parámetro, se ejecuta un análisis rápido. Verá aparecer, en la parte superior de la consola de Windows PowerShell, el avance de la operación:

```
Start-MpScan
   0/1 terminado

[
       ]
  Windows Defender analiza su PC
        Esto puede tardar unos minutos según el tipo de examen
seleccionado.

[ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
     ]

     Análisis rápido
```

Ejemplo: lanzar un análisis sobre una carpeta concreta

```
PS C:\Windows\system32> Start-MpScan -ScanType CustomScan -ScanPath
D:\Temp
```

Con este comando, se analizarán todas las carpetas y archivos presentes en la carpeta D:\Temp.

Ejemplo: lanzar un análisis sobre varios elementos concretos

El parámetro **-ScanPath** no permite especificar varias carpetas. Sin embargo, siempre puede lanzar varias veces el comando **Start-MpScan** utilizando un bucle `Foreach` que le pase los distintos elementos para analizar. He aquí un ejemplo:

```
PS C:\Windows\system32> $path =
@('D:\Temp',$env:USERPROFILE,'D:\ShareFolder\autoGeneratedFile.exe'
)
PS C:\Windows\system32> $path | Foreach-Object {
>> If (Test-Path $_) {Start-MpScan -ScanType CustomScan -ScanPath
$_}
>> }
```

Windows Defender Offline

Windows Defender Offline (WDO) permite lanzar un análisis antes del inicio de Windows. En efecto, algunos programas maliciosos son difíciles de eliminar cuando Windows está iniciado. Para resolver este problema, Windows Defender Offline permite lanzar un análisis antes de cargar Windows para detectar mejor y eliminar las amenazas. Esta operación requiere un reinicio del equipo, y el análisis dura unos 15 minutos.

WDO también puede lanzarse con el cmdlet **Start-MpWDOScan**. Este se utiliza fácilmente, sin especificar ningún parámetro.

Ejemplo: lanzar la ejecución de WDO

```
PS C:\Windows\system32> Start-MpWDOScan
```

Una vez ejecutado el comando, el equipo reinicia y lanza Windows Defender Offline antes de iniciar Windows. Se realiza un análisis de tipo rápido para detectar y eliminar el software malicioso.

4. Administrar las amenazas detectadas

Es posible utilizar tres cmdlets dedicados a la gestión de las amenazas detectadas: **Get-MpThreatDetection**, **Get-MpThreat** y **Remove-MpThreat**. Cada cmdlet aporta su conjunto de información y la posibilidad de lanzar una limpieza sobre los elementos detectados como potencialmente peligrosos.

Por último, **Get-MpThreatCatalog** permite mostrar el catálogo completo de amenazas que es posible detectar con Windows

Defender. Este catálogo se actualiza regularmente con cada descarga de una nueva versión de las definiciones del antispyware y del antivirus.

a. Recuperar los eventos ligados a la detección de amenazas

El primer cmdlet, **`Get-MpThreatDetection`**, muestra el conjunto de elementos potencialmente peligrosos detectados por Windows Defender. Puede tratarse de amenazas todavía activas o no.

Parámetro	Descripción
`-ThreatID` `<Int64[]>`	Designa el ID de la amenaza o las amenazas que devolverá el comando.

Ejemplo: recuperar el conjunto de amenazas detectadas

```
PS C:\Windows\system32> Get-MpThreatDetection


ActionSuccess                 : True
AdditionalActionsBitMask      : 0
AMProductVersion              : 4.10.14393.0
CleaningActionID              : 1
CurrentThreatExecutionStatusID: 0
DetectionID                             : {A8CFE49B-F7F8-4935-8E66-
1D6A435E131E}
DetectionSourceTypeID         : 2
DomainUser                    : AUTORIDAD NT\SERVICE LOCAL
InitialDetectionTime          : 27/12/2016 15:59:48
LastThreatStatusChangeTime    : 27/12/2016 16:00:06
ProcessName                   : Unknown
RemediationTime               : 27/12/2016 16:00:06
Resources                     :
{containerfile:_D:\Temp\Win32.Polip.a\Win32.Polip.a_infected.exe,

file:_D:\Temp\Win32.Polip.a\Win32.Polip.a_infected.exe->
[PCleanHelper]}
ThreatID                      : 2147577513
ThreatStatusErrorCode         : 0
ThreatStatusID                : 2
PSComputerName                :
```

Así, cada detección genera un evento como el que acabamos de ver. Para interpretarlos mejor, he aquí una descripción de la información principal:

Propiedad	Descripción
`ActionSuccess`	Indica si la acción de limpieza ha finalizado con éxito (`True`) o con fallo (`False`).

Propiedad	Descripción
`RemediationTime`	Indica la fecha y la hora de la acción emprendida para la resolución. Si no se indica ninguna fecha de resolución, se espera una acción manual por parte del usuario.
`ThreatID`	Número de identificación del software malintencionado.
`ThreatStatusID`	Indica cuál es el estado de la amenaza. He aquí los posibles valores con su identificador asociado: • 0: desconocido • 1: detectado • 2: limpiado • 3: cuarentena • 4: eliminado • 5: autorizado • 6: bloqueado • 10x: acción errónea

En el ejemplo anterior, podemos interpretar el evento así: la acción de limpieza de la amenaza 2147577513 ha sido correcta, la resolución se ha realizado el 27/12/2016 a 16 h.

b. Recuperar el histórico de amenazas detectadas

El segundo cmdlet, **`Get-MpThreat`**, permite mostrar el conjunto de amenazas detectadas por Windows Defender. Se obtiene el histórico completo, independientemente de si se trata de amenazas activas o no.

Parámetro	Descripción
`-ThreatID` `<Int64[]>`	Designa el ID de la amenaza o las amenazas que devolverá el comando.

Ejemplo: recuperar el histórico de una amenaza detectada mediante su identificador

Reutilizando el número de identificación de la amenaza devuelta por **`Get-MpThreatDetection`**, es posible obtener el histórico vinculado a dicha amenaza.

```
PS C:\Windows\system32> Get-MpThreat -ThreatID 2147577513

CategoryID      : 42
```

```
DidThreatExecute: False
IsActive        : True
Resources       :
{containerfile:_D:\Temp\Win32.Polip.a\Win32.Polip.a_infected.exe,

file:_D:\Temp\Win32.Polip.a\Win32.Polip.a_infected.exe->
[PCleanHelper]}
RollupStatus    : 1
SchemaVersion   : 1.0.0.0
SeverityID      : 5
ThreatID        : 2147577513
ThreatName      : Virus:Win32/Polip.A
TypeID          : 0
PSComputerName  :
```

He aquí un resumen de los elementos más importantes:

Propiedad	Descripción
`DidThreatExecute`	Indica si la amenaza se ha ejecutado (`True`) o no (`False`).
`IsActive`	Indica si la amenaza está activa (`True`) o no (`False`).
`SeverityID`	Indica el nivel de gravedad de la amenaza. He aquí la lista: • 0: desconocido • 1: bajo • 2: moderado • 4: elevado • 5: grave
`ThreatID`	Número de identificación del software malicioso.
`ThreatName`	Nombre del software malicioso.

La información más importante es `IsActive`. Si vale `True`, significa que el software malicioso está activo todavía y que es preciso emprender las acciones necesarias para retirar esta amenaza.

Para recuperar todo el histórico de amenazas detectadas en el puesto de trabajo, ejecute el cmdlet **`Get-MpThreat`** sin ningún parámetro.

5. Limpiar o eliminar las amenazas detectadas

Por último, para lanzar una limpieza de las amenazas que se han detectado en algún análisis previo de Windows Defender, se utiliza el cmdlet **`Remove-MpThreat`**.

Este cmdlet se utiliza sin ningún parámetro y se encarga de eliminar todas las amenazas activas.

Ejemplo: eliminar las amenazas activas detectadas

```
PS C:\Windows\system32> Remove-MpThreat
```

Una vez ejecutado el comando, Windows Defender ejecuta una limpieza eliminando o limpiando el software malicioso encontrado. La operación puede llevar varios minutos antes de que la consola de Windows PowerShell devuelva el control.

Compruebe, una vez terminada la ejecución del comando, si todavía existen amenazas activas mediante el cmdlet **Get-MpThreat**:

```
PS C:\Windows\system32> Get-MpThreat | ? {$_.IsActive}
```

6. Consultar el catálogo de amenazas

Cada actualización de las definiciones del antispyware y del antivirus actualiza el catálogo que contiene todas las referencias a las amenazas conocidas que Windows Defender es capaz de detectar en sus análisis.

Es posible consultar este catálogo mediante el cmdlet **Get-MpThreatCatalog**.

Parámetro	Descripción
-ThreatID <Int64[]>	Designa el ID de la amenaza o las amenazas que devolverá el comando.

Ejemplo: mostrar todo el catálogo

```
PS C:\Windows\system32> Get-MpThreatCatalog


CategoryID     : 39
SeverityID     : 5
ThreatID       : 1596
ThreatName     : TrojanSpy:Win32/AcidBattery
TypeID         : 0
PSComputerName:

CategoryID     : 4
SeverityID     : 5
ThreatID       : 1600
ThreatName     : TrojanDownloader:Win32/AcidReign
TypeID         : 0
PSComputerName:

CategoryID     : 6
SeverityID     : 5
ThreatID       : 1604
ThreatName     : Backdoor:Win32/AckCmd
TypeID         : 0
PSComputerName:

CategoryID     : 11
SeverityID     : 2
ThreatID       : 1605
ThreatName     : Dialer:Win32/Aconti
TypeID         : 0
```

```
PSComputerName:
[...]
```

La lista es bastante exhaustiva. Así, para la versión 1.233.3331.0 de las definiciones, ¡el catálogo está compuesto de 156 505 referencias!

Ejemplo: mostrar la amenaza asociada a su número de identificación

Utilizando el parámetro **-ThreatID** con el número de identificación asociado a una amenaza detectada previamente, puede recuperar con rapidez la información asociada a este software malicioso.

```
PS C:\Windows\system32> Get-MpThreatCatalog -ThreatID 2147577513


CategoryID     : 42
SeverityID     : 5
ThreatID       : 2147577513
ThreatName     : Virus:Win32/Polip.A
TypeID         : 0
PSComputerName:
```

Vemos aquí el virus llamado Polip.A y su nivel de gravedad.

7. Administrar los parámetros y las preferencias

Para finalizar con esta sección dedicada a Windows Defender, he aquí cómo administrar a través de Windows PowerShell el conjunto de parámetros y de preferencias de Windows Defender. Para ello, se utilizan los cuatro cmdlets con el nombre ***-MpPreference**.

a. Obtener la configuración y los parámetros

El cmdlet **Get-MpPreference** permite recuperar el conjunto de parámetros que componen Windows Defender. El cmdlet se utiliza sin ningún parámetro.

Ejemplo: mostrar el conjunto de parámetros de Windows Defender

```
PS C:\Windows\system32> Get-MpPreference


CheckForSignaturesBeforeRunningScan          : False
ComputerID                                    : 47337C95-D959-4294-
8458-62C9221328EB
DisableArchiveScanning                       : False
DisableAutoExclusions                        : False
DisableBehaviorMonitoring                    : False
DisableBlockAtFirstSeen                      : False
DisableCatchupFullScan                       : True
DisableCatchupQuickScan                      : True
```

```
DisableEmailScanning                            : True
DisableIntrusionPreventionSystem                :
DisableIOAVProtection                           : False
DisablePrivacyMode                              : False
DisableRealtimeMonitoring                       : False
DisableRemovableDriveScanning                   : True
DisableRestorePoint                             : True
DisableScanningMappedNetworkDrivesForFullScan: True
DisableScanningNetworkFiles                     : False
DisableScriptScanning                           : False
ExclusionExtension                              :
ExclusionPath                                   :
ExclusionProcess                                :
HighThreatDefaultAction                         : 0
LowThreatDefaultAction                          : 0
MAPSReporting                                   : 2
ModerateThreatDefaultAction                     : 0
PUAProtection                                   : 0
QuarantinePurgeItemsAfterDelay                  : 90
RandomizeScheduleTaskTimes                      : True
RealTimeScanDirection                           : 0
RemediationScheduleDay                          : 0
RemediationScheduleTime                         : 02:00:00
ReportingAdditionalActionTimeOut                : 10080
ReportingCriticalFailureTimeOut                 : 10080
ReportingNonCriticalTimeOut                     : 1440
ScanAvgCPULoadFactor                            : 50
ScanOnlyIfIdleEnabled                           : True
ScanParameters                                  : 1
ScanPurgeItemsAfterDelay                        : 15
ScanScheduleDay                                 : 0
ScanScheduleQuickScanTime                       : 00:00:00
ScanScheduleTime                                : 02:00:00
SevereThreatDefaultAction                       : 0
SignatureAuGracePeriod                          : 1440
SignatureDefinitionUpdateFileSharesSources      :
SignatureDisableUpdateOnStartupWithoutEngine : False
SignatureFallbackOrder                                                :
MicrosoftUpdateServer|MMPC
SignatureFirstAuGracePeriod                     : 120
SignatureScheduleDay                            : 8
SignatureScheduleTime                           : 01:45:00
SignatureUpdateCatchupInterval                  : 1
SignatureUpdateInterval                         : 0
SubmitSamplesConsent                            : 1
ThreatIDDefaultAction_Actions                   :
ThreatIDDefaultAction_Ids                       :
UILockdown                                      : False
UnknownThreatDefaultAction                      : 0
PSComputerName                                  :
```

La mayoría de los parámetros pueden modificarse mediante el cmdlet **`Set-MpPreference`**, que veremos a continuación. Para comprender mejor su rol y los posibles parámetros, consulte la tabla resumen de la sección Modificar la configuración y los parámetros.

b. Modificar la configuración y los parámetros

Para modificar la configuración y el comportamiento de Windows Defender, se utiliza el cmdlet **`Set-MpPreference`**. Este cmdlet admite un gran número de parámetros; cada uno permite modificar un elemento de la lista devuelta por **`Get-MpPreference`**. La siguiente tabla presenta los parámetros principales.

Parámetro	Descrip
-DisableArchiveScanning <Boolean>	Indica si los comprimidos .zip o .cab analizarse o Si vale entonces excluirán archivos.
-DisableIOAVProtection <Boolean>	Define si Defender todos los descargados archivos adj Si vale entonces característic desactivada
-DisablePrivacyMode <Boolean>	Indica confidencial activa o opción prohibir usuarios cor histórico amenazas. Si vale entonces confidencial está activa.
-DisableRealtime Monitoring <Boolean>	Se corresp la **Protección tiempo r** Windows Esta cara permite ide impedir instalación ejecución programas maliciosos. Si vale entonces característic desactivada

Parámetro	Descrip
-DisableRemovableDrive-Scanning <Boolean>	Indica s unidades e como dis memorias (USB, tarje etc.) analizarse un análisis d Si vale Tru unidades excluirán análisis.
-DisableRestorePoint <Boolean>	Indica s necesario c punto restauración sistema a eliminar, ej poner en cu los e detectados. Si vale T desactiva la de un pu restauración
-DisableScanningMappedNetworkDrivesForFullScan <Boolean>	Indica s unidades conectadas deben a durante un completo. Si vale Tru unidades excluirán análisis.
-ExclusionExtension <String[]>	Define extensiones archivos excluirán análisis.
-ExclusionPath <String[]>	Define las acceso a los o carpetas excluirán análisis. Si una carpet sus eleme excluirán.

Parámetro	Descrip
-ExclusionProcess <String[]>	Define los específicos excluirán análisis. Se los archivos por program los proceso Para procesos, u parámetro **Exclusior**
-HighThreatDefaultAction <ThreatAction>	Especifica e resolución llevará a automáticar cuando se una amer nivel eleva posibles val Quarantir Remove, Ig
-LowThreatDefaultAction <ThreatAction>	Especifica e resolución llevará a automáticar cuando se una amer nivel baj posibles val Quarantir Remove, Ig
-MAPSReporting <MAPSReportingType>	Se corresp la **Protección en la n** Windows Este también se como (*Microsoft Protection* que permite automáticar programas maliciosos Microsoft. información permite nuevas de y proteger

Parámetro	Descrip
	conjunto de de los progr seguridad Microsoft. Los posibles para esta op los siguiente • 0: des No se ninguna informac Microsoft • 1: su básica. informac enviada correspor el detectad proveniei acción emprend (eliminac otra) resultadc última fallo). • 2: su avanzada este m envía informac suplemer la su básica, incluyen ubicación software, nombre archivos, funcionar del softw manera e afecta al

Parámetro	Descrip
-ModerateThreatDefault Action <ThreatAction>	Especifica e resolución llevará a automáticar cuando se una amer nivel moder posibles val Quarantir Remove, Ig
-QuarantinePurgeItems AfterDelay <UInt32>	Indica el nú días qu guardarán elementos cuarentena ser purgad valor es 0, los eleme guardan indefinidam
-RealTimeScanDirection <ScanDirection>	Indica comportami la protecc tiempo real entradas y s los presentes unidades N posibles val • 0: ana archivos entrante salientes • 1: únicamer archivos entrante • 2: únicamer archivos salientes

Parámetro	Descrip
-SevereThreatDefault Action <ThreatAction>	Especifica e resolución llevará a automáticar cuando se una amer nivel grav posibles val Quarantir Remove, Ig
-SubmitSamplesConsent <ConsentType>	Se corresp la opción **E muestras automátic** Windows Esta opción invitar al u validar o nc de muest archivos sospechosos parámetro MAPSReport está des entonces Defender p consentimie usuario en de los valores sigu • 0: pedir una conf antes de las mues • 1: env muestras peligro automáti . • 2: nunc muestras • 3: envia las automáti .

Parámetro	Descrip
-UILockdown <Boolean>	Permite cualquier modificació configuració Windows De través de la gráfica (inc un administ Si vale entonces la gráfica sombreada opciones n accesibles.
-UnknownThreatDefault Action <ThreatAction>	Especifica e resolución llevará a automáticar cuando se una amer nivel des Los posible son: Quar Remove, Ig

Ejemplo: activar el análisis de las unidades extraíbles durante un escaneo completo

```
PS C:\Windows\system32> Set-MpPreference
-DisableRemovableDriveScanning:$false
PS                    C:\WINDOWS\system32>                    (Get-
MpPreference).DisableRemovableDriveScanning
False
```

Ejemplo: definir exclusiones en archivos, carpetas y extensiones de archivo

```
PS C:\Windows\system32> Set-MpPreference -ExclusionPath
C:\SecureFolder,'C:\Program        Files\myProg\test.exe'        -
ExclusionExtension
obj,rar,dgml
PS C:\WINDOWS\system32> Get-MpPreference | Select-Object
ExclusionPath,ExclusionExtension

ExclusionPath
ExclusionExtension
-------------                                          --------------
----
{C:\Program  Files\myProg\test.exe,  C:\SecureFolder}  {dgml,  obj,
rar}
```

Ejemplo: prohibir el acceso a los parámetros de Windows Defender

```
PS C:\Windows\system32> Set-MpPreference -UILockdown:$true
```

Así, si abre Windows Defender en **Configuración**, el conjunto de opciones aparecen sombreadas y no están accesibles:

El acceso al Centro de seguridad de Windows Defender está, ahora, bloqueado

c. Agregar y eliminar exclusiones

Si bien el cmdlet **`Set-MpPreference`** permite definir exclusiones, existen otros cmdlets que permiten modificar esta configuración: **`Add-MpPreference`** y **`Remove-MpPreference`**; la diferencia radica en el hecho de que estos comandos agregan o eliminan únicamente la exclusión o las exclusiones indicadas sin afectar a las demás que ya estuvieran configuradas. A diferencia de **`Add-MpPreference`**, **`Set-MpPreference`** borrará toda la configuración de las exclusiones implementada con el nuevo conjunto que se haya definido en la llamada al comando.

Los parámetros disponibles para **`Add-MpPreference`** son, por tanto, muy similares a los de **`Set-MpPreference`**:

Parámetro	Descripción
`-ExclusionExtension` `<String[]>`	Agrega las extensiones de archivos que se excluirán de los análisis.
`-ExclusionPath` `<String[]>`	Agrega las rutas de acceso a los archivos o carpetas que se excluirán de los análisis. Si se especifica una carpeta, se excluirán todos los elementos presentes en ella.

Parámetro	Descripción
-ExclusionProcess <String[]>	Agrega los procesos específicos que se excluirán de los análisis. Se excluyen los archivos abiertos por programas, y no los procesos en sí. Para excluir procesos, utilice el parámetro **-ExclusionPath**.

Ejemplo: agregar una extensión de archivo en las exclusiones

```
PS C:\Windows\system32> Add-MpPreference -ExclusionExtension
data
PS C:\Windows\system32> Get-MpPreference | Select-Object
ExclusionExtension

ExclusionExtension
------------------
{data, dgml, obj, rar}
```

De manera inversa, **Remove-MpPreference** elimina únicamente los elementos de exclusión que se especifiquen en el comando. Este cmdlet permite también eliminar las acciones por defecto en caso de detección de un software malicioso. He aquí los parámetros:

Parámetro	Descripción
-ExclusionExtension <String[]>	Elimina las extensiones de archivos de la lista de exclusión.
-ExclusionPath <String[]>	Elimina las rutas de acceso a los archivos o carpetas de la lista de exclusión.
-ExclusionProcess <String[]>	Elimina los procesos específicos de la lista de exclusión.
-HighThreatDefaultAction	Elimina la resolución automática (acción por defecto) para las amenazas de nivel elevado.

Parámetro	Descripción
-LowThreatDefaultAction	Elimina la resolución automática (acción por defecto) para las amenazas de nivel bajo.
-ModerateThreatDefaultAction	Elimina la resolución automática (acción por defecto) para las amenazas de nivel moderado.
-SevereThreatDefaultAction	Elimina la resolución automática (acción por defecto) para las amenazas de nivel grave.
-UnknownThreatDefaultAction	Elimina la resolución automática (acción por defecto) para las amenazas de nivel desconocido.

Ejemplo: eliminar una ruta de acceso excluida del análisis

```
PS C:\WINDOWS\system32> Remove-MpPreference -ExclusionPath
'C:\Program Files\myProg\test.exe'
```

Ejemplo: eliminar la acción por defecto para las amenazas de nivel bajo

```
PS C:\WINDOWS\system32> Remove-MpPreference
-LowThreatDefaultAction
```

Administrar usuarios y grupos locales

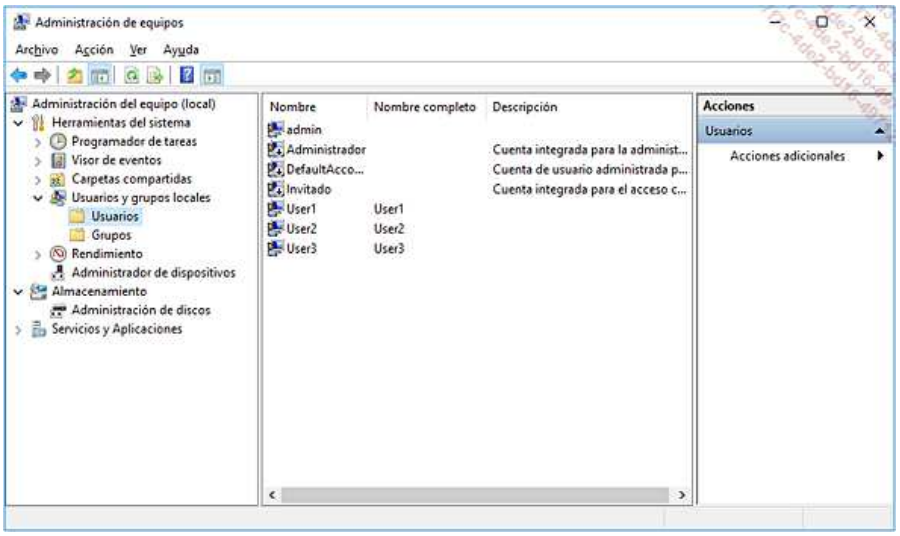
Para que un usuario pueda iniciar una sesión en Windows o acceder a un recurso (carpeta compartida, archivo, impresora compartida, etc.), es necesario crear previamente una cuenta de usuario. Esta puede ser local, es decir, conocida únicamente en el puesto de trabajo en el que se creará, o bien pertenecer a un directorio Active Directory, donde tendrá la posibilidad de conectarse a cualquier equipo del dominio, siempre y cuando posea los permisos correspondientes.

En esta sección veremos la gestión de las cuentas de usuario y los grupos locales. He aquí un breve resumen de los cmdlets y su rol:

- **`Add-LocalGroupMember`**: agrega un miembro en un grupo local.
- **`Disable-LocalUser`**: permite desactivar una cuenta de usuario.
- **`Enable-LocalUser`**: permite activar una cuenta de usuario.
- **`Get-LocalGroup`**: muestra el conjunto de grupos de seguridad locales presentes en el puesto de trabajo.
- **`Get-LocalGroupMember`**: recupera los miembros presentes en un grupo local.
- **`Get-LocalUser`**: muestra el conjunto de cuentas de usuario locales presentes en el puesto de trabajo.
- **`New-LocalGroup`**: crea un nuevo grupo de seguridad local.
- **`New-LocalUser`**: crea una nueva cuenta de usuario local.
- **`Remove-LocalGroup`**: elimina un grupo de seguridad local.
- **`Remove-LocalGroupMember`**: elimina un miembro de un grupo local.
- **`Remove-LocalUser`**: elimina una cuenta de usuario local.
- **`Rename-LocalGroup`**: renombra un grupo de seguridad local.
- **`Rename-LocalUser`**: renombra una cuenta de usuario local (y, por consiguiente, su identificador de login).
- **`Set-LocalGroup`**: permite modificar las propiedades de un grupo de seguridad local.

- **`Set-LocalUser`**: permite modificar las propiedades de una cuenta de usuario local.

A través de la interfaz gráfica, la gestión de usuarios y grupos locales se realiza desde la consola **Administración de equipos**. Para acceder a esta, haga clic con el botón derecho en el botón **Inicio** y seleccione **Administración de equipos** en la lista de accesos directos. En la ventana que se abre, seleccione **Usuarios y grupos locales** en el árbol.



Administración de usuarios y grupos locales a través de la interfaz gráfica

1. Usuarios

En esta sección veremos la administración de usuarios locales: cómo crear una nueva cuenta de usuario y también cómo eliminar, desactivar o incluso modificar una cuenta existente.

a. Obtener la lista de cuentas de usuario

Para obtener la lista de cuentas de usuario locales creadas en el puesto de trabajo, se utiliza el cmdlet **`Get-LocalUser`**. He aquí los parámetros:

Parámetro	Descripción

Parámetro	Descripción
-Name <String[]>	Designa el nombre de la cuenta o las cuentas de usuario que se devolverán.
-SID <SecurityIdentifier[]>	Designa el identificador de seguridad (de tipo SID - *Security Identifiers*) de la cuenta o las cuentas de usuario que se devolverán.

Ejemplo: mostrar todas las cuentas de usuario locales

```
PS C:\Windows\system32> Get-LocalUser

Name           Enabled Description
----           ------- -----------
Admin          True
Administrador  False   Cuenta de usuario de administración
DefaultAccount  False     Cuenta de usuario administrada por el
sistema.
Invitado       False   Cuenta de usuario invitado
```

Ejemplo: recuperar toda la información correspondiente a una cuenta de usuario

```
PS C:\Windows\system32> Get-LocalUser -Name Admin | Format-List
*


AccountExpires         :
Description            :
Enabled                : True
FullName               :
PasswordChangeableDate: 18/11/2016 11:24:22
PasswordExpires        :
UserMayChangePassword : True
PasswordRequired       : False
PasswordLastSet        : 17/11/2016 11:24:22
LastLogon              : 03/02/2017 13:31:16
Name                   : Admin
SID                     : S-1-5-21-1798178853-4209864326-1796974730-
1001
PrincipalSource        : Local
ObjectClass            : Usuario
```

De este modo, puede recuperar información como las fechas de expiración de la cuenta (`AccountExpires`) o de la contraseña (`PasswordExpires`), y también el identificador de seguridad de la cuenta (`SID`). También se obtiene información adicional, como la fecha de última conexión con esta cuenta (`LastLogon`).

b. Crear una cuenta de usuario local

El cmdlet **New-LocalUser** permite crear una nueva cuenta de usuario. Esta, una vez creada, no formará parte de ningún grupo, y por consiguiente no poseerá ningún permiso en el puesto de trabajo.

Los distintos parámetros que ofrece este cmdlet permiten configurar la cuenta de usuario tras su creación:

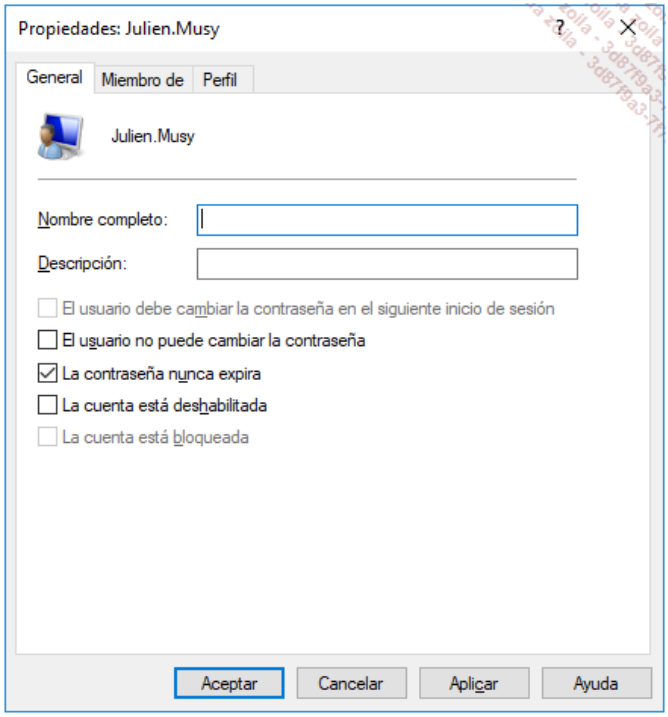
Parámetro	Descripción
-AccountExpires <DateTime>	Especifica la fecha y hora de expiración de la cuenta de usuario.
-AccountNeverExpires	Indica que la cuenta nunca expira.
-Description <String>	Especifica una descripción para esta cuenta de usuario. La descripción no debe superar los 48 caracteres.
-Disabled	Indica que la cuenta de usuario creada estará desactivada.
-FullName <String>	Especifica el nombre completo para esta cuenta de usuario.
-Name <String>	Especifica el nombre para esta cuenta de usuario. Se trata del login.
-NoPassword	Indica que la cuenta de usuario no tiene contraseña.
-Password <SecureString>	Especifica la contraseña asociada a la cuenta de usuario creada.
-PasswordNeverExpires	Indica que la contraseña de la cuenta de usuario nunca expira.

Parámetro	Descripción
-UserMayNotChangePassword	Indica que el usuario no puede cambiar su contraseña.

Ejemplo: crear una cuenta de usuario

```
PS C:\Windows\system32> $pwd = ConvertTo-SecureString "P@$$w0rd"

-AsPlainText -Force
PS C:\Windows\system32> New-LocalUser -Name Julien.Musy -Password
$pwd
-PasswordNeverExpires
```

Se crea la cuenta de usuario local y estará visible desde la consola Usuarios y grupos locales. Si se visualizan las propiedades de la cuenta de usuario que se acaba de crear, he aquí el resultado:



Propiedades de la cuenta de usuario creada

Ejemplo: crear una cuenta de usuario con una fecha de expiración

```
PS C:\Windows\system32> $pwd = ConvertTo-SecureString "P@$$w0rd"

-AsPlainText -Force
PS C:\Windows\system32> $date = (Get-Date).AddDays(7)
```

```
PS C:\Windows\system32> New-LocalUser -Name Frederic -Password
$pwd
-PasswordNeverExpires -AccountExpires $date
```

La nueva cuenta de usuario creada tiene una validez de una semana. Pasado este tiempo, el usuario no podrá iniciar sesión en el puesto de trabajo.

c. Modificar una cuenta de usuario local

Para modificar una cuenta de usuario existente, disponemos del cmdlet **`Set-LocalUser`**. Los parámetros son similares a los presentados en la creación de una cuenta de usuario:

Parámetro	Descripción
`-AccountExpires` `<DateTime>`	Especifica la fecha y hora de expiración de la cuenta de usuario.
`-AccountNeverExpires`	Indica que la cuenta nunca expira.
`-Description` `<String>`	Especifica una descripción para esta cuenta de usuario. La descripción no debe superar los 48 caracteres.
`-FullName` `<String>`	Especifica el nombre completo para esta cuenta de usuario.
`-Name` `<String>`	Designa el nombre de la cuenta de usuario que se modificará.
`-Password` `<SecureString>`	Especifica la contraseña asociada a la cuenta de usuario creada.
`-PasswordNeverExpires` `<Boolean>`	Indica que la contraseña de la cuenta de usuario nunca expira (`$True`) o bien si debe expirar (`$False`).
`-SID` `<SecurityIdentifier>`	Designa el identificador de seguridad de la cuenta de usuario que se modificará.

Parámetro	Descripción
-UserMayChangePassword <Boolean>	Indica si el usuario puede cambiar su contraseña ($True) o no ($False).

Ejemplo: modificar las propiedades de una cuenta de usuario

```
PS C:\Windows\system32> Set-LocalUser -Name Julien.Musy -FullName
"Julien Musy" -Description "Workstation Administrator"
```

En este ejemplo, las propiedades Nombre completo (FullName) y Descripción (Description) de la cuenta de usuario Julien.Musy se modifican o agregan si no existe esta información.

Ejemplo: cambiar la contraseña de una cuenta de usuario

```
PS C:\Windows\system32> Set-LocalUser -Name Julien.Musy -Password
(ConvertTo-SecureString -String "m07d3p455" -AsPlainText -Force)
```

d. Renombrar una cuenta de usuario local

Rename-LocalUser permite cambiar el nombre de la cuenta de usuario y, por consiguiente, su identificador de login. Este cmdlet se utiliza haciendo referencia a la cuenta de usuario que se renombrará mediante su nombre actual o SID, indicando a continuación el nuevo nombre con el parámetro **-NewName**.

Parámetro	Descripción
-Name <String>	Designa el nombre actual de la cuenta de usuario que se renombrará.
-NewName <String>	Especifica el nuevo nombre para la cuenta de usuario, y por consiguiente su nuevo identificador de login.
-SID <SecurityIdentifier>	Designa el identificador de seguridad de la cuenta de usuario que se renombrará.

Ejemplo: renombrar una cuenta de usuario

La cuenta de usuario se renombra a ws.admin, y en lo sucesivo deberá utilizar este identificador para iniciar una sesión con esta

```
PS C:\Windows\system32> Rename-LocalUser -Name Julien.Musy
-NewName ws.admin
```

cuenta. Sin embargo, observe que, si bien el nombre ha cambiado, la ruta de acceso a su perfil de usuario no lo ha hecho. He aquí las variables de entorno de la sesión de usuario tras renombrarla:

```
PS C:\Users\Julien.Musy> $env:USERNAME
ws.admin
PS C:\Users\Julien.Musy> $env:USERPROFILE
C:\Users\Julien.Musy
```

¡De ahí la importancia de utilizar correctamente las variables de entorno en los scripts!

e. Activar una cuenta de usuario local

En caso de tener cuentas de usuario desactivadas, por ejemplo tras la creación de una nueva cuenta con el parámetro **-Disabled** de **New-LocalUser**, el cmdlet **Enable-LocalUser** permite reactivarla. Las cuentas desactivadas poseen un indicador (flecha hacia abajo) en su icono. Recuerde que el cmdlet **Get-LocalUser** también permite consultar rápidamente el conjunto de cuentas desactivadas:

```
PS C:\WINDOWS\system32> Get-LocalUser | ? {$_.Enabled -eq
$false}

Name            Enabled Description
----            ------- -----------
Administrador    False   Cuenta de usuario de administración
DefaultAccount   False    Cuenta de usuario administrada por el
sistema.
Invitado         False   Cuenta de usuario invitado
Ludovic.White    False
```

He aquí los parámetros de **Enable-LocalUser**:

Parámetro	Descripción
-Name <String[]>	Designa el nombre de la cuenta o las cuentas de usuario que se activarán.

Parámetro	Descripción
-SID <SecurityIdentifier[]>	Designa el identificador de seguridad de la cuenta o las cuentas de usuario que se activarán.

Ejemplo: activar una cuenta de usuario

```
PS C:\Windows\system32> Enable-LocalUser -Name Ludovic.White
```

f. Desactivar una cuenta de usuario local

De manera inversa, puede desactivar en cualquier momento una cuenta de usuario. Esta operación se realiza mediante el cmdlet **Disable-LocalUser**. Los parámetros son idénticos:

Parámetro	Descripción
-Name <String[]>	Designa el nombre de la cuenta o las cuentas de usuario que se desactivarán.
-SID <SecurityIdentifier[]>	Designa el identificador de seguridad de la cuenta o las cuentas de usuario que se desactivarán.

Ejemplo: desactivar una cuenta de usuario

```
PS C:\Windows\system32> Disable-LocalUser -Name Ludovic.White
```

g. Eliminar una cuenta de usuario local

Para terminar, la última operación que es posible realizar sobre las cuentas de usuario locales es la eliminación. **Remove-LocalUser** permite llevar a cabo esta tarea, indicando el nombre o el SID de la cuenta que se ha de eliminar.

Observe, sin embargo, que si bien la cuenta se ha eliminado, no ocurre así con el perfil de usuario y el conjunto de datos que contiene.

Parámetro	Descripción

Parámetro	Descripción
-Name <String[]>	Designa el nombre de la cuenta o las cuentas de usuario que se eliminarán.
-SID <SecurityIdentifier[]>	Designa el identificador de seguridad de la cuenta o las cuentas de usuario que se eliminarán.

Ejemplo: eliminar una cuenta de usuario

```
PS C:\Windows\system32> Remove-LocalUser -Name Ludovic.White
```

Tenga precaución: no se pide confirmación durante la ejecución de este cmdlet. Puede agregar el parámetro **-Confirm** para solicitar una confirmación.

Del mismo modo, tenga en cuenta que, si recrea a continuación una nueva cuenta con el mismo nombre que la que acaba de eliminar, esta no recuperará el entorno del usuario de la cuenta anterior. En efecto, el SID será diferente y, por consiguiente, cuando el usuario se autentique con esta nueva cuenta, Windows lo identificará como una nueva persona y creará un nuevo entorno de usuario, de modo que la ruta de acceso al perfil del usuario será diferente (Windows agregará el nombre del equipo en el nombre de la carpeta):

```
PS C:\Users\Ludovic.White.W10> $env:USERNAME
Ludovic.White
PS C:\Users\Ludovic.White.W10> $env:USERPROFILE
C:\Users\Ludovic.White.W10
```

2. Grupos

Los grupos permiten agrupar un conjunto de objetos (cuentas de usuario, entidades de seguridad integradas, etc.), y facilitar así la gestión y la administración.

El mejor ejemplo es la agrupación de cuentas de usuario en un mismo grupo, que representa una actividad o entidad. Será más sencillo otorgar o retirar permisos sobre un recurso al conjunto de usuarios presentes en estos grupos.

Por último, debe saber que los grupos predefinidos durante una instalación de Windows (Administradores, Usuarios, etc.) no

pueden eliminarse. En efecto, estos grupos poseen permisos a nivel de la máquina que asignan permisos específicos a las cuentas de usuario que serán miembros de estos grupos predefinidos. Por ejemplo, el grupo de Administradores dará un acceso completo a la máquina Windows a los miembros de este grupo.

a. Obtener la lista de grupos locales

El cmdlet **Get-LocalGroup** permite mostrar el conjunto de grupos locales presentes en el puesto de trabajo.

Parámetro	Descripción
-Name <String[]>	Designa el nombre del grupo o de los grupos que se devolverán.
-SID <SecurityIdentifier[]>	Designa el identificador de seguridad del grupo o de los grupos de seguridad que se devolverán.

Ejemplo: recuperar el conjunto de grupos

```
PS C:\Windows\system32> Get-LocalGroup

Name                                               Description
----                                               -----------
Administradores                                         Los miembros del
grupo Administradores ...
Administradores de Hyper-V                         Los miembros de este
grupo disponen de un...
Duplicadores                                         Tiene en cuenta la
replicación de archi...
IIS_IUSRS                                             Grupo integrado
utilizado por los servic...
Invitados                                               Los miembros del
grupo Invitados disponen...
Lectores de registros de eventos                   Los miembros de este
grupo pueden leer...
Operadores de asistencia de control de acceso Los miembros de este
grupo pueden leer...
Operadores criptográficos                            Los miembros están
autorizados a realizar...
Operadores de configuración de red                 Los miembros de este
grupo disponen de...
Operadores de copia de seguridad                        Los miembros del
grupo Operadores de su...
System Managed Accounts Group                      Los miembros de este
grupo se administran...
Usuarios                                                 Los usuarios no
pueden realizar...
Usuarios avanzados                                          Los usuarios
```

```
avanzados están inc...
Usuarios de administración remota             Los miembros de este
grupo tienen acceso...
Usuarios del Monitor de rendimiento           Los miembros de este
grupo puede acceder...
Usuarios de Escritorio remoto                 Los miembros de este
grupo disponen de ...
Usuarios del registro de rendimiento          Los miembros de este
grupo pueden plani...
Usuarios del modelo COM distribuido             Los miembros tiene
permisos para ejecut...
```

Se obtiene el conjunto de grupos presentes en el puesto de trabajo. En el caso anterior, se trata del conjunto de grupos predefinidos por Windows 10 (los grupos predefinidos pueden variar en función de la versión de Windows).

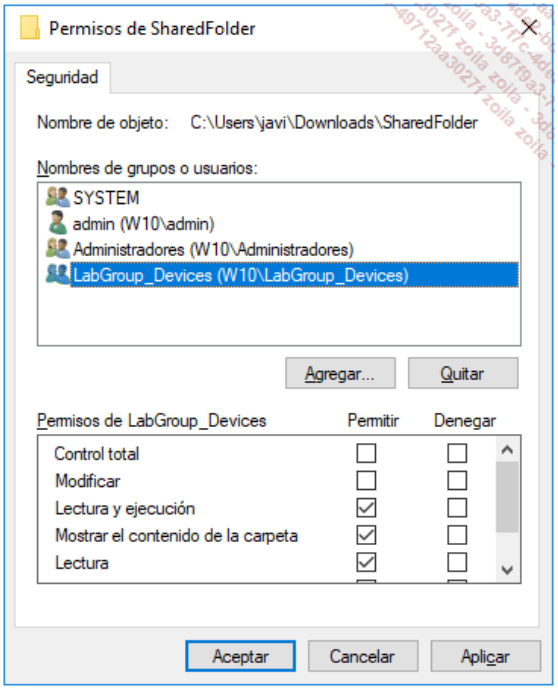
Ejemplo: mostrar las propiedades de un grupo

```
PS C:\Windows\system32> Get-LocalGroup -Name Administradores |
Format-List *


Description     : Los miembros del grupo Administradores disponen
de acceso
                  completo e ilimitado al equipo y al dominio
Name            : Administradores
SID             : S-1-5-32-544
PrincipalSource: Local
ObjectClass     : Grupo
```

Como puede observar, también existe un SID para los grupos de seguridad. Windows detecta los grupos a través de este identificador y determina así los permisos asignados a los grupos.

b. Crear un grupo local

La creación de un grupo local se realiza mediante el cmdlet **New-LocalGroup**. Como ocurre con las cuentas de usuario, un grupo recién creado no poseerá ningún permiso en la máquina. Debe saber también que no es posible incluir un grupo local como miembro de otro grupo local. Los grupos locales creados pueden, sin embargo, configurarse para definir permisos NTFS sobre carpetas o archivos, también sobre recursos compartidos. De este modo, todos los miembros presentes en este grupo estarán afectados por la configuración que se haya definido.



Asignación de permisos NTFS en un grupo local

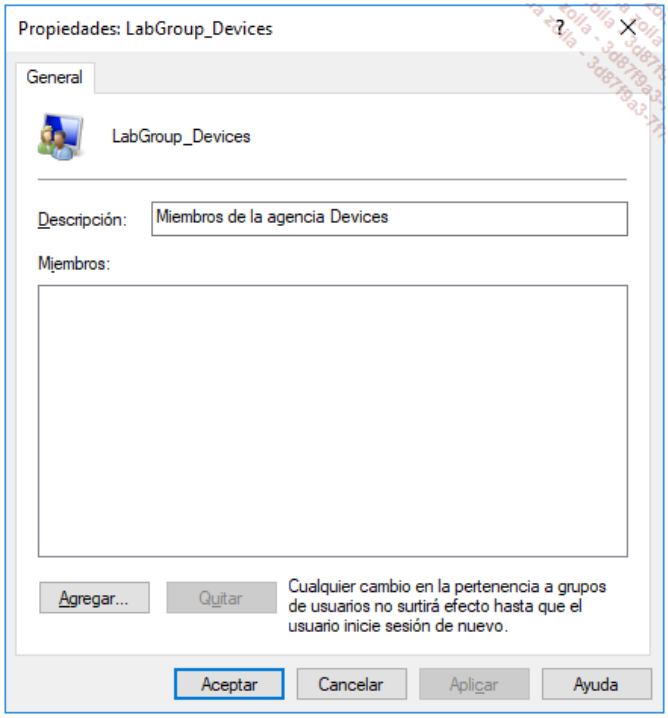
He aquí los parámetros de **`New-LocalGroup`**:

Parámetro	Descripción
`-Description` `<String>`	Introduce una descripción para el grupo creado.
`-Name` `<String>`	Especifica el nombre del grupo que se va a crear.

Ejemplo: crear un grupo

```
PS C:\Windows\system32> New-LocalGroup -Name LabGroup_Devices
-Description "Miembros la agencia Devices"
```

Se ha creado el grupo LabGroup_Devices. Desde la interfaz gráfica de Windows, verá aparecer el grupo de la siguiente manera:



Propiedades del grupo creado

c. Modificar las propiedades de un grupo local

Para modificar las propiedades de un grupo creado, puede utilizar el cmdlet **`Set-LocalGroup`**. Con este comando solo es posible modificar la descripción del grupo. He aquí los parámetros:

Parámetro	Descripción
`-Description` `<String>`	Define la nueva descripción del grupo indicado.
`-Name` `<String>`	Designa el nombre del grupo que se va a modificar.
`-SID` `<SecurityIdentifier>`	Designa el identificador de seguridad del grupo que se va a modificar.

Ejemplo: crear un grupo

```
PS C:\Windows\system32> Set-LocalGroup -Name LabGroup_Devices
-Description "Miembros con acceso al recurso compartido Lab"
```

La descripción del grupo se reemplaza por la nueva descripción indicada en el comando.

d. Renombrar un grupo local

Para editar el nombre de un grupo, se utiliza el cmdlet **Rename-LocalGroup**. Será necesario indicar el grupo que se va a renombrar, mediante su nombre actual o SID, y definir a continuación el nuevo nombre que se le asignará.

Parámetro	Descripción
-Name <String>	Designa el nombre del grupo que se va a renombrar.
-NewName <String>	Especifica el nuevo nombre que se asignará al grupo.
-SID <SecurityIdentifier>	Designa el identificador de seguridad del grupo que se va a renombrar.

Ejemplo: renombrar un grupo

```
PS C:\Windows\system32> Rename-LocalGroup -Name LabGroup_Devices
-NewName BU_Devices
```

e. Eliminar un grupo local

La última acción posible sobre los grupos es la eliminación. Esta operación se realiza mediante el cmdlet **Remove-LocalGroup**. Observe que los objetos presentes en el grupo que se elimina no se verán afectados. He aquí los parámetros:

Parámetro	Descripción
-Name <String[]>	Designa el nombre del grupo o de los grupos que hay que eliminar.
-SID <SecurityIdentifier[]>	Designa el identificador de seguridad del grupo o de los grupos que hay que eliminar.

Ejemplo: eliminar un grupo

```
PS C:\Windows\system32> Remove-LocalGroup -Name BU_Devices
```

Se elimina el grupo BU_Devices. Tenga precaución, pues no se pide confirmación; puede añadir una solicitud de confirmación de eliminación agregando el parámetro **-Confirm**.

Las referencias a este grupo eliminado definidas en la seguridad NTFS dejan de estar activas y, como ocurre con las cuentas de usuario, el hecho de recrear un grupo con el mismo nombre no es una solución reversible: el SID será necesariamente diferente. Habrá que volver a aplicar todos los permisos NTFS.

3. Administrar los miembros de los grupos locales

Una vez creado un grupo local, es preciso agregar los miembros, pues en caso contrario el grupo no tiene ningún interés. Veremos cómo agregar objetos (cuentas de usuarios locales o de dominio, grupos de dominio, etc.) y también cómo eliminarlos de los grupos.

a. Agregar miembros en un grupo local

Es posible agregar miembros en un grupo mediante el cmdlet **`Add-LocalGroupMember`**. A continuación, es preciso indicar el grupo, utilizando su nombre o SID, así como la cuenta o las cuentas de usuario que serán miembros de dicho grupo. He aquí los parámetros:

Parámetro	Descripción
`-Group` `<LocalGroup>`	Designa el grupo al que se agregarán los miembros.
`-Member` `<LocalPrincipal[]>`	Indica el objeto o los objetos que se agregarán al grupo indicado. Los objetos especificados pueden definirse por su nombre, SID o LocalPrincipal.
`-Name` `<String>`	Designa el nombre del grupo al que se agregarán los miembros.
`-SID` `<SecurityIdentifier>`	Designa el SID del grupo al que se agregarán los miembros.

Ejemplo: agregar un miembro al grupo Administradores

```
PS C:\Windows\system32> Add-LocalGroupMember -Group
"Administradores"
```

```
-Member "Julien.Musy"
```

Puede definir el grupo sobre el que desea realizar la operación bien mediante el parámetro **-Group** o bien mediante **-Name**, o incluso **-SID**.

La cuenta de usuario local Julien.Musy se ha agregado al grupo predefinido Administradores. En lo sucesivo, el usuario puede iniciar una sesión en el puesto de trabajo con todos los permisos.

Ejemplo: agregar miembros a un grupo local

```
PS C:\Windows\system32> Add-LocalGroupMember -Group
"LabGroup_Devices"
-Member "CONTOSO\Julien","CONTOSO\Benjamin","CONTOSO\Octavio",
"CONTOSO\GG DSI"
```

El alcance de **Add-LocalGroupMember** no se limita únicamente a las cuentas locales. También es posible agregar cuentas de Microsoft, Azure Active Directory e incluso provenientes de otros dominios.

Ejemplo: agregar miembros a un grupo local con un objeto LocalPrincipal

```
PS C:\Windows\system32> Get-LocalGroupMember -Group
LabGroup_Devices
-Member * | Add-LocalGroupMember -Group LabGroup_Server
```

El resultado devuelto por el cmdlet **Get-LocalGroupMember** es de tipo LocalPrincipal, de modo que puede transferirse al cmdlet **Add-LocalGroupMember** para agregar miembros a un grupo.

b. Obtener los miembros presentes en un grupo local

Para recuperar el conjunto de objetos que son miembros de un grupo, se utiliza **Get-LocalGroupMember**.

Parámetro	Descripción
-Group <LocalGroup>	Designa el grupo cuyos miembros se recuperarán.

Parámetro	Descripción
-Member <String>	Indica el objeto miembro del grupo que se recuperará. El objeto especificado puede definirse por su nombre o SID, y se admite el uso del carácter comodín * (*wildcard*).
-Name <String>	Designa el nombre del grupo cuyos miembros se recuperarán.
-SID <SecurityIdentifier>	Designa el SID del grupo cuyos miembros se recuperarán.

Ejemplo: recuperar todos los miembros de un grupo local

```
PS      C:\Windows\system32>      Get-LocalGroupMember      -Group
"LabGroup_Devices"

ObjectClass Name             PrincipalSource
----------- ----             ---------------
Usuario     CONTOSO\Benjamin ActiveDirectory
Grupo       CONTOSO\GG DSI   ActiveDirectory
Usuario     CONTOSO\Julien   ActiveDirectory
Usuario     CONTOSO\Octavio  ActiveDirectory
Usuario     W10\Julien.Musy  Local
```

El comando anterior devuelve el conjunto de miembros presentes en el grupo LabGroup_Devices. El resultado del comando indica también claramente de qué tipo de objeto se trata, así como el origen del objeto (Local, Active Directory, etc.).

Ejemplo: recuperar los miembros (cuentas de usuario) de un grupo proveniente de un dominio específico

```
PS      C:\Windows\system32>      Get-LocalGroupMember      -Group
"LabGroup_Devices"
-Member CONTOSO\* | Where-Object {$_.ObjectClass -eq "Usuario"}

ObjectClass Name             PrincipalSource
----------- ----             ---------------
Usuario     CONTOSO\Benjamin ActiveDirectory
Usuario     CONTOSO\Julien   ActiveDirectory
Usuario     CONTOSO\Octavio  ActiveDirectory
```

En este ejemplo, solo se muestran las cuentas de usuario provenientes del dominio CONTOSO.

c. Eliminar miembros de un grupo local

Para terminar, la última acción que es posible realizar es la eliminación de los miembros presentes en un grupo. Esta operación se lleva a cabo mediante **Remove-LocalGroupMember**. Como en los anteriores cmdlets de esta sección, hay que indicar el grupo, así como el miembro o los miembros que se eliminarán.

Parámetro	Descripción
-Group <LocalGroup>	Designa el grupo cuyos miembros se eliminarán.
-Member <LocalPrincipal[]>	Indica el objeto o los objetos que se eliminarán en el grupo indicado. Los objetos especificados pueden definirse mediante su nombre, SID o LocalPrincipal.
-Name <String>	Designa el nombre del grupo cuyos miembros se eliminarán.
-SID <SecurityIdentifier>	Designa el SID del grupo cuyos miembros se eliminarán.

Ejemplo: eliminar un miembro presente en un grupo

```
PS C:\Windows\system32> Remove-LocalGroupMember -Group
LabGroup_Devices
-Member "W10\Julien.Musy","CONTOSO\Octavio"
```

Los miembros definidos en el comando se eliminan del grupo. Estas cuentas ya no poseen los permisos que tuvieran por ser miembros de este grupo.

Compartición de carpetas

En la actualidad, compartimos cada vez más archivos, y generalmente más voluminosos.Una de las formas de compartir archivos es ubicarlos en un recurso compartido de red. Así, desde otro dispositivo (ordenador, tableta…), tendrá acceso a este recurso compartido para consultar o recuperar los archivos.

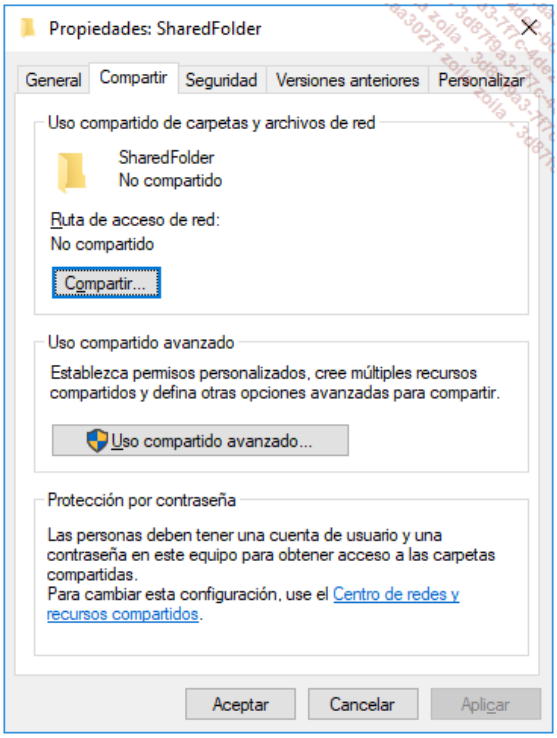
Esta compartición de recursos se realiza mediante el protocolo SMB (*Server Message Block*) y los puestos cliente acceden desde una dirección utilizando la nomenclatura UNC (*Uniform Naming Convention*), de tipo: *servidor**recurso_compartido**carpeta**archivo*.

He aquí el conjunto de cmdlets que permiten administrar los recursos compartidos de carpetas:

- **`New-SmbShare`**: permite crear el recurso compartido de red a nivel de una carpeta.
- **`Get-SmbShare`**: recupera información relativa al conjunto de carpetas compartidas.
- **`Set-SmbShare`**: permite aplicar nuevas configuraciones en un recurso compartido de red.
- **`Remove-SmbShare`**: permite eliminar un recurso compartido de red.
- **`Block-SmbShareAccess`**: agrega permisos de tipo «Denegar».
- **`Unblock-SmbShareAccess`**: elimina permisos de tipo «Denegar».
- **`Grant-SmbShareAccess`**: agrega permisos de tipo «Autorizar».
- **`Revoke-SmbShareAccess`**: elimina permisos de tipo «Autorizar».
- **`Get-SmbShareAccess`**: muestra el detalle de las autorizaciones actuales sobre un recurso compartido de red.

La activación de un recurso compartido se realiza desde las propiedades de una carpeta. Haga clic con el botón derecho en una carpeta y seleccione **Propiedades**. En la ventana emergente, haga clic en la pestaña **Compartir**. El botón **Uso compartido**

avanzado permitirá definir con más detalle las opciones de compartición.



Ventana de propiedades de una carpeta abierta por la pestaña ***Compartir***

1. Crear un recurso compartido de red

El cmdlet **`New-SmbShare`** permite habilitar un recurso compartido de red sobre una carpeta y definir un cierto número de parámetros durante su implementación. He aquí un resumen de los parámetros disponibles:

Parámetro	Descripción
`-ChangeAccess` `<String[]>`	Designa las cuentas de usuario o los grupos a los que se concederán permisos de modificación.

Parámetro	Descripción
-ConcurrentUserLimit <UInt32>	Especifica el número máximo de usuarios que pueden conectarse al recurso compartido simultáneamente. El valor predeterminado es 20.
-Description <String>	Permite introducir una descripción para el recurso compartido de red. No debe superar los 256 caracteres.
-FullAccess <String[]>	Designa las cuentas de usuario o los grupos a los que se concederán permisos de control total.
-Name <String>	Especifica un nombre para el recurso compartido de red.
-NoAccess <String[]>	Designa las cuentas de usuario o los grupos a los que se denegarán el conjunto de permisos (control total, modificación, lectura).
-Path <String>	Indica la ruta de acceso a la carpeta que se va a compartir.
-ReadAccess <String[]>	Designa las cuentas de usuario o los grupos a los que se concederán permisos de lectura.

Ejemplo 1: crear un recurso compartido de red

La creación de un recurso compartido de red mediante **New-SmbShare** requiere como mínimo dos datos: la ruta de acceso a la carpeta compartida (**-Path**) y el nombre que se le asignará (**-Name**).

```
PS C:\Windows\system32> New-SmbShare -Path D:\Temp\ -Name Temp

Name ScopeName Path    Description
---- --------- ----    -----------
Temp *         D:\Temp
```

Si bien la carpeta compartida es la carpeta D:\Temp\, el nombre asignado puede ser diferente, y será este último el que visualicen

los clientes que se conecten al recurso compartido.

De manera similar, es posible crear un recurso compartido de red oculto, agregando el carácter $ al final del nombre:

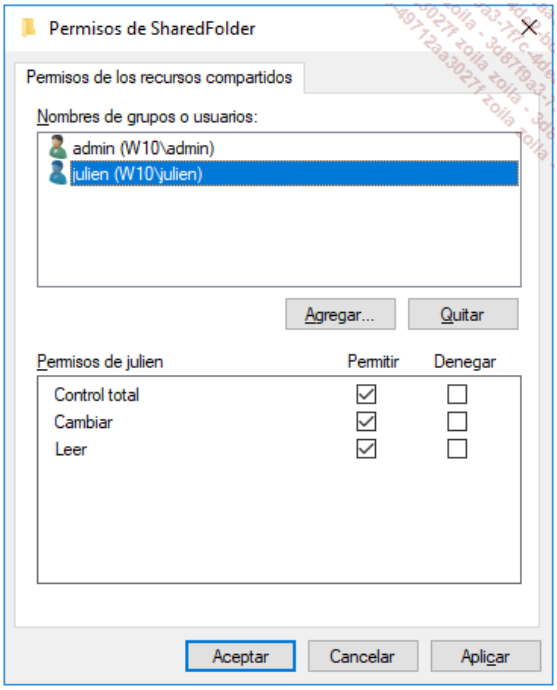
```
PS C:\Windows\system32> New-SmbShare -Path D:\ShareFolder\ -Name
Share$
-Description "Recurso compartido oculto"

Name   ScopeName Path           Description
----   --------- ----           -----------
Share$ *         D:\ShareFolder Recurso compartido oculto
```

Ejemplo 2: crear un recurso compartido definiendo los permisos

Por defecto, tras la creación del recurso compartido, si no se especifica ningún permiso en el comando, se configuran los siguientes permisos sobre el recurso compartido: permiso de acceso de solo lectura al grupo Todos. Sin embargo, tras la ejecución de **New-SmbShare**, es posible definir otros permisos utilizando los parámetros: **-FullAccess**, **-ChangeAccess**, **-ReadAccess** y **-NoAccess**.

El siguiente ejemplo permite asignar permisos de Control total a uno o varios usuarios, ya sean usuarios del dominio o locales:

```
PS C:\Windows\system32> New-SmbShare -Path D:\ShareFolder\ -Name
ShareFolder -FullAccess CONTOSO\julien,Admin

Name        ScopeName Path           Description
----        --------- ----           -----------
ShareFolder *         D:\ShareFolder
```

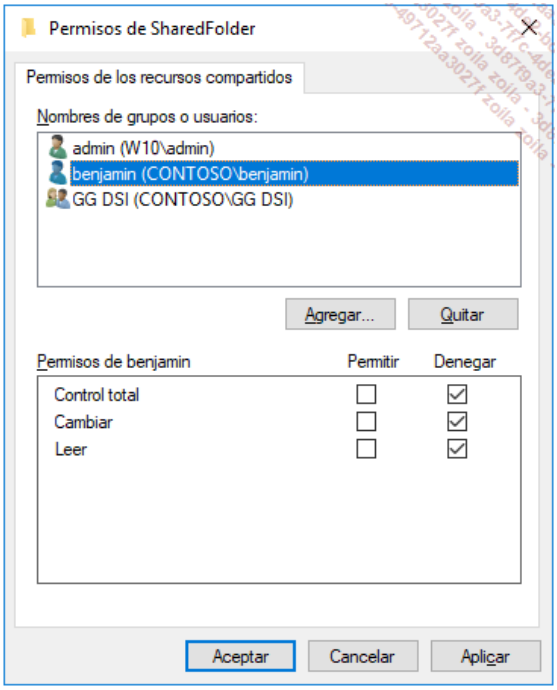


Creación de un recurso compartido con permisos de acceso

Ejemplo 3: crear un recurso compartido configurando sus permisos

```
PS D:\> New-SmbShare -Path D:\ShareFolder\ -Name ShareFolder -
FullAccess
Admin -ReadAccess "CONTOSO\GG DSI" -NoAccess CONTOSO\benjamin

Name        ScopeName Path           Description
----        --------- ----           -----------
ShareFolder *         D:\ShareFolder
```

En este caso, se han asignado permisos de lectura al grupo GG DSI, y se ha denegado el permiso de acceso para la cuenta de Benjamin. Si Benjamin pertenece al grupo GG DSI, la prohibición es prioritaria, y por tanto no tendrá acceso al recurso compartido.



Creación de un recurso compartido con denegación de acceso

2. Recuperar información de los recursos compartidos de red

El cmdlet **`Get-SmbShare`** permite recuperar el conjunto de recursos compartidos de red activos en el puesto de trabajo, así como su información.

Parámetro	Descripción
`-ConcurrentUserLimit` `<UInt32>`	Enumera los recursos compartidos con este límite en el número de conexiones de usuario simultáneas.
`-Name` `<String[]>`	Especifica el nombre del recurso o de los recursos compartidos que se van a enumerar.

Parámetro	Descripción
-Special <Boolean[]>	Enumera los recursos compartidos especiales. Se trata de recursos compartidos de administración remota, por defecto, e IPC remoto. Estos recursos compartidos se muestran por defecto.

Ejemplo: mostrar todos los recursos compartidos activos

```
PS D:\> Get-SmbShare

Name         ScopeName Path           Description
----         --------- ----           -----------
ADMIN$       *         C:\WINDOWS     Administración remota
C$              *            C:\                Recurso compartido por
defecto
D$              *            D:\                Recurso compartido por
defecto
IPC$         *                        IPC remoto
ShareFolder  *         D:\ShareFolder
Temp$        *         D:\Temp
```

Sin especificar ningún parámetro, se muestra el conjunto de recursos compartidos, incluyendo los recursos compartidos especiales.

Ejemplo: mostrar los recursos compartidos activos, excluyendo los recursos compartidos especiales

```
PS D:\> Get-SmbShare -Special:$false

Name         ScopeName Path           Description
----         --------- ----           -----------
ShareFolder  *         D:\ShareFolder
Temp$        *         D:\Temp
```

Ejemplo: mostrar las propiedades de un recurso compartido

```
PS D:\> Get-SmbShare -Name ShareFolder | Format-List -Property
Name,Description,Path,ConcurrentUserLimit,CurrentUsers,EncryptData,

Special


Name                : ShareFolder
Description         :
Path                : D:\ShareFolder
ConcurrentUserLimit: 0
CurrentUsers        : 0
EncryptData         : False
Special             : False
```

Si la propiedad `ConcurrentUserLimit` vale `0`, esto quiere decir que el número máximo de usuarios que pueden conectarse simultáneamente se ha configurado sobre este recurso compartido, que es de 20 para Windows.

3. Modificar un recurso compartido de red existente

También es posible modificar la configuración de los recursos compartidos de red activos mediante el cmdlet **`Set-SmbShare`**. Sin embargo, la administración de los permisos sobre el recurso compartido no se realiza mediante este comando. Para modificar los permisos, se utilizan los llamados **`*-SmbShare Access`** que se describen en la sección Administrar los permisos del recurso compartido SMB.

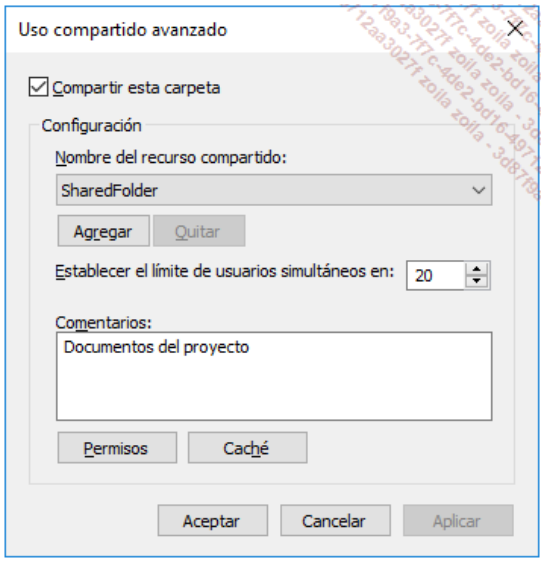
He aquí un resumen de los parámetros de **`Set-SmbShare`** más importantes:

Parámetro	Descripción
`-ConcurrentUserLimit` `<UInt32>`	Especifica el número máximo de usuarios que pueden conectarse al recurso compartido simultáneamente. El valor debe estar comprendido entre 1 y 20.
`-Description` `<String>`	Permite introducir una descripción para el recurso compartido de red. No debe superar los 256 caracteres.
`-Force`	Ejecuta el comando sin pedir confirmación.
`-Name` `<String[]>`	Designa el nombre del recurso o de los recursos compartidos cuya configuración se va a modificar.

Ejemplo: cambiar la descripción del recurso compartido

```
PS D:\> Set-SmbShare -Name ShareFolder -Description "Documentos del
proyecto"
-Force
```

Así, la modificación de la descripción tiene efecto inmediatamente; el parámetro **-Force** se utiliza para omitir la confirmación, que por defecto se pide.



Ventana de uso compartido avanzado con la nueva descripción

Ejemplo: cambiar el límite de conexiones de usuario simultáneas

```
PS D:\> Get-SmbShare -Name ShareFolder | Set-SmbShare -
ConcurrentUserLimit
5 -Force
```

Este segundo ejemplo permite cambiar el límite de conexiones usuario simultáneas sobre el recurso compartido, configurando el valor a 5. Como se muestra, es posible combinar los cmdlets **Get-SmbShare** y **Set-SmbShare**.

4. Eliminar un recurso compartido de red

El último cmdlet dedicado a la administración de recursos compartidos es **Remove-SmbShare**, y permite desactivar un recurso compartido. Este cmdlet se utiliza de una manera muy sencilla, indicando el nombre del recurso compartido que se desea eliminar. He aquí los distintos parámetros disponibles:

Parámetro	Descripción
-Force	Ejecuta el comando sin pedir confirmación.

Parámetro	Descripción
-Name <String[]>	Designa el nombre del recurso o de los recursos compartidos sobre los que se realizará la acción.

Ejemplo: eliminar un recurso compartido

```
PS D:\> Remove-SmbShare -Name Temp$ -Force
```

Como con **Set-SmbShare**, se pide una confirmación por defecto. El parámetro **-Force** permite omitir este paso.

Ejemplo: eliminar todos los recursos compartidos activos en un puesto de trabajo

```
PS D:\> Get-SmbShare -Special:$false | Remove-SmbShare -Force
```

5. Administrar los permisos del recurso compartido SMB

Una vez creado el recurso compartido y activo con el cmdlet **New-SmbShare**, es posible administrar los permisos mediante los cmdlets llamados ***-SmbShareAccess**.

a. Recuperar los permisos del recurso compartido

El cmdlet **Get-SmbShareAccess** permite recuperar el conjunto de permisos configurados en el recurso compartido y visualizar así qué cuentas o grupos poseen permisos de acceso.

Parámetro	Descripción
-Name <String[]>	Designa el nombre del recurso o de los recursos compartidos sobre los que se realizará la acción.

Ejemplo: recuperar los permisos del recurso compartido

```
PS D:\> Get-SmbShareAccess -Name ShareFolder

Name              ScopeName  AccountName          AccessControlType
AccessRight
----              ---------  -----------          -----------------
--
ShareFolder *          CONTOSO\benjamin Deny              Full
ShareFolder *          W10\Admin        Allow             Full
ShareFolder *          CONTOSO\GG DSI   Allow             Read
```

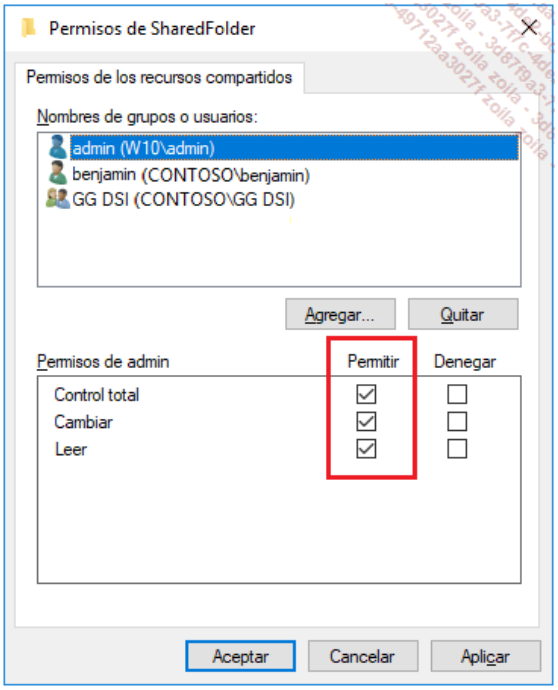
Así, podemos ver que las cuentas Admin y GG DSI poseen permisos de tipo «Permitir», con distintos niveles: control total para la cuenta Admin y de lectura para el grupo GG DSI. La cuenta de Benjamin, en cambio, posee permisos de tipo «Denegar».

b. Agregar o revocar permisos de acceso al recurso compartido

Es posible agregar permisos de acceso al recurso compartido con ayuda del cmdlet **`Grant-SmbShareAccess`**. Este cmdlet permite agregar una o varias cuentas o grupos y configurar el nivel de acceso asociado a cada uno de ellos, y también modificar los permisos de las cuentas ya asignadas. He aquí los parámetros necesarios para el uso de este cmdlet:

Parámetro	Descripción
`-AccessRight` `<ShareAccessRight>`	Expresa el nivel de acceso que se asignará a las cuentas indicadas. Los posibles valores son `Full`, `Change` o `Read`.
`-AccountName` `<String[]>`	Especifica las cuentas o grupos a los que se asignarán los permisos.
`-Force`	Ejecuta el comando sin pedir una confirmación.
`-Name` `<String[]>`	Designa el nombre del recurso o de los recursos compartidos sobre los que se realizará la acción.

`Grant-SmbShareAccess` y **`Revoke-SmbShareAccess`**, que veremos a continuación, permiten administrar los permisos de tipo «Permitir» sobre el recurso compartido.

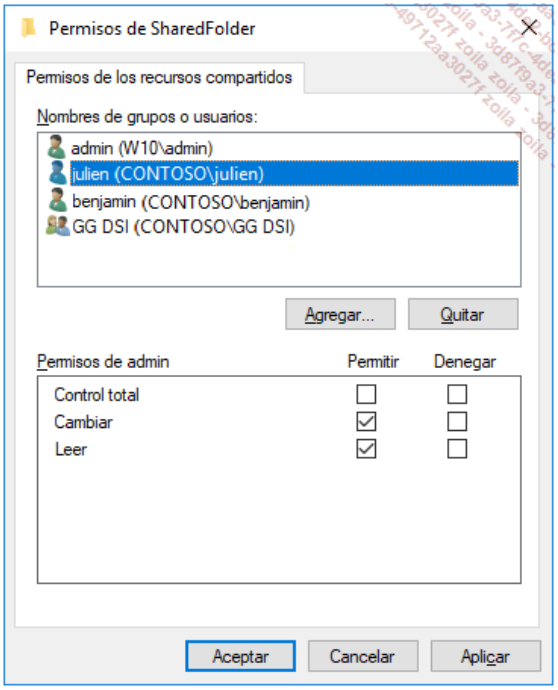


Administrar permisos de tipo «Permitir» sobre el recurso compartido

Ejemplo: configurar permisos para una cuenta sobre un recurso compartido

```
PS  D:\>  Grant-SmbShareAccess  -Name  ShareFolder  -AccountName
CONTOSO\julien
-AccessRight Change -Force

Name              ScopeName  AccountName          AccessControlType
AccessRight
----         --------- -----------      ----------------- ---------
--
ShareFolder *         CONTOSO\benjamin Deny               Full
ShareFolder *         W10\Admin        Allow              Full
ShareFolder *         CONTOSO\GG DSI   Allow              Read
ShareFolder *         CONTOSO\julien   Allow              Change
```

En este ejemplo, se agrega la cuenta de usuario CONTOSO\julien a los permisos del recurso compartido, y se le asigna el nivel de permiso para «Cambiar».



Agregar una nueva cuenta de usuario con un nivel de permiso para Cambiar

Si la cuenta indicada por el comando ya posee permisos, estos se sobrescriben con el nuevo nivel de permisos que se haya especificado.

Ejemplo: configurar permisos para varias cuentas sobre varios recursos compartidos

```
PS D:\> Grant-SmbShareAccess -Name ShareFolder,Temp$ -AccountName

CONTOSO\julien,CONTOSO\octavio -AccessRight Full -Force

Name              ScopeName  AccountName          AccessControlType
AccessRight
----              ---------- -----------      -----------------
--
Temp$        *          Todos             Allow              Read
Temp$        *          CONTOSO\julien    Allow              Full
Temp$        *          CONTOSO\octavio   Allow              Full
ShareFolder  *          CONTOSO\benjamin  Deny               Full
ShareFolder  *          W10\Admin         Allow              Full
ShareFolder  *          CONTOSO\GG DSI    Allow              Read
ShareFolder  *          CONTOSO\julien    Allow              Full
ShareFolder  *          CONTOSO\octavio   Allow              Full
```

Si **Grant-SmbShareAccess** permite agregar o redefinir nuevos permisos, el cmdlet **Revoke-SmbShareAccess** permite revocar todos los permisos de tipo «Permitir» de la cuenta o las cuentas indicadas. He aquí los parámetros para utilizar este cmdlet:

Parámetro	Descripción
-AccountName <String[]>	Especifica las cuentas o grupos sobre los que se revocarán los permisos.
-Force	Ejecuta el comando sin pedir una confirmación previa.
-Name <String[]>	Designa el nombre del recurso o de los recursos compartidos sobre los que se llevará a cabo la acción.

Ejemplo: revocar los permisos de una cuenta sobre un recurso compartido

```
PS D:\> Revoke-SmbShareAccess -Name ShareFolder -AccountName
CONTOSO\julien -Force

Name              ScopeName  AccountName          AccessControlType
AccessRight
----         --------- -----------      ----------------- ---------
--
ShareFolder *         CONTOSO\benjamin Deny              Full
ShareFolder *         W10\Admin        Allow             Full
ShareFolder *         CONTOSO\GG DSI   Allow             Read
ShareFolder *         CONTOSO\octavio  Allow             Full
```

Así, la cuenta CONTOSO\julien ya no está presente en la lista de control de acceso al recurso compartido ShareFolder.

De forma similar al ejemplo mostrado con **Grant-SmbShareAccess**, es posible revocar los permisos de varias cuentas de usuario y grupos sobre varios recursos compartidos con un único comando.

Ejemplo: revocar los permisos sobre un recurso compartido para todas las cuentas de un dominio

```
PS D:\> $ssa = Get-SmbShareAccess -Name ShareFolder | Where-Object

{($_.AccountName).StartsWith("CONTOSO\")}      |      Select-Object
AccountName
PS D:\> Revoke-SmbShareAccess -Name ShareFolder -AccountName
$ssa.AccountName -Force

Name              ScopeName  AccountName          AccessControlType
```

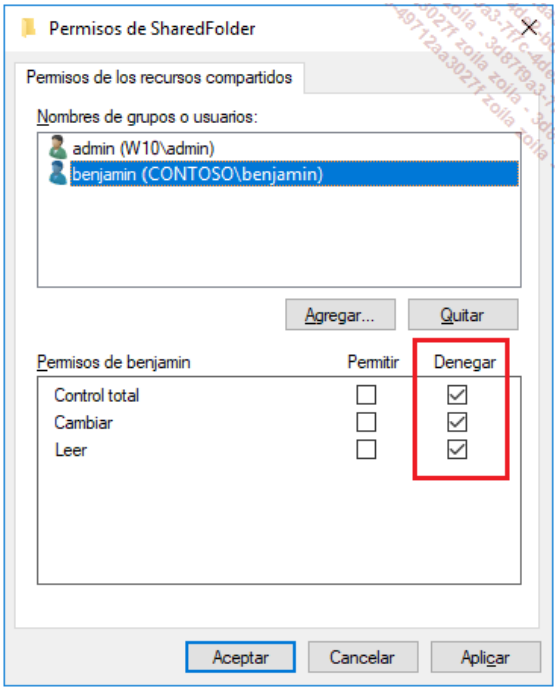
```
AccessRight
----         --------- -----------      ----------------- ---------
--
ShareFolder *         CONTOSO\benjamin Deny               Full
ShareFolder *         W10\Admin        Allow              Full
```

Observe el hecho de que el acceso CONTOSO\benjamin siga estando presente; se debe a que se trata de un permiso de tipo «Denegar». Este tipo de permiso se administra mediante los cmdlets que veremos a continuación.

c. Bloquear o desbloquear permisos de acceso al recurso compartido

Del mismo modo a como hemos visto antes la administración de permisos de acceso de tipo «Permitir» sobre un recurso compartido, he aquí cómo administrar los permisos de acceso de tipo «Denegar». Hay que tener en cuenta que los permisos para este tipo de acceso son prioritarios en caso de producirse algún conflicto con los permisos definidos con «Permitir». Por ello, cuando una cuenta de usuario o grupo poseen un permiso de tipo «Denegar», serán incapaces de acceder al recurso compartido.

Los cmdlets **`Block-SmbShareAccess`** y **`Unblock-SmbShareAccess`** permiten asignar o revocar permisos sobre un recurso compartido de tipo «Denegar».



Administrar permisos de tipo «Denegar» sobre el recurso compartido

Block-SmbShareAccess permite prohibir el acceso a los recursos compartidos para una o varias cuentas. He aquí los parámetros:

Parámetro	Descripción
-AccountName <String[]>	Especifica las cuentas o grupos sobre los que se deniega el acceso al recurso compartido.
-Force	Ejecuta el comando sin pedir una confirmación previa.
-Name <String[]>	Designa el nombre del recurso o de los recursos compartidos sobre los que se llevará a cabo la acción.

Ejemplo: denegar el acceso al recurso compartido para una cuenta de usuario

```
PS D:\> Block-SmbShareAccess -Name ShareFolder -AccountName
CONTOSO\julien
-Force

Name              ScopeName  AccountName          AccessControlType
AccessRight
----              ---------  -----------      -----------------
--
ShareFolder *          CONTOSO\julien   Deny              Full
ShareFolder *          CONTOSO\benjamin Deny              Full
ShareFolder *          W10\Admin        Allow             Full
```

Observe que el nivel asignado es Control total (Full). En efecto, **Block-SmbShareAccess** asigna directamente este nivel de permiso. Sea cual sea el nivel asignado mediante la interfaz gráfica, la cuenta no podrá acceder al recurso compartido.

De manera inversa, para revocar la prohibición de acceso al recurso compartido, se utiliza el cmdlet **Unblock-SmbShareAccess**. La cuenta o las cuentas especificadas se retirarán de la lista de control de acceso del recurso compartido.

He aquí los parámetros para este cmdlet:

Parámetro	Descripción
-AccountName <String[]>	Especifica las cuentas o grupos sobre los que se revocará la prohibición de acceso al recurso compartido.
-Force	Ejecuta el comando sin pedir una confirmación previa.
-Name <String[]>	Designa el nombre del o de los recursos compartidos sobre los que se llevará a cabo la acción.

Ejemplo: revocar la prohibición de acceso al recurso compartido para una cuenta de usuario

```
PS D:\> Unblock-SmbShareAccess -Name ShareFolder -AccountName
CONTOSO\julien
-Force

Name              ScopeName  AccountName           AccessControlType
AccessRight
----         --------- -----------      ----------------- ---------
--
ShareFolder *         CONTOSO\benjamin Deny              Full
ShareFolder *         W10\Admin        Allow             Full
```

Introducción

Como administrador, a menudo tendrá que recopilar información en los puestos de trabajo que ejecutan los scripts. Saber recopilar información, pero sobre todo saber dónde encontrarla, es primordial, pues sirve, por ejemplo, para implementar estructuras condicionales (`If`, `ElseIf`, `Else`) en un script. En efecto, en función de una o varias condiciones, puede hacer que el script no ejecute las mismas líneas de comandos.

La recopilación de información también puede servir para auditar los puestos de trabajo, y así recuperar información sobre los distintos componentes de hardware que los componen. Esto puede permitir, por ejemplo, reparar el conjunto de equipos que posean un componente que el fabricante haya declarado como defectuoso, y llevar a cabo rápidamente su sustitución.

Tras la lectura de este capítulo, sabrá cómo recuperar la mayoría de la información, tanto sobre el hardware como sobre el software. Esto le permitirá realizar correctamente el conjunto de tareas que quiera llevar a cabo en función de la máquina que ejecuta las líneas de comandos PowerShell.

Buscar archivos

Windows integra de manera nativa una herramienta de búsqueda de archivos accesible desde el Explorador de archivos. Con Windows PowerShell, es posible realizar este tipo de búsqueda, únicamente con el cmdlet **`Get-Child Item`**, visto anteriormente. La búsqueda está cada vez más presente en Windows, y la cantidad de datos es cada vez mayor. Conviene saber realizar búsquedas avanzadas con Windows PowerShell.

1. Buscar archivos en función de sus propiedades

Comenzaremos por una búsqueda en función del tipo de archivo, por ejemplo los archivos de datos de Outlook, cuya extensión es .pst. Para buscar la existencia de estos archivos en el conjunto del disco C:, escriba:

Ejemplo:

```
PS C:\Windows\system32> Get-ChildItem C:\ -Filter *.pst -Recurse


    Directory: C:\CopiaDeSeguridad


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         24/03/2014     17:53  190989312 Archive2013.pst


    Directory: C:\Users\Julien\Outlook


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         11/12/2011     13:45 1527792640 Archive2013.pst
-a---         29/01/2012     15:21 1003635712 Outlook.PST
```

De este modo, puede buscar todos los tipos de archivo, desde el punto del árbol seleccionado. En efecto, resulta inútil buscar en una partición completa los datos si uno sabe dónde se encuentran: esto permite ahorrar una cantidad de tiempo considerable.

Por supuesto, es posible realizar una búsqueda de todos los archivos que ocupen más de 100 MB.

Ejemplo:

```
PS C:\> Get-ChildItem -Path . -Recurse | Where-Object {$_.Length
-gt 104857600}


    Directory: C:\Archivos\Windows 7\Service Pack 1


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---          23/11/2010      02:49   947070088 windows6.1-KB976932-
X64.exe


    Directory: C:\Archivos\Windows 7\XPmode


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---           27/03/2011       13:45   492597008  WindowsXPMode_en-
us.exe
[...]
```

El valor expresado por `Length` se define en bytes.

1 024 bytes = 1 KB, 1 024 * 1 024 bytes = 1 MB, 1024 * 1024 * 100 bytes = 100 MB.

También es posible buscar archivos en función de su fecha de última modificación. En el siguiente ejemplo, el resultado devuelve todos los archivos que no se han modificado desde hace más de un año respecto a la fecha actual.

Ejemplo:

```
PS C:\> $date = Get-Date -Year (((Get-Date).Year)-1)
PS   C:\>   Get-ChildItem   -Path   .   -Recurse   |   Where-Object
{$_.LastWriteTime
-lt $date}


    Directory: C:\Archivos\Windows 7\Service Pack 1


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---          23/11/2010      02:49   947070088 windows6.1-KB976932-
X64.exe


    Directory: C:\Archivos\Windows 7\XPmode


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
```

```
-a---           27/03/2011       13:45   492597008 WindowsXPMode_en-
us.exe
[...]
```

2. Utilizar el resultado de la búsqueda

La búsqueda se utiliza generalmente con dos objetivos:

- Bien para encontrar un archivo.
- Bien para realizar una acción sobre un conjunto de archivos.

Este segundo punto es el que vamos a ver a continuación. Puede utilizar directamente la lista de archivos devuelta por **Get-ChildItem**. En efecto, es posible enviar este flujo directamente a otro cmdlet que definirá la acción que desea realizar sobre todos estos archivos.

Por ejemplo, para eliminar el conjunto de archivos temporales presentes en una carpeta y sus subcarpetas, basta con escribir un único comando.

Ejemplo:

```
PS C:\Temp> Get-ChildItem -Path . -Filter *.tmp -Recurse | Remove-
Item
```

Para copiar archivos, escriba:

Ejemplo:

```
PS C:\Carpeta2014> Get-ChildItem -Path . -Recurse | Copy-Item
-Destination \\server01\backup$ -Recurse -Force
```

3. Buscar en el contenido de los archivos

Pero la búsqueda puede ir todavía más lejos, con la búsqueda de información directamente en los archivos de tipo ASCII, es decir, el conjunto de archivos de texto, scripts, diversos archivos de configuración, etc. Esto puede resultar práctico para encontrar un archivo que contenga una cadena de caracteres específica. Para ello, utilice el cmdlet **Select-String**, que realizará la búsqueda de la expresión indicada.

Parámetro	Descripción

Parámetro	Descripción
-CaseSensitive	Búsqueda que hace distinción de mayúsculas y minúsculas. Por defecto, la búsqueda no hace distinción de mayúsculas y minúsculas.
-Path <String[]>	Indica la ruta de acceso a los archivos donde se realizará la búsqueda. Se permite el uso del carácter comodín «*».
-Pattern <String[]>	Especifica el texto que hay que buscar. Puede ser una cadena de caracteres o una expresión regular.

Ejemplo:

```
PS C:\Temp> Select-String -Pattern "Win32_" -Path .\serial.vbs

serial.vbs:8:Set colItems = objWMIService.ExecQuery("Select * from
Win32_BIOS",,48)
serial.vbs:59:Set colItems = objWMIService.ExecQuery("Select * from
Win32_ComputerSystem",,48)
serial.vbs:107:Set IPConfigSet = objWMIService.ExecQuery("Select *
from Win32_NetworkAdapterConfiguration Where IPEnabled=TRUE")
```

El resultado de **Select-String** devuelve, para cada expresión encontrada, el nombre del archivo, el número de la línea y el contenido de la línea correspondiente.

Por defecto, la búsqueda de la expresión que se indica a través del parámetro **-Pattern** no hace distinción de mayúsculas y minúsculas. Para que la búsqueda haga distinción de mayúsculas y minúsculas, es preciso agregar el parámetro **-CaseSensitive**.

Asociando los cmdlets **Get-ChildItem** y **Select-String**, puede realizar una búsqueda más extensa. El siguiente ejemplo escruta el contenido de los archivos con extensiones .txt, .vbs y .ps1 presentes en la carpeta C:\Temp y sus subcarpetas, y comprueba si contienen la palabra «Sanabria».

Ejemplo:

El cmdlet **Get-ChildItem** es más que un simple comando `dir`.

```
PS C:\Temp> Get-ChildItem .\* -Include *.txt,*.vbs,*.ps1 -Recurse
| Select-String -Pattern "Sanabria" -CaseSensitive

PXE.txt:5:# Site Sanabria
SetQuota.ps1:If ($site -eq "Sanabria")
```

Utilizado correctamente, **Get-ChildItem** se convierte en un verdadero motor de búsqueda, tanto en el árbol de archivos como en el registro de Windows. Unido a otros cmdlets, permite realizar alguna acción con una única línea de comando sobre el conjunto de archivos que resultan del comando **Get-ChildItem**.

Comparar objetos

Compare-Object permite comparar dos objetos entre sí: el primero se indica mediante el parámetro **-ReferenceObject**, mientras que el segundo se indica con el parámetro **-DifferenceObject**. De este modo, es posible comparar el contenido de dos archivos de texto, registros, etc. **Compare-Object** está bien adaptado para comparar el contenido de dos elementos que contengan texto, o propiedades de objetos.

1. Comparar dos archivos de texto

El resultado devuelto por **Compare-Object** es un indicador para cada elemento comparado. Así, todo elemento correspondiente a la referencia se indica con el símbolo **<=**, y todo elemento correspondiente a la diferencia, con el símbolo **=>**. Especificando el parámetro **-IncludeEqual**, todo lo que sea común a ambos objetos de referencia y de diferencia se mostrará por pantalla con el indicador **==**.

Ejemplo 1: comparación de dos archivos de texto

```
PS C:\Windows\system32> Compare-Object -ReferenceObject (Get-
Content
C:\Temp\ref.txt) -DifferenceObject (Get-Content C:\Temp\dif.txt)

InputObject
SideIndicator
-----------                                                    -------
------
Este archivo se llama dif.txt, y es el objeto de diferencia. =>
Este archivo se llama ref.txt, y es el objeto de referencia.  <=
```

Ambos archivos solo se diferencian en una línea. Así, el símbolo devuelto por `SideIndicator` indica con claridad qué línea está únicamente presente en función del objeto de referencia o de diferencia.

Si retomamos el ejemplo anterior, agregando el parámetro **-IncludeEqual**, verá que se muestran otras líneas de los archivos de texto:

```
PS C:\Windows\system32> Compare-Object -ReferenceObject (Get-
Content
```

```
C:\Temp\ref.txt) -DifferenceObject (Get-Content C:\Temp\dif.txt)
-IncludeEqual

InputObject
 SideIndicator
-----------                                                    ------
-------
Esto es un archivo de texto.                                   ==
                                                               ==
"Hello World!"                                                 ==
Este archivo se llama dif.txt, y es el objeto de diferencia. =>
Este archivo se llama ref.txt, y es el objeto de referencia. <=
```

Ahora se muestran los elementos comunes a ambos archivos, con el símbolo **==**.

Ejemplo 2: comparación de los servicios de Windows

En este segundo ejemplo, va a comparar la lista completa de servicios de Windows, para diferenciar lo que ha cambiado de un momento a otro.

```
PS       C:\Windows\system32>       Get-Service       |       Select
Status,Name,DisplayName |
Export-Csv C:\Temp\export_ref.csv
PS C:\Windows\system32> Stop-Service Spooler
PS       C:\Windows\system32>       Get-Service       |       Select
Status,Name,DisplayName |
Export-Csv C:\Temp\export_dif.csv
PS   C:\Windows\system32>   Compare-Object   -ReferenceObject   (Get-
Content
C:\Temp\export_ref.csv) -DifferenceObject (Get-Content
C:\Temp\export_dif.csv)

InputObject
SideIndicator
-----------
-------------
"Stopped","Spooler","Administrador de trabajos de impresión"
  =>
"Running","Spooler","Administrador de trabajos de impresión"
  <=
```

El resultado indica la diferencia entre ambos archivos. Puede observar claramente con el indicador **=>** que el servicio Administrador de trabajos de impresión se ha detenido.

Con este método podrá comparar fácilmente archivos de texto (*.txt, *.ini, *.csv, *.xml, *.reg, etc.).

2. Comparar dos archivos de datos

Generalmente, cuando compara archivos de datos, lo hace principalmente basándose en el criterio de la última fecha de

modificación (`LastWriteTime`), es decir, en las propiedades del archivo, y no a partir de su contenido.

Ejemplo: comparación de dos archivos sobre su fecha de modificación

```
PS C:\Windows\system32> $ref = Get-ItemProperty
\\servidor01\d$\miArchivo.exe
PS C:\Windows\system32> $dif = Get-ItemProperty
C:\Temp\miArchivo.exe
PS C:\Windows\system32> $ref.LastWriteTime

lunes 2 de junio de 2014 10:55:00


PS C:\Windows\system32> $dif.LastWriteTime

domingo 1 de junio de 2014 14:24:32


PS C:\Windows\system32> $ref.LastWriteTime -eq $dif.LastWriteTime
False
```

Si parte del principio de que el archivo de referencia es el que existe en el servidor, es necesario realizar una copia para actualizar el archivo presente en el puesto de trabajo.

También puede decir que el archivo más reciente es el correcto: en este caso, basta con comparar las fechas:

```
PS C:\Windows\system32> $ref = Get-ItemProperty
\\servidor01\d$\miArchivo.exe
PS      C:\Windows\system32>      $dif      =      Get-ItemProperty
C:\Contoso\miArchivo.exe
PS C:\Windows\system32> $ref.LastWriteTime

domingo 1 de junio de 2014 10:55:00


PS C:\Windows\system32> $dif.LastWriteTime

domingo 1 de junio de 2014 14:24:32


PS     C:\Windows\system32>     If     ($ref.LastWriteTime     -gt
$dif.LastWriteTime)
>> {
>>                       Copy-Item     \\servidor01\d$\miArchivo.exe
C:\Contoso\miArchivo.exe
-Force
>> }
>> Else
>> {
>>                             Copy-Item     C:\Contoso\miArchivo.exe
 \\servidor01\d$\miArchivo.exe
-Force
>> }
>>
```

El ejemplo anterior compara la última fecha de modificación de ambos archivos. Si el archivo del servidor es más reciente que el existente en el puesto de trabajo, se copiará en el equipo local. Se llevará a cabo la operación inversa si el archivo del puesto de trabajo es más reciente que el del servidor.

Por supuesto, la propiedad `LastWriteTime` no es la única. Existen muchas otras, que puede utilizar para realizar comparaciones o recopilar información para realizar una tarea diferente.

Utilizando el cmdlet **`Get-ItemProperty`**, puede mostrar el conjunto de propiedades que componen un archivo. He aquí un ejemplo para un archivo DLL:

```
PS C:\WINDOWS\system32> Get-ItemProperty . \user32.dll | fl *


PSPath           :
Micro-
soft.PowerShell.Core\FileSystem::C:\Windows\System32\user32.dll
PSParentPath                                                    :
Microsoft.PowerShell.Core\FileSystem::C:\Windows\System32
PSChildName      : user32.dll
PSDrive          : C
PSProvider       : Microsoft.PowerShell.Core\FileSystem
Mode             : -a----
VersionInfo                :   File:
C:\Windows\System32\user32.dll
                   InternalName:     user32
                   OriginalFilename: user32
                                FileVersion:          10.0.14393.0
(rs1_release.160715-1616)
                         FileDescription:  DLL cliente de la API de
usuario
de Windows multi-usuario
                     Product:          Sistema operativo Microsoft®
Windows®
                   ProductVersion:   10.0.14393.0
                   Debug:            False
                   Patched:          False
                   PreRelease:       False
                   PrivateBuild:     False
                   SpecialBuild:     False
                   Language:         Español (España)

BaseName         : user32
Target           : {C:\Windows\WinSxS\amd64_microsoft-windows-
user32_31bf3856ad364e35_10.0.14393.576_none_06f0ff920b5e
                    5489\user32.dll}
LinkType         : HardLink
Name             : user32.dll
Length           : 1461200
DirectoryName    : C:\Windows\System32
Directory        : C:\Windows\System32
IsReadOnly       : False
Exists           : True
```

```
FullName          : C:\Windows\System32\user32.dll
Extension         : .dll
CreationTime      : 27/12/2016 10:55:41
CreationTimeUtc   : 27/12/2016 09:55:41
LastAccessTime    : 27/12/2016 10:55:41
LastAccessTimeUtc: 27/12/2016 09:55:41
LastWriteTime     : 09/12/2016 11:10:58
LastWriteTimeUtc : 09/12/2016 10:10:58
Attributes        : Archivo
```

He aquí una tabla que resume las principales propiedades asociadas a un archivo, con una descripción de cada una de ellas:

Propiedad	Descripción
`VersionInfo`	Proporciona información complementaria correspondiente al propio archivo: versión, descripción, etc.
`BaseName`	Nombre del archivo (sin su extensión).
`Name`	Nombre del archivo (incluyendo su extensión).
`Length`	Tamaño del archivo, expresado en bytes.
`DirectoryName` `Directory`	Ruta de acceso a la carpeta donde se encuentra el archivo.
`IsReadOnly`	Valor booleano que indica si el archivo es de solo lectura o no.
`FullName`	Ruta de acceso al archivo.
`Extension`	Extensión del archivo.
`CreationTime`	Fecha de creación del archivo.
`LastAccessTime`	Fecha de último acceso al archivo.
`LastWriteTime`	Fecha de última modificación.

Así, recuperar el número de versión del archivo DLL es bien sencillo:

```
PS C:\Windows\system32> $file = Get-ItemProperty .\user32.dll
PS C:\WINDOWS\system32> $file.VersionInfo.FileVersion
10.0.14393.0 (rs1_release.160715-1616)
PS C:\WINDOWS\system32> $file.VersionInfo.ProductVersion
10.0.14393.0
```

En cuanto a la propiedad `Length`, puede expresarla fácilmente en megabytes o gigabytes, con la siguiente operación:

```
PS C:\Temp> $file = Get-ItemProperty .\WindowsTechnicalPreview-x64-
EN-US.iso
PS C:\Temp> $file.Length/1MB
3910,5390625
PS C:\Temp> [math]::Round($file.Length/1GB,2)
3,82
```

Todas estas propiedades pueden recuperarse y explotarse en función de nuestros objetivos.

Recuperar la información general del equipo

Es posible recuperar la información general del equipo a través del cmdlet **Get-ComputerInfo**. El conjunto de información devuelto por este cmdlet es una consolidación de diversos elementos que pueden recuperarse a través de consultas WMI: información acerca del hardware (procesador, tarjeta de red, etc.), la BIOS, el sistema operativo, incluyendo su configuración del sistema (nombre, dominio, etc.), y también parámetros regionales (reloj, idioma y región).

Si bien el uso de consultas WMI se explica en la sección Obtener información mediante las clases WMI, **Get-ComputerInfo** permite recuperar rápidamente información genérica relativa al sistema, sin necesidad de poseer conocimientos previos acerca de las clases WMI.

Get-ComputerInfo solo permite mostrar información, de modo que puede utilizarse sin ningún parámetro.

Sin embargo, el cmdlet puede recibir un parámetro para filtrar las propiedades devueltas por el cmdlet.

Parámetro	Descripción
-Property <String[]>	Permite recuperar las propiedades correspondientes a la cadena de caracteres especificada. Se admite el uso del carácter comodín * (*wildcard*).

He aquí un ejemplo completo de la información que se obtiene con **Get-ComputerInfo** en una máquina Hyper-V:

```
PS C:\Windows\system32> Get-ComputerInfo


WindowsBuildLabEx                                           :
14393.693.amd64fre.rs1_release.161220-1747
WindowsCurrentVersion                                       : 6.3
WindowsEditionId                                                    :
Professional
WindowsInstallationType                                     : Client
WindowsInstallDateFromRegistry                                : 16/02/2017
19:34:48
WindowsProductId                                                 : 00331-
```

```
10000-00001-AA339
WindowsProductName                                  : Windows 10
Pro
WindowsRegisteredOrganization                          : Contoso
Corp.
WindowsRegisteredOwner                              : Admin
WindowsSystemRoot                                             :
C:\WINDOWS
BiosCharacteristics                                 : {3, 9, 15,
16...}
BiosBIOSVersion                                      : {VRTUAL -
1, Hyper-V
UEFI Release v1.0, EDK II - 10000}
BiosBuildNumber                                     :
BiosCaption                                            : Hyper-V
UEFI Release v1.0
BiosCodeSet                                         :
BiosCurrentLanguage                                 :
BiosDescription                                        : Hyper-V
UEFI Release v1.0
BiosEmbeddedControllerMajorVersion                  : 255
BiosEmbeddedControllerMinorVersion                  : 255
BiosFirmwareType                                    : Uefi
BiosIdentificationCode                              :
BiosInstallableLanguages                            :
BiosInstallDate                                     :
BiosLanguageEdition                                 :
BiosListOfLanguages                                 :
BiosManufacturer                                       : Microsoft
Corporation
BiosName                                               : Hyper-V
UEFI Release v1.0
BiosOtherTargetOS                                   :
BiosPrimaryBIOS                                     : True
BiosReleaseDate                                     : 26/11/2012
01:00:00
BiosSerialNumber                                    : 8848-2762-
6391-8131-
0233-4446-53
BiosSMBIOSBIOSVersion                                  : Hyper-V
UEFI Release v1.0
BiosSMBIOSMajorVersion                              : 2
BiosSMBIOSMinorVersion                              : 4
BiosSMBIOSPresent                                   : True
BiosSoftwareElementState                            : Running
BiosStatus                                          : OK
BiosSystemBiosMajorVersion                          : 1
BiosSystemBiosMinorVersion                          : 0
BiosTargetOperatingSystem                           : 0
BiosVersion                                            : VRTUAL -
1
CsAdminPasswordStatus                               : Unknown
CsAutomaticManagedPagefile                          : True
CsAutomaticResetBootOption                          : True
CsAutomaticResetCapability                          : True
CsBootOptionOnLimit                                 :
CsBootOptionOnWatchDog                              :
CsBootROMSupported                                  : True
CsBootStatus                                           : {0, 0, 0,
127...}
CsBootupState                                             : Normal
boot
CsCaption                                           : LAB-WS01
```

```
CsChassisBootupState                                    : Safe
CsChassisSKUNumber                                      :
CsCurrentTimeZone                                       : 60
CsDaylightInEffect                                      : False
CsDescription                                                : AT/AT
COMPATIBLE
CsDNSHostName                                           : LAB-WS01
CsDomain                                                                :
contoso.local
CsDomainRole                                                            :
MemberWorkstation
CsEnableDaylightSavingsTime                             : True
CsFrontPanelResetStatus                                 : Unknown
CsHypervisorPresent                                     : True
CsInfraredSupported                                     : False
CsInitialLoadInfo                                       :
CsInstallDate                                           :
CsKeyboardPasswordStatus                                : Unknown
CsLastLoadInfo                                          :
CsManufacturer                                             : Microsoft
Corporation
CsModel                                                       : Virtual
Machine
CsName                                                  : LAB-WS01
CsNetworkAdapters                                          : {Ethernet
2, Ethernet}
CsNetworkServerModeEnabled                              : True
CsNumberOfLogicalProcessors                             : 2
CsNumberOfProcessors                                    : 1
CsProcessors                                               : {Intel(R)
Core(TM)
i5-4300M CPU @ 2.60GHz}
CsOEMStringArray                                                        :
{[MS_VM_CERT/SHA1/
9b80ca0d5dd061ec9da4e494f4c3fd1196270c22],
00000000000000000000000000000000, To be filled by OEM}
CsPartOfDomain                                          : True
CsPauseAfterReset                                       : -1
CsPCSystemType                                          : Desktop
CsPCSystemTypeEx                                        : Desktop
CsPowerManagementCapabilities                           :
CsPowerManagementSupported                              :
CsPowerOnPasswordStatus                                 : Unknown
CsPowerState                                            : Unknown
CsPowerSupplyState                                      : Safe
CsPrimaryOwnerContact                                   :
CsPrimaryOwnerName                                      : Admin
CsResetCapability                                       : Other
CsResetCount                                            : -1
CsResetLimit                                            : -1
CsRoles                                                                 :
{LM_Workstation, LM_Server, NT}
CsStatus                                                : OK
CsSupportContactDescription                             :
CsSystemFamily                                                : Virtual
Machine
CsSystemSKUNumber                                       : None
CsSystemType                                               : x64-based
PC
CsThermalState                                          : Safe
CsTotalPhysicalMemory                                                   :
2144366592
CsPhyicallyInstalledMemory                              : 2097152
```

```
CsUserName                                                          :
CONTOSO\Julien
CsWakeUpType                                                        :
PowerSwitch
CsWorkgroup                                                 :
OsName                                                        : Microsoft
Windows 10

 Profesional
OsType                                                      : WINNT
OsOperatingSystemSKU                                        : 48
OsVersion                                                           :
10.0.14393
OsCSDVersion                                                :
OsBuildNumber                                               : 14393
OsHotFixes                                                   : {KB3194623,
KB3199986,
                                                               KB3202790,
KB3211320...}
OsBootDevice                                                        :
\Device\HarddiskVolume2
OsSystemDevice                                                      :
\Device\HarddiskVolume4
OsSystemDirectory                                                   :
C:\WINDOWS\system32
OsSystemDrive                                               : C:
OsWindowsDirectory                                                  :
C:\WINDOWS
OsCountryCode                                               : 33
OsCurrentTimeZone                                           : 60
OsLocaleID                                                  : 040c
OsLocale                                                    : es-ES
OsLocalDateTime                                               : 23/02/2017
17:18:49
OsLastBootUpTime                                              : 23/02/2017
06:01:55
OsUptime                                                            :
11:16:53.0317714
OsBuildType                                                         :
Multiprocessor Free
OsCodeSet                                                   : 1252
OsDataExecutionPreventionAvailable                          : True
OsDataExecutionPrevention32BitApplications                  : True
OsDataExecutionPreventionDrivers                            : True
OsDataExecutionPreventionSupportPolicy                      : OptIn
OsDebug                                                     : False
OsDistributed                                               : False
OsEncryptionLevel                                           : 256
OsForegroundApplicationBoost                                : Maximum
OsTotalVisibleMemorySize                                    : 2094108
OsFreePhysicalMemory                                        : 1007756
OsTotalVirtualMemorySize                                    : 2749468
OsFreeVirtualMemory                                         : 1430844
OsInUseVirtualMemory                                        : 1318624
OsTotalSwapSpaceSize                                        :
OsSizeStoredInPagingFiles                                   : 655360
OsFreeSpaceInPagingFiles                                    : 615688
OsPagingFiles                                                       :
{C:\pagefile.sys}
OsHardwareAbstractionLayer                                          :
10.0.14393.206
OsInstallDate                                                 : 16/02/2017
20:34:48
```

```
OsManufacturer                                          : Microsoft
Corporation
OsMaxNumberOfProcesses                                             :
4294967295
OsMaxProcessMemorySize                                             :
137438953344
OsMuiLanguages                                        : {es-ES, en-
US}
OsNumberOfLicensedUsers                               : 0
OsNumberOfProcesses                                   : 56
OsNumberOfUsers                                       : 7
OsOrganization                                            : Contoso
Corp.
OsArchitecture                                        : 64 bits
OsLanguage                                            : es-ES
OsProductSuites                                                    :
{TerminalServicesSingleSession}
OsOtherTypeDescription                                :
OsPAEEnabled                                          :
OsPortableOperatingSystem                             : False
OsPrimary                                             : True
OsProductType                                                      :
WorkStation
OsRegisteredUser                                      : Admin
OsSerialNumber                                            : 00331-
10000-00001-AA339
OsServicePackMajorVersion                             : 0
OsServicePackMinorVersion                             : 0
OsStatus                                              : OK
OsSuites                                                           :
{TerminalServices,
                                                          TerminalSe
rvicesSingleSession}
OsServerLevel                                         :
KeyboardLayout                                        : es-ES
TimeZone                                               : (UTC+01:00)
Bruselas, Copenhague,
Madrid, París
LogonServer                                             : \\CONTOSO-
DC01
PowerPlatformRole                                     : Desktop
HyperVisorPresent                                     : True
HyperVRequirementDataExecutionPreventionAvailable     :
HyperVRequirementSecondLevelAddressTranslation        :
HyperVRequirementVirtualizationFirmwareEnabled        :
HyperVRequirementVMMonitorModeExtensions              :
DeviceGuardSmartStatus                                : Off
DeviceGuardRequiredSecurityProperties                 :
DeviceGuardAvailableSecurityProperties                :
DeviceGuardSecurityServicesConfigured                 :
DeviceGuardSecurityServicesRunning                    :
DeviceGuardCodeIntegrityPolicyEnforcementStatus       :
DeviceGuardUserModeCodeIntegrityPolicyEnforcementStatus:
```

Para conocer con más detalle a qué se corresponden los elementos devueltos, he aquí un resumen de las propiedades más importantes:

- `WindowsCurrentVersion`: versión del sistema operativo.
- `WindowsEditionId`: edición de Windows instalada.

- `WindowsInstallDateFromRegistry`: fecha de instalación de Windows en el equipo.
- `WindowsProductName`: nombre del sistema operativo. Esta información se obtiene del valor del registro `ProductName` presente en HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion.
- `WindowsRegisteredOrganization`: nombre de la empresa a la que pertenece el equipo. Esta información se obtiene del valor del registro `RegisteredOrganization` presente en HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion.
- `WindowsRegisteredOwner`: nombre del usuario del equipo. Esta información se obtiene del valor del registro `RegisteredOwner` presente en HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion.
- `WindowsSystemRoot`: se corresponde con la carpeta del sistema donde se encuentra instalado Windows.
- `BiosFirmwareType`: tipo de firmware ejecutado durante el arranque del equipo, `BIOS` o `UEFI`.
- `BiosManufacturer`: fabricante y desarrollador de la BIOS.
- `BiosName`: nombre de la BIOS.
- `BiosReleaseDate`: fecha de publicación de la BIOS.
- `BiosSerialNumber`: número de serie registrado en la BIOS.
- `BiosSMBIOSBIOSVersion`: versión de la BIOS.
- `CsDNSHostName`: nombre DNS del host.
- `CsDomain`: nombre del dominio o del grupo de trabajo al que se encuentra asociado el equipo.
- `CsDomainRole`: indica si el equipo está integrado en un dominio (`Member Workstation`) o en un grupo de trabajo (`StandaloneWorkstation`).
- `CsManufacturer`: nombre del fabricante del equipo.
- `CsModel`: nombre del modelo del equipo.
- `CsName`: nombre del equipo.
- `CsNetworkAdapters`: enumera los nombres de las interfaces de red.

- `CsNumberOfLogicalProcessors`: número de procesadores lógicos.
- `CsNumberOfProcessors`: número de procesadores (sockets).
- `CsProcessors`: nombre del procesador presente en la máquina.
- `CsPartOfDomain`: indica si el equipo está integrado en un dominio o no.
- `CsPCSystemType`: indica si es un equipo fijo (`Desktop`) o portátil (`Mobile`).
- `CsSystemType`: tipo de arquitectura del sistema operativo, 32 o 64 bits.
- `CsPhysicallyInstalledMemory`: tamaño de la memoria instalada en la placa base (expresado en kilobytes).
- `CsUserName`: cuenta autenticada en la máquina.
- `CsWorkgroup`: nombre del grupo de trabajo en el que se encuentra el equipo.
- `OsName`: nombre del sistema operativo instalado.
- `OsVersion`: versión del sistema operativo.
- `OsBuildNumber`: versión de la build.
- `OsHotFixes`: enumera el conjunto de actualizaciones (KB) instaladas.
- `OsBootDevice`: indica la ubicación (disco/partición) de los archivos de arranque.
- `OsSystemDevice`: indica la ubicación (disco/partición) de los archivos de sistema.
- `OsSystemDirectory`: indica la carpeta de sistema del sistema operativo.
- `OsSystemDrive`: indica la unidad donde se encuentra instalado el sistema operativo.
- `OsWindowsDirectory`: indica la carpeta de Windows.
- `OsCountryCode`: indicador del país seleccionado en los formatos de fecha, de hora o de los números.
- `OsCurrentTimeZone`: valor correspondiente a la zona horaria configurada en Windows. Para conocer la correspondencia, consulte la lista que encontrará en la

siguiente página: https://msdn.microsoft.com/en-us/library/ms912391

- `OsLocaleID`: identificador de la configuración regional (en hexadecimal), correspondiente al país seleccionado en los formatos de fecha, de hora o de los números.
- `OsLocale`: identificador de la configuración regional, correspondiente al país seleccionado en los formatos de fecha, de hora o de los números.
- `OsLocalDateTime`: fecha y hora local.
- `OsLastBootUpTime`: fecha y hora de inicio del sistema operativo.
- `OsUptime`: tiempo de funcionamiento desde el arranque del sistema operativo.
- `OsTotalVisibleMemorySize`: tamaño de la memoria física visible por el sistema operativo, expresado en kilobytes.
- `OsFreePhysicalMemory`: memoria física disponible para el sistema operativo, expresada en kilobytes.
- `OsTotalVirtualMemorySize`: tamaño de la memoria virtual total (memoria física + archivo de paginación), expresado en kilobytes.
- `OsFreeVirtualMemory`: memoria virtual disponible, expresada en kilobytes.
- `OsInUseVirtualMemory`: memoria virtual utilizada, expresada en kilobytes.
- `OsSizeStoredInPagingFiles`: tamaño del archivo de paginación, expresado en kilobytes.
- `OsFreeSpaceInPagingFiles`: tamaño disponible en el archivo de paginación, expresado en kilobytes.
- `OsPagingFiles`: ubicación del archivo de paginación.
- `OsHardwareAbstractionLayer`: versión de la capa de abstracción de hardware (HAL) - aplicación intermedia entre el sistema operativo y los componentes de hardware del equipo.
- `OsInstallDate`: fecha y hora de instalación del sistema operativo.
- `OsManufacturer`: nombre del fabricante del sistema operativo.
- `OsMuiLanguages`: lista de paquetes de idioma instalados.

- `OsNumberOfProcesses`: número de procesos en ejecución.
- `OsNumberOfUsers`: número de sesiones de usuario para las que el sistema operativo guarda actualmente información de estado.
- `OsOrganization`: nombre de la empresa a la que pertenece el equipo. Esta información se obtiene del valor del registro `RegisteredOrganization` presente en HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion.
- `OsArchitecture`: tipo de arquitectura del sistema operativo.
- `OsLanguage`: nombre del idioma del sistema operativo (tras su instalación).
- `OsRegisteredUser`: nombre del usuario del equipo. Esta información se obtiene del valor del registro `RegisteredOwner` presente en HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion.
- `KeyboardLayout`: idioma del dispositivo de entrada (teclado). El nombre que se muestra se corresponde con el primer método de introducción del idioma activo en sus preferencias de idioma.
- `TimeZone`: zona horaria configurada.
- `LogonServer`: nombre del controlador de dominio para la autenticación.
- `PowerPlatformRole`: indica si es un equipo fijo (`Desktop`) o portátil (`Mobile`).
- `HyperVisorPresent`: indica si existe un hipervisor (`True`) o no (`False`).

Las variables de entorno

Como ha visto en el capítulo Las unidades de Windows PowerShell, existe una unidad que permite leer las variables de entorno (Env:). Contiene las variables de entorno propias del sistema operativo.

Estas variables dinámicas y globales al sistema permiten pasar información entre los distintos procesos ejecutados por Windows. Proporcionan información en función de la configuración de Windows en el puesto de trabajo.

Tomemos un ejemplo sencillo, con la variable **$env:USERPROFILE**: esta variable indica la ruta de acceso al perfil del usuario que tiene la sesión abierta. Si se autentica con otra cuenta de usuario, la ruta de acceso es diferente.

Así, para un script PowerShell que pueda ejecutarse sobre puestos de trabajo con configuraciones completamente diferentes, conviene utilizar siempre que sea posible las variables de entorno.

Estas variables se definen automáticamente cuando se inicia una sesión de PowerShell. Para ver el conjunto de variables de entorno y obtener un resumen, puede ejecutar el siguiente comando:

```
PS C:\Windows\system32> Get-ChildItem Env:

Name                           Value
----                           -----
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\Admin\AppData\Roaming
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
CommonProgramW6432             C:\Program Files\Common Files
COMPUTERNAME                   LAB-W10
ComSpec                        C:\WINDOWS\system32\cmd.exe
HOMEDRIVE                      C:
HOMEPATH                       \Users\Julien
LOCALAPPDATA                   C:\Users\Julien\AppData\Local
LOGONSERVER                    \\FABRIKAM-DC01
NUMBER_OF_PROCESSORS           4
OneDrive                       C:\Users\Julien\OneDrive
OS                             Windows_NT
Path                           C:\Windows\system32;C:\Windows;
                               C:\Windows\System32\Wbem;
                               C:\Windows\System32\WindowsPo...
PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;
                               .JSE;.WSF;.WSH;.MSC;.CPL
PROCESSOR_ARCHITECTURE         AMD64
PROCESSOR_IDENTIFIER              Intel64 Family 6 Model 60 Stepping
3,
                               GenuineIntel
```

```
PROCESSOR_LEVEL                 6
PROCESSOR_REVISION              3c03
ProgramData                     C:\ProgramData
ProgramFiles                    C:\Program Files
ProgramFiles(x86)               C:\Program Files (x86)
ProgramW6432                    C:\Program Files
PSModulePath                    C:\Users\Julien\Documents\
                                WindowsPowerShell\Modules;

 C:\ProgramFiles\WindowsPowerShell...
PUBLIC                          C:\Users\Public
SystemDrive                     C:
SystemRoot                      C:\WINDOWS
TEMP                            C:\Users\Julien\AppData\Local\Temp
TMP                             C:\Users\Julien\AppData\Local\Temp
USERDNSDOMAIN                   FABRIKAM.LOCAL
USERDOMAIN                      FABRIKAM
USERDOMAIN_ROAMINGPROFILE       FABRIKAM
USERNAME                        Julien
USERPROFILE                     C:\Users\Julien
windir                          C:\WINDOWS
```

La siguiente tabla resume las variables de entorno más importantes:

Nombre	Descripción
`$env:ProgramFiles`	Permite recuperar la ruta de acceso donde se instalan las aplicaciones. En una versión de 64 bits de Windows, esta variable apunta a la carpeta de instalación de aplicaciones de 64 bits.
`$env:ProgramFiles(x86)`	Permite recuperar la ruta de acceso donde se instalan las aplicaciones de 32 bits.
`$env:SystemDrive`	Indica la raíz de la unidad donde se ha instalado el sistema operativo Windows.
`$env:SystemRoot` `$env:windir`	Ruta de acceso a la carpeta del sistema operativo.

Nombre	Descripción
`$env:USERNAME`	Indica el usuario en curso autenticado en el puesto de trabajo. Puede tratarse tanto de un usuario local como de un usuario de dominio.
`$env:USERPROFILE`	Indica la ruta de acceso al perfil del usuario autenticado actualmente en el puesto de trabajo. En esta carpeta se almacena toda la información relativa a la cuenta.
`$env:OneDrive`	Indica la ruta de acceso a la carpeta OneDrive del usuario autenticado actualmente.
`$env:Path`	Rutas de acceso separadas por comas. Se trata de las rutas donde se buscan programas y DLL.
`$env:COMPUTERNAME`	Indica el nombre del equipo.
`$env:USERDNSDOMAIN`	Indica el nombre DNS del dominio al que está unido el puesto de trabajo.
`$env:USERDOMAIN`	Igual que la variable anterior, pero con un nombre corto.

He aquí un ejemplo sencillo que muestra lo prácticas que resultan las variables de entorno. La ruta de acceso al perfil del usuario es diferente en función del usuario conectado. En este caso, se recomienda encarecidamente utilizar la variable de entorno `$env:USERPROFILE`, por no decir que resulta obligatorio.

Ejemplo: copiar un archivo en el Escritorio del usuario

```
PS C:\Windows\system32> Copy-Item \\servidor1\d$\GuiaBienvenida.pdf
$env:USERPROFILE\Desktop
```

Esta línea de comando funciona en todas las versiones de Windows, sea cual sea el idioma del sistema operativo instalado y

el usuario autenticado en la máquina.

En el caso de un script que se desplegará en los puestos de trabajo, a través de una GPO, por ejemplo, el uso de las variables de entorno es obligatorio y aporta una gran flexibilidad en el desarrollo de scripts.

Obtener información gracias a las clases WMI

Los objetos WMI permiten buscar información. Información que puede estar vinculada con la máquina (BIOS, configuración del hardware, etc.) y con Windows (configuración, aplicaciones instaladas, etc.), todo ello accesible a través de un único cmdlet: **`Get-WmiObject`**.

WMI (*Windows Management Instrumentation*) es una API (*Application Programming Interface*) creada en 1998 por Microsoft. Su objetivo es simplificar la administración en un entorno distribuido, para facilitar el intercambio de información en un entorno compuesto de equipos no homogéneos.

La base WMI está compuesta de clases, que representan una familia de elementos. Cada una de estas clases contiene información que es fácil de explotar con Windows PowerShell. También es posible interactuar con el sistema operativo, el hardware y las aplicaciones a través de WMI a través de métodos, aunque este no es el objetivo de este capítulo. Veremos un conjunto de clases y sus propiedades para recuperar información y, a continuación, explotar estos datos en función de sus objetivos.

La lista de clases es extensa, muy extensa; para hacerse una idea, puede ejecutar el siguiente comando:

```
PS C:\Windows\system32> Get-WmiObject -List


   NameSpace: ROOT\cimv2

Name                                Methods          Properties
----                                -------          ----------
CIM_Indication                      {}                {CorrelatedIndic
CIM_ClassIndication                 {}                {ClassDefinition
CIM_ClassDeletion                   {}                {ClassDefinition
CIM_ClassCreation                   {}                {ClassDefinition
[...]
```

¡Y se trata únicamente de las clases presentes en el espacio de nombres por defecto, es decir, ROOT\cimv2! En efecto, viendo el enorme número de clases, y en un intento de organización, estas se agrupan en distintos espacios de nombres. La mayoría de las clases, que representan los recursos de un equipo y de su sistema operativo, están incluidas en el espacio de nombres ROOT\cimv2.

Si recuperar información como el fabricante de la máquina, el número de serie, la versión de la BIOS y otros muchos datos todavía le parece complicado, debe saber que nunca ha sido tan sencillo con Windows PowerShell y su cmdlet **Get-WmiObject**.

1. El cmdlet Get-WmiObject

Antes de entrar de lleno en el asunto, es importante describir el rol del cmdlet **Get-WmiObject**. Permite intercambiar información con la capa WMI, bien desde el sistema local o desde un sistema remoto.

He aquí una presentación de los parámetros más comunes de **Get-WmiObject**:

Parámetro	Descripción
-Class <String>	Define el nombre de la clase de la que se recupera la información.
-ComputerName <String[]>	Define el nombre del equipo sobre el que se aplica la consulta WMI. Por defecto, se trata del equipo local.
-Credential <PSCredential>	Contiene la información de autenticación, permitiendo ejecutar el comando con una cuenta diferente.
-Filter <String>	Especifica una cláusula Where que se utiliza como filtro. Utiliza la sintaxis del lenguaje de consultas WMI (WQL).
-NameSpace <String>	Define el espacio de nombres en el que se encuentra la clase. Por defecto, se trata de ROOT\cimv2.

Parámetro	Descripción
-List	Muestra el conjunto de clases presentes en el espacio de nombres indicado (**-NameSpace**). Si se omite, se utiliza el valor por defecto.
-Property <String[]>	Define el nombre de la propiedad que se va a recuperar.

2. Explotar algunas clases WMI

Las clases WMI son muy numerosas, de modo que es imposible abordarlas todas en este libro. Sin embargo, algunas son mucho más importantes que otras, y resulta fundamental conocerlas y saber qué tipo de información es posible obtener.

Cada clase posee propiedades, y es a partir de estos elementos como se puede recuperar información para explotarla a continuación. Pueden existir varias decenas de propiedades para una única clase; es inútil enumerarlas todas.

Para facilitar la lectura en las siguientes secciones, los resultados de las consultas WMI están filtrados mediante el cmdlet **Format-List**, seguido de las propiedades más interesantes y de una explicación para cada una de ellas.

Debe saber que siempre tiene la posibilidad de enumerar el conjunto de propiedades asociadas a una clase WMI pasando el resultado (mediante el operador pipe) de la consulta WMI al cmdlet **Format-List -Property ***:

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_Bios |
Format-List *


PSComputerName       : W10
Status               : OK
Name                 : Default System BIOS
Caption              : Default System BIOS
SMBIOSPresent        : True
__GENUS              : 2
__CLASS              : Win32_BIOS
__SUPERCLASS         : CIM_BIOSElement
__DYNASTY            : CIM_ManagedSystemElement
[...]
```

Si conoce la propiedad que le interesa, puede utilizar **Get-WmiObject** de la siguiente manera:

```
(Get-WmiObject nombre_de_la_clase).nombre_de_la_propiedad
```

Ejemplo: recuperar el modelo del equipo + estructura condicional

```
PS C:\Windows\system32> $model  = (Get-WmiObject
Win32_ComputerSystem).Model
PS C:\Windows\system32> If ($model -eq "Latitude E6410")
>> {
>>     # Código a ejecutar si la condición es verdadera
>> }
>> # Poner otras condiciones si hace falta (ElseIf, Else)
```

a. Información vinculada a la BIOS

Veremos en primer lugar la información que puede recuperar a nivel de la BIOS de la máquina. Para ello, se utiliza la clase **Win32_Bios**:

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_BIOS |
Format-List
Manufacturer,Name,BIOSVersion,Version,ListOfLanguages,
PrimaryBIOS,ReleaseDate,SerialNumber,SMBIOSBIOSVersion,
SMBIOSMajorVersion,SMBIOSMinorVersion


Manufacturer      : Dell Inc.
Name              : BIOS Date: 05/04/12 18:50:20 Ver: A01.00
BIOSVersion       : {DELL   - 1072009, BIOS Date: 05/04/12
                     18:50:20 Ver: A01.00 }
Version           : DELL   - 1072009
ListOfLanguages   : {en|US|iso8859-1}
PrimaryBIOS       : True
ReleaseDate       : 20120504000000.000000+000
SerialNumber      : 4VKXC5J
SMBIOSBIOSVersion : A01
SMBIOSMajorVersion: 2
SMBIOSMinorVersion: 7
```

He aquí un resumen de las propiedades devueltas por la consulta WMI:

- `Manufacturer`: nombre del fabricante de este elemento de software.
- `Name`: nombre utilizado para identificar este elemento de software.
- `BIOSVersion`: información complementaria acerca de la versión de la BIOS.
- `Version`: versión de la BIOS.

- `ListOfLanguages`: enumera los idiomas disponibles para la interfaz de la BIOS.
- `PrimaryBIOS`: si vale `True`, se trata de la BIOS primaria del equipo.
- `ReleaseDate`: fecha de construcción de la BIOS (en formato UTC).
- `SerialNumber`: número de serie asignado a este elemento de software.
- `SMBIOSBIOSVersion`: versión de la BIOS reportada por SMBIOS (*System Management BIOS*).
- `SMBIOSMajorVersion`: número de versión mayor de la SMBIOS.
- `SMBIOSMinorVersion`: número de versión menor de la SMBIOS.

He aquí otro ejemplo; esta vez se trata de una máquina virtual que se ejecuta sobre Hyper-V:

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_BIOS |
Format-List
Manufacturer,Name,BIOSVersion,Version,ListOfLanguages,PrimaryBIOS,

ReleaseDate,SerialNumber,SMBIOSBIOSVersion,SMBIOSMajorVersion,
SMBIOSMinorVersion


Manufacturer      : American Megatrends Inc.
Name              : BIOS Date: 05/23/12 17:15:53  Ver: 09.00.06
BIOSVersion       : {VRTUAL - 5001223, BIOS Date: 05/23/12
17:15:53  Ver: 09.00.06, BIOS Date: 05/23/12 17:15:53
                     Ver: 09.00.06}
Version           : VRTUAL - 5001223
ListOfLanguages   : {enUS}
PrimaryBIOS       : True
ReleaseDate       : 20120523000000.000000+000
SerialNumber      : 9714-9231-9339-3295-6217-4089-24
SMBIOSBIOSVersion : 090006
SMBIOSMajorVersion: 2
SMBIOSMinorVersion: 3
```

b. Información vinculada al equipo que trabaja con Windows

En esta ocasión, recuperaremos información correspondiente al equipo que trabaja con Windows. La clase utilizada es **`Win32_ComputerSystem`**:

He aquí un resumen de las propiedades devueltas por la consulta WMI:

```
PS C:\Windows\system32> Get-WmiObject Win32_ComputerSystem |
Format-List Domain,PartOfDomain,Manufacturer,Model,Name,
SystemType,PrimaryOwnerName,UserName,NumberOfProcessors,
NumberOfLogicalProcessors,TotalPhysicalMemory


Domain                     : CONTOSO.LOCAL
PartOfDomain               : True
Manufacturer               : Dell Inc.
Model                      : OptiPlex 7010
Name                       : W10
SystemType                 : x64-based PC
PrimaryOwnerName           : OptiPlex 7010
UserName                   : W10\Admin
NumberOfProcessors         : 1
NumberOfLogicalProcessors: 4
TotalPhysicalMemory        : 8466911232
```

- `Domain`: nombre del dominio al que pertenece el equipo. Si el equipo no forma parte de un dominio, se obtiene el nombre del grupo de trabajo.
- `PartOfDomain`: si vale `True`, el equipo es miembro de un dominio. Si vale `False`, el equipo no forma parte de un dominio.
- `Manufacturer`: nombre del fabricante del equipo (placa base).
- `Model`: nombre del producto asignado por el fabricante.
- `Name`: nombre del equipo asignado por Windows, permite identificarlo en la red.
- `SystemType`: tipo de sistema operativo. En el caso de un sistema Windows de 64 bits, el valor es «x64-based PC». Si el sistema operativo es de 32 bits, el valor es «x86-based PC».
- `PrimaryOwnerName`: nombre del propietario principal del sistema.
- `UserName`: nombre del usuario autenticado actualmente (sesión abierta).
- `NumberOfProcessors`: número de procesadores físicos (sockets) disponibles en el sistema.
- `NumberOfLogicalProcessors`: número de procesadores lógicos disponibles en el sistema.
- `TotalPhysicalMemory`: tamaño total de la memoria física, en bytes. Tenga precaución: el resultado devuelto es el valor total tras restar los espacios de memoria reservados

por los distintos componentes de hardware que componen el equipo (por ejemplo: la tarjeta gráfica integrada, etc.).

He aquí el resultado devuelto en una máquina virtual Hyper-V:

```
PS C:\Windows\system32> Get-WmiObject Win32_ComputerSystem
| Format-List Domain,PartOfDomain,Manufacturer,Model,Name,
SystemType,PrimaryOwnerName,UserName,NumberOfProcessors,
NumberOfLogicalProcessors,TotalPhysicalMemory


Domain                    : WORKGROUP
PartOfDomain              : False
Manufacturer              : Microsoft Corporation
Model                     : Virtual Machine
Name                      : W10-VM
SystemType                : x64-based PC
PrimaryOwnerName          : Admin
UserName                  : W10-VM\Admin
NumberOfProcessors        : 1
NumberOfLogicalProcessors: 4
TotalPhysicalMemory       : 2147012608
```

c. Información vinculada al equipo (producto)

La clase **`Win32_ComputerSystemProduct`** permite recuperar información vinculada al producto equipo. He aquí un ejemplo, siempre con **`Get-WmiObject`**:

```
PS C:\Windows\system32> Get-WmiObject Win32_ComputerSystemProduct
| Format-List IdentifyingNumber,Name,Vendor,Version,Caption,UUID


IdentifyingNumber: 4VKXC5J
Name              : OptiPlex 7010
Vendor            : Dell Inc.
Version           : 01
Caption           : Computer System Product
UUID              : 4C4C4544-0056-4B10-8058-B4C04F43354A
```

Las propiedades se definen así:

- `IdentifyingNumber`: número de identificación del producto.
- `Name`: nombre común del producto.
- `Vendor`: nombre del fabricante del producto.
- `Version`: número de versión del producto.
- `Caption`: descripción corta del producto.
- `UUID`: identificador único universal (*Universal Unique IDentifier* - UUID) para este producto. Codificado en 128 bits,

este identificador es único en el mundo.

d. Información vinculada al procesador (CPU)

Veamos el procesador (CPU) con el que se equipa el puesto de trabajo. Para ello, se utiliza la clase **`Win32_Processor`**:

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_Processor |
Format-List Caption,DeviceID,Manufacturer,MaxClockSpeed,Name,
CurrentClockSpeed,AddressWidth,DataWidth,L2CacheSize,L3CacheSize,
NumberOfCores,NumberOfLogicalProcessors,Status


Caption                   : Intel64 Family 6 Model 58 Stepping 9
DeviceID                  : CPU0
Manufacturer              : GenuineIntel
MaxClockSpeed             : 3301
Name                            : Intel(R) Core(TM) i5-3550 CPU @
3.30GHz
CurrentClockSpeed         : 3301
AddressWidth              : 64
DataWidth                 : 64
L2CacheSize               : 1024
L3CacheSize               : 6144
NumberOfCores             : 4
NumberOfLogicalProcessors: 4
Status                    : OK
```

He aquí un resumen de las propiedades devueltas por la consulta WMI:

- `Caption`: descripción corta del procesador (CPU).
- `DeviceID`: identificador único del procesador en el sistema.
- `Manufacturer`: nombre del fabricante del procesador.
- `MaxClockSpeed`: velocidad máxima del procesador, en MHz.
- `Name`: nombre con que se conoce el procesador.
- `CurrentClockSpeed`: velocidad actual del procesador, en MHz.
- `AddressWidth`: ancho del bus de direcciones, en bits. En un sistema operativo de 32 bits, el valor es 32, y en un sistema operativo de 64 bits es 64.
- `DataWidth`: ancho del bus de datos. En un procesador de 32 bits, el valor es 32, y en un procesador de 64 bits es 64.
- `L2CacheSize`: tamaño de la caché de nivel 2 del procesador, en kilobytes.

- `L3CacheSize`: tamaño de la caché de nivel 3 del procesador, en kilobytes.
- `NumberOfCores`: número de núcleos del procesador.
- `NumberOfLogicalProcessors`: número de procesadores lógicos.
- `Status`: estado actual del objeto procesador (OK, Error, Degraded, etc.).

e. Información vinculada a los discos

Con la clase **`Win32_DiskDrive`**, es posible recuperar información vinculada a los discos (físicos o virtuales) y demás dispositivos de almacenamiento (discos duros externos, llaves USB):

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_DiskDrive |
Format-List
Caption,Model,DeviceID,SerialNumber,FirmwareRevision,
InterfaceType,Partitions,Size


Caption          : ST250DM000-1BD141
Model            : ST250DM000-1BD141
DeviceID         : \\.\PHYSICALDRIVE0
SerialNumber     :             V9GY4AER
FirmwareRevision: KC45
InterfaceType    : IDE
Partitions       : 2
Size             : 250056737280

Caption          : WDC WD5000BPKT-00PK4T0
Model            : WDC WD5000BPKT-00PK4T0
DeviceID         : \\.\PHYSICALDRIVE1
SerialNumber     :     W -DXW128AC20283
FirmwareRevision: 01.0
InterfaceType    : IDE
Partitions       : 1
Size             : 500105249280

Caption          : Kingston DataTraveler G2 USB Device
Model            : Kingston DataTraveler G2 USB Device
DeviceID         : \\.\PHYSICALDRIVE1
SerialNumber     : 00123F54ACC0F9A0D61A0D61
FirmwareRevision: 1.00
InterfaceType    : USB
Partitions       : 1
Size             : 4005711360
```

El ejemplo anterior muestra que el equipo contiene tres discos: dos discos duros y una llave USB. He aquí el detalle de cada una de las propiedades devueltas:

- `Caption`: descripción corta del disco.

- `Model`: nombre y número del modelo asignados por el fabricante.
- `DeviceID`: identificador único que identifica al disco.
- `SerialNumber`: número de serie del disco asignado por el fabricante.
- `FirmwareRevision`: revisión del firmware del disco.
- `InterfaceType`: tipo de interfaz del disco (los posibles valores son: SCSI, HDC, IDE, USB, 1394).
- `Partitions`: número de particiones del disco reconocidas por el sistema operativo.
- `Size`: tamaño del disco (expresado en bytes).

f. Información vinculada a los servicios

Ya hemos visto la manipulación de los servicios a través de cmdlets de Windows PowerShell. Esto también es posible a través del objeto WMI **`Win32_Service`**. He aquí un ejemplo para consultar el conjunto de servicios de Windows en un puesto de trabajo:

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_Service


ExitCode : 0
Name     : AdobeARMservice
ProcessId: 1712
StartMode: Auto
State    : Running
Status   : OK

ExitCode : 0
Name     : AeLookupSvc
ProcessId: 0
StartMode: Manual
State    : Stopped
Status   : OK

ExitCode : 1077
Name     : ALG
ProcessId: 0
StartMode: Manual
State    : Stopped
Status   : OK
[...]
```

El resultado del comando devuelve el conjunto de servicios presentes en el puesto de trabajo. Si conoce el nombre del servicio, lo mejor es utilizar el parámetro **`-Filter`**, que permite mostrar únicamente el servicio correspondiente:

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_Service
-Filter          "Name='wuauserv'"          |          Format-List
Name,DisplayName,PathName,
ExitCode,ProcessId,StartName,StartMode,State,Status


Name       : wuauserv
DisplayName: Windows Update
PathName   : C:\Windows\system32\svchost.exe -k netsvcs
ExitCode   : 0
ProcessId  : 1080
StartName  : LocalSystem
StartMode  : Auto
State      : Running
Status     : OK
```

He aquí un detalle de cada una de las propiedades devueltas:

- `Name`: identificador único del servicio, que da una pista acerca de la funcionalidad que se administra.
- `DisplayName`: nombre del servicio tal y como se muestra en la consola «Servicios».
- `PathName`: ruta de acceso completa al archivo binario que implementa el servicio.
- `ExitCode`: código de error de Windows que define el error encontrado durante el arranque o la parada del servicio.
- `ProcessId`: número del PID (*Process Identifier*) del servicio.
- `StartName`: nombre de la cuenta con que se ejecuta el servicio.
- `StartMode`: modo de inicio del servicio.
- `State`: define el estado de la ejecución del servicio (Running, Stopped...).
- `Status`: define el estado del servicio (OK, Starting, Stopping, Error, Degraded...).

g. Información vinculada a las carpetas compartidas

El objeto WMI **`Win32_Share`** permite mostrar el conjunto de unidades y carpetas compartidas activas en el puesto de trabajo.

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_Share

Name                    Path                Description
```

```
----                    ----                  -----------
ADMIN$                  C:\Windows            Remote Admin
C$                      C:\                   Default share
D$                      D:\                   Default share
CarpetaCompartida               D:\CarpetaCompartida  Mi  carpeta
compartida
IPC$                                          Remote IPC
```

Explicación de las propiedades:

- `Name`: nombre del recurso compartido.
- `Path`: ruta de acceso local a la carpeta compartida.
- `Description`: descripción asociada al recurso compartido.

Por defecto, los recursos compartidos ADMIN$, IPC$ y el conjunto de unidades de datos (C$, D$...) están compartidas para fines de administración.

Cada recurso compartido posee una propiedad `Type`, que permite identificar el tipo de recurso compartido. Esta propiedad puede mostrarse con el siguiente comando:

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_Share |
Format-Table Name,Type

Name
Type
----                                                                    ---
-
ADMIN$
2147483648
C$
2147483648
D$
2147483648
F$
2147483648
IPC$
2147483651
miRecurso1$
0
Share$                                                                   0
```

En realidad, **`Win32_Share`** no devuelve únicamente las carpetas compartidas activas del puesto de trabajo. Existen otros recursos que pueden compartirse, como por ejemplo una cola de impresión.

He aquí una tabla resumen de los distintos valores que puede tomar la propiedad `Type` y el tipo de recurso que tiene asociado:

Valor	Tipo de recurso

Valor	Tipo de recurso
0 (0x0)	Disco duro
1 (0x1)	Cola de impresión
2 (0x2)	Dispositivo
3 (0x3)	IPC (*Interprocess Communications*)
2147483648 (0x80000000)	Disco duro - administración
2147483649 (0x80000001)	Cola de impresión - administración
2147483650 (0x80000002)	Dispositivo - administración
2147483651 (0x80000003)	IPC - administración

Para conocer, por ejemplo, el conjunto de carpetas compartidas, puede definir un filtro basado en la propiedad `Type`, con el valor a 0. Por ejemplo:

```
PS C:\Windows\system32> Get-WmiObject Win32_Share -Filter
"Type='0'"

Name                    Path                        Description
----                    ----                        -----------
CarpetaCompartida           D:\CarpetaCompartida        Mi carpeta
compartida
```

h. Información vinculada a los comandos de arranque

Cada vez que inicia una sesión en Windows, se ejecutan programas automáticamente. Para conocer el conjunto de aplicaciones que se ejecutan durante el arranque, se utiliza la clase **`Win32_StartupCommand`**. Esta devuelve una lista con toda la información necesaria:

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_StartupCommand


Command                                 User
Caption
-------                                 ----                                ----
---
C:\Windows\SysWOW64\OneDriveS...    AUTORIDAD    NT\SERVICIO    LOCAL
 OneDriveSetup
C:\Windows\SysWOW64\OneDriveS...   AUTORIDAD   NT\SERVICIO   DE   RED
OneDriveSetup
"C:\Users\Admin\AppData\Local...  W10\Admin
```

```
 OneDrive
"%ProgramFiles%\Windows Defen... Public
WindowsDefender
[...]
```

He aquí el detalle de las propiedades devueltas:

- `Command`: indica el comando o la ruta de acceso al ejecutable que se ejecuta durante el arranque.
- `User`: indica el usuario con el que se ha ejecutado el comando de arranque.
- `Caption`: descripción corta del comando de arranque.

Tenga precaución: en el caso de un sistema operativo Windows de 64 bits, **`Win32_StartupCommand`** no muestra las entradas configuradas, por ejemplo, en: HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\ Windows\ CurrentVersion\Run.

Para estar seguros de consultar todos los comandos de arranque, puede ir al **Administrador de tareas** y, a continuación, consultar la pestaña **Inicio** (Windows 8 y versiones posteriores). En Windows 7, utilice el programa msconfig.exe, pestaña **Inicio**.

i. Información vinculada al sistema operativo

La clase WMI **`Win32_OperatingSystem`** permite, como su nombre indica, obtener información correspondiente al sistema operativo Windows instalado en el puesto de trabajo. La lista de propiedades de esta clase es muy larga, pero he aquí las más relevantes:

```
PS C:\Windows\system32> Get-WmiObject Win32_OperatingSystem |
Format-List Caption,Manufacturer,SystemDirectory,Organization,
Version,BuildNumber,SerialNumber,OSLanguage,OSArchitecture,
RegisteredUser,InstallDate


Caption        : Microsoft Windows 10 Profesional
Manufacturer   : Microsoft Corporation
SystemDirectory: C:\Windows\system32
Organization   : Fabrikam Corp.
Version        : 10.0.14393
BuildNumber    : 14393
SerialNumber   : 00331-10000-00001-AA129
OSLanguage     : 1036
OSArchitecture : 64 bits
RegisteredUser : Admin
InstallDate    : 20161026195616.000000+120
```

- `Caption`: descripción corta del sistema operativo.
- `Manufacturer`: nombre del fabricante que ha desarrollado el sistema operativo.
- `SystemDirectory`: carpeta de sistema del sistema operativo.
- `Organization`: nombre de la empresa a la que pertenece el equipo. Esta información se obtiene directamente de la clave del registro HKEY_ LOCAL_ MACHINE\SOFTWARE\Microsoft\Windows NT\ CurrentVersion, valor RegisteredOrganization.
- `Version`: número de versión del sistema operativo.
- `BuildNumber`: número de *build* (versión) del sistema operativo.
- `SerialNumber`: número de serie de identificación del sistema operativo.
- `OSLanguage`: idioma del sistema operativo instalado (en formato LCID).
- `OSArchitecture`: arquitectura del sistema operativo instalado (32 bits o 64 bits).
- `RegisteredUser`: nombre del usuario del sistema operativo. Esta información se obtiene directamente de la clave del registro HKEY_LOCAL_ MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion, valor RegisteredOwner.
- `InstallDate`: fecha de instalación de Windows en el puesto de trabajo.

Respeto a la propiedad `InstallDate`, para convertir la fecha en un formato literal, utilice el siguiente comando:

```
PS C:\Windows\system32> ([WMI]'').ConvertToDateTime((Get-
WmiObject Win32_OperatingSystem).InstallDate)

miércoles 26 de octubre de 2016 19:56:16
```

En cuanto a la propiedad `Version`, he aquí los números de sistema operativo Windows desde Windows 7:

Sistema operativo	Número de versión
Windows 7	6.1.7600

Sistema operativo	Número de versión
Windows 7 con Service Pack 1	6.1.7601
Windows 8	6.2.9200
Windows 8.1	6.3.9600
Windows 10	10.0.10240
Windows 10 (November Update)	10.0.10586
Windows 10 (Anniversary Update)	10.0.14393
Windows 10 (Creators Update)	10.0.15063

Una de las novedades de Windows 10 es que este sistema operativo evoluciona regularmente en el tiempo. Sus actualizaciones no contienen únicamente revisiones de seguridad o de aplicaciones, sino que aportan también nuevas características. Por ello, el número de versión también ha evolucionado.

j. Información vinculada a la unidad de CD/DVD

La presencia o no de una unidad de CD/DVD conectada al equipo y reconocida por el sistema operativo puede identificarse mediante la clase WMI **`Win32_CDROMDrive`**:

```
PS C:\Windows\system32> Get-WmiObject Win32_CDROMDrive |
Format-List Name,Drive,Manufacturer,VolumeName,MediaLoaded,
MediaType,MfrAssignedRevisionLevel


Name                      : TSSTcorp DVD-ROM TS-H353B
Drive                     : F:
Manufacturer              : (Standard CD-ROM drives)
VolumeName                : J_CPRA_X64ESEV_ES-ES_DV5
MediaLoaded               : True
MediaType                 : DVD-ROM
MfrAssignedRevisionLevel: D500
```

La información que se obtiene es la siguiente:

- `Name`: nombre de la unidad de CD/DVD.
- `Drive`: letra de la unidad de CD/DVD.
- `Manufacturer`: nombre del fabricante de la unidad de CD/DVD.

- `VolumeName`: nombre del volumen de CD o DVD presente en la unidad.
- `MediaLoaded`: indica si se ha cargado algún medio extraíble en la unidad (vale `True` si hay algún medio presente).
- `MediaType`: tipos de medios que pueden utilizarse con la unidad (DVD-ROM, DVD Writer, etc.).
- `MfrAssignedRevisionLevel`: versión del firmware integrado en la unidad CD/DVD.

También se obtiene información relativa a las unidades de CD/DVD virtuales.

k. Información vinculada a la placa base

La clase WMI **`Win32_BaseBoard`** recupera información a nivel de la placa base con que se equipa el puesto de trabajo:

```
PS C:\Windows\system32> Get-WmiObject Win32_BaseBoard | Format-List
Manufacturer,Name,SerialNumber,Product,Version,Status


Manufacturer: Dell Inc.
Name        : Base Board
SerialNumber: /4VKXC5J/CN7360424B00FP/
Product     : 0773VG
Version     : A00
Status      : OK
```

He aquí una descripción de las propiedades resultado del comando anterior:

- `Manufacturer`: nombre del fabricante de la placa base.
- `Name`: etiqueta con la que se conoce el objeto.
- `SerialNumber`: número de serie asignado por la empresa que produce la placa base.
- `Product`: número del producto definido por el fabricante. En ciertos casos, puede tratarse del nombre/modelo del producto.
- `Version`: número de revisión de la placa base.
- `Status`: indica el estado del objeto (OK, Error, Degraded...).

l. Información vinculada a la batería

Es posible recuperar información relativa a otro componente de hardware con el que se equipa no solo ordenadores portátiles, sino también tabletas. Se trata de la batería. Para ello, se utiliza la clase WMI **`Win32_Battery`**:

```
PS C:\Windows\system32> Get-WmiObject Win32_Battery | Format-
List
Caption,Name,DesignVoltage,DeviceID,EstimatedChargeRemaining,
EstimatedRunTime


Caption                  : Batería interna
Name                     : DELL H139106
DesignVoltage            : 13186
DeviceID                 : 3572Samsung SDIDELL H139106
EstimatedChargeRemaining: 85
EstimatedRunTime         : 71582788
```

He aquí una explicación de cada una de las propiedades:

- `Caption`: descripción corta de la batería.
- `Name`: nombre de la batería.
- `DesignVoltage`: voltaje específico de la batería, en milivoltios.
- `DevideID`: identificador de la batería.
- `EstimatedChargeRemaining`: estimación de la carga total, en porcentaje.
- `EstimatedRunTime`: estimación, en minutos, del tiempo restante hasta el agotamiento completo de la batería. El equipo portátil debe funcionar con batería para que la estimación sea correcta. En caso contrario, se devuelve un valor por defecto: 71582788.

¿Ordenador fijo o portátil?

Cuando escribe un script, y en función de su objetivo, puede ser necesario conocer si el script se está ejecutando en un puesto de trabajo fijo o portátil. Si tiene un parque de ordenadores muy homogéneo, puede mantener un listado con los modelos de los equipos que posee en su parque informático. En el caso de que su parque no sea tan homogéneo, tiene la posibilidad de detectar, con **`Win32_Battery`**, si un puesto de trabajo posee batería o no.

En un puesto de trabajo fijo, el comando no devuelve ningún resultado. Así, puede utilizar la siguiente estructura condicional:

```
If (Get-WmiObject Win32_Battery)
{
       # Procesamiento a realizar en el caso de un ordenador
portátil
    Write-Host "¡Es un ordenador portátil!"
}
Else
{
    # Procesamiento a realizar en el caso de un ordenador fijo
    Write-Host "¡Es un ordenador fijo!"
}
```

En el caso de tabletas que trabajan con Windows (Surface y Surface Pro, por ejemplo), se detectan como ordenadores portátiles. Como poseen batería, la consulta WMI devuelve obligatoriamente un resultado.

m. Información vinculada a la tarjeta de red

La tarjeta de red es un elemento importante que forma parte de un puesto de trabajo. La clase WMI **`Win32_NetworkAdapter`** permite recuperar información acerca del conjunto de interfaces de red físicas y también virtuales. Para enumerar únicamente las tarjetas de red de tipo Ethernet, el comando filtra los resultados basándose en la propiedad `AdapterType`:

```
PS C:\Windows\system32> Get-WmiObject Win32_NetworkAdapter | ?
{$_.AdapterType -match "Ethernet"} | Format-List ServiceName,
DeviceID,Name,Manufacturer,Description,AdapterType,NetConnectionID
,MACAddress,Speed


ServiceName     : e1iexpress
DeviceID        : 0
Name            : Conexión de red Intel(R) 82577LM Gigabit
Manufacturer    : Intel Corporation
Description     : Conexión de red Intel(R) 82577LM Gigabit
InterfaceIndex : 24
AdapterType     : Ethernet 802.3
NetConnectionID: Ethernet
MACAddress      : 00:26:B9:E0:9F:C1
Speed           : 9223372036854775807

ServiceName     : NETwNs64
DeviceID        : 2
Name            : Intel(R) Centrino(R) Advanced-N 6200 AGN
Manufacturer    : Intel Corporation
Description     : Intel(R) Centrino(R) Advanced-N 6200 AGN
InterfaceIndex : 25
AdapterType     : Ethernet 802.3
NetConnectionID: Wi-Fi
MACAddress      : 00:23:14:D0:CA:40
Speed           : 130000000
```

He aquí un detalle de las propiedades devueltas:

- `ServiceName`: nombre del servicio de la tarjeta de red.
- `DeviceID`: identificador único de la tarjeta de red (dispositivo) en el sistema.
- `Name`: nombre de la tarjeta de red.
- `Manufacturer`: nombre del fabricante de la tarjeta de red.
- `Description`: descripción de la tarjeta de red.
- `InterfaceIndex`: índice que identifica de manera única a la interfaz de red.
- `AdapterType`: tipo de tecnología de red utilizada (Ethernet 802.3, Token Ring 802.5, 1394, etc.).
- `NetConnectionID`: nombre de la conexión de red, tal y como aparece en Conexiones de red en el Panel de control de Windows.
- `MACAddress`: dirección MAC (*Media Access Control*) de la tarjeta de red.
- `Speed`: estimación del ancho de banda medido en bit/s.

n. Información vinculada a la memoria

Otro componente de hardware con el que se equipa a los ordenadores es la memoria. La clase WMI que vamos a utilizar es **`Win32_PhysicalMemory`**:

```
PS C:\Windows\system32> Get-WmiObject Win32_PhysicalMemory |
Format-List BankLabel,DeviceLocator,Capacity,Manufacturer,
PartNumber,SerialNumber,Speed


BankLabel    : BANK 0
DeviceLocator: DIMM_A
Capacity     : 2147483648
Manufacturer : 830B
PartNumber   : NT2GC64B8HC0NS-CG
SerialNumber : AC09186C
Speed        : 1067

BankLabel    : BANK 2
DeviceLocator: DIMM_B
Capacity     : 2147483648
Manufacturer : 830B
PartNumber   : NT2GC64B8HC0NS-CG
SerialNumber : A10A186E
Speed        : 1067
```

He aquí una descripción de cada una de las propiedades:

- `BankLabel`: nombre de la ubicación física (slot de memoria).
- `DeviceLocator`: indicador de la ubicación de la memoria.
- `Capacity`: capacidad total de la memoria física, en bytes.
- `Manufacturer`: nombre del fabricante de la memoria física.
- `PartNumber`: referencia del producto asignada por el fabricante.
- `SerialNumber`: número de serie asignado por el fabricante.
- `Speed`: velocidad de la memoria física, en nanosegundos.

o. Información vinculada a las aplicaciones

La clase WMI **`Win32_Product`** permite recuperar una lista de los programas y aplicaciones instalados en el puesto de trabajo. Tenga precaución, sin embargo, pues **`Win32_Product`** enumera únicamente las aplicaciones instaladas a través de Windows Installer (packages MSI):

```
PS C:\Windows\system32> Get-WmiObject -Class Win32_Product


IdentifyingNumber: {90150000-0090-040C-1000-0000000FF1CE}
Name              : Microsoft DCF MUI (Spanish) 2013
Vendor            : Microsoft Corporation
Version           : 15.0.4569.1506
Caption           : Microsoft DCF MUI (Spanish) 2013

IdentifyingNumber: {90150000-0011-0000-1000-0000000FF1CE}
Name              : Microsoft Office Professional Plus 2013
Vendor            : Microsoft Corporation
Version           : 15.0.4569.1506
Caption           : Microsoft Office Professional Plus 2013

IdentifyingNumber: {90150000-00A1-040C-1000-0000000FF1CE}
Name              : Microsoft OneNote MUI (Spanish) 2013
Vendor            : Microsoft Corporation
Version           : 15.0.4569.1506
Caption           : Microsoft OneNote MUI (Spanish) 2013

IdentifyingNumber: {90150000-00C1-0000-1000-0000000FF1CE}
Name              : Microsoft Office 32-bit Components 2013
Vendor            : Microsoft Corporation
Version           : 15.0.4569.1506
Caption           : Microsoft Office 32-bit Components 2013
[...]
```

El comando devuelve cinco propiedades para cada uno de los programas instalados en el puesto de trabajo. He aquí una descripción:

- `IdentifyingNumber`: identificador (de tipo GUID) propio de cada programa.
- `Name`: nombre común del programa.
- `Vendor`: nombre del fabricante del programa.
- `Version`: número de versión del programa.
- `Caption`: descripción corta del producto.

Actualizaciones

Otro aspecto importante en un puesto de trabajo es poder conocer las actualizaciones instaladas. Están todas referenciadas por un número de KB (*Knowlegde Base*). Todos los segundos martes del mes, Microsoft publica generalmente actualizaciones que se aplicarán sobre los puestos de trabajo, para mejorar la seguridad y, en ocasiones, las características. Se trata de lo que se conoce más comúnmente como *Patch Tuesday*.

1. Mostrar las actualizaciones instaladas

Gracias a PowerShell, es fácil recuperar la lista de actualizaciones instaladas. Con un único comando, se muestra diversa información: el identificador de la revisión (número de KB), la cuenta de usuario que ha realizado la actualización, así como la fecha de instalación. Se utiliza el cmdlet **`Get-HotFix`**.

He aquí los parámetros:

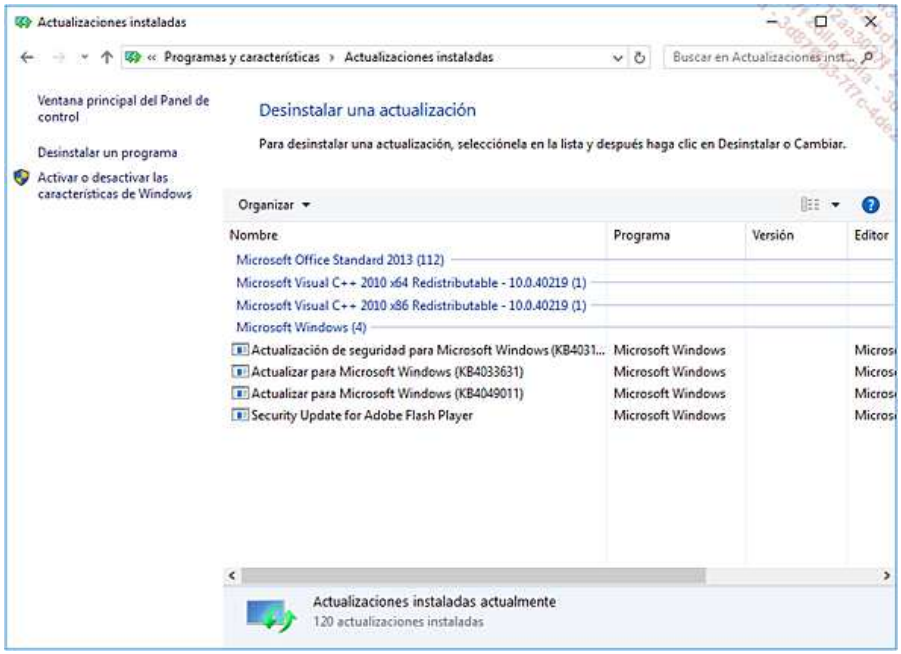
Parámetro	Descripción
`-ComputerName` `<String[]>`	Especifica uno o varios equipos remotos.
`-Credential` `<PSCredential>`	Especifica una cuenta de usuario que posee permisos para ejecutar esta acción. El valor por defecto es el usuario actual.
`-Description` `<String[]>`	Obtiene únicamente las revisiones con una descripción concreta. Se admiten los caracteres comodín.
`-Id` `<String[]>`	Obtiene únicamente las revisiones con los ID de revisión específicos.

Ejemplo: recupera la lista de KB instaladas en el equipo local

```
PS C:\Windows\system32> Get-HotFix
```

```
Source       Description         HotFixID      InstalledBy
 InstalledOn
------      -----------       --------    -----------           -----
------
W10           Update               KB3176935    AUTORIDAD  NT\Sistema
 26/10/2016 00:00:00
W10           Update               KB3176937    AUTORIDAD  NT\Sistema
 26/10/2016 00:00:00
W10           Update               KB3199209    AUTORIDAD  NT\Sistema
 27/10/2016 00:00:00
W10           Update               KB3199986    AUTORIDAD  NT\Sistema
 01/11/2016 00:00:00
W10             Security  Update   KB3214628     AUTORIDAD  NT\Sistema
 12/01/2017 00:00:00
W10             Security  Update   KB3213986     AUTORIDAD  NT\Sistema
 12/01/2017 00:00:00
[...]
```

El resultado devuelto por este cmdlet indica únicamente aquellas actualizaciones instaladas para el sistema operativo Microsoft Windows. No se muestran las revisiones para Microsoft Office y demás productos.



Actualizaciones instaladas

Por defecto, el cmdlet **`Get-HotFix`** consulta el puesto local, aunque también es posible ejecutar el cmdlet en equipos remotos:

```
PS C:\Windows\system32> Get-HotFix -ComputerName W7,W8

Source   Description         HotFixID         InstalledBy
 InstalledOn
------  -----------      --------      -----------          -------
```

```
----
W7        Update               KB2819745       W7\Admin
10/08/2014
W7        Hotfix              KB2534111
10/08/2014
W7         Update               KB2809215       W7\Admin
  10/08/2014
W7         Hotfix               KB2872035       W7\Admin
  10/08/2014
W7       Update            KB976902      W7\Administrador
W8       Update            KB2899189_... AUTORIDAD NT\Sistema
W8       Update            KB2843630     AUTORIDAD NT\Sistema
W8       Security Update   KB2862152     AUTORIDAD NT\Sistema
W8       Security Update   KB2868626     AUTORIDAD NT\Sistema
W8       Security Update   KB2876331     AUTORIDAD NT\Sistema
[...]
```

Por supuesto, es posible personalizar el resultado para recuperar únicamente la información deseada. Por ejemplo, para recuperar únicamente los números de KB, escriba:

```
PS C:\Windows\system32> $kb = Get-HotFix | Select-Object HotfixID
PS C:\WINDOWS\system32> $kb.HotfixID
KB3176935
KB3176937
KB3199209
KB3199986
KB3214628
KB3213986
```

Y para guardar esta información en un archivo de texto, escriba:

```
PS C:\Windows\system32> $filename = '\\servername\temp$\' +
$env:COMPUTERNAME + '.txt'
PS C:\Windows\system32> $kb = Get-HotFix | Select-Object HotfixID
PS C:\WINDOWS\system32> $kb.HotfixID | Out-File -FilePath $filename
```

En el ejemplo, el resultado del comando se escribe en un archivo de texto con el nombre de la máquina que ha ejecutado el script. El archivo se guarda en el recurso compartido de red `temp$` del servidor `servername`.

El siguiente ejemplo comprueba, para cada puesto de trabajo incluido en la variable **`$tabPC`**, si se ha instalado o no la KB3213986. En caso negativo, se escribe una línea en un archivo de texto indicando el nombre del equipo que no contiene la KB instalada.

```
PS C:\Windows\system32> $tabPC = @("localhost","PC-1","PC-2")
PS C:\Windows\system32> $tabPC | % {
>> If (!(Get-HotFix -Id KB3213986 -ComputerName $_ -ErrorAction
SilentlyContinue))
>> {
>>     Add-Content $_ -Path C:\Temp\KB3213986-faltante.txt
>> }
```

```
>> }
>>
```

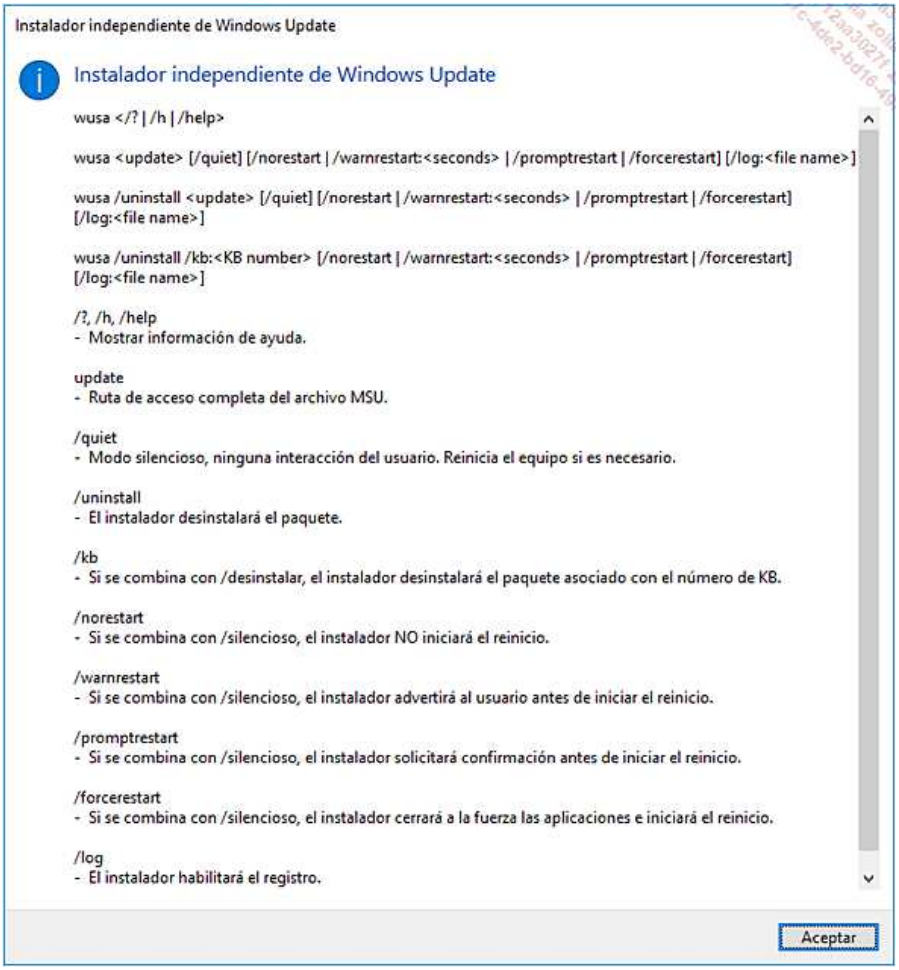
Existen varias posibilidades para recuperar la lista de KB instaladas en los puestos de trabajo. El rol WSUS (*Windows Server Update Services*), presente en Windows Server, puede proveer un informe detallado de las actualizaciones instaladas en cada equipo.

2. Interactuar con las actualizaciones con wusa.exe

Si bien no existe un cmdlet propiamente dicho para instalar y desinstalar KB, conviene saber cómo llevar a cabo esta tarea por línea de comandos con Windows PowerShell. Esto puede resultar práctico en los scripts, por ejemplo.

El Instalador independiente de Windows Update es el ejecutable wusa.exe. Puede consultar la ayuda de este programa con el siguiente comando:

```
PS C:\Windows\system32> wusa.exe /?
```



La ayuda del Instalador independiente de Windows Update

a. Instalar una actualización

Para instalar una KB, hay que indicar la ruta de acceso al archivo de update, que es un archivo .msu.

Ejemplo: instalación de una KB por línea de comandos

```
PS C:\Windows\system32> $msuFile = "C:\Temp\WindowsTH-RSAT_WS2016-
x64.msu"
PS C:\Windows\system32> Start-Process wusa.exe -ArgumentList
"$msuFile /quiet /norestart" -Wait
```

b. Desinstalar una actualización

Para desinstalar una KB, nos basaremos en el mismo principio, a excepción de indicar el parámetro `/uninstall`. Existen dos posibilidades para realizar esta operación, bien especificando el

número de KB que se ha de desinstalar o bien indicando la ruta de acceso al archivo .msu (que posee la información correspondiente a la KB que se ha de desinstalar).

Ejemplo 1: desinstalar una KB pasando como parámetro el número de KB

El parámetro `/kb:<KB number>` debe incluirse en la línea de comando para indicar el número de KB que se va a desinstalar.

```
PS C:\Windows\system32> Start-Process wusa.exe -ArgumentList
"/uninstall /kb:2693643 /quiet /norestart" -Wait
```

Ejemplo 2: desinstalar una KB utilizando un archivo .msu

En este caso, se indica un archivo idéntico al que se utilizó para instalar la KB.

```
PS C:\Windows\system32> $msuFile = "C:\Temp\WindowsTH-RSAT_WS2016-
x64.msu"
PS C:\Windows\system32> Start-Process wusa.exe -ArgumentList
"/uninstall $msuFile /quiet /norestart" -Wait
```

Introducción

En este capítulo aprenderá a administrar ciertos componentes de un puesto de trabajo, en particular la tarjeta de red, que es uno de los elementos más importantes en la actualidad de cualquier ordenador. Veremos también la administración de discos y la gestión de la energía.

A continuación veremos los dispositivos, en particular las operaciones de administración básica sobre los dispositivos de impresión en Windows.

Las tarjetas de red

Es posible configurar los dispositivos de red desde Windows 8. Si desea gestionar esta característica en Windows 7, utilice el ejecutable netsh.exe. En Windows 7 solo está disponible el cmdlet **`Test-Connection`**.

La configuración de red en un puesto de trabajo resulta algo indispensable en la actualidad. Incluso aunque la mayoría de los parámetros de red los proporcionan automáticamente los servidores DHCP (*Dynamic Host Configuration Protocol*), puede ser necesario, en ciertas situaciones, configurar los parámetros manualmente, tanto la parte IP como DNS. También puede ser necesario comprobar la accesibilidad de un puesto remoto para saber si se encuentra o no conectado.

Los cmdlets estudiados son los siguientes:

- **`Test-Connection`**: envía paquetes con peticiones eco ICMP (equivalentes al comando `ping`).
- **`Get-NetAdapter`**: recupera la lista de adaptadores de red (Ethernet, Wi-Fi, etc.).
- **`Get-NetIPInterface`**: recupera la lista de interfaces IP configuradas.
- **`Get-NetIPAddress`**: recupera la dirección IP de una interfaz.
- **`Get-NetRoute`**: muestra la tabla de enrutamiento.
- **`New-NetIPAddress`**: configura una nueva dirección IP en una interfaz.
- **`Remove-NetIPAddress`**: elimina la dirección IP de una interfaz.
- **`Remove-NetRoute`**: elimina una entrada de la tabla de enrutamiento.
- **`Set-NetIPInterface`**: modifica uno o varios parámetros de una interfaz.
- **`Get-DnsClientServerAddress`**: recupera las direcciones IP de los servidores DNS.

- **Set-DnsClientServerAddress**: define las direcciones de los servidores DNS.
- **Resolve-DnsName**: resuelve los nombres DNS o IP.
- **Get-DnsClientCache**: recupera la caché DNS del puesto de trabajo.
- **Clear-DnsClientCache**: vacía la caché DNS.
- **Register-DnsClient**: renueva la inscripción del cliente DNS.
- **Add-VpnConnection**: agrega una nueva conexión VPN.
- **Get-VpnConnection**: recupera la información de conexión de una conexión VPN.
- **Set-VpnConnection**: modifica los parámetros de una conexión VPN.
- **Remove-VpnConnection**: elimina una conexión VPN.

1. Realizar un ping

Si durante la ejecución de un script tiene que realizar interacciones con distintas máquinas a través de la red, puede resultar interesante en primer lugar saber si están todas en línea. En caso contrario, el script puede desarrollarse de manera totalmente incongruente y tener consecuencias nefastas.

Para ello se utiliza el cmdlet **Test-Connection**. He aquí algunos de los parámetros disponibles:

Parámetro	Descripción
-ComputerName <String[]>	Especifica uno o varios equipos a los que se enviará la petición eco ICMP.
-Count <Int32>	Especifica el número de peticiones eco que se van a enviar. El valor por defecto es 4.

Ejemplo: envío de un ping a un equipo remoto

```
PS C:\Windows\system32> Test-Connection -ComputerName W10

Source        Destination     IPV4Address      IPV6Address      Bytes
 Time(ms)
```

```
------      -----------    -----------     -----------     -----  ----
----
W7          W10            192.168.1.8     ::1             32     0
W7          W10            192.168.1.8     ::1             32     0
W7          W10            192.168.1.8     ::1             32     0
W7          W10            192.168.1.8     ::1             32     0
```

En el caso de un script, lo más sencillo es utilizar la variable **$?**, que devuelve el estado (éxito o error) de la última operación. Si el comando se desarrolla correctamente, entonces devuelve el valor `True`; en caso contrario, se trata del valor `False`.

Ejemplo: comprobar la presencia de una máquina remota

Si la máquina remota no responde, el script termina automáticamente con el comando **Exit**.

```
PS  C:\Windows\system32>  Test-Connection  -ComputerName  W10  -Count
1
PS C:\Windows\system32> If (!$?) {
>> # No hay respuesta al ping
>> Exit
>> }
```

No olvide que la no-respuesta al comando **Test-Connection** puede estar causada por diversos motivos. Puede provenir de un problema de red, por supuesto, pero también simplemente porque haya algún firewall activo en la máquina remota. El ejemplo anterior no es general y depende de la política de seguridad implementada en su empresa.

La otra solución consiste en utilizar un objeto .NET Framework, con la clase `Ping`, presente en el espacio de nombres System.Net.NetworkInformation.

```
PS C:\Windows\system32> $ping =
New-Object System.Net.NetworkInformation.Ping
PS C:\Windows\system32> $server = 'servidor01'
PS C:\Windows\system32> $reply = $ping.send($server)
PS C:\Windows\system32> If ($reply.Status -eq 'Success')
>> {
>>     Write-Host $server is online !
>> }
>> Else
>> {
>>     Write-Host $server is offline ! -ForegroundColor Red
>>     Exit
>> }
>>
```

El ejemplo anterior envía un paquete con una petición eco ICMP al servidor01. Si este responde, el script sigue con su desarrollo

normal. En caso contrario, el script detiene su ejecución gracias al comando **`Exit`**.

- En la definición de la variable `$server`, puede utilizar bien el nombre del servidor o bien una dirección IP (de tipo IPv4 o IPv6).

2. Modificar los parámetros de la tarjeta de red

En esta sección, veremos cómo recuperar la información de configuración de las tarjetas de red presentes en un puesto de trabajo y, por supuesto, cómo modificar o parametrizar la configuración IP de una tarjeta de red.

También se abordará la tabla de enrutamiento, pues en ella se encuentra la configuración IP de la puerta de enlace predeterminada configurada en las tarjetas de red.

a. Parametrizar la configuración IP

Para comenzar, puede resultar interesante conocer los adaptadores de red que existen en el puesto de trabajo. Generalmente, un equipo fijo está compuesto de una única tarjeta de red, mientras que un equipo portátil posee una tarjeta Wi-Fi, además de una tarjeta de red. **`Get-NetAdapter`** permite enumerar estas interfaces y ver su información complementaria, como por ejemplo su estado, la dirección MAC o su velocidad teórica. He aquí un resumen de los parámetros:

Parámetro	Descripción
`-InterfaceDescription` `<String[]>` **`-ifDesc`** `<String[]>`	Especifica la descripción del adaptador de red.
`-IncludeHidden`	Incluye los adaptadores de red ocultos. Por defecto, solo se muestran los adaptadores visibles.

Parámetro	Descripción
-InterfaceIndex <UInt32[]> **-ifIndex** <UInt32[]>	Especifica el número de índice del adaptador de red.
-Name <String[]>	Especifica el nombre del adaptador de red.
-Physical	Devuelve todos los adaptadores de red físicos.

Ejemplo: obtener los distintos adaptadores de red de un puesto de trabajo

```
PS C:\Windows\system32> Get-NetAdapter -Physical | Select-Object
Name,InterfaceDescription,ifIndex,Status


Name           InterfaceDescription                          ifIndex
Status
----         --------------------                        ------- -----
-
Ethernet    Conexión de red Intel(R) 82577LM Gi...         3 Up
Wi-Fi          Intel(R)  Centrino(R)  Advanced-N  6200...          4
Disconnected
```

Las explicaciones y los ejemplos que se muestran a continuación se basan en una tarjeta de red Ethernet con una dirección IP de tipo IPv4. Para la configuración de una tarjeta de red Wi-Fi, el principio será idéntico.

Windows PowerShell permite recuperar y modificar los parámetros de la tarjeta de red Ethernet del puesto de trabajo. Se explicarán varios cmdlets, comenzando por el cmdlet **Get-NetIPInterface**, que permite recuperar las distintas interfaces IP de los diferentes adaptadores de red presentes en el puesto de trabajo.

Parámetro	Descripción
-AddressFamily <AddressFamily[]>	Especifica la versión de IP que hay que recuperar. Los posibles valores son: IPv4, IPv6.

Parámetro	Descripción
-ConnectionState <ConnectionState[]>	Especifica el estado de las conexiones para las interfaces que están conectadas físicamente a una red.
-Dhcp <Dhcp[]>	Especifica el valor DHCP para una interfaz IP. Los posibles valores son: Enabled, Disabled.
-InterfaceAlias <String[]> **-ifAlias** <String[]>	Especifica uno o varios alias de interfaces de red.
-InterfaceIndex <UInt32[]> **-ifIndex** <UInt32[]>	Especifica uno o varios números de índice de interfaces de red.

Ejemplo 1: mostrar el conjunto de interfaces IP del puesto de trabajo

```
PS C:\Windows\system32> Get-NetIPInterface | Select-Object
ifIndex,InterfaceAlias,AddressFamily,Dhcp,ConnectionState


ifIndex        : 6
InterfaceAlias : Teredo Tunneling Pseudo-Interface
AddressFamily  : IPv6
Dhcp           : Enabled
ConnectionState: Connected

ifIndex        : 5
InterfaceAlias : isatap.{8F148250-588B-4319-A005-D041D0AE52C6}
AddressFamily  : IPv6
Dhcp           : Disabled
ConnectionState: Disconnected

ifIndex        : 4
InterfaceAlias : Wi-Fi
AddressFamily  : IPv6
Dhcp           : Enabled
ConnectionState: Disconnected
```

```
ifIndex        : 1
InterfaceAlias : Loopback Pseudo-Interface 1
AddressFamily  : IPv6
Dhcp           : Disabled
ConnectionState: Connected

ifIndex        : 3
InterfaceAlias : Ethernet
AddressFamily  : IPv6
Dhcp           : Enabled
ConnectionState: Connected

ifIndex        : 4
InterfaceAlias : Wi-Fi
AddressFamily  : IPv4
Dhcp           : Enabled
ConnectionState: Disconnected

ifIndex        : 1
InterfaceAlias : Loopback Pseudo-Interface 1
AddressFamily  : IPv4
Dhcp           : Disabled
ConnectionState: Connected

ifIndex        : 3
InterfaceAlias : Ethernet
AddressFamily  : IPv4
Dhcp           : Enabled
ConnectionState: Connected
```

Ejemplo 2: mostrar las interfaces IP en función de los parámetros

Los distintos parámetros de **Get-NetIPInterface** permiten filtrar la petición que recupera las distintas interfaces IP. El siguiente ejemplo muestra únicamente las interfaces IP de tipo IPv4 sobre los adaptadores de red físicos.

```
PS C:\Windows\system32> Get-NetAdapter -Physical |
Get-NetIPInterface -AddressFamily IPv4 | Format-List


InterfaceIndex                 : 4
InterfaceAlias                 : Wi-Fi
AddressFamily                  : IPv4
Forwarding                     : Disabled
Advertising                    : Disabled
NlMtu(Bytes)                   : 1500
AutomaticMetric                : Enabled
InterfaceMetric                : 5
NeighborDiscoverySupported     : Yes
NeighborUnreachabilityDetection: Enabled
BaseReachableTime(ms)          : 30000
ReachableTime(ms)              : 17000
RetransmitTime(ms)             : 1000
DadTransmits                   : 3
DadRetransmitTime(ms)          : 1000
RouterDiscovery                : ControlledByDHCP
ManagedAddressConfiguration    : Enabled
OtherStatefulConfiguration     : Enabled
```

```
WeakHostSend                    : Disabled
WeakHostReceive                 : Disabled
IgnoreDefaultRoutes             : Disabled
AdvertisedRouterLifetime        : 00:30:00
AdvertiseDefaultRoute           : Disabled
CurrentHopLimit                 : 0
ForceArpNdWolPattern            : Disabled
DirectedMacWolPattern           : Disabled
EcnMarking                      : AppDecide
Dhcp                            : Enabled
ConnectionState                 : Disconnected
PolicyStore                     : ActiveStore
CompartmentId                   : 1

InterfaceIndex                  : 3
InterfaceAlias                  : Ethernet
AddressFamily                   : IPv4
Forwarding                      : Disabled
Advertising                     : Disabled
NlMtu(Bytes)                    : 1500
AutomaticMetric                 : Enabled
InterfaceMetric                 : 20
NeighborDiscoverySupported      : Yes
NeighborUnreachabilityDetection: Enabled
BaseReachableTime(ms)           : 30000
ReachableTime(ms)               : 32500
RetransmitTime(ms)              : 1000
DadTransmits                    : 3
DadRetransmitTime(ms)           : 1000
RouterDiscovery                 : ControlledByDHCP
ManagedAddressConfiguration     : Enabled
OtherStatefulConfiguration      : Enabled
WeakHostSend                    : Disabled
WeakHostReceive                 : Disabled
IgnoreDefaultRoutes             : Disabled
AdvertisedRouterLifetime        : 00:30:00
AdvertiseDefaultRoute           : Disabled
CurrentHopLimit                 : 0
ForceArpNdWolPattern            : Disabled
DirectedMacWolPattern           : Disabled
EcnMarking                      : AppDecide
Dhcp                            : Enabled
ConnectionState                 : Connected
PolicyStore                     : ActiveStore
CompartmentId                   : 1
```

Por defecto, Windows posee varias interfaces IP. Algunas provienen de adaptadores de red virtuales, como Teredo Tunneling Pseudo-Interface e ISATAP, que sirven para gestionar el protocolo IPv6.

La interfaz de Loopback representa la dirección de bucle local del puesto de trabajo. Existe como interfaz IPv4 (correspondiente a la dirección 127.0.0.1) e IPv6 (correspondiente a la dirección ::1).

A esto se agregan dos interfaces físicas, Ethernet y Wi-Fi. Cada una de ellas tiene dos interfaces posibles: la primera en IPv4 y la segunda en IPv6. En función de los equipos, pueden existir otras

interfaces: un puesto de trabajo que posea varias tarjetas de red Ethernet, o incluso Bluetooth, por ejemplo.

Ahora que conoce el nombre de la tarjeta (o su número de índice), así como el tipo o los tipos de interfaces IP vinculadas a ella, la siguiente etapa es saber qué dirección IP está configurada en una interfaz concreta. Para ello, se utiliza el cmdlet **`Get-NetIPAddress`**. He aquí una tabla con los parámetros más importantes de este cmdlet:

Parámetro	Descripción
`-AddressFamily` `<AddressFamily[]>`	Especifica la versión IP que se ha de recuperar. Los posibles valores son: `IPv4`, `IPv6`.
`-InterfaceAlias` `<String[]>` **`-ifAlias`** `<String[]>`	Especifica uno o varios alias de interfaces de red.
`-InterfaceIndex` `<UInt32[]>` **`-ifIndex`** `<UInt32[]>`	Especifica uno o varios números de índice de interfaces de red.
`-IPAddress` `<String[]>`	Especifica la dirección IPv4 o IPv6.
`-PrefixLength` `<Byte[]>`	Especifica la máscara de subred.
`-PrefixOrigin` `<PrefixOrigin[]>`	Especifica el origen del prefijo de la dirección IP. Los posibles valores son: `Manual`, `WellKnown`, `DHCP`, `RouterAdvertisement`.
`-SuffixOrigin` `<SuffixOrigin[]>`	Especifica el origen del sufijo de la dirección IP. Los posibles valores son: `Manual`, `WellKnown`, `DHCP`, `Link`, `Random`.

Observación correspondiente a los parámetros **`-PrefixOrigin`** y **`-SuffixOrigin`**: la dirección IP está dividida en dos partes, la dirección de red (que corresponde a `PrefixOrigin`) y la dirección del host (que corresponde a `SuffixOrigin`).

Ejemplo 1: recuperar la dirección IP sobre la tarjeta Ethernet IPv4

```
PS     C:\Windows\system32>     Get-NetIPAddress     -InterfaceAlias
Ethernet
-AddressFamily IPv4


IPAddress         : 192.168.0.48
InterfaceIndex    : 3
InterfaceAlias    : Ethernet
AddressFamily     : IPv4
Type              : Unicast
PrefixLength      : 24
PrefixOrigin      : Dhcp
SuffixOrigin      : Dhcp
AddressState      : Preferred
ValidLifetime     : 9.17:51:47
PreferredLifetime: 9.17:51:47
SkipAsSource      : False
PolicyStore       : ActiveStore
```

La propiedad `IPAddress` muestra la dirección IP configurada en la interfaz de red. Es posible recuperarla de la siguiente manera:

```
PS    C:\Windows\system32>    $ifIndex3    =    Get-NetIPAddress    -
InterfaceAlias
Ethernet -AddressFamily IPv4
PS C:\Windows\system32> $ifIndex3.IPAddress
192.168.0.48
```

Ejemplo 2: recuperar las direcciones IP que se han configurado manualmente

```
PS C:\Windows\system32> Get-NetIPAddress -AddressFamily IPv4
-PrefixOrigin Manual -SuffixOrigin Manual


IPAddress         : 192.168.1.8
InterfaceIndex    : 4
InterfaceAlias    : Wi-Fi
AddressFamily     : IPv4
Type              : Unicast
PrefixLength      : 24
PrefixOrigin      : Manual
SuffixOrigin      : Manual
AddressState      : Preferred
ValidLifetime     : Infinite ([TimeSpan]::MaxValue)
PreferredLifetime: Infinite ([TimeSpan]::MaxValue)
SkipAsSource      : False
PolicyStore       : ActiveStore
```

El valor de la máscara de subred está definido por `PrefixLength`. Este se provee en formato CDIR (*Classless*

Inter-Domain Routing), que se corresponde con el número de bits de mayor peso utilizados en la máscara de subred. Así, en el ejemplo anterior, la configuración IP para la tarjeta de red Ethernet es 192.168.1.8/24.

He aquí algunas correspondencias entre `PrefixLength` y la máscara de subred:

- /24 es una máscara de subred 255.255.255.0,
- /16 es una máscara de subred 255.255.0.0,
- /8 es una máscara de subred 255.0.0.0.

Para que un equipo pueda comunicarse con otras redes, la interfaz de red debe tener configurada una puerta de enlace predeterminada. Sin embargo, esta dirección no puede recuperarse mediante el cmdlet **`Get-NetIPAddress`**. Se utiliza otro cmdlet, **`Get-NetRoute`**, cuya función es mostrar la tabla de enrutamiento, para buscar el valor de la puerta de enlace predeterminada.

Parámetro	Descripción
`-AddressFamily` `<AddressFamily[]>`	Especifica la versión IP que se ha de recuperar. Los posibles valores son: `IPv4`, `IPv6`.
`-DestinationPrefix` `<String[]>`	Especifica una o varias direcciones de red de destino.
`-InterfaceAlias` `<String[]>` **`-ifAlias`** `<String[]>`	Especifica uno o varios alias de interfaces de red.
`-InterfaceIndex` `<UInt32[]>` **`-ifIndex`** `<UInt32[]>`	Especifica uno o varios números de índice de interfaces de red.
`-NextHop` `<String[]>`	Especifica uno o varios valores que representan el próximo salto.

Ejemplo: recuperar la tabla de enrutamiento del puesto de trabajo

```
PS C:\Windows\system32> Get-NetRoute

ifIndex  DestinationPrefix            NextHop            RouteMetric
PolicyStore
-------  -----------------       -------       ----------- ----------
-
3          255.255.255.255/32        0.0.0.0                      256
ActiveStore
1          255.255.255.255/32        0.0.0.0                      256
ActiveStore
3          224.0.0.0/4                0.0.0.0                      256
ActiveStore
1          224.0.0.0/4                0.0.0.0                      256
ActiveStore
[...]
```

La tabla de enrutamiento indica todas las entradas configuradas de las distintas interfaces que posee el puesto de trabajo; a continuación, las que han podido configurarse automáticamente por el sistema, y por último aquellas que se han introducido manualmente.

Como su nombre indica, la puerta de enlace predeterminada configura la dirección IP hacia la que hay que transmitir los paquetes IP cuando no puede aplicarse ninguna otra ruta.

En la tabla de enrutamiento, la dirección de red de destino (`Destination Prefix`) para la puerta de enlace predeterminada se corresponde a: 0.0.0.0/0. Por consiguiente, especificando el número de índice de la interfaz y la dirección de red de destino por defecto, puede recuperar el valor IP de la puerta de enlace predeterminada (correspondiente a `NextHop`):

Ejemplo: recuperar la puerta de enlace predeterminada para la tarjeta Ethernet

```
PS C:\Windows\system32> Get-NetRoute -InterfaceAlias Ethernet
-DestinationPrefix 0.0.0.0/0

ifIndex  DestinationPrefix        NextHop                RouteMetric
PolicyStore
-------  -----------------     -------         ----------- ----------
-
3          0.0.0.0/0                 192.168.0.254                 0
ActiveStore
```

Es importante saber cómo recuperar la puerta de enlace predeterminada de una interfaz de red, pues esta información es necesaria para modificar los parámetros de configuración a continuación.

Para configurar una nueva dirección IP manualmente en la tarjeta de red, se utiliza el cmdlet **`New-NetIPAddress`**. Para

ello, necesita conocer la interfaz sobre la que se va a definir la nueva configuración, la nueva dirección IP que se va a aplicar, la máscara de subred en formato CDIR y, opcionalmente, la dirección IP de la puerta de enlace predeterminada. He aquí una tabla resumen de algunos de los parámetros disponibles para **New-NetIPAddress**:

Parámetro	Descripción
-DefaultGateway <String>	Especifica una dirección IPv4 o IPv6 para la puerta de enlace predeterminada.
-InterfaceAlias <String> **-ifAlias** <String>	Especifica uno o varios alias de interfaces de red.
-InterfaceIndex <UInt32> **-ifIndex** <UInt32>	Especifica uno o varios números de índice de interfaces de red.
-IPAddress <String>	Especifica la dirección IP que se va a crear. Esta puede ser de tipo IPv4 o IPv6.
-PrefixLength <Byte>	Especifica la máscara de subred.

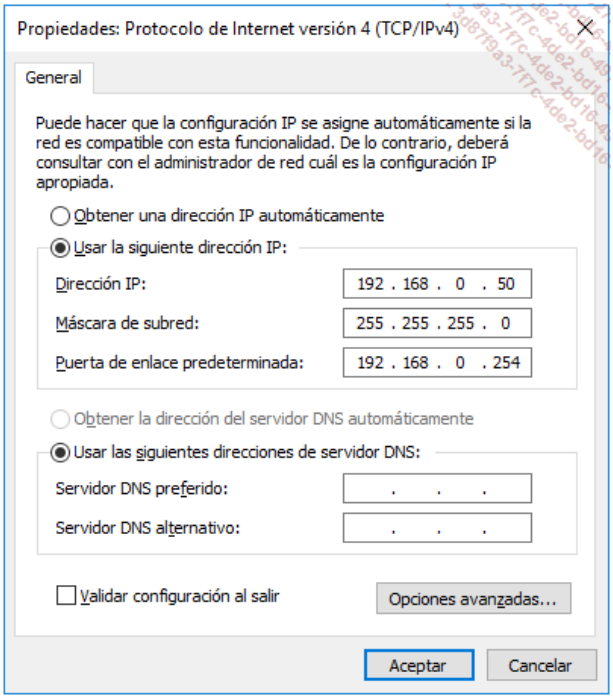
Importante: antes de ejecutar el cmdlet **New-NetIPAddress**, asegúrese de que la tarjeta de red está bien conectada: cable Ethernet conectado o Wi-Fi conectada a un punto de acceso. En caso contrario, obtendrá un error.

Ejemplo: configuración de una nueva dirección IP

```
PS C:\Windows\system32> New-NetIPAddress -IPAddress 192.168.0.50
-InterfaceAlias Ethernet -DefaultGateway 192.168.0.254
-PrefixLength 24


IPAddress         : 192.168.0.50
InterfaceIndex    : 3
InterfaceAlias    : Ethernet
AddressFamily     : IPv4
Type              : Unicast
PrefixLength      : 24
PrefixOrigin      : Manual
SuffixOrigin      : Manual
AddressState      : Tentative
ValidLifetime     : Infinite ([TimeSpan]::MaxValue)
PreferredLifetime: Infinite ([TimeSpan]::MaxValue)
```

```
SkipAsSource     : False
PolicyStore      : ActiveStore

IPAddress        : 192.168.0.50
InterfaceIndex   : 3
InterfaceAlias   : Ethernet
AddressFamily    : IPv4
Type             : Unicast
PrefixLength     : 24
PrefixOrigin     : Manual
SuffixOrigin     : Manual
AddressState     : Invalid
ValidLifetime    : Infinite ([TimeSpan]::MaxValue)
PreferredLifetime: Infinite ([TimeSpan]::MaxValue)
SkipAsSource     : False
PolicyStore      : PersistentStore
```

Ahora, la dirección IP está configurada manualmente en las propiedades del protocolo IPv4 de la tarjeta de red Ethernet:



Configuración de la dirección IP de la tarjeta de red Ethernet

Si ya existe una configuración IP introducida de forma manual en la tarjeta de red que desea configurar, se recomienda encarecidamente eliminar esta configuración (dirección IP, máscaras de subred y puerta de enlace predeterminada) antes de ejecutar el cmdlet **`New-NetIPAddress`**. Para realizar esta operación, utilice el cmdlet **`Remove-NetIPAddress`**.

Para eliminar la configuración IP de la tarjeta de red Ethernet, se utiliza el cmdlet **Remove-NetIPAddress**. Sin embargo, para eliminar correctamente todos los parámetros, hay que indicar la puerta de enlace predeterminada si esta existe.

En caso contrario, se elimina la configuración IP, pero la puerta de enlace predeterminada sigue estando configurada: hay que utilizar el comando **Remove-NetRoute** para eliminar la entrada correspondiente de la tabla de enrutamiento.

Remove-NetIPAddress recibe los mismos parámetros que su comando equivalente que permite configurar nuevas direcciones IP. He aquí un resumen de los parámetros:

Parámetro	Descripción
-DefaultGateway <String>	Especifica una dirección IPv4 o IPv6 para la puerta de enlace predeterminada.
-InterfaceAlias <String[]> **-ifAlias** <String[]>	Especifica uno o varios alias de interfaces de red.
-InterfaceIndex <UInt32[]> **-ifIndex** <UInt32[]>	Especifica uno o varios números de índice de interfaces de red.
-IPAddress <String[]>	Especifica la dirección IP que se ha de eliminar. Esta puede ser de tipo IPv4 o IPv6.
-PrefixLength <Byte[]>	Especifica la máscara de subred.

Ejemplo: eliminar la configuración IP configurada manualmente

```
PS C:\Windows\system32> Remove-NetIPAddress -IPAddress
192.168.0.50 -InterfaceAlias Ethernet -PrefixLength 24
-DefaultGateway 192.168.0.254 -Confirm:$false
```

Se ha eliminado la configuración que se ha realizado sobre la tarjeta de red Ethernet, pero el modo de configuración sigue siendo manual: si existe algún servidor DHCP en su red, la tarjeta Ethernet no intenta recuperar una nueva dirección.

Si se omite el parámetro **-DefaultGateway** en la ejecución del comando **Remove-NetIPAddress**, hay que utilizar **Remove-**

NetRoute para eliminar la dirección IP de la puerta de enlace predeterminada:

```
PS C:\Windows\system32> Get-NetRoute -InterfaceAlias Ethernet
-AddressFamily IPv4

ifIndex  DestinationPrefix              NextHop           RouteMetric
 PolicyStore
-------  -----------------        -------      -----------  --------
---
3          255.255.255.255/32         0.0.0.0                     256
 ActiveStore
3          224.0.0.0/4                 0.0.0.0                     256
 ActiveStore
3          0.0.0.0/0                     192.168.0.254           256
 ActiveStore


PS    C:\Windows\system32>    Remove-NetRoute    -DestinationPrefix
0.0.0.0/0
-InterfaceIndex 3 -Confirm:$false
```

Para modificar una dirección IP configurada en una interfaz de red (IPv4 o IPv6), debe utilizar los cmdlets **Remove-NetIPAddress** y **New-NetIPAddress**. No es posible cambiar una dirección IP sin utilizar estos dos comandos.

Por último, si simplemente desea volver a una configuración por DHCP de la tarjeta de red Ethernet, puede ejecutar el siguiente comando:

```
PS C:\Windows\system32> Set-NetIPInterface -InterfaceIndex 3
-Dhcp Enabled
```

b. Parametrizar la configuración DNS

Ya hemos explicado las acciones relativas a la configuración IP de una tarjeta de red, pero nos queda por abordar la configuración DNS. En efecto, para que los puestos de trabajo tengan en cuenta los nombres de los distintos elementos presentes en la red, hay que definir un servidor DNS en la configuración de la tarjeta de red Ethernet.

Para ello, debe utilizar los cmdlets dedicados a la administración del cliente DNS. Por esta razón, describiremos a continuación los cmdlets **Get-DnsClientServerAddress** y **Set-DnsClientServerAddress**.

Get-DnsClientServerAddress permite recuperar la configuración del cliente DNS en cada una de las interfaces (que

se han enumerado mediante el comando **Get-NetIPInterface**). He aquí los parámetros disponibles:

Parámetro	Descripción
-AddressFamily <AddressFamily[]>	Especifica la versión IP que se va a recuperar. Los posibles valores son: IPv4, IPv6.
-InterfaceAlias <String[]>	Especifica uno o varios alias de interfaces de red.
-InterfaceIndex <UInt32[]>	Especifica uno o varios números de índice de interfaces de red.

Ejemplo: obtener la lista de servidores DNS configurados en las interfaces de red

```
PS C:\Windows\system32> Get-DnsClientServerAddress

InterfaceAlias               Interface Address ServerDirecciones
                             Index     Family
--------------               --------- ------- ---------------
Ethernet                             3 IPv4    {}
Ethernet                               3 IPv6    {fec0:0:0:ffff::1,
fec0...
isatap.{A7B8971D-3013-427...         4 IPv4    {}
isatap.{A7B8971D-3013-427...           4 IPv6    {fec0:0:0:ffff::1,
fec0...
Loopback Pseudo-Interface 1          1 IPv4    {}
Loopback Pseudo-Interface 1          1 IPv6    {}
Teredo Tunneling Pseudo-I...         5 IPv4    {}
Teredo Tunneling Pseudo-I...         5 IPv6    {}
```

Como no se ha configurado ninguna dirección IP de servidor DNS en el puesto de trabajo, Windows asigna automáticamente tres direcciones IPv6 de servidores DNS. Estas tres direcciones son: FEC0:0:0:FFFF::1, FEC0:0:0:FFFF::2, FEC0:0:0:FFFF::3.

Para configurar manualmente las direcciones IP de los servidores DNS a través de Windows PowerShell, se utiliza el cmdlet **Set-DnsClientServerAddress**. Hay que especificar sobre qué interfaz de red se aplica el comando y a continuación escribir las direcciones IP de los servidores DNS.

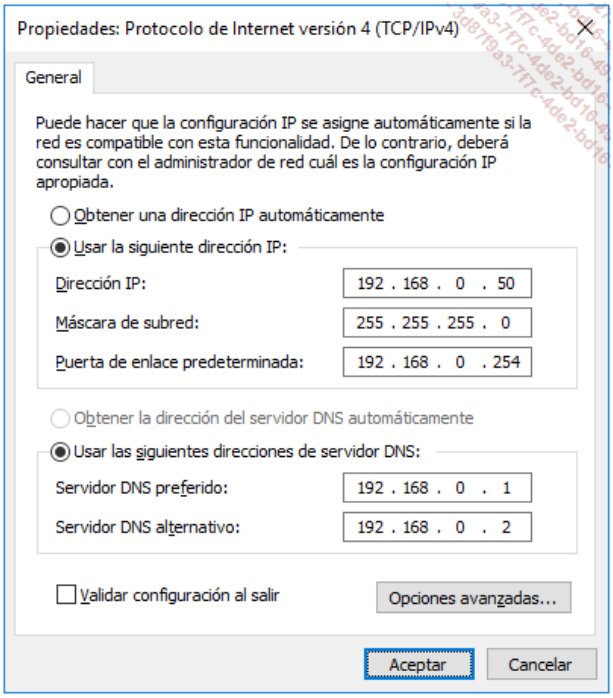
He aquí un resumen de los parámetros disponibles:

Parámetros	Descripción
-InterfaceAlias <String[]>	Especifica uno o varios alias de interfaces de red.
-InterfaceIndex <UInt32[]>	Especifica uno o varios números de índice de interfaces de red.
-ResetServerDirecciones	Reinicia las direcciones IP de los servidores DNS al valor por defecto.
-ServerDirecciones <String[]>	Especifica una lista de direcciones IP de los servidores DNS para la interfaz de red correspondiente.
-Validate	Valida que las direcciones IP se correspondan con los servidores DNS antes de configurarlas en la interfaz de red.

Ejemplo: configuración de dos servidores DNS para la tarjeta de red Ethernet

```
PS C:\Windows\system32> Set-DnsClientServerAddress -
InterfaceAlias
Ethernet -ServerDirecciones 192.168.0.1, 192.168.0.2
PS C:\Windows\system32> Get-DnsClientServerAddress -
InterfaceAlias
Ethernet -AddressFamily IPv4

InterfaceAlias         Interface Address ServerDirecciones
                       Index     Family
--------------         --------- ------- ---------------
Ethernet                                3 IPv4      {192.168.0.1,
192.168.0.2}
```

En la interfaz gráfica, se obtiene la siguiente configuración:



Configuración de las direcciones IP de los servidores DNS

Para terminar, veamos cómo reinicializar la configuración DNS cliente (en este caso, eliminar el conjunto de direcciones de los servidores DNS informados). Esto se lleva a cabo mediante el parámetro **`-ResetServerAddresses`**. Debe saber que, si durante su ejecución, la tarjeta de red está configurada con la opción **Obtener una dirección IP automáticamente**, entonces la sección DNS se configurará según la opción **Obtener la dirección del servidor DNS automáticamente**. En caso contrario, si se configura una dirección IP manualmente, entonces la sección DNS se borrará y la opción se mantendrá en **Usar las siguientes direcciones de servidor DNS**.

Ejemplo: eliminar las direcciones IP de los servidores DNS configurados

```
PS C:\Windows\system32> Get-DnsClientServerAddress

InterfaceAlias               Interface Address ServerDirecciones
                             Index     Family
--------------               --------- ------- ---------------
Ethernet                                   3 IPv4     {192.168.0.1,
192.168.0.2}
Ethernet                               3 IPv6     {}
isatap.{9104766D-7110-4BB...               6 IPv4     {192.168.0.1,
192.168.0.2}
isatap.{9104766D-7110-4BB...           6 IPv6     {}
Loopback Pseudo-Interface 1            1 IPv4     {}
```

```
Loopback Pseudo-Interface 1          1 IPv6    {}


PS C:\Windows\system32> Set-DnsClientServerAddress -InterfaceIndex
3
-ResetServerDirecciones
PS C:\Windows\system32> Get-DnsClientServerAddress

InterfaceAlias               Interface Address ServerDirecciones
                             Index     Family
--------------               --------- ------- ---------------
Ethernet                             3 IPv4    {}
Ethernet                               3 IPv6    {fec0:0:0:ffff::1,
fec0...
isatap.{9104766D-7110-4BB...         6 IPv4    {}
isatap.{9104766D-7110-4BB...           6 IPv6    {fec0:0:0:ffff::1,
fec0...
Loopback Pseudo-Interface 1          1 IPv4    {}
Loopback Pseudo-Interface 1          1 IPv6    {}
```

3. Resolución de nombres DNS

Para resolver el nombre de una máquina en la red y así obtener su dirección IP, puede utilizar el cmdlet **`Resolve-DnsName`**. Veremos en primer lugar un resumen de algunos parámetros:

Parámetro	Descripción
`-CacheOnly`	Resuelve la petición utilizando únicamente la caché local.
`-DnsOnly`	Resuelve la petición utilizando solamente el protocolo DNS. No se consideran los protocolos LLMNR y NetBIOS.
`-Name` `<String>`	Indica el nombre DNS o la dirección IP a resolver.
`-NoHostsFile`	No considera el archivo hosts para la resolución de la petición.
`-Server` `<String[]>`	Especifica el nombre o los nombres de los servidores DNS que se utilizan en la resolución de la petición.
`-Type` `<RecordType>`	Indica el tipo de petición DNS que debe entregarse. Por defecto, el tipo es A_AAAA: se consultan los registros A y AAAA. Los posibles valores de este parámetro son: `NS`, `MX`, `MF`, `CNAME`, `MB`, `PTR`, `SRV`, `DNAME`, etc.

Ejemplo 1: resolución del nombre DNS de un puesto de trabajo o servidor

```
PS C:\Windows\system32> Resolve-DnsName -Name SERVER01

Name                   Type   TTL   Section    IPAddress
----                   ----   ---   -------    ---------
SERVER01                            AAAA     30       Answer
fe80::1936:e3f0:ab7:a7c0
SERVER01               A      30    Answer     192.168.0.11
```

Los registros DNS de tipo A devuelven una dirección IPv4 de 32 bits, mientras que los registros de tipo AAAA devuelven una dirección IPv6 de 128 bits.

Puede utilizarse este mismo comando para resolver la dirección de un dominio de Internet.

Ejemplo 2: resolución del nombre DNS de una dirección de Internet

```
PS C:\Windows\system32> Resolve-DnsName www.bing.com

Name                           Type   TTL   Section    NameHost
----                           ----   ---   -------    --------
www.bing.com                             CNAME   978     Answer
any.edge.bing.com

Name      : any.edge.bing.com
QueryType : A
TTL       : 978
Section   : Answer
IP4Address: 204.79.197.200


Name                  : bing.com
QueryType             : SOA
TTL                   : 978
Section               : Authority
NameAdministrator     : msnhst.microsoft.com
SerialNumber          : 2014062401
TimeToZoneRefresh     : 1800
TimeToZoneFailureRetry: 900
TimeToExpiration      : 2419200
DefaultTTL            : 3600
```

Resolve-DnsName funciona también en el sentido opuesto, especificando una dirección IP, ya sea pública o privada.

Ejemplo 3: resolución de una dirección IP

```
PS C:\Windows\system32> Resolve-DnsName 192.168.0.11

Name                           Type   TTL   Section    NameHost
----                           ----   ---   -------    --------
11.0.168.192.in-addr.arpa.     PTR    1200  Question   SERVER01
```

El registro de tipo PTR es un puntero que devuelve el nombre de máquina (FQDN) de una dirección IP concreta.

Pero **Resolve-DnsName** no termina aquí… Este cmdlet ofrece una multitud de parámetros que podrán ayudarle a encontrar el problema cuando algún puesto de trabajo no resuelva correctamente los nombres DNS.

Ejemplo 4: encontrar la causa de la no-resolución de un nombre de máquina

En el caso de una máquina inalcanzable, cuando el puesto de trabajo no sea capaz de resolver el nombre de la máquina remota con la dirección IP correcta, **Resolve-DnsName** puede ayudarnos a encontrar la causa.

Ejemplo con el parámetro **-NoHostsFile**.

```
PS C:\Windows\system32> Resolve-DnsName -Name W7

Name                                           Type    TTL    Section
 IPAddress
----                                        ----    ---    -------     ------
---
W7                                             A       80280 Answer
192.168.1.19


PS C:\Windows\system32> Resolve-DnsName -Name W7 -NoHostsFile

Name                                           Type    TTL    Section
 IPAddress
----                                        ----    ---    -------     ------
---
W7.CONTOSO.LOCAL                               A       1200   Answer
192.168.1.7
```

Cuando el comando **Resolve-DnsName** no toma en consideración el archivo hosts del puesto de trabajo, entonces se resuelve otra dirección IP. De modo que conviene echar un vistazo al archivo hosts y actualizarlo o limpiar las entradas incorrectas.

Con la ayuda de los distintos parámetros de **Resolve-DnsName**, resulta más sencillo diagnosticar un problema de resolución de nombre DNS en un puesto de trabajo.

4. Consultar la caché DNS

Cada puesto de trabajo conserva una caché de direcciones DNS y de su dirección IP asociada, con el objetivo de no volver a enviar

una petición nueva al servidor DNS para una dirección o nombre que ya se haya consultado previamente. Esta caché puede consultarse mediante el cmdlet **Get-DnsClientCache**, que devuelve una lista de correspondencias entre los nombres de dominio y las respectivas direcciones IP.

He aquí los parámetros disponibles de **Get-DnsClientCache** que permitirán filtrar la petición:

Parámetro	Descripción
-Data <String[]>	Devuelve los registros de la columna Data.
-DataLength <UInt16[]>	Devuelve los registros de la columna DataLength.
-Entry <String[]>	Devuelve los registros de la columna Entry.
-Name <String[]>	Devuelve los registros de la columna RecordName.
-Section <Section[]>	Devuelve los registros de la columna Section. Los posibles valores son: Answer, Authority, Additional.
-Status <Status[]>	Devuelve los registros de la columna Status. Los posibles valores son: Success, NotExist, NoRecords.
-TimeToLive <UInt32[]>	Devuelve los registros de la columna TimeToLive, en segundos.
-Type <Type[]>	Devuelve los registros de la columna RecordType. Los posibles valores son: A, NS, CNAME, SOA, PTR, MX, AAAA, SRV.

Ejemplo 1: recuperar la caché DNS

```
PS C:\Windows\system32> Get-DnsClientCache | Format-List


Entry     : technet.microsoft.com
RecordName: technet.microsoft.com
RecordType: CNAME
Status    : Success
```

```
Section    : Answer
TimeToLive: 602
DataLength: 8
Data       : technet.microsoft.akadns.net

Entry      : technet.microsoft.com
RecordName: technet.microsoft.akadns.net
RecordType: A
Status     : Success
Section    : Answer
TimeToLive: 602
DataLength: 4
Data       : 157.56.148.23

Entry      : i4.services.social.microsoft.com
RecordName: i4.services.social.microsoft.com
RecordType: CNAME
Status     : Success
Section    : Answer
TimeToLive: 646
DataLength: 8
Data       : i.s1.social.ms.akadns.net

Entry      : i4.services.social.microsoft.com
RecordName: i.s1.social.ms.akadns.net
RecordType: CNAME
Status     : Success
Section    : Answer
TimeToLive: 646
DataLength: 8
Data       : a1363.g.akamai.net

Entry      : i4.services.social.microsoft.com
RecordName: a1363.g.akamai.net
RecordType: A
Status     : Success
Section    : Answer
TimeToLive: 646
DataLength: 4
Data       : 2.16.117.72

Entry      : i4.services.social.microsoft.com
RecordName: a1363.g.akamai.net
RecordType: A
Status     : Success
Section    : Answer
TimeToLive: 646
DataLength: 4
Data       : 2.16.117.75

[...]
```

Puede ver el resultado en formato tabla sin utilizar el cmdlet **`Format-List`**.

Ejemplo 2: búsqueda filtrada en la caché DNS

Se trata de filtrar las entradas que contienen bing.com, con un registro de tipo A y cuyo estado sea correcto.

```
PS C:\Windows\system32> Get-DnsClientCache -Entry *bing.com
-Type A -Status Success | Format-List


Entry     : www.bing.com
RecordName: any.edge.bing.com
RecordType: A
Status    : Success
Section   : Answer
TimeToLive: 455
DataLength: 4
Data      : 204.79.197.200

Entry     : ssl.bing.com
RecordName: a-0001.a-msedge.net
RecordType: A
Status    : Success
Section   : Answer
TimeToLive: 229
DataLength: 4
Data      : 204.79.197.200
```

- Puede utilizar el carácter comodín * (wildcard) para realizar una búsqueda extendida si no conoce exactamente la dirección buscada.

Vaciar la caché DNS

En determinadas situaciones (por ejemplo, tras el cambio de dirección IP de un servidor), o si ha diagnosticado que la caché DNS de un puesto de trabajo plantea algún problema, puede vaciar la caché local para que se reconfigure correctamente con nuevos datos.

Para vaciar y reinicializar la caché, existe un cmdlet específico para llevar a cabo esta funcionalidad: **`Clear-DnsClientCache`**. Su uso es muy sencillo, pues no requiere ningún parámetro:

```
PS C:\Windows\system32> Clear-DnsClientCache
```

- Las entradas precargadas del archivo hosts no se eliminan. Para retirarlas, edite el archivo hosts y lleve a cabo las modificaciones correspondientes.

5. Renovar la inscripción del cliente DNS

Para renovar manualmente la inscripción dinámica de los nombres DNS y de las direcciones IP configuradas en el equipo, se utiliza el

cmdlet **Register-DnsClient**. Este comando permite resolver el problema de un nombre DNS que no se haya podido inscribir o algún problema de actualización dinámica entre un puesto de trabajo y el servidor o los servidores DNS configurados, sin necesidad de reiniciar el equipo.

No se utiliza ningún parámetro específico con este cmdlet:

```
PS C:\Windows\system32> Register-DnsClient
```

Un puesto de trabajo puede tener una o varias tarjetas de red, de modo que **Register-DnsClient** provoca el registro del conjunto de direcciones IP presentes en el conjunto de tarjetas.

6. Las conexiones VPN

Las conexiones VPN le permiten establecer un vínculo seguro con la red de su empresa pasando por Internet. Tanto si se encuentra en la sede de algún cliente o en cualquier otro lugar equipado con una conexión a Internet, tiene la posibilidad de conectarse a la red de su empresa a través de una conexión VPN, y así acceder al conjunto de puestos de trabajo y servidores.

Windows PowerShell permite gestionar las conexiones VPN y esto es lo que veremos a continuación.

a. Crear una conexión VPN

Es posible crear una conexión VPN en un puesto de trabajo por línea de comando Windows PowerShell de manera muy simple utilizando **Add-VpnConnection**.

Debe configurarse el cliente VPN en función de la configuración del servidor VPN: método de autenticación, encriptación…

Para hacerse una idea de los parámetros, he aquí una tabla resumen:

Parámetro	Descripción

Parámetro	Descripción
-AllUserConnection	Indica que la conexión VPN se crea en la libreta de teléfonos global. Esta conexión será visible para todos los usuarios del puesto de trabajo.
-AuthenticationMethod <String[]>	Especifica el método de autenticación que se ha de utilizar para la conexión VPN. Los posibles valores son: PAP, CHAP, MSCHAPv2, EAP.
-DnsSuffix <String>	Especifica el sufijo DNS para la conexión VPN.
-EncryptionLevel <String>	Especifica el nivel de encriptación para la conexión VPN. Los posibles valores son: NoEncryption, Optional, Required, Maximum.
-Force	Indica que la clave precompartida (PSK) se provee a través de un canal no seguro si se utiliza el protocolo L2TP.
-IdleDisconnectSeconds <UInt32>	Indica el tiempo de inactividad, en segundos, antes de que la conexión VPN se corte.
-L2tpPsk <String>	Especifica el valor de la clave precompartida (PSK) que se va a utilizar para la autenticación L2TP. Si no se especifica este parámetro, se utiliza un certificado para L2TP.
-Name <String>	Indica el nombre de la conexión VPN.
-RememberCredential	Indica que se guardan las credenciales de la conexión VPN.

Parámetro	Descripción
-ServerAddress <String>	Indica la dirección del servidor VPN. Esta dirección puede ser de tipo URL, IPv4 o IPv6.
-SplitTunneling	Activa la característica Split Tunneling. Por defecto, está desactivada.
-TunnelType <String>	Especifica el tipo de túnel utilizado para la conexión VPN. Los posibles valores son: PPTP, L2TP, SSTP, IKEv2, Automatic.
-UseWinlogonCredential	Indica que el método de autenticación es de tipo MSCHAPv2 o EAP MSCHAPv2, y que se utiliza automáticamente la autenticación Windows cuando se establece la conexión VPN.

He aquí algunos ejemplos de creación de conexión VPN con los distintos parámetros vistos anteriormente.

Ejemplo 1:

Un primer ejemplo, únicamente con los parámetros **-Name** y **-ServerAddress**, muestra los distintos parámetros que se proveen por defecto durante la creación de una conexión VPN sin definir el modo de autenticación, el nivel de encriptación o incluso el tipo de túnel.

```
PS C:\Users\Admin> Add-VpnConnection -Name myVPN -ServerAddress
172.16.1.1
PS C:\Users\Admin> Get-VpnConnection -Name myVPN


Name                  : myVPN
ServerAddress         : 172.16.1.1
AllUserConnection     : False
Guid                  : {E85305E5-1D2C-4135-AADC-C0C74FA05B41}
TunnelType            : Automatic
AuthenticationMethod  : {MsChapv2}
EncryptionLevel       : Optional
L2tpIPsecAuth         : Certificate
UseWinlogonCredential: False
EapConfigXmlStream    :
ConnectionStatus      : Disconnected
```

```
RememberCredential   : False
SplitTunneling       : False
DnsSuffix            :
IdleDisconnectSeconds: 0
```

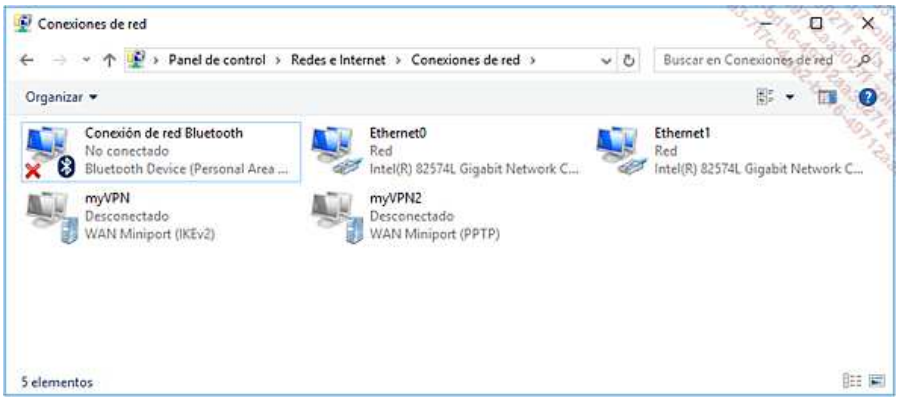
Ejemplo 2:

Se muestra un segundo ejemplo, que utiliza el método de autenticación de tipo MSCHAPv2.

```
PS C:\Users\Admin> Add-VpnConnection -Name "myVPN 2"
-ServerAddress 172.16.1.1 -TunnelType PPTP -EncryptionLevel
Required -AuthenticationMethod MSChapv2 -UseWinlogonCredential
-SplitTunneling -AllUserConnection -RememberCredential
PS C:\Windows\system32> Get-VpnConnection -Name "myVPN 2"
-AllUserConnection


Name                 : myVPN 2
ServerAddress        : 172.16.1.1
AllUserConnection    : True
Guid                 : {C4606643-CBCB-4661-81D1-A0FC8C8E1F74}
TunnelType           : Pptp
AuthenticationMethod : {MsChapv2}
EncryptionLevel      : Required
L2tpIPsecAuth        :
UseWinlogonCredential: True
EapConfigXmlStream   :
ConnectionStatus     : Disconnected
RememberCredential   : True
SplitTunneling       : True
DnsSuffix            :
IdleDisconnectSeconds: 0
```

Este comando crea una nueva conexión VPN llamada myVPN 2. Esta conexión utiliza el método de autenticación MSCHAPv2 (**-AuthenticationMethod**). Los parámetros suplementarios permiten:

- Utilizar una autenticación Windows (**-UseWinLoginCredential**).
- Utilizar la característica Split Tunneling (**-SplitTunneling**).
- Registrar la conexión VPN en la libreta de teléfonos global.
- La conexión estará accesible para el conjunto de usuarios del puesto de trabajo (**-AllUserConnection**).
- Poner en caché la información de autenticación una vez establecida la primera conexión (**-RememberCredential**).



Las conexiones VPN se crean y están presentes en Conexiones de red

Como hemos visto en los dos ejemplos anteriores, el cmdlet **`Get-VpnConnection`** permite recuperar la información detallada de cada una de las conexiones VPN presentes en el puesto de trabajo. He aquí los parámetros:

Parámetro	Descripción
`-AllUserConnection`	Indica que la conexión VPN se recupera desde la libreta de teléfonos global.
`-Name` `<String[]>`	Especifica uno o varios nombres de conexiones VPN.

b. Modificar una conexión VPN

`Set-VpnConnection` permite actualizar una conexión VPN existente. Este cmdlet utiliza los mismos parámetros que **`Add-VpnConnection`**: basta con indicar el nombre de la conexión VPN que quiere modificar y agregar los parámetros con los nuevos valores. A modo de recordatorio, he aquí una tabla resumen de los parámetros:

Parámetros	Descripción
`-AllUserConnection`	Indica que la conexión VPN está accesible para todos los usuarios del puesto de trabajo.

Parámetros	Descripción
-AuthenticationMethod <String[]>	Especifica el método de autenticación que se ha de utilizar para la conexión VPN. Los posibles valores son: PAP, CHAP, MSCHAPv2, EAP.
-DnsSuffix <String>	Especifica el sufijo DNS para la conexión VPN.
-EncryptionLevel <String>	Especifica el nivel de encriptación para la conexión VPN. Los posibles valores son: NoEncryption, Optional, Required, Maximum.
-Force	Indica que la clave precompartida (PSK) se provee a través de un canal no seguro, si se utiliza el protocolo L2TP.
-IdleDisconnectSeconds <UInt32>	Indica el tiempo de inactividad, en segundos, antes de que se corte la conexión VPN.
-L2tpPsk <String>	Especifica el valor de la clave precompartida (PSK) que se ha de utilizar para la autenticación L2TP. Si no se especifica este parámetro, se utiliza un certificado para L2TP.
-Name <String>	Indica el nombre de la conexión VPN.

Parámetros	Descripción
-RememberCredential <Boolean>	Indica que se guardan las credenciales de la conexión VPN.
-ServerAddress <String>	Indica la dirección del servidor VPN. Esta dirección puede ser de tipo URL, IPv4 o IPv6.
-SplitTunneling <Boolean>	Activa la característica Split Tunneling. Por defecto, está desactivada.
-TunnelType <String>	Especifica el tipo de túnel utilizado para la conexión VPN. Los posibles valores son: PPTP, L2TP, SSTP, IKEv2, Automatic.
-UseWinlogonCredential <Boolean>	Indica que el método de autenticación es de tipo MSCHAPv2 o EAP MSCHAPv2, y que se utiliza automáticamente la autenticación Windows cuando se establece la conexión VPN.

Ejemplo: modificar una conexión VPN existente

```
PS C:\Windows\system32> Set-VpnConnection -Name myVPN -
TunnelType
l2tp -EncryptionLevel Required -AuthenticationMethod EAP
-SplitTunneling:$true -L2tpPsk "P@$$w0rd" -Force
-RememberCredential:$true
PS C:\Windows\system32> Get-VpnConnection myVPN


Name                  : myVPN
ServerAddress         : 172.16.1.1
AllUserConnection     : False
Guid                  : {E85305E5-1D2C-4135-AADC-C0C74FA05B41}
TunnelType            : L2tp
AuthenticationMethod : {Eap}
```

```
EncryptionLevel      : Required
L2tpIPsecAuth        : Psk
UseWinlogonCredential: False
EapConfigXmlStream   : #document
ConnectionStatus     : Disconnected
RememberCredential   : True
SplitTunneling       : True
DnsSuffix            :
IdleDisconnectSeconds: 0
```

En lo sucesivo, la conexión myVPN tendrá en cuenta el método de autenticación EAP (**-AuthenticationMethod**). El tipo de túnel utilizado pasa a ser L2TP (**-TunnelType**), y no se utiliza certificado, de modo que hay que precisar el valor de la clave precompartida (PSK) para la autenticación L2TP (**-L2tpPsk**). Los parámetros adicionales permiten:

- Activar la característica Split Tunneling (**-SplitTunneling:$true**).
- Activar la copia de seguridad de las credenciales tras la primera conexión con éxito (**-RememberCredential:$true**).

c. Eliminar una conexión VPN

El cmdlet **Remove-VpnConnection** permite eliminar cualquier conexión VPN configurada en Windows. Para ello, hay que indicar el nombre de la conexión VPN y, si está presente en la libreta de teléfonos global, especificarlo con el parámetro correspondiente.

Parámetro	Descripción
-AllUserConnection	Indica que la conexión VPN presente en la libreta de teléfonos global se eliminará.
-Name <String[]>	Especifica uno o varios nombres de conexiones VPN.

Ejemplo 1: eliminar una conexión VPN

```
PS C:\Windows\system32> Remove-VpnConnection -Name myVPN

Confirmar
Deleting VPN connection myVPN. Do you want to continue?
[S] Sí  [N] No  [U] Suspender  [?] Ayuda (el valor por defecto
es 'S'): S
```

Ejemplo 2: eliminar una conexión VPN presente en la libreta de teléfonos global

```
PS C:\Windows\system32> Remove-VpnConnection -Name "myVPN 2"
-AllUserConnection -Force
```

La gestión de discos

Es posible administrar discos a través de cmdlets Windows PowerShell desde Windows 8. En Windows 7, puede utilizar el ejecutable diskpart.exe, que permite realizar la mayoría de las acciones que se describen en esta sección.

La gestión de discos, elementos imprescindibles para almacenar datos, es primordial en el seno de Windows. Windows PowerShell permite administrar los discos duros físicos y también los discos duros virtuales.

He aquí los cmdlets que se estudiarán:

- **`Get-Disk`**: devuelve la lista de los discos visibles por el sistema operativo.
- **`Get-Partition`**: devuelve una lista de todas las particiones visibles.
- **`Resize-Partition`**: permite redimensionar el tamaño de una partición.
- **`Get-ParticionesupportSize`**: recupera los tamaños mínimo y máximo soportados por la partición.
- **`New-Partition`**: crea una nueva partición.
- **`Format-Volume`**: permite dar formato a una unidad.
- **`Add-PartitionAccessPath`**: agrega una letra de unidad a una partición.
- **`Remove-PartitionAccessPath`**: elimina la letra de unidad de una partición.
- **`Initialize-Disk`**: permite inicializar un disco.
- **`Set-Disk`**: permite realizar acciones sobre el disco (cambiar la tabla de particiones, configurar un disco fuera de conexión o de solo lectura).
- **`Clear-Disk`**: elimina la configuración del disco (¡y los datos!).
- **`Optimize-Volume`**: permite optimizar el rendimiento de las unidades (desfragmentación de los datos, TRIM).

- **Repair-Volume**: permite buscar y corregir los errores del sistema de archivos en las unidades.
- **Remove-Partition**: elimina una partición.
- **New-VHD**: crea un nuevo disco duro virtual.
- **Mount-VHD**: monta un disco duro virtual.
- **Optimize-VHD**: compacta los datos de un disco duro virtual de tipo extensión dinámica (recuperar el espacio en disco).
- **Convert-VHD**: convierte un disco duro virtual (de VHD a VHDX o de tamaño fijo a extensión dinámica).
- **Resize-VHD**: permite redimensionar el tamaño máximo de un disco duro virtual.

1. Los discos duros

Los discos duros se corresponden con todos los tipos de almacenamiento presentes y conectados al equipo. Estos discos pueden ser discos duros (físicos o virtuales) y también discos extraíbles: llaves USB, tarjetas SDHC/SDXC, etc.

El cmdlet **Get-Disk** permite recuperar el conjunto de discos duros conectados al equipo que son visibles por el sistema operativo Windows.

Parámetro	Descripción
-FriendlyName <String[]>	Recupera el disco especificado por su nombre.
-Number <UInt32[]>	Recupera el disco por su número.

Ejemplo: recuperar los discos conectados al puesto de trabajo

```
PS C:\Windows\system32> Get-Disk | Format-List
Number,FriendlyName,OperationalStatus,Size,PartitionStyle


Number           : 0
FriendlyName     : WDC WD2500BEKT-75A25T0
OperationalStatus: Online
Size             : 250063733391
PartitionStyle   : MBR

Number           : 1
FriendlyName     : Kingston DataTraveler G2 USB Device
OperationalStatus: Online
```

```
Size              : 4005057003
PartitionStyle    : MBR
```

En este ejemplo, existen dos discos, que están conectados al equipo. Observe que cada disco tiene asignado un número: el disco 0 se corresponde con el disco duro principal del equipo, el disco 1 es una llave USB.

Esta numeración es importante, pues en las siguientes operaciones (formato, gestión de las particiones, etc.) habrá que indicar el número de disco sobre el que se llevará a cabo la acción. Habrá que tener mucha precaución con las manipulaciones, pues existe el riesgo de perder los datos.

En un disco duro, puede haber una o varias particiones. Para visualizar el conjunto de particiones en el conjunto de discos, se utiliza el comando **`Get-Partition`**.

Parámetro	Descripción
`-DiskNumber` `<UInt32[]>`	Especifica el número del disco.
`-DriveLetter` `<Char[]>`	Especifica la letra de la unidad de la partición o del volumen.
`-PartitionNumber` `<UInt32[]>`	Especifica el número de la partición.

Ejemplo: mostrar el conjunto de particiones presentes en el puesto de trabajo

```
PS C:\Windows\system32> Get-Partition

    Disk Number: 0

PartitionNumber  DriveLetter Offset                 Size Type
---------------  ----------- ------                 ---- ----
1                            1048576              500 MB IFS
2                C           525336576          81.51 GB IFS


   Disk Number: 1

PartitionNumber  DriveLetter Offset                 Size Type
---------------  ----------- ------                 ---- ----
1                G           32256                3.73 GB FAT32
```

a. Redimensionar una partición

Para redimensionar una partición, es decir, para reducir o aumentar el tamaño de una partición existente sin tener que

eliminarla, utilice el cmdlet **Resize-Partition**. Hay que especificar la partición que se quiere modificar e indicar el nuevo tamaño deseado.

Parámetro	Descripción
-DiskNumber <UInt32[]>	Especifica el número del disco.
-DriveLetter <Char[]>	Especifica la letra de la unidad de la partición o del volumen.
-PartitionNumber <UInt32[]>	Especifica el número de la partición.
-Size <UInt64>	Especifica el nuevo tamaño de la partición. Las unidades de valores posibles son: Bytes (por defecto), KB, MB, GB, TB.

Ejemplo: redimensionar una partición

En este ejemplo, la partición número 3 del disco 0 se redimensiona a 100 GB.

```
PS C:\Windows\system32> Resize-Partition -DiskNumber 0
-PartitionNumber 3 -Size 100GB
```

Pero ¿cómo saber el tamaño de la partición mínimo o máximo que puede configurarse en una partición existente? Si desea redimensionar una partición para que sea lo más grande posible o, a la inversa, lo más pequeña posible, entonces es necesario recuperar previamente este valor límite, valor definido de forma automática por el sistema.

Para ello, se utiliza el cmdlet **Get-PartitionsupportedSize**.

Parámetros	Descripción
-DiskNumber <UInt32[]>	Especifica el número del disco.
-DriveLetter <Char[]>	Especifica la letra de la unidad de la partición o del volumen.

Parámetros	Descripción
-PartitionNumber <UInt32[]>	Especifica el número de la partición.

Ejemplo: recuperar los valores de los tamaños mínimo y máximo soportados

```
PS C:\Windows\system32> Get-PartitionsupportedSize -DiskNumber 0
-PartitionNumber 3

    SizeMin      SizeMax
    -------      -------
74232483840 135996112896
```

Así, resulta muy sencillo modificar el tamaño de una partición asignándole uno u otro valor extremo.

Los valores devueltos están expresados en bytes.

Ejemplo: redimensionar una partición a su tamaño máximo

```
PS C:\Windows\system32> $size = Get-PartitionsupportedSize
-DiskNumber 0 -PartitionNumber 3
PS C:\Windows\system32> Resize-Partition -DiskNumber 0
-PartitionNumber 3 -Size $size.SizeMax
```

b. Crear una nueva partición

Para crear una nueva partición se utiliza el cmdlet **New-Partition**. Basta con indicar el número de disco sobre el que se creará la nueva partición y, por supuesto, definir el tamaño deseado.

Parámetro	Descripción
-AssignDriveLetter	Asigna automáticamente una letra de unidad a la nueva partición.
-DiskNumber <UInt32[]>	Especifica el número del disco.
-DriveLetter <Char>	Asigna una letra de unidad específica a la nueva partición.

Parámetro	Descripción
-Size <UInt64>	Especifica el tamaño de la nueva partición. Las posibles unidades de valores son: Bytes (por defecto), KB, MB, GB, TB.
-UseMaximumSize	Crea la partición con el tamaño más grande posible en el disco especificado.

Ejemplo 1: crear una partición con el mayor tamaño posible

```
PS C:\Windows\system32> New-Partition -DiskNumber 0
-UseMaximumSize
```

En el ejemplo anterior, no se asigna ninguna letra de unidad una vez creada la partición. El segundo ejemplo resuelve esta carencia.

Ejemplo 2: crear una partición con un tamaño y una letra de unidad concretos

```
PS C:\Windows\system32> New-Partition -DiskNumber 0 -Size 100GB
-DriveLetter F
```

Pero tenga precaución, **New-Partition** se contenta únicamente con crear una partición. Una vez realizada la operación, la partición no está todavía lista para alojar datos, pues carece de formato.

c. Formatear una partición

Format-Volume permite formatear particiones (y también volúmenes). Puede servir para formatear particiones existentes o recién creadas tras el uso de **New-Partition**.

Para indicar la partición que se ha de formatear, existen dos posibilidades:

- Bien especificando directamente la letra de unidad.
- Bien pasando la instancia CIM que se obtiene con **Get-Partition** o **New-Partition**.

Format-Volume admite varios parámetros que conviene conocer. Estos parámetros se deben utilizar dependiendo de las necesidades y los objetivos buscados:

Parámetro	Descripción
-Compress	Activa la compresión en volúmenes NTFS.
-DriveLetter <Char[]>	Indica la unidad que se ha de formatear. Si algún volumen existente ya posee una letra de unidad, puede especificarla directamente para indicar la unidad que se ha de formatear.
-FileSystem <String>	Indica con qué sistema de archivos se formateará la unidad. Los posibles valores son: NTFS, exFAT, FAT32, FAT.
-FileSystemLabel <String[]>	Indica el volumen que se ha de formatear mediante su etiqueta.
-Full	Realiza un formateo completo. Un formateo completo escribe en cada sector del disco duro y lleva mucho más tiempo que el formateo rápido. Este último se utiliza por defecto si no se especifica este parámetro.
-NewFileSystemLabel <String>	Especifica la nueva etiqueta que se asignará al volumen.
-Partition <CimInstance>	Indica la partición sobre la que se realiza el formateo. Es posible recuperar el objeto CIM mediante los cmdlets **Get-Partition** y **New-Partition**.

Ejemplo 1: formatear una partición

```
PS C:\Windows\system32> $partition = Get-Partition -DiskNumber 0

-PartitionNumber 4
PS C:\Windows\system32> Format-Volume -Partition $partition
```

```
-Confirm:$false | Format-List DriveLetter,DriveType,FileSystem,
FileSystemLabel,HealthStatus,OperationalStatus,Path,Size,SizeRemai
ning


DriveLetter       :
DriveType         : Fixed
FileSystem        : NTFS
FileSystemLabel   :
HealthStatus      : Healthy
OperationalStatus : OK
Path                        :  \\?\Volume{572550c4-61d9-11e4-826b-
00155db46f0a}\
Size              : 61761629716
SizeRemaining     : 61654255534
```

Por defecto, un nuevo volumen formateado tiene como sistema de archivos NTFS. Por supuesto, es posible seleccionar otro sistema de archivos gracias al parámetro **-FileSystem** (consulte la tabla anterior). Además, **Format-Volume** envía sistemáticamente una petición de confirmación antes de ejecutar el comando, salvo si se indica expresamente lo contrario, como en el ejemplo, a través del parámetro **-Confirm:$false**.

Ejemplo 2: crear y formatear una partición en una sola línea de comando

```
PS    C:\Windows\system32>    New-Partition    -DiskNumber    0    -
UseMaximumSize
-DriveLetter E | Format-Volume -NewFileSystemLabel "DATA"
-Confirm:$false | Format-List DriveLetter,DriveType,FileSystem,
FileSystemLabel,HealthStatus,OperationalStatus,Path,Size,SizeRemai
ning


DriveLetter       : E
DriveType         : Fixed
FileSystem        : NTFS
FileSystemLabel   : DATA
HealthStatus      : Healthy
OperationalStatus : OK
Path                        :  \\?\Volume{13e3f755-6663-11e4-826b-
00155db46f0a}\
Size              : 59687038976
SizeRemaining     : 59581702144
```

Observe que, en función del tamaño de la partición y del volumen que se ha de formatear, no estarán disponibles algunos tipos de sistemas de archivos. En estos casos, obtendrá un mensaje de error explicando el motivo del rechazo.

d. Asignar una letra de unidad a una partición

Para acceder a las unidades de datos, es necesario utilizar una letra de unidad. Para asignar una nueva letra a un volumen, puede utilizar el cmdlet **Add-PartitionAccessPath**.

Parámetro	Descripción
-AccessPath <String>	Asigna una letra de unidad específica a la partición.
-AssignDriveLetter	Asigna la siguiente letra de unidad disponible a la partición.
-DiskNumber <UInt32[]>	Especifica el número del disco.
-PartitionNumber <UInt32[]>	Especifica el número de la partición.

Ejemplo 1: asignar una letra de unidad automáticamente

```
PS C:\Windows\system32> Add-PartitionAccessPath -DiskNumber 0
-PartitionNumber 4 -AssignDriveLetter
```

En el ejemplo anterior, no se indica ninguna letra de unidad, pues el parámetro **-AssignDriveLetter** asigna automáticamente la primera letra de unidad disponible por orden alfabético. Si desea asignar una letra de unidad específica, debe utilizar el parámetro **-AccessPath** seguido de la letra de unidad correspondiente (en formato X: o X:\).

Ejemplo 2: asignar una letra de unidad específica a una partición

```
PS C:\Windows\system32> Add-PartitionAccessPath -DiskNumber 0
-PartitionNumber 4 -AccessPath E:\
```

Una partición(o volumen) solo puede tener una letra de unidad. Si repite este comando, o bien si ya existe una letra de unidad asignada al volumen correspondiente, Windows PowerShell le devuelve un error.

e. Eliminar la letra de unidad de una partición

De manera inversa al cmdlet **Add-PartitionAccessPath**, **Remove-PartitionAccessPath** sirve para eliminar la letra de unidad de las particiones.

Parámetro	Descripción
-AccessPath <String>	Indica la letra de unidad que se ha de eliminar (en formato X: o X:\).
-DiskNumber <UInt32[]>	Especifica el número del disco.
-PartitionNumber <UInt32[]>	Especifica el número de la partición.

Hay que indicar la partición sobre la que se realizará la operación y, a continuación, especificar la letra de unidad que se ha de eliminar mediante el parámetro **-AccessPath**.

Ejemplo: eliminar la letra de unidad de una partición

```
PS C:\Windows\system32> Remove-PartitionAccessPath -DiskNumber 0
-PartitionNumber 4 -AccessPath E:\
```

f. Inicializar un disco

Cuando agrega un nuevo disco duro físico, o un disco duro virtual creado recientemente, es preciso inicializar el disco. Esta inicialización tiene como objetivo crear una tabla de particiones de tipo MBR (*Master Boot Record*) o GPT (*GUID Partition Table*).

En este caso concreto, esta inicialización es la primera etapa obligatoria para utilizar el nuevo disco. Se lleva a cabo mediante el cmdlet **Initialize-Disk**.

Parámetro	Descripción
-FriendlyName <String[]>	Especifica el nombre del disco que se ha de inicializar.
-Number <UInt32[]>	Especifica el número del disco que se ha de inicializar.
-PartitionStyle <PartitionStyle>	Indica el tipo de tabla de particiones que se ha de crear. Los posibles valores son: MBR, GPT.

Ejemplo: inicializar un disco

```
PS C:\Windows\system32> Initialize-Disk -Number 1
PS   C:\Windows\system32>   Get-Disk   -Number   1   |   Format-List
DiskNumber,
PartitionStyle,ProvisioningType,OperationalStatus,HealthStatus,Bus
Type,
FirmwareVersion,FriendlyName,Guid,Manufacturer,Model,Number,
NumberOfPartition,SerialNumber,Size


DiskNumber        : 1
PartitionStyle    : GPT
ProvisioningType  : Thin
OperationalStatus: Online
HealthStatus      : Healthy
BusType           : SAS
FirmwareVersion   : 1.0
FriendlyName      : Msft Virtual Disk
Guid              : {33fc835a-4a31-46cb-80d3-c2a33adbc4c2}
Manufacturer      : Msft
Model             : Virtual Disk
Number            : 1
SerialNumber      :
Size              : 136365211648
```

Por defecto, si no se indica el parámetro **-PartitionStyle**, la tabla de particiones creada es de tipo GPT.

Pero ¿cómo saber si un disco necesita ser inicializado o no? Un disco virgen que no posee tabla de particiones es de tipo RAW. Conviene revisar esta propiedad para saber si es preciso ejecutar o no el cmdlet **Initialize-Disk**:

```
PS C:\Windows\system32> Get-Disk | Where-Object {$_.PartitionStyle

-eq "RAW"} | Select-Object Number,Model,Size,PartitionStyle


          Number  Model                                              Size
PartitionStyle
       ------ -----                                  ---- -------------
-
            1 Virtual Disk               21474836480 RAW
```

Como muestra el ejemplo anterior, es necesario inicializar el disco número 1.

Ejemplo: inicializar todos los discos no inicializados

En caso de que quiera inicializar todos los discos que deban serlo, puede hacerlo con una única línea de comando pasando el conjunto de discos no inicializados al comando **Initialize-Disk** mediante un pipe.

```
PS C:\Windows\system32> Get-Disk | Where-Object
{$_.PartitionStyle -eq "RAW"} | Initialize-Disk
```

g. Modificar las propiedades de un disco

El cmdlet **Set-Disk** permite realizar varios tipos de acciones diferentes sobre un disco determinado. Antes de ver qué es posible hacer, he aquí una descripción de los parámetros de este cmdlet:

Parámetro	Descripción
-IsOffline <Boolean>	Especifica si el disco debe ponerse fuera de línea o no.
-IsReadOnly <Boolean>	Define el disco en modo de solo lectura o en modo de lectura y escritura.
-Number <UInt32>	Especifica el número del disco donde se llevará a cabo la acción.
-PartitionStyle <PartitionStyle>	Especifica el tipo de tabla de particiones que se ha de definir. Los posibles valores son: MBR o GPT.

Convertir la tabla de particiones (MBR o GPT)

Para convertir la tabla de particiones de un disco, por ejemplo de GPT a MBR o a la inversa, se utiliza el parámetro **-PartitionStyle** especificando el nuevo tipo que se va a configurar. Tenga precaución: para poder llevar a cabo este cambio, no debe existir ninguna partición en el disco.

Ejemplo 1: cambiar la tabla de particiones del disco a MBR

```
PS C:\Windows\system32> Set-Disk -Number 1 -PartitionStyle MBR
```

Ejemplo 2: operación inversa para convertir un disco de MBR a GPT

```
PS C:\Windows\system32> Set-Disk -Number 1 -PartitionStyle GPT
```

Esta operación no puede realizarse sobre un disco considerado como crítico (disco de sistema).

Dejar un disco fuera de conexión

El parámetro **-IsOffline** permite dejar un disco fuera de conexión. Basta con definir el valor booleano a verdadero o falso en función de las necesidades.

Ejemplo 1: dejar un disco fuera de conexión

```
PS C:\Windows\system32> Set-Disk -Number 1 -IsOffline:$true
```

Ejemplo 2: dejar un disco en línea

```
PS C:\Windows\system32> Set-Disk -Number 1 -IsOffline:$false
```

Configurar un disco de solo lectura

El parámetro **-IsReadOnly** permite configurar un disco de solo lectura. Basta con definir el valor booleano a verdadero o falso en función de las necesidades.

Observe que, una vez activada esta opción, ya no podrá realizarse ninguna modificación, ya sea a nivel de los archivos y carpetas, ni tampoco a nivel de las particiones. El acceso a los datos para su lectura queda, por supuesto, autorizado.

Ejemplo 1: configurar un disco de solo lectura

```
PS C:\Windows\system32> Set-Disk -Number 1 -IsReadOnly:$true
```

Ejemplo 2: configurar un disco en modo lectura-escritura

```
PS C:\Windows\system32> Set-Disk -Number 1 -IsReadOnly:$false
```

h. Limpiar un disco

Clear-Disk es el cmdlet definitivo para limpiar un disco duro, físico o virtual. Esta acción tiene como objetivo eliminar toda la información de las particiones y reinicializa el disco, eliminando también el conjunto de datos que contiene. Debido al **carácter irreversible de este comando, conviene ser extremadamente cauto durante su ejecución**.

Antes de limpiar un disco, es necesario indicar el disco que se eliminará y, si contiene una o varias particiones, habrá que autorizar al comando a eliminar los datos (**-RemoveData**).

Parámetros	Descripción
-FriendlyName <String[]>	Especifica el nombre del disco que se ha de borrar.
-Number <UInt32[]>	Especifica el número del disco que se ha de borrar.
-RemoveData	Permite eliminar todos los datos presentes en el disco.

Ejemplo: limpiar un disco que posee particiones y datos

```
PS C:\Windows\system32> Clear-Disk -Number 1 -RemoveData
-Confirm:$false
```

Por defecto, debido a su aspecto crítico, la ejecución de este cmdlet **Clear-Disk** pide una confirmación. Puede omitir esta petición de confirmación agregando el parámetro **-Confirm:$false**.

i. Optimizar una partición

El cmdlet **Optimize-Volume** sirve para optimizar un volumen, realizando por ejemplo una desfragmentación. En función de la partición indicada a **Optimize-Volume**, este cmdlet sabe cuáles son las acciones óptimas que es preciso realizar en función del tipo de disco duro que la contiene. He aquí un resumen de algunos parámetros del cmdlet:

Parámetro	Descripción
-Analyze	Analiza la fragmentación del volumen especificado. Realiza únicamente un análisis e indica el estado actual de optimización del volumen.

Parámetro	Descripción
-Defrag	Indica al cmdlet que inicializa la función de desfragmentación en el volumen especificado. Su objetivo es consolidar las regiones fragmentadas y mejorar el rendimiento de lectura y de escritura secuenciales.
-DriveLetter <Char[]>	Indica la letra de la unidad de datos en la que se llevará a cabo la optimización.
-Retrim	Lanza el comando TRIM, que indica al controlador de disco SSD cuáles son los bloques de datos que ya no se utilizan y que pueden purgarse.
-SlabConsolidate	Indica al cmdlet que ejecute la consolidación de secciones para optimizar las asignaciones de secciones y así reducir el número de secciones utilizadas.

En función del tipo de almacenamiento donde se encuentre la partición en la que deba llevarse a cabo la operación de optimización, **Optimize-Volume** define automáticamente los parámetros adecuados si no se han especificado. Los comandos de optimización en función del tipo de almacenamiento se definen de la siguiente manera:

- Discos duros físicos, discos duros virtuales (tamaño fijo): **-Analyze -Defrag**.
- Discos duros virtuales (tamaño dinámico): **-Analyze -SlabConsolidate -Retrim**.
- SSD compatible con TRIM: **-Retrim**.
- SSD no compatible con TRIM: no se realiza ninguna acción.

Para lanzar una optimización del volumen, basta con indicar la letra de la unidad de datos sobre la que se lleva a cabo la tarea.

```
PS C:\Windows\system32> Optimize-Volume -DriveLetter D
```

En función del disco y del trabajo que ha de realizar **Optimize-Volume**, la ejecución del comando puede resultar

muy larga. Hasta que termine, no se devolverá el control a Windows PowerShell.

j. Corregir los errores de una partición

Repair-Volume es el equivalente del comando chkdsk.exe. Esta herramienta permite escanear y corregir los errores detectados en una partición, por ejemplo los clústeres defectuosos, o incluso la detección de archivos huérfanos. He aquí los parámetros de **Repair-Volume**:

Parámetro	Descripción
-DriveLetter <Char[]>	Especifica la letra de la unidad donde se llevará a cabo la acción.
-FileSystemLabel <String[]>	Especifica la etiqueta de la partición donde se llevará a cabo la acción.
-OfflineScanAndFix	Desmonta el volumen para el escaneo y repara todos los errores encontrados.
-Scan	Escanea el volumen sin realizar ninguna reparación. Todos los errores reportados se indican en el archivo de sistema $corrupt.
-SpotFix	Desmonta el volumen y corrige los errores reportados en el archivo de sistema $corrupt.

Ejemplo 1: escanear y reparar una partición

```
PS C:\Windows\system32> Repair-Volume -DriveLetter D
-OfflineScanAndFix
```

Ejemplo 2: escanear y reportar los errores únicamente

```
PS C:\Windows\system32> Repair-Volume -DriveLetter D -Scan
```

Por último, cuando puede desmontarse el volumen o la partición, puede ejecutar el comando de reparación sobre los errores encontrados previamente con **-Scan**.

```
PS C:\Windows\system32> Repair-Volume -DriveLetter D -SpotFix
```

k. Eliminar una partición

Remove-Partition permite eliminar particiones. Tenga precaución, pues al eliminar una partición se elimina también el conjunto de datos contenidos en ella, y esta acción es irreversible. ¡Conviene ser muy prudente!

Parámetro	Descripción
-DriveLetter <Char[]>	Especifica la letra de la unidad que identifica la partición que se ha de eliminar.
-DiskNumber <UInt32[]>	Especifica el número del disco.
-PartitionNumber <UInt32[]>	Especifica el número de la partición.

Ejemplo 1: eliminar una partición

```
C:\Windows\system32> Remove-Partition -DiskNumber 0
-PartitionNumber 4 -WhatIf
WhatIf: This will erase all data on disk 0 partition 3.

C:\Windows\system32> Remove-Partition -DiskNumber 0
-PartitionNumber 4 -Confirm:$false
```

Debido al carácter crítico de este comando, se pide una confirmación. Es posible omitirla utilizando el parámetro **-Confirm:$false**. Por otro lado, no olvide que puede simular el resultado de este comando utilizando el parámetro **-WhatIf**, disponible en un gran número de cmdlets.

Ejemplo 2: eliminar una partición identificada por su letra de unidad

```
C:\Windows\system32> Remove-Partition -DriveLetter H
-Confirm:$false
```

2. Los discos duros virtuales

Esta sección detalla los cmdlets dedicados a los discos duros virtuales. Los discos duros virtuales son archivos .vhd o .vhdx,

cada formato presenta sus ventajas y sus limitaciones. Ambos tipos de discos duros virtuales están soportados de manera nativa desde Windows 8: es posible crear, montar y desmontar discos duros virtuales desde el administrador de discos. Respecto a Windows 7, solo están soportados los archivos con extensión .vhd.

Los cmdlets que hemos visto antes correspondientes a la administración de los discos también pueden utilizarse: una vez montado el disco duro virtual, este es visible y puede manipularse como cualquier otro disco.

Antes de detallar las posibles tareas de administración con Windows PowerShell sobre los discos duros virtuales, he aquí dos tablas resumen. La primera compara las ventajas y las limitaciones de los formatos VHD y VHDX:

	Ventajas	Limitaciones
VHD	Soportado de manera nativa desde Windows 7 en los puestos de trabajo, y desde Windows Server 2008 R2 en los servidores.	El tamaño máximo del disco duro virtual es de 2040 GB.
VHDX	El tamaño supera sobradamente los 2040 GB (el tamaño máximo es 64 TB) y VHDX resiste los eventos vinculados con fallos de corriente.	Soportado desde Windows 8 y Windows Server 2012.

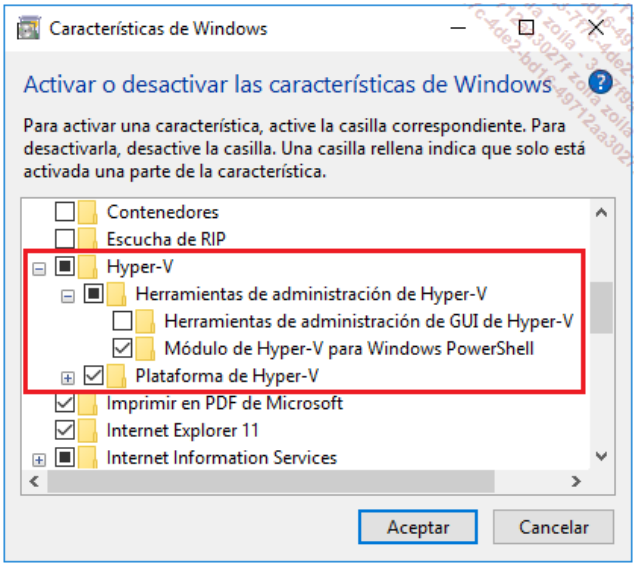
La segunda tabla explica las diferencias entre los dos posibles tipos de discos duros virtuales, de tamaño fijo y de extensión dinámica:

	Descripción
Tamaño fijo	El archivo ocupa directamente el tamaño máximo tras la creación del disco duro virtual.
Extensión dinámica	El tamaño del archivo aumenta conforme se escriben los datos hasta alcanzar el tamaño máximo definido.

Los cmdlets de administración de los formatos VHD y VHDX están disponibles con el módulo Hyper-V. Conviene instalar como mínimo las siguientes características:

- Módulo Hyper-V para Windows PowerShell.
- Plataforma Hyper-V.

La administración de las características de Windows se realiza desde la opción **Programas y características** del **Panel de control** de Windows.



Lista de las características de Windows que hay que instalar para utilizar los cmdlets VHD

Se requiere un reinicio del equipo para finalizar la instalación.

Por último, para terminar con esta presentación, debe saber que, si ejecuta Windows desde una máquina virtual, no es posible instalar la característica **Plataforma Hyper-V**. Por consiguiente, no es posible administrar discos duros virtuales con Windows PowerShell en un entorno virtualizado. Si intenta utilizar los cmdlets específicos de la administración de discos duros virtuales, obtendrá un error:

```
PS C:\Windows\system32> New-VHD -Path C:\Temp\VHD-1.vhdx
-SizeBytes 10GB -Dynamic
New-VHD: el rol Hyper-V no está instalado en el host de
destino. Agréguelo a ese host y ejecute el cmdlet de nuevo.
```

a. Crear un disco duro virtual

Para crear un disco duro virtual, es preciso definir las tres características que lo componen: tamaño máximo, formato (VHD o VHDX) y tipo (de tamaño fijo o de extensión dinámica). Un

disco duro virtual, al ser un archivo, requiere también una ubicación.

New-VHD permite crear un nuevo disco duro virtual. Se trata del mismo cmdlet que se usa para crear archivos VHD o VHDX. El formato seleccionado se reconoce directamente gracias a la extensión del nombre del archivo que se ha indicado en la ruta de acceso (**-Path**). Los dos primeros parámetros son el tamaño máximo (**-SizeBytes**) y el tipo de disco duro virtual deseado (**-Fixed** o **-Dynamic**). He aquí una tabla resumen de estos parámetros:

Parámetro	Descripción
-Path <String[]>	Ruta de acceso a los archivos de los discos duros virtuales.
-SizeBytes <UInt64>	Indica el tamaño máximo, en bytes, del disco duro virtual que se creará.
-Dynamic	Especifica que el nuevo disco duro virtual es de extensión dinámica.
-Fixed	Especifica que el nuevo disco duro virtual es de tamaño fijo.

Ejemplo 1: crear un disco duro virtual VHD

He aquí un ejemplo que permite crear un disco duro virtual en formato VHD, con un tamaño máximo de 10 GB y de tipo tamaño fijo.

```
PS C:\Windows\system32> New-VHD -Path C:\Temp\DiscoVirtual.vhd
-SizeBytes 10GB -Fixed


ComputerName            : W10
Path                    : C:\Temp\DiscoVirtual.vhd
VhdFormat               : VHD
VhdType                 : Fixed
FileSize                : 10737418752
Size                    : 10737418240
MinimumSize             :
LogicalSectorSize       : 512
PhysicalSectorSize      : 512
BlockSize               : 0
ParentPath              :
DiskIdentifier          : D9AC3CFA-5E37-4152-AC18-0A04CB4C1BA8
FragmentationPercentage: 0
Alignment               : 1
Attached                : False
```

```
DiskNumber             :
Number                 :
```

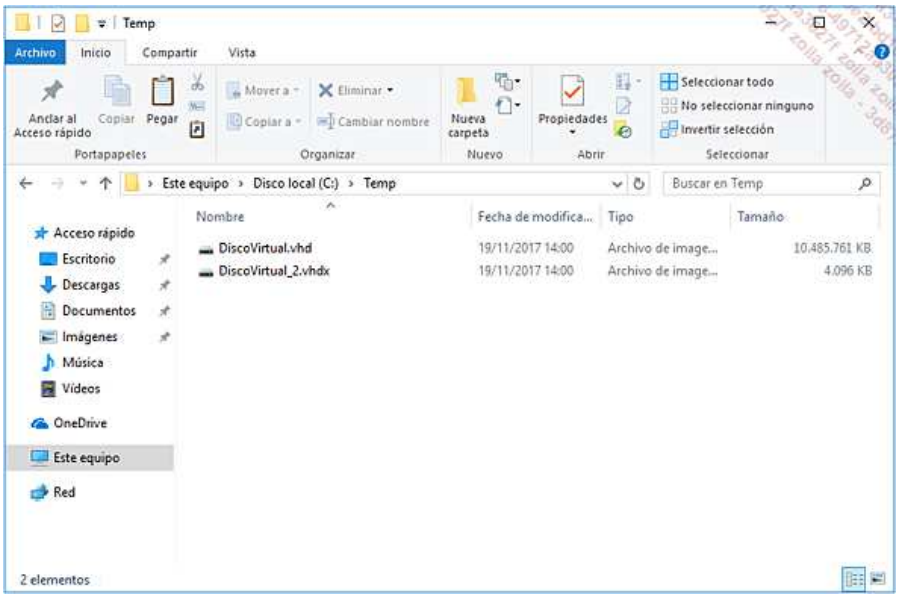
Se muestra una barra de progreso en la parte superior de la ventana de Windows PowerShell para indicarle el estado de la creación del disco duro virtual. Una vez realizada la operación, el disco duro virtual está disponible y listo para montarse.

Ejemplo 2: crear un disco duro virtual VHDX

En este segundo ejemplo, se crea un disco duro virtual en formato VHDX, de 1 terabyte de tamaño, y de tipo extensión dinámica.

```
PS C:\Users\Administrator> New-VHD C:\Temp\DiscoVirtual_2.vhdx
-SizeBytes 1TB -Dynamic


ComputerName            : W10
Path                    : C:\Temp\DiscoVirtual_2.vhdx
VhdFormat               : VHDX
VhdType                 : Dynamic
FileSize                : 4194304
Size                    : 1099511627776
MinimumSize             :
LogicalSectorSize       : 512
PhysicalSectorSize      : 4096
BlockSize               : 33554432
ParentPath              :
DiskIdentifier          : B3629502-5B63-43B3-8274-6CDFADC8FC43
FragmentationPercentage: 0
Alignment               : 1
Attached                : False
DiskNumber              :
Number                  :
```

En el Explorador de archivos, los dos discos duros virtuales están presentes: el de tamaño fijo ocupa el espacio definido durante su creación, mientras que el disco duro virtual de tipo extensión dinámica ocupa solo 4 megabytes tras su creación.



Diferencia de tamaño tras la creación entre los dos tipos de discos duros virtuales

b. Montar un disco duro virtual

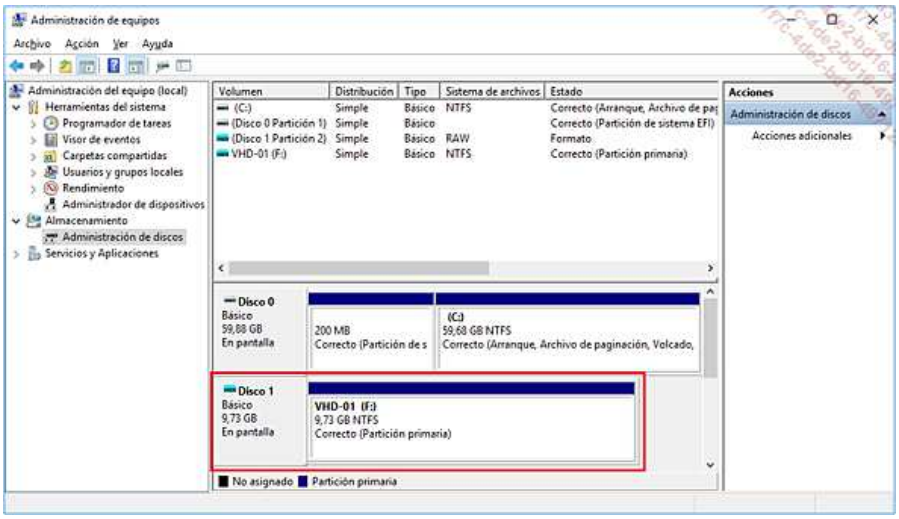
Para montar un disco duro virtual, recién creado o no, hay que utilizar el cmdlet **`Mount-VHD`**. Además de que es obligatorio indicar la ruta de acceso al disco duro virtual que desea montar, hay disponibles dos opciones para montar el disco duro virtual. Veámoslas en la tabla resumen de los parámetros:

Parámetro	Descripción
`-NoDriveLetter`	El disco duro virtual no posee ninguna letra de unidad cuando se monta.
`-Path` `<String[]>`	Indica la ruta de acceso al archivo de disco duro virtual.
`-ReadOnly`	El disco duro virtual montado está accesible únicamente en modo de solo lectura.

Ejemplo: montar un disco duro virtual

```
PS C:\Windows\system32> Mount-VHD -Path C:\Temp\DiscoVirtual.vhd
```

El disco duro virtual está disponible de inmediato y puede verse desde la interfaz Administración de discos:



Disco duro virtual montado en la Administración de discos

c. Compactar un disco duro virtual de tipo extensión dinámica

Cuando se acaba de crear un disco duro virtual de tipo extensión dinámica, este posee un tamaño de 4 megabytes. Este tamaño aumenta conforme se escriben los datos en él, hasta alcanzar el tamaño máximo fijado definido en su creación. Sin embargo, el espacio ocupado por el disco duro virtual no disminuye jamás, incluso aunque se elimine la totalidad de los datos contenidos en él.

Es aquí donde resulta útil el comando **`Optimize-VHD`**: este cmdlet permite reducir el tamaño de un disco duro virtual ocupando solo el espacio necesario en función de los datos contenidos en él. Esta operación solo puede llevarse a cabo bajo alguna de las dos condiciones siguientes:

- Cuando el disco virtual no está montado.
- Cuando el disco virtual está montado, pero en modo de solo lectura.

Si se cumple alguna de estas dos condiciones, basta con ejecutar el comando de optimización especificando la ruta de acceso al archivo VHD o VHDX:

Parámetro	Descripción

Parámetro	Descripción
-Mode <VhdCompactMode>	Define el modo de optimización que se ha de aplicar a un disco duro virtual. En un disco VHD, el modo por defecto es `Full`. En un disco VHDX, el modo por defecto es `Quick`. • `Full`: escanea los bloques vacíos y recupera los bloques no utilizados. • `Quick`: recupera los bloques no utilizados sin escanear los bloques vacíos.
-Path <String[]>	Indica la ruta de acceso al archivo del disco duro virtual.

Ejemplo: compactar un disco duro virtual

En este ejemplo, el disco duro virtual se monta previamente en modo de solo lectura.

```
PS C:\Windows\system32> Mount-VHD -Path
C:\Temp\DiscoVirtual_2.vhdx -ReadOnly
```

A continuación, se ejecuta el comando **Optimize-Volume** para compactar el disco duro virtual.

```
PS C:\Windows\system32> Optimize-VHD -Path
C:\Temp\DiscoVirtual_2.vhdx
```

Se muestra una barra de progreso en la parte superior de Windows PowerShell durante el tiempo que dura la operación de optimización.

Una vez finalizada la tarea, el archivo correspondiente al disco duro virtual tiene un tamaño comparable a los datos que contiene.

d. Convertir un disco duro virtual

En la introducción a los discos duros virtuales, hemos visto las ventajas y las limitaciones de cada uno de los formatos VHD y VHDX. Con ayuda del cmdlet **Convert-VHD**, es posible convertir un disco duro virtual de un formato a otro. Tenga precaución, sin

embargo, al realizar la conversión en caliente: se crea un nuevo disco duro virtual y, a continuación, se realiza una copia de los datos. Compruebe bien el espacio en disco disponible antes de llevar a cabo esta operación.

He aquí una tabla resumen de los parámetros de **Convert-VHD**:

Parámetro	Descripción
-DeleteSource	Significa que el disco duro virtual de origen se eliminará tras la copia de los datos.
-DestinationPath <String>	Indica el nuevo archivo de disco duro virtual que se creará. La extensión del archivo determina el tipo de disco duro virtual.
-Path <String>	Indica el origen del disco duro virtual que se ha de convertir.
-VHDType <VHDType>	Define el tipo del disco duro virtual convertido. El valor por defecto está determinado por el tipo de disco duro virtual de origen. Los posibles valores son: Fixed o Dynamic.

Ejemplo 1: convertir un disco duro virtual VHD a VHDX

El tipo de disco duro virtual es idéntico al tipo de disco duro virtual de origen. El tamaño máximo del disco duro virtual también se preserva.

```
PS C:\Windows\system32> Convert-VHD -Path
C:\Temp\DiscoVirtual.vhd -DestinationPath
C:\Temp\DiskConverted.vhdx
```

Esta operación puede resultar muy larga. Todo depende de la cantidad de datos presentes en el disco duro virtual de origen. Windows PowerShell no le devolverá el control hasta que finalice la ejecución del comando.

Ejemplo 2: convertir un disco duro virtual VHDX a VHD especificando su tipo

Este segundo ejemplo permite convertir un disco VHDX a VHD, definiendo claramente el tipo de disco duro virtual deseado. Una vez copiados los datos, se elimina el disco duro virtual de origen.

```
PS C:\Windows\system32> Convert-VHD -Path
C:\Temp\DiscoVirtual_2.vhdx -DestinationPath
C:\Temp\DiskConverted_2.vhd -VHDType Fixed -DeleteSource
```

e. Redimensionar un disco duro virtual

Es posible modificar el tamaño máximo definido durante la creación de un disco duro virtual mediante el cmdlet **Resize-VHD**. Sea cual sea el formato, VHD o VHDX, es posible aumentar el límite de tamaño, con un tamaño máximo de 2040 GB para el VHD, y de 64 TB para el VHDX. También puede reducir el tamaño, pero únicamente con los discos duros virtuales de formato VHDX.

He aquí una tabla resumen de los parámetros:

Parámetro	Descripción
-Path <String[]>	Especifica la ruta de acceso al archivo del disco duro virtual que debe redimensionarse.
-SizeBytes <Uint64>	Indica el nuevo tamaño máximo que se aplica al disco duro virtual.
-ToMinimumSize	Indica que el nuevo tamaño máximo estará reducido al mínimo automáticamente. Este parámetro solo puede aplicarse en los discos duros virtuales de formato VHDX.

Ejemplo 1: aumentar el tamaño máximo de un disco duro virtual

```
PS C:\Windows\system32> Resize-VHD -Path
C:\Temp\DiscoVirtual.vhd -SizeBytes 25GB
```

Ejemplo 2: reducir el tamaño máximo de un disco duro virtual VHDX al mínimo

En este segundo ejemplo, se reduce al mínimo el tamaño máximo del disco duro virtual VHDX.

```
PS C:\Windows\system32> Resize-VHD C:\Temp\DiscoVirtual_2.vhdx
-ToMinimumSize
```

La gestión de la energía

La gestión de la energía de los puestos de trabajo es un asunto que preocupa a cada vez más empresas y que como mínimo conviene abordar. Sin embargo, en la actualidad Windows PowerShell no posee cmdlets específicos para la gestión de la energía. El ejecutable powercfg.exe permite administrar el plan de energía en un sistema local. Pero Windows PowerShell también permite realizar algunas acciones a través del objeto WMI Win32_PowerPlan.

1. Mostrar los planes de energía

Para ver todos los planes de energía configurados en un puesto de trabajo, se utiliza un consumidor WMI para consultar la clase **`Win32_PowerPlan`** presente en el espacio de nombres root\cimv2\power.

```
PS C:\Windows\system32> Get-CimInstance -Namespace root\cimv2\power

-ClassName Win32_PowerPlan


Caption       :
Description     : Equilibra automáticamente el rendimiento con el
consumo
                  de energía en el hardware que lo permita.
ElementName   : Uso normal
InstanceID          :  Microsoft:PowerPlan\{381b4222-f694-41f0-9685-
ff5bb260df2e}
IsActive      : False
PSComputerName:

Caption       :
Description     : Mejora el rendimiento, pero puede utilizar más
energía.
ElementName   : Alto rendimiento
InstanceID          :  Microsoft:PowerPlan\{8c5e7fda-e8bf-4a96-9a85-
a6e23a8c635c}
IsActive      : True
PSComputerName:

Caption       :
Description   : Reduce el rendimiento del equipo cuando sea posible

                  para ahorrar energía.
ElementName   : Ahorro de energía
InstanceID          :  Microsoft:PowerPlan\{a1841308-3541-4fab-bc81-
f71556f20b4a}
IsActive      : False
PSComputerName:
```

Get-CimInstance y **Get-WMIObject** se utilizan de una manera similar. Ambos cmdlets son consumidores WMI.

El comando anterior devuelve todos los planes de energía configurados en el puesto de trabajo. El nombre de los distintos modos está definido por la propiedad `ElementName`, y viendo la propiedad `IsActive` es posible saber si se trata del modo activo (`True`) o no (`False`).

El consumidor WMI utilizado aquí, **Get-CimInstance**, también permite ejecutar comandos en los equipos remotos, y por consiguiente recuperar la lista de los distintos planes de energía. Para ello se requieren tres etapas:

1. Cree un objeto con las credenciales (nombre de usuario y contraseña). La cuenta utilizada debe ser una cuenta con permisos de administración en la máquina remota.

2. Cree una sesión CIM que represente la conexión entre el equipo local y el equipo remoto. Para ello, debe especificar la información que permite establecer la conexión, como el nombre del equipo remoto y las credenciales.

3. Ejecute el comando y recupere la instancia CIM.

He aquí un ejemplo que permite recuperar el plan de energía activo en un equipo remoto:

```
PS C:\Windows\system32> $cred = Get-Credential -Credential
CONTOSO\Administrador
PS C:\Windows\system32> $cim = New-CimSession -ComputerName W10
-Credential $cred
PS C:\Windows\system32> Get-CimInstance -Name root\cimv2\power
-Class  Win32_PowerPlan  -Filter  "IsActive  =  'True'"  -CimSession
$cim


Caption       :
Description     : Mejora el rendimiento, pero puede utilizar más
energía.
ElementName   : Alto rendimiento
InstanceID          :  Microsoft:PowerPlan\{8c5e7fda-e8bf-4a96-9a85-
a6e23a8c635c}
IsActive      : True
PSComputerName: W10
```

El uso de **Get-CimInstance** o de **New-CimSession** en puestos de trabajo remotos (**-ComputerName**) requiere que el servicio WinRM esté configurado y arrancado en los puestos

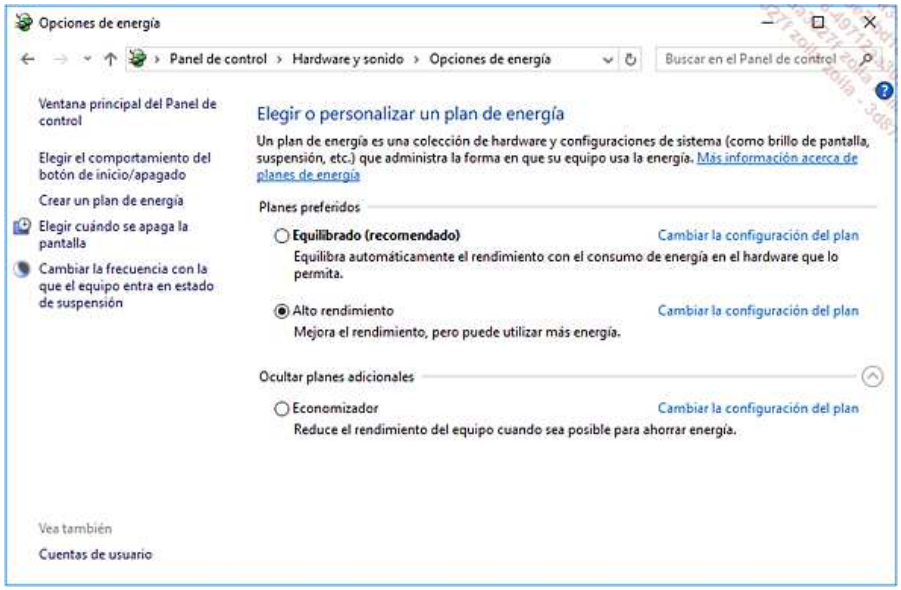
remotos (`winrm quickconfig`). En efecto, estos cmdlets crean una sesión temporal en el equipo correspondiente mediante el protocolo WS-Management.

Además, el parámetro **`-ComputerName`** de **`Get-CimInstance`** y de **`New-CimSession`** puede recibir varios puestos de trabajo, lo que devuelve los planes de energía de cada uno de los equipos correspondientes. La propiedad `PSComputerName` indica el nombre del puesto de trabajo al que corresponde el resultado.

2. Escoger un plan de energía

Para bascular de un plan de energía a otro modo, se utiliza el método `Activate` de la clase **`Win32_PowerPlan`**. Para llevar a cabo esta operación, hay que invocar el método con el cmdlet **`Invoke-CimMethod`**, pasándole como parámetro el plan de energía que se desea activar. He aquí un ejemplo para activar el modo de Alto rendimiento:

```
PS C:\Windows\system32> $pp = Get-CimInstance -Name
root\cimv2\power
-Class Win32_PowerPlan -Filter "ElementName = 'Alto rendimiento'"
PS C:\Windows\system32> Invoke-CimMethod -InputObject $pp -
MethodName
Activate

ReturnValue PSComputerName
----------- --------------
       True
```



El modo Alto rendimiento está ahora activo

También es posible realizar esta acción en un puesto remoto especificando el parámetro **-CimSession** y pasando una variable que contenga las credenciales de conexión al cmdlet **Invoke-CimMethod**.

La gestión de las impresoras

Desde Windows 8, es posible administrar las impresoras con cmdlets de Windows PowerShell. Si desea administrar las impresoras con Windows 7, puede utilizar el ejecutable printui.exe, que ofrece las mismas posibilidades.

La impresora es uno de los dispositivos más habituales en los puestos de trabajo, ya sea en el mundo profesional o para un uso personal. Windows PowerShell permite administrar las impresoras en los puestos de trabajo únicamente por línea de comandos: agregar un controlador, agregar un puerto, agregar una impresora y otros muchos puntos son los que abordaremos en esta sección.

Los cmdlets estudiados para la administración de las impresoras son los siguientes:

- **`Get-PrinterDriver`**: recupera la lista de controladores de impresión instalados.
- **`Add-PrinterDriver`**: agrega un controlador de impresión.
- **`Remove-PrinterDriver`**: elimina un controlador de impresión.
- **`Add-PrinterPort`**: crea un nuevo puerto de impresión.
- **`Get-PrinterPort`**: recupera la lista de puertos de impresión.
- **`Remove-PrinterPort`**: elimina un puerto de impresión.
- **`Add-Printer`**: agrega una nueva impresora.
- **`Get-Printer`**: recupera la lista de impresoras configuradas en el puesto de trabajo.
- **`Remove-Printer`**: elimina una impresora.

1. Los controladores de impresión

Antes de poder agregar una impresora en un puesto de trabajo, hay que comprender esta parte dedicada a la administración de controladores de impresión. Estos controladores son programas que permiten interactuar con el sistema operativo y el dispositivo: resultan, por tanto, elementos obligatorios para poder utilizar una impresora.

a. Conocer los controladores de impresión instalados

Para recuperar en cualquier momento la información correspondiente a los controladores de impresión instalados en un puesto de trabajo, puede utilizar el cmdlet **Get-PrinterDriver**. Este comando devuelve una lista de controladores con su nombre, el entorno en el que trabajan, el número de versión mayor y el fabricante.

Parámetro	Descripción
-Name <String[]>	Especifica el nombre del controlador o de los controladores de impresión que se han de recuperar.
-PrinterEnvironment <String[]>	Especifica el entorno del controlador de impresión.

Ejemplo: obtener la lista de controladores de impresión instalados

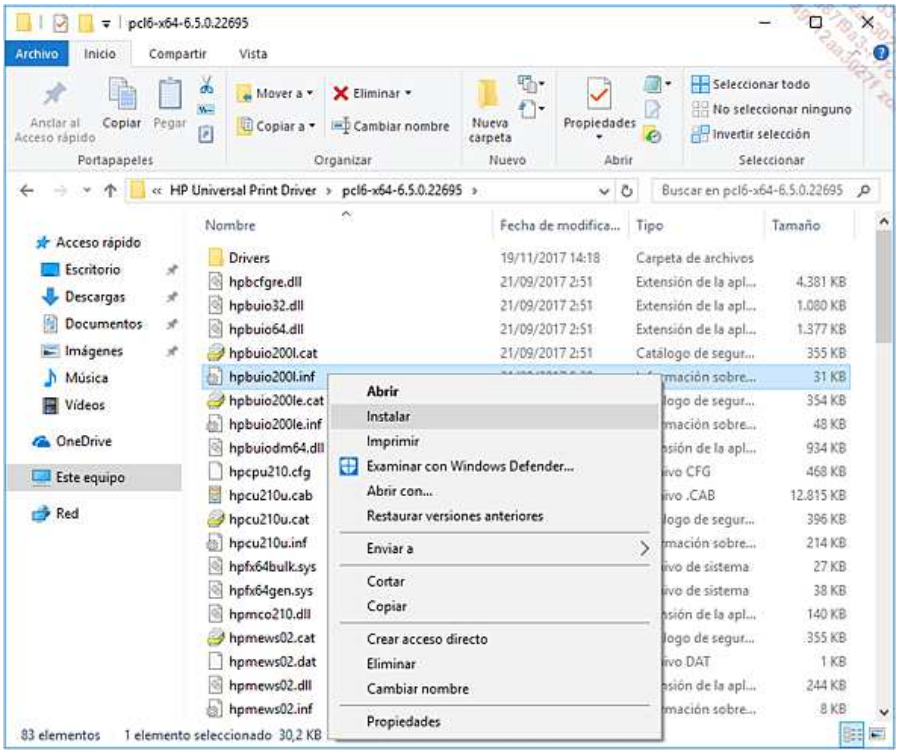
```
PS C:\Windows\system32> Get-PrinterDriver

Name                                PrinterEnvironment MajorVersion
Manufacturer
----                            ------------------ ------------ -----
-------
Send to Microsoft OneNote...  Windows x64           4
 Microsoft
Microsoft XPS Document Wr...  Windows x64           4
 Microsoft
Microsoft Print To PDF          Windows x64           4
 Microsoft
Microsoft Desktop Easy Pr...  Windows x64           3
 Microsoft
Microsoft Shared Fax Driver   Windows x64           3
 Microsoft
Microsoft enhanced Point ...  Windows x64           3
 Microsoft
Microsoft enhanced Point ...  Windows NT x86       3
 Microsoft
```

Siempre tiene la posibilidad de obtener información complementaria de cada uno de los controladores pasándole el objeto instanciado con **Get-PrinterDriver** a **Format-List**:

```
PS C:\Windows\system32> Get-PrinterDriver -Name "Microsoft XPS
Document Writer v4" | Format-List *
```

b. Agregar un controlador de impresión

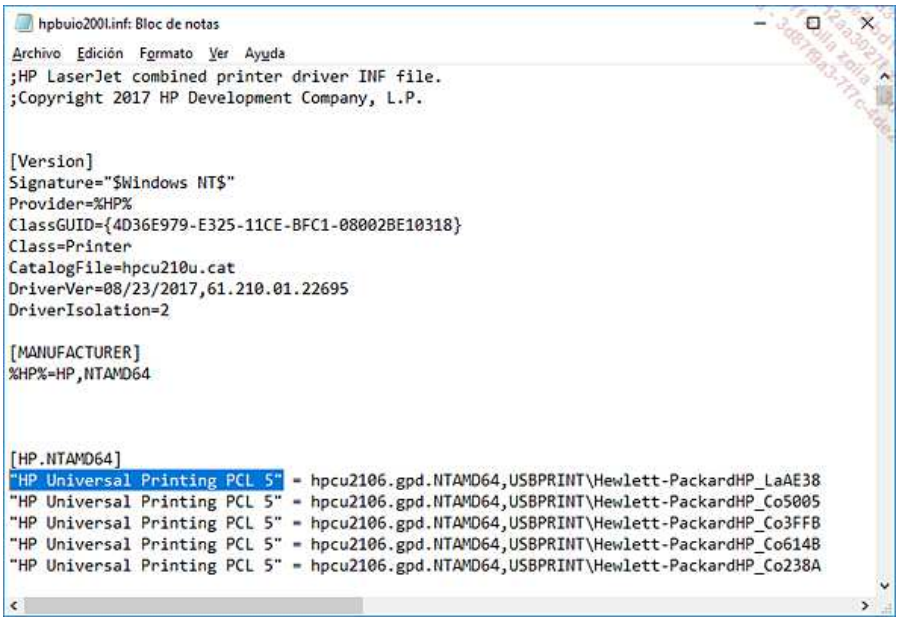
Previamente, es necesario que los controladores estén presentes en el almacén de controladores de Windows. Si no es el caso, lleve a cabo la instalación del controlador, bien utilizando el programa proporcionado por el fabricante de la impresora, o bien utilizando el archivo .inf (haga clic con el botón derecho en el archivo y, a continuación, haga clic en **Instalar**, o bien ejecute el comando `pnputil -i -a C:\Temp\PrinterDriver.inf`).



Instalación de controladores mediante el archivo .inf (contiene el nombre de los controladores)

Una vez instalados los controladores en el almacén de controladores de Windows, es preciso instalar los controladores de impresión para poder agregar la impresora más adelante. Para llevar a cabo esta etapa, es preciso conocer el nombre de la impresora asociada al controlador previamente instalado: la información se encuentra en el archivo .inf.

Abra el archivo .inf del controlador de impresión con el Bloc de notas y busque lo que pueda corresponder al nombre del controlador previamente instalado (generalmente al inicio del archivo):



Archivo .inf que contiene el nombre de los controladores

En el ejemplo anterior, el nombre del controlador de impresión es HP Universal Printing PCL 5.

Add-PrinterDriver permite instalar un controlador de impresión. Durante la ejecución de este cmdlet, este comando busca directamente en el almacén de controladores de Windows el controlador asociado al nombre de la impresora. El parámetro **-Name** recibe como valor el nombre de este controlador.

Parámetro	Descripción
-Name <String>	Nombre del controlador de impresión que se ha de agregar.

Ejemplo: agregar un controlador de impresión

He aquí un ejemplo para instalar el controlador de impresión HP Universal Printing PCL 5.

```
PS C:\Windows\system32> Add-PrinterDriver -Name "HP Universal
Printing
PCL 5"
PS C:\Windows\system32> Get-PrinterDriver

Name                                PrinterEnvironment MajorVersion
Manufacturer
----                          ------------------ ------------ -----
-------
Send to Microsoft OneNote... Windows x64           4
 Microsoft
```

```
Microsoft XPS Document Wr... Windows x64           4
 Microsoft
Microsoft Print To PDF         Windows x64           4
 Microsoft
HP Universal Printing PCL 5  Windows x64        3            HP
Microsoft Desktop Easy Pr... Windows x64           3
 Microsoft
Microsoft Shared Fax Driver   Windows x64           3
 Microsoft
Microsoft enhanced Point ... Windows x64           3
 Microsoft
Microsoft enhanced Point ... Windows NT x86       3
 Microsoft
```

El controlador HP Universal Printing PCL 5 está ahora instalado y listo para poder utilizarlo para agregar una impresora en **Dispositivos e impresoras** del **Panel de control** de Windows.

c. Eliminar un controlador de impresión

Antes de pasar a la administración de los puertos de impresión, veamos rápidamente cómo eliminar un controlador de impresión. Esta operación se lleva a cabo con el cmdlet **Remove-PrinterDriver**.

Parámetro	Descripción
-Name <String[]>	Nombre del controlador de impresión que se ha de eliminar.
-RemoveFromDriverStore	Elimina también el controlador del almacén de controladores de Windows.

Ejemplo: eliminar un controlador de impresión

```
PS C:\Windows\system32> Remove-PrinterDriver -Name "HP Universal
Printing
PCL 5" -RemoveFromDriverStore
PS C:\Windows\system32> Get-PrinterDriver

Name                                PrinterEnvironment MajorVersion
Manufacturer
----                          ------------------ ------------ -----
-------
Send to Microsoft OneNote... Windows x64           4
 Microsoft
Microsoft XPS Document Wr... Windows x64           4
 Microsoft
Microsoft Print To PDF         Windows x64           4
 Microsoft
Microsoft Desktop Easy Pr... Windows x64           3
 Microsoft
Microsoft Shared Fax Driver   Windows x64           3
```

```
 Microsoft
Microsoft enhanced Point ... Windows x64          3
 Microsoft
Microsoft enhanced Point ... Windows NT x86       3
 Microsoft
```

Si intenta agregar de nuevo el controlador de impresión habiendo utilizado el parámetro **-RemoveFromDriverStore**, obtendrá un mensaje de error en el que se le indicará que el controlador ya no se encuentra en el almacén de controladores:

```
PS C:\Windows\system32> Add-PrinterDriver -Name "HP Universal
Printing PCL 5"
Add-PrinterDriver: el controlador especificado no existe en
el almacén de controladores.
```

De modo que es necesario volver a instalar el controlador (utilizando el archivo .inf) en el almacén de controladores de Windows, como se ha descrito anteriormente (consulte la sección Agregar un controlador de impresión).

2. Los puertos de impresión

Antes de agregar una impresora en un puesto de trabajo, es preciso saber qué puerto utilizar para dialogar con el dispositivo de impresión. A continuación veremos la administración (creación y eliminación) de los puertos de impresión.

a. Obtener la lista de puertos

Existe un cierto número de puertos creados por defecto en el puesto de trabajo. Para consultarlos y obtener información relativa a ellos, se utiliza el cmdlet **Get-PrinterPort**.

Parámetro	Descripción
-Name <String[]>	Especifica el nombre del puerto o de los puertos de impresión que se han de recuperar.

Ejemplo: obtener la lista de puertos configurados

```
PS C:\Windows\system32> Get-PrinterPort

Name                 ComputerName            Description
 PortMonitor
----           ------------         -----------           ----------
-
COM1:                                   Puerto local           Local
Monitor
COM2:                                   Puerto local           Local
```

```
Monitor
COM3:                                Puerto local            Local
Monitor
COM4:                                Puerto local            Local
Monitor
FILE:                                Puerto local            Local
Monitor
IR                                   Puerto local            Local
Monitor
LPT1:                                Puerto local            Local
Monitor
LPT2:                                Puerto local            Local
Monitor
LPT3:                                Puerto local            Local
Monitor
nul:                                 Puerto local            Local
Monitor
PORTPROMPT:                          Puerto local            Local
Monitor
SHRFAX:                                     Puerto de Monitor....
Microsoft Shared ...
```

b. Crear un nuevo puerto

Actualmente, la gran mayoría de las impresoras están conectadas a través de una red IP en la empresa, pues esta configuración ofrece una mayor flexibilidad. Es el caso que vamos a estudiar aquí.

Para comunicarse con la impresora, hay que crear un puerto de impresión en el puesto de trabajo antes de pasar a crear la impresora. Es posible agregar un nuevo puerto a través del cmdlet **Add-PrinterPort**.

Parámetro	Descripción
-Name <String>	Especifica el nombre del puerto de impresión que se ha de agregar.
-PortNumber <UInt32>	Especifica el número de puerto TCP/IP que se ha de asignar al puerto de impresión.
-PrinterHostAddress <String>	Especifica la dirección IP de la impresora que se va a configurar en el puerto de impresión.

Ejemplo: agregar un puerto de impresión

```
PS C:\Windows\system32> Add-PrinterPort -Name "IP_192.168.0.50"

-PrinterHostAddress 192.168.0.50
```

```
PS C:\WINDOWS\system32> Get-PrinterPort -Name "IP_192.168.0.50"

Name                     ComputerName           Description
 PortMonitor
----                 ------------         -----------          ---
--------
IP_192.168.0.50                                  Port TCP/IP standard
TCPMON.DLL


PS C:\Windows\system32> Get-PrinterPort -Name "IP_192.168.0.50" |

Format-List

Name               : IP_192.168.0.50
ComputerName       :
Description        : Puerto TCP/IP estándar
Protocol           : RAW
PrinterHostAddress: 192.168.0.50
PrinterHostIP      :
PortNumber         : 9100
SNMPIndex          : 0
SNMPCommunity      :
SNMPEnabled        : False
LprQueueName       :
LprByteCounting    : False
```

El puerto por defecto es el 9100. Si la impresora debe comunicarse a través de algún otro número de puerto TCP/IP, especifíquelo utilizando el parámetro **-PrinterHostAddress**.

c. Eliminar un puerto

Es posible eliminar un puerto simplemente pasándole el nombre del puerto que se ha de eliminar al cmdlet **Remove-PrinterPort**.

Parámetro	Descripción
-Name <String[]>	Especifica el nombre del puerto de impresión que se ha de eliminar.

Ejemplo: eliminar un puerto de impresión

```
PS C:\Windows\system32> Remove-PrinterPort -Name "IP_192.168.0.50"
```

3. Las impresoras

Ahora que ya hemos abordado la administración de los controladores de impresión y de los puertos de impresión, entremos directamente a ver cómo administrar las impresoras.

a. Agregar una impresora

Agregar una impresora es la última etapa antes de poder ejecutar tareas de impresión desde el puesto de trabajo en el que se configure. Para llevar a cabo esta operación, el administrador debe conocer dos elementos:

- El nombre del controlador de impresión previamente instalado.
- El puerto de impresión (incluyendo su dirección IP) que debe utilizar el puesto de trabajo para comunicarse con la impresora.

El cmdlet **`Add-Printer`** permite agregar la impresora en **Dispositivos e impresoras** del **Panel de control** de Windows. He aquí un resumen de los distintos parámetros disponibles para **`Add-Printer`**:

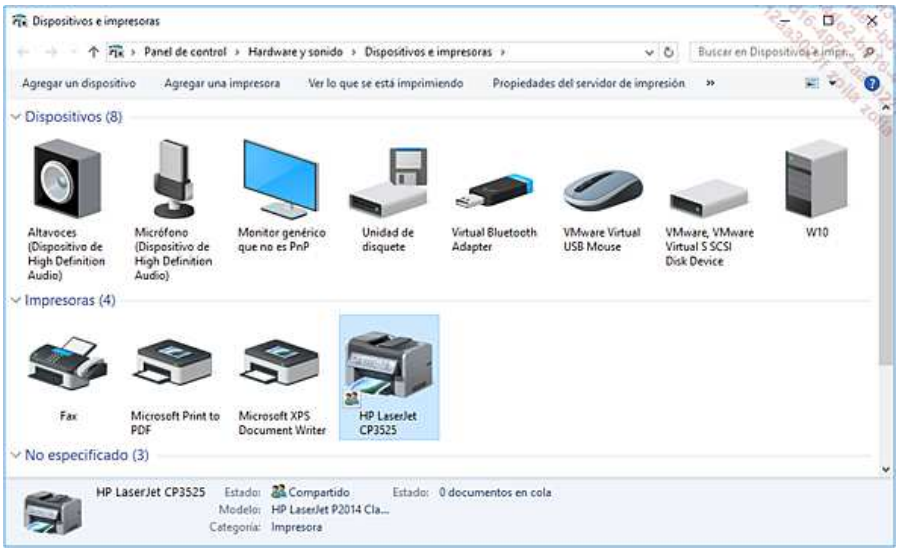
Parámetro	Descripción
`-Comment` `<String>`	Agrega el texto en el campo **Comentarios** de la impresora.
`-DriverName` `<String>`	Especifica el nombre del controlador de impresión que se va a utilizar.
`-Location` `<String>`	Especifica la ubicación de la impresora.
`-Name` `<String>`	Especifica el nombre de la impresora que se va a agregar.
`-PortName` `<String>`	Especifica el nombre del puerto que se utilizará para comunicarse con la impresora.

Ejemplo: agregar una impresora

Es preciso indicar la siguiente información: el nombre de la impresora que se agrega (**`-Name`**), el nombre del controlador de la impresora (**`-DriverName`**) y, por último, el nombre del puerto, incluyendo la dirección IP (**`-PortName`**).

```
PS C:\Windows\system32> Add-Printer -Name "HP LaserJet CP3525"
-DriverName "HP Universal Printing PCL 5" -PortName
"IP_192.168.0.50"
PS C:\Windows\system32> Get-Printer | Select-Object
Name,DriverName,PortName
```

```
Name                                DriverName
PortName
----                                ----------                                -
-------
HP LaserJet CP3525                  HP Universal Printing PCL 5
 IP_192.168.0.50
Enviar a OneNote 2013               Send to Microsoft OneNote 15 Dr...
nul:
Microsoft XPS Document Writer Microsoft XPS Document Writer v4
PORTPROMPT:
Microsoft Print to PDF         Microsoft Print To PDF
PORTPROMPT:
Fax                                  Microsoft Shared Fax Driver
 SHRFAX:
```



*La impresora aparece a continuación en **Dispositivos e impresoras***

La aplicación **Configuración** de Windows 10 dispone también de una entrada **Impresoras y escáneres**, donde puede encontrar las impresoras instaladas y configuradas. Para acceder, abra la aplicación **Configuración** y, a continuación, haga clic en **Dispositivos** y luego en **Impresoras y escáneres**.

*La impresora también aparece en la aplicación **Configuración** de Windows 10*

La impresora está lista para recibir tareas de impresión ejecutadas desde el puesto de trabajo.

En el ejemplo anterior, **`Get-Printer`** permite enumerar las impresoras configuradas en el puesto de trabajo y obtener información acerca de ellas. Este cmdlet se utiliza de una manera muy sencilla, con o sin parámetros:

Parámetro	Descripción
`-Name` `<String[]>`	Especifica el nombre de la impresora sobre la que se desea recuperar información.

Ejemplo: mostrar la información de una impresora

```
PS C:\Windows\system32> Get-Printer -Name "HP LaserJet CP3525" |

Format-List
Name,Type,DriverName,PortName,PrintProcessor,Shared,Published


Name          : HP LaserJet CP3525
Type          : Local
DriverName    : HP Universal Printing PCL 5
PortName      : IP_192.168.0.50
PrintProcessor: hpcpp180
Shared        : False
Published     : False
```

b. Modificar los parámetros de una impresora

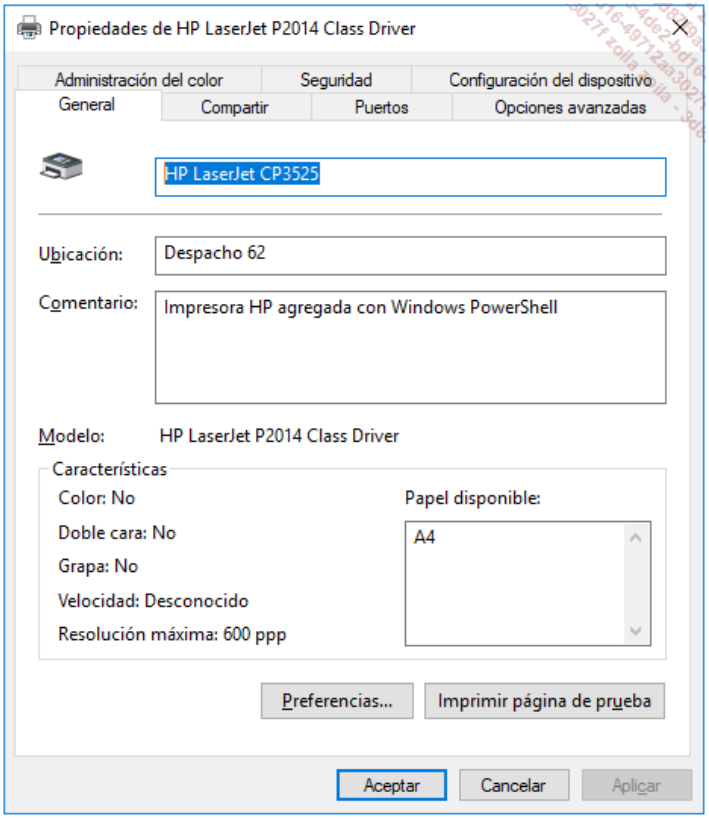
Set-Printer permite actualizar la configuración de una impresora en un puesto de trabajo. Los parámetros son parecidos a los de **Add-Printer**. He aquí una tabla resumen de algunos parámetros de **Set-Printer**:

Parámetro	Descripción
-Comment <String>	Actualiza el texto en el campo **Comentarios** de las propiedades de la impresora.
-DriverName <String>	Especifica el nombre del controlador de impresión que se ha de utilizar.
-Location <String>	Informa el campo **Ubicación** en las propiedades de la impresora.
-Name <String[]>	Indica el nombre de la impresora sobre la que se realizarán las modificaciones.
-PortName <String>	Especifica el nombre del puerto que se ha de utilizar para comunicarse con la impresora.

Ejemplo: modificar los parámetros de una impresora

```
PS C:\Windows\system32> Set-Printer -Name "HP LaserJet CP3525"
-Comment "Impresora HP agregada con Windows PowerShell"
-Location "Despacho 62" -DriverName "HP Universal Printing PCL 5"
```

Se han realizado las modificaciones correspondientes sobre la impresora indicada:



Actualización de las propiedades de la impresora HP LaserJet CP3525

c. Renombrar una impresora

El cmdlet **`Rename-Printer`** permite renombrar la impresora y de este modo presentarla bajo otro nombre a los usuarios.

El uso de este cmdlet es muy sencillo y requiere solamente dos parámetros:

Parámetro	Descripción
`-Name` `<String>`	Indica el nombre de la impresora que se renombrará.
`-NewName` `<String>`	Indica el nuevo nombre que se asignará a la impresora.

Ejemplo: renombrar una impresora

```
PS C:\Windows\system32> Rename-Printer -Name "HP LaserJet
CP3525"
-NewName "IMP-FRPARB1F5"
```

d. Eliminar una impresora

Del mismo modo que es posible agregar una impresora, también es posible eliminarla. Este es el rol de **`Remove-Printer`**. Basta con indicar, simplemente, el nombre de la impresora que se ha de eliminar.

Parámetro	Descripción
`-Name` `<String[]>`	Especifica el nombre de la impresora que se ha de eliminar.

Ejemplo: eliminar una impresora

```
PS C:\Windows\system32> Remove-Printer -Name "HP LaserJet CP3525"
```

A continuación, se elimina la impresora de la ventana **Dispositivos e impresoras**. La configuración de los puertos de impresión (**`Get-PrinterPort`**) y de los controladores de impresión (**`Get-PrinterDriver`**) no se ve afectada.

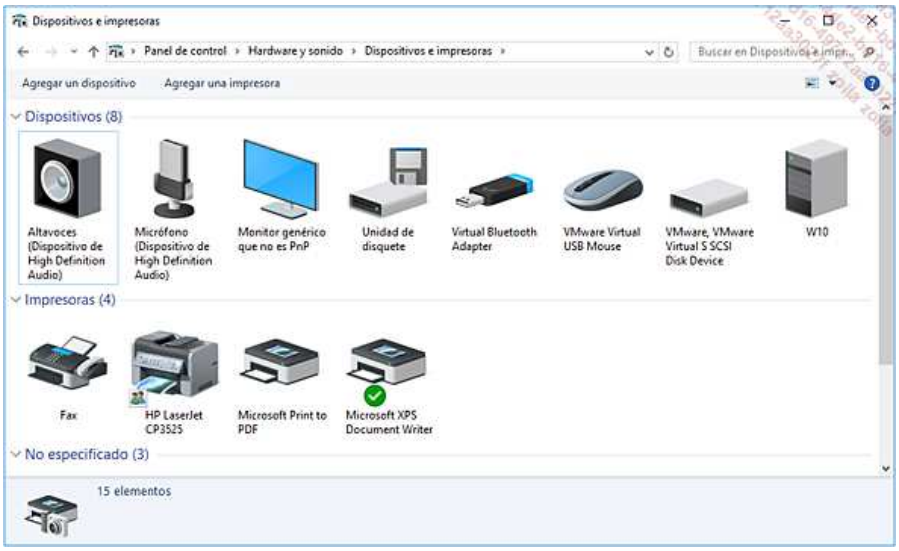
e. Definir una impresora predeterminada

Definir una impresora predeterminada permite, cuando algún usuario desea imprimir un documento, que el dispositivo de impresión elegido esté seleccionado automáticamente. En general, cuando un puesto de trabajo posee varias impresoras configuradas, la impresora predeterminada es aquella que se utiliza con una mayor frecuencia.

Para cambiar y definir la nueva impresora predeterminada, se utiliza una petición WMI, con la clase **`Win32_Printer`**. El método utilizado es `SetDefaultPrinter`.

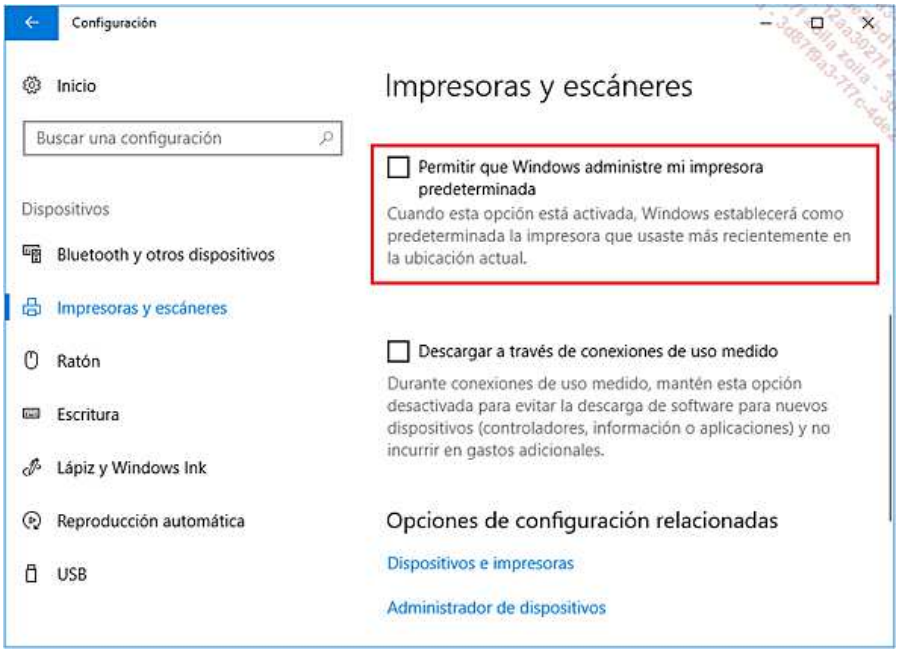
Ejemplo 1: cambiar la impresora predeterminada por la impresora virtual XPS

```
PS C:\Windows\system32> (Get-WmiObject -Class Win32_Printer
-Filter "Name='Microsoft XPS Document
Writer'").SetDefaultPrinter()
```



La impresora predeterminada es ahora Microsoft XPS Document Writer

Tenga precaución con la siguiente particularidad de Windows 10: para que la configuración de la impresora predeterminada se tenga en cuenta correctamente, hay que deshabilitar primero la opción **Permitir que Windows administre mi impresora predeterminada**. Esta opción se encuentra en la sección **Impresoras y escáneres** de la sección **Dispositivos** de la aplicación **Configuración**.



Permitir que Windows administre la impresora predeterminada

Es posible deshabilitar esta opción a través del registro, y por lo tanto también a través de una línea de comando PowerShell:

```
PS C:\WINDOWS\system32> Set-ItemProperty
"HKCU:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows"
-Name LegacyDefaultPrinterMode -Value 1
```

Si fuera necesario, para volver a habilitar esta opción, vuelva a ejecutar la misma línea de comando, pero definiendo el valor a 0.

Ejemplo 2: cambiar la impresora predeterminada por la impresora LaserJet CP3525

```
PS C:\Windows\system32> (Get-WmiObject -Class Win32_Printer
-Filter "Name='HP LaserJet CP3525'").SetDefaultPrinter()
```

Administrar las aplicaciones instaladas

En el capítulo Buscar y recopilar información, ha visto que es posible enumerar las aplicaciones instaladas en un puesto de trabajo, principalmente con la consulta WMI **`Win32_Product`**. Sin embargo, este método enumera únicamente las aplicaciones instaladas mediante Windows Installer.

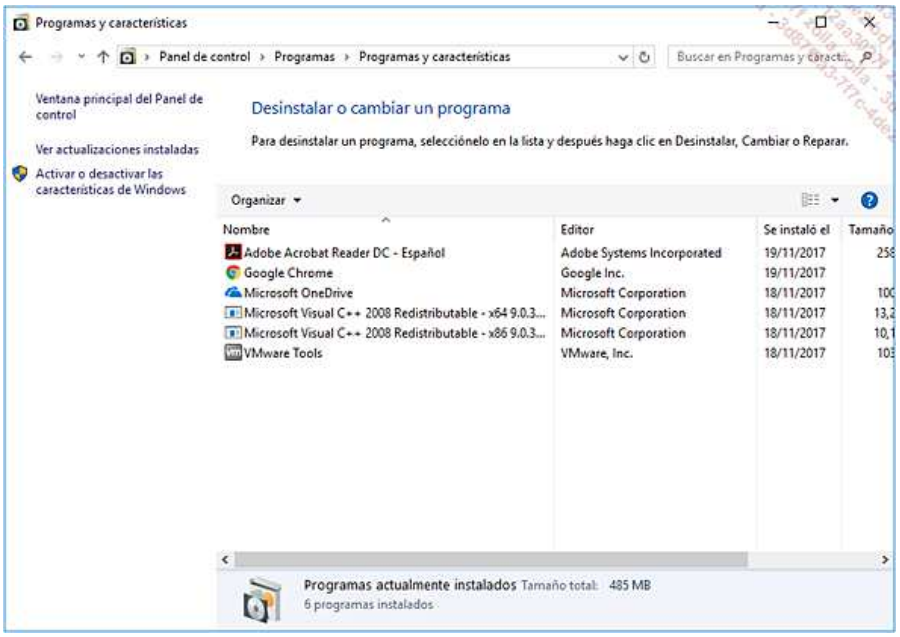
Existen otros métodos, de los cuales hay uno particularmente interesante: utilizar el registro. Este método presenta varias ventajas:

- Las manipulaciones en el registro con Windows PowerShell son sencillas, con los cmdlets llamados **`*-Item*`** (a excepción de **`Invoke-Item`**), como hemos visto en el capítulo Las unidades de Windows PowerShell.
- Cada una de las aplicaciones instaladas en el equipo posee una clave de registro con cierta información asociada. Están disponibles en las siguientes claves de registro:

Arquitectura Windows	Clave de registro
Windows 32 bits	HKLM:\SOFTWARE\Microsoft\Windows\ CurrentVersion\ Uninstall
Windows 64 bits	HKLM:\SOFTWARE\Microsoft\Windows\ CurrentVersion\ Uninstall HKLM:\SOFTWARE\Wow6432Node\Microsoft\ Windows\ CurrentVersion\Uninstall

Hay un par de datos muy interesantes relativos a cada aplicación instalada: el valor `DisplayName`, que se corresponde con el nombre de la aplicación tal y como está registrada en Programas y características del Panel de control. La segunda, que permite lanzar la desinstalación de la aplicación, es el valor `UninstallString`. A continuación veremos todo esto con más detalle.

A nivel de Windows, esto corresponde a **Desinstalar o cambiar un programa**, presente en **Programas y características** del **Panel de control**.



Lista de aplicaciones instaladas

1. Enumerar las aplicaciones instaladas

En primer lugar, ¿cómo elaborar una lista de las aplicaciones instaladas? Para ello hay que recuperar el dato registrado en el valor del registro `DisplayName`. Este valor se encuentra en las subclaves de HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall y HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\ Uninstall para cada programa instalado. Es posible recuperar otra información, como el número de versión de la aplicación (mediante el valor `DisplayVersion`) o la fecha de instalación del programa (mediante el valor `InstallDate`).

He aquí un ejemplo para recuperar la lista de aplicaciones instaladas:

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*    |    ?
{$_.DisplayName
-ne $null} | Select-Object DisplayName, DisplayVersion,InstallDate
| Format-Table

DisplayName
                    DisplayVersion InstallDate
-----------                                                -------------
- -----------
Microsoft Visual Studio 2010 Tools for Office Runt... 10.0.50903
```

```
Módulo de idioma de Microsoft Visual Studio 2010 T... 10.0.50903
Mozilla Firefox 50.1.0 (x64 fr)                       50.1.0
Notepad++ (64-bit x64)                                7.3.1
Microsoft Office Standard 2013
15.0.4569.1506
Microsoft Visual C++ 2010 x64 Redistributable - 1...
10.0.40219 20170107
paint.net
                          4.0.13 20170120
Microsoft Visual Studio 2010 Tools for Office Runt...
10.0.50903 20170107
Microsoft Office Standard 2013
15.0.4569.1506 20170112
[...]
```

Si el ejemplo anterior muestras las aplicaciones de 64 bits instaladas, hay que hacer lo mismo para mostrar las aplicaciones de 32 bits, pero con la siguiente clave de registro: HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\ CurrentVersion\Uninstall.

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninsta
ll\* | ?
{$_.DisplayName -ne $null} | Select-Object
 DisplayName,DisplayVersion,InstallDate |
Format-Table

DisplayName
DisplayVersion InstallDate
-----------                                           -------------
- -----------
Mozilla Maintenance Service                          50.1.0
Adobe Acrobat Reader DC - Español
15.023.20053 20170120
Microsoft Visual C++ 2010 x86 Redistributable - 1...
10.0.40219 20170107
[...]
```

En función de sus necesidades, **Format-Table** puede reemplazarse por **Out-File** o **Export-Csv**, que permiten respectivamente exportar el resultado del comando a un archivo de texto o a un archivo en formato CSV (*Comma-Separated Value*). En ambos cmdlets es preciso agregar los parámetros necesarios para su funcionamiento.

Ejemplo 1: exportar el resultado a un archivo de texto

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*
| ? {$_.DisplayName -ne $null} | Select-Object
DisplayName,DisplayVersion,
Publisher,InstallDate | Out-File -FilePath C:\Temp\exportSoft.txt
```

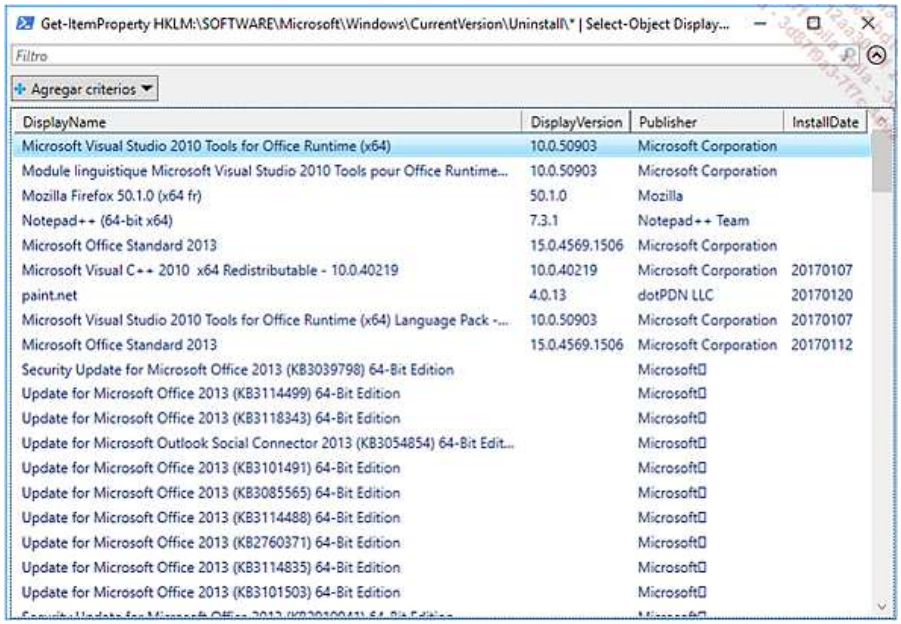
Ejemplo 2: exportar el resultado a un archivo CSV

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*
|     ?     {$_.DisplayName     -ne     $null}     |     Select-Object
DisplayName,DisplayVersion,
Publisher,InstallDate | Export-Csv -Path C:\Temp\exportSoft.csv
```

Ejemplo 3: mostrar el resultado en una ventana interactiva

También puede utilizar **`Out-GridView`**, que muestra el resultado de manera gráfica y de forma interactiva:

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*
|     ?     {$_.DisplayName     -ne     $null}     |     Select-Object
DisplayName,DisplayVersion,
Publisher,InstallDate | Out-GridView
```



*Resultado de la exportación de datos con **`Out-GridView`***

2. Filtrar las consultas

El número de aplicaciones instaladas puede ser elevado en un puesto de trabajo, de modo que conviene poder filtrar las consultas para obtener un resultado más conveniente.

a. Filtrar por nombre

A nivel del registro, el valor `DisplayName` contiene el nombre de la aplicación instalada en el equipo. Con ayuda de **`Where-`**

Object (representado por el carácter «**?**» en el siguiente ejemplo), es posible realizar una búsqueda basándose en el valor de `DisplayName`.

Ejemplo: filtrar las aplicaciones que contienen la cadena de caracteres «paint»

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\* |?
{$_.DisplayName -like "*paint*"} | Select-Object
DisplayName,DisplayVersion,InstallDate

DisplayName DisplayVersion InstallDate
----------- -------------- -----------
paint.net   4.0.13         20170120
```

Observe que con el operador **-like** en **Where-Object**, el valor buscado puede contener el carácter comodín «*****» (wildcard). Con el operador **-match**, tiene la posibilidad de incluir una expresión regular para llevar a cabo su búsqueda. En ambos casos, esto permite extender la búsqueda sin tener que conocer el nombre exacto de la aplicación buscada.

b. Filtrar por fabricante

Es posible filtrar por fabricante, lo que permite obtener una lista de las aplicaciones que han sido editadas por la misma empresa. Para ello, debe realizar una búsqueda sobre el valor de `Publisher`.

Ejemplo 1: filtrar las aplicaciones del fabricante «Adobe»

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninst
all\*
| ? {$_.Publisher -like "*Adobe*"} | Select-Object
DisplayName,DisplayVersion,InstallDate,Publisher

DisplayName                DisplayVersion InstallDate Publisher
-----------                -------------- ----------- ---------
Adobe Acrobat Reader DC... 15.023.20053     20170120       Adobe
Systems Incorporated
```

Ejemplo 2: filtrar las aplicaciones cuyo fabricante no sea «Microsoft»

En el caso de que existan muchas aplicaciones del mismo fabricante, es posible filtrar para excluir el fabricante correspondiente del resultado. Esto se lleva a cabo con ayuda del operador **-notlike**:

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\* | ?
{$_.Publisher -notlike "Microsoft*"} | Select-Object
DisplayName,DisplayVersion,InstallDate,Publisher

DisplayName                                DisplayVersion  InstallDate
Publisher
-----------                          -------------- ----------- -------
--
Mozilla          Firefox          50.1.0          (x64          fr)
50.1.0                       Mozilla
Notepad++     (64-bit     x64)
7.3.1                        Notepad++ Team
paint.net                          4.0.13          20170120     dotPDN
LLC
```

3. Desinstalar una aplicación

Para lanzar el asistente que permite desinstalar una aplicación (o, en ocasiones, su desinstalación automática), hay que recuperar el valor de `UninstallString` presente en la clave de registro de la aplicación. El dato registrado en este valor del registro contiene una cadena de caracteres que representa la línea de comando que se ejecuta cuando se hace clic en el botón **Desinstalar** en la ventana **Programas y características**.

```
PS C:\Windows\system32> Get-ItemProperty
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\* | ?
{$_.DisplayName -like "paint.net"} | Select-Object
DisplayName,UninstallString

DisplayName UninstallString
----------- ---------------
paint.net   MsiExec.exe /X{6AC1101E-7561-43C9-BEEA-4AB1D220D8FF}
```

Debe ejecutarse esta línea de comando para lanzar el programa de desinstalación. Es posible utilizar el cmdlet **`Start-Process`** (específico para esto) tal y como se muestra a continuación, introduciendo la línea de comando manualmente:

```
PS C:\Windows\system32> Start-Process MsiExec.exe -ArgumentList
"/X{6AC1101E-7561-43C9-BEEA-4AB1D220D8FF}"
```

En el caso de un script, es posible proceder de la siguiente manera. Preste atención, sin embargo, pues no se trata de algo general, y el comando **`Start-Process`** debe adaptarse en función del dato presente en `UninstallString`.

Ejemplo: lanzar el proceso de desinstalación

En este ejemplo, se recuperan las propiedades del registro `DisplayName` y `UninstallString` de la aplicación paint.net y

se almacenan en la variable `$appli`. A continuación, se divide en dos la cadena de caracteres contenida en el valor `UninstallString` con ayuda del método `Split(" ")` para permitir la ejecución del comando con **`Start-Process`**:

- `Split(" ")[0] = MsiExec.exe`
- `Split(" ")[1] = /X{6AC1101E-7561-43C9-BEEA-4AB1D220D8FF}`

```
PS C:\Windows\system32> $appli = Get-ItemProperty
HKLM:\SOFTWARE\\Microsoft\Windows\CurrentVersion\Uninstall\* | ?
{$_.DisplayName -like "paint.net"} | Select-Object
DisplayName,UninstallString
PS C:\Windows\system32> Start-Process
$appli.UninstallString.Split(" ")[0] -ArgumentList
$appli.UninstallString.Split(" ")[1]
```

En este caso, y para hacer que la desinstalación de la aplicación sea completamente silenciosa, bastará con incluir los parámetros `/quiet` y `/passive` de Windows Installer. Esto produce, por ejemplo, el siguiente comando:

```
PS C:\Windows\system32> $wiexe = $appli.UninstallString.Split(" ")
[0]
PS C:\Windows\system32> $uargs = $appli.UninstallString.Split(" ")
[1] + "
/quiet /passive"
PS C:\Windows\system32> Start-Process $wiexe -ArgumentList $uargs
```

Lanzar un ejecutable

Para lanzar un ejecutable o un script, hay que indicar la ruta absoluta o la ruta relativa al archivo que desea ejecutar, salvo si está presente en alguna carpeta de la variable de entorno `PATH`.

Así, para lanzar archivos ejecutables o scripts presentes en la ubicación actual, estos deben estar precedidos de «**.**».

Ejemplo:

```
PS C:\Temp> .\myScript.ps1
```

El cmdlet **`Start-Process`** permite arrancar procesos y por lo tanto lanzar ejecutables. En este caso concreto, no es obligatorio precisar la ruta relativa si el archivo se encuentra en la carpeta actual:

Ejemplo:

```
PS C:\Temp> Start-Process MediaCreationToolx64.exe
```

Para lanzar un script de Windows PowerShell desde otro entorno (símbolo del sistema o algún otro programa), debe invocarse el motor de script powershell.exe, seguido del script que se ha de ejecutar.

He aquí dos ejemplos que permiten lanzar un script de Windows PowerShell desde el símbolo del sistema (cmd.exe):

Ejemplo 1: lanzar un script de PowerShell desde el símbolo del sistema

```
C:\Temp>powershell.exe .\debug.ps1
** ¡Inicio del script! **
Hello World!

viernes 26 de septiembre de 2014 15:20:01
** ¡Fin del script! **
```

Ejemplo 2: mismo escenario que antes, pero con argumentos

En el caso de que se pasen argumentos en la ejecución del script, basta con indicarlos a continuación, separándolos con un carácter como mínimo.

```
C:\Temp>powershell.exe .\param.ps1 info1 1234
Primer parámetro: info1
Segundo parámetro: 1234
```

Windows Installer (MSI)

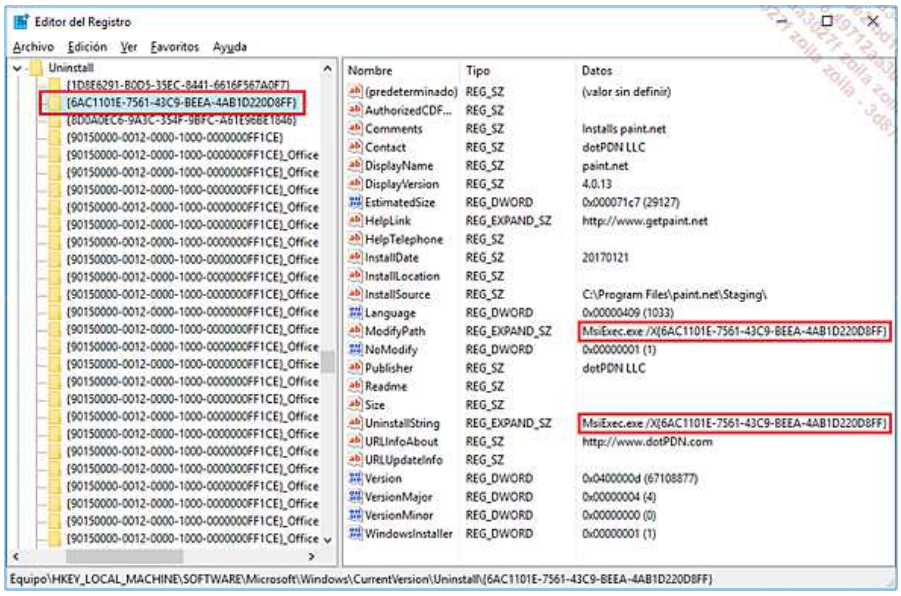
Saber instalar, actualizar y desinstalar aplicaciones por línea de comandos, y en ocasiones de manera remota, es algo fundamental para un administrador de puestos de trabajo. A continuación veremos cómo interactuar con los paquetes MSI, bien en un puesto de trabajo local, o también en uno o varios equipos remotos.

1. ¿Qué es Windows Installer?

Windows Installer (anteriormente conocido como Microsoft Installer) es un motor de instalación de aplicaciones que se utiliza para instalar, actualizar o desinstalar aplicaciones. El archivo que permite ejecutar Windows Installer es msiexec.exe, mientras que los paquetes que sirven para la instalación de las aplicaciones y de las actualizaciones son, respectivamente, archivos con las extensiones .msi y .msp.

Cada aplicación instalada con Windows Installer está identificada por un GUID, que se referencia a continuación en el registro. Estos identificadores se corresponden con las subclaves presentes en:

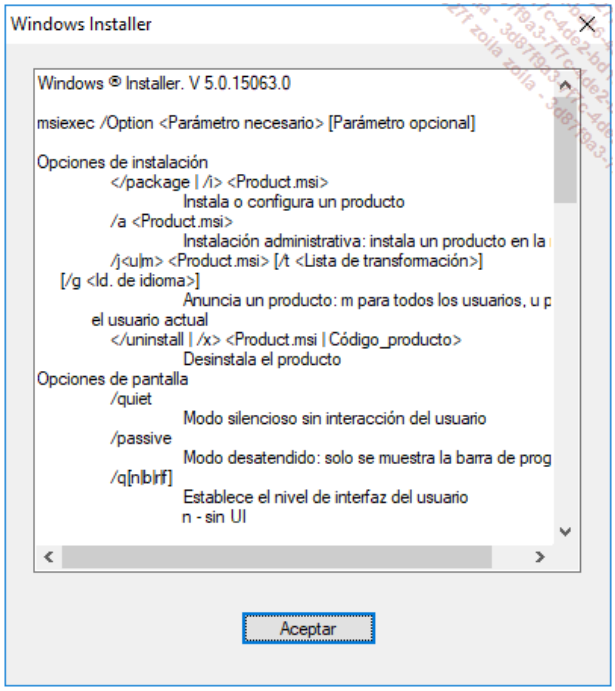
- HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
- HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\ Uninstall



Clave GUID de la aplicación paint.net en el Editor del Registro

También puede consultar la ayuda del ejecutable msiexec.exe. Este posee muchos parámetros, muy prácticos, que conviene conocer. Para abrir la ayuda, indique el parámetro `/help` en la llamada al ejecutable.

```
PS C:\Windows\system32> Start-Process msiexec.exe -ArgumentList "/help"
```



2. Instalar un paquete MSI

El principal interés de un paquete MSI para un administrador de sistemas es, sin lugar a dudas, la posibilidad de instalar una aplicación de manera totalmente silenciosa: con el objetivo de evitar cualquier tipo de molestia al usuario, y también para no tener que llevar a cabo la instalación manual de la aplicación en cada puesto, evitando también los posibles errores manuales.

En primer lugar, veremos cómo instalar un paquete MSI en un puesto de trabajo de manera local. Conviene saber que no existe ningún cmdlet dedicado a esta operación, pues es en realidad el archivo msiexec.exe el encargado de gestionar el conjunto del proceso.

Ejemplo 1: instalar un paquete MSI en modo silencioso

```
PS C:\Windows\system32> msiexec.exe /i C:\Temp\mySoft.msi /qn
```

En este ejemplo, los parámetros significan lo siguiente:

- `/i`: instala o configura una aplicación. Este parámetro puede sustituirse por `/package`, que posee exactamente el mismo rol.
- `/qn`: significa «quiet» (modo silencioso) + «no UI» (sin interfaz de usuario). Así, el proceso de instalación se lleva a cabo de una manera completamente silenciosa, sin ninguna interacción con el usuario. Además, no aparece absolutamente nada por pantalla.

Pero ¿por qué no lanzar la instalación de una aplicación en un puesto de trabajo remoto? Aquí es donde interviene la funcionalidad de PowerShell Remoting: permitiendo ejecutar esta misma línea de comando, pero sobre uno o varios puestos remotos. Para ello debe utilizar el cmdlet **`Invoke-Command`**, que es capaz de ejecutar un bloque de script (**`-ScriptBlock`**) en la máquina o las máquinas remotas indicadas.

Ejemplo 2: instalar una aplicación en varios puestos de trabajo remotos

```
PS C:\Windows\system32> Invoke-Command -ComputerName
PC01,PC02,PC03 -ScriptBlock {Start-Process -FilePath msiexec.exe
-ArgumentList "/i C:\Temp\mySoft.msi /qn" -Wait -Verb RunAs}
```

El archivo MSI debe copiarse previamente en los puestos de trabajo remotos. La ruta de acceso informada en el comando indica, por tanto, la ubicación del paquete MSI en el puesto de trabajo remoto. El paquete MSI también puede ponerse a disposición del comando en un recurso compartido de red.

Para obtener más información acerca del funcionamiento de **`Invoke-Command`**, consulte el capítulo Administración remota de puestos de trabajo.

3. Aplicar una actualización (.msp)

Las aplicaciones incluidas en un paquete MSI pueden recibir actualizaciones. Estas tienen la extensión de archivo .msp. Su uso es muy sencillo, pues basta con especificar sobre qué aplicación debe aplicarse esta actualización. Para ello, puede utilizar msiexec.exe, esta vez con el parámetro `/update`.

Ejemplo: actualizar una aplicación en un puesto local

```
PS C:\Windows\system32> msiexec.exe /update
C:\Temp\mySoft_patch.msp
```

4. Desinstalar una aplicación con un paquete MSI

Por último, para desinstalar una aplicación en un puesto de trabajo, debe poseer necesariamente el archivo .msi que ha servido para instalar la aplicación o conocer su código de producto. El parámetro de msiexec.exe que sirve para eliminar una aplicación de Windows es `/x` o `/uninstall`, como prefiera:

Ejemplo 1: desinstalar una aplicación con el archivo MSI

```
PS C:\Windows\system32> msiexec.exe /x C:\Temp\mySoft.msi
```

Ejemplo 2: desinstalar una aplicación con la clave GUID correspondiente

Recuerde que la clave GUID correspondiente al código de producto de la aplicación que desea desinstalar se encuentra en el registro. Consulte la sección ¿Cómo encontrar rápidamente el código de producto de una aplicación? que se describe más adelante.

Ejemplo 3: desinstalar una aplicación en puestos de trabajo

```
PS C:\Windows\system32> Start-Process msiexec.exe -ArgumentList
"/x {4498748D-F54C-4B84-AD4D-F8DA827FF65E} /qn"
```

remotos

```
PS C:\Windows\system32> Invoke-Command -ComputerName
PC01,PC02,PC03 -ScriptBlock {Start-Process -FilePath msiexec.exe
-ArgumentList "/x C:\Temp\mySoft_patch.msi /qn" -Wait -Verb RunAs}
```

¿Cómo encontrar rápidamente el código de producto de una aplicación?

Cada aplicación instalada en Windows a través de Windows Installer posee un código de producto, que de hecho es una clave GUID. Para encontrarla rápidamente, siga estas etapas:

- Abra el editor del registro.
- Abra la clave de registro siguiente: HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall (si se trata de una aplicación de 32 bits y trabaja con una versión de Windows de 64 bits) o bien HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall (para las aplicaciones de 64 bits).
- En el menú, haga clic en **Edición** y luego en **Buscar**... (o utilice la combinación de teclas [Ctrl] **F**).
- En el campo de búsqueda, introduzca el nombre de la aplicación que desea desinstalar (utilizando el mismo nombre de la aplicación que puede encontrar en Programas y características del Panel de control).
- Si la búsqueda produce algún resultado, consulte el dato de la propiedad de registro `DisplayName` y asegúrese de que se trata de la aplicación correcta que desea desinstalar. Si es así, la clave GUID se encuentra en la ruta de la clave de registro, y también está presente en el dato contenido en la propiedad `UninstallString` (consulte la captura de pantalla de la sección ¿Qué es Windows Installer? de este capítulo, por ejemplo).

Esta búsqueda de información puede realizarse con Windows PowerShell. Puede encontrar el conjunto de explicaciones y ejemplos en la sección Desinstalar una aplicación.

Presentación

Un administrador de paquetes es un sistema que permite administrar el conjunto del ciclo de vida de las aplicaciones desplegadas de esta manera en el puesto de trabajo. Presente desde hace mucho tiempo en Linux, esta característica permite abarcar todas las fases de recuperación, instalación, actualización, etc., de las aplicaciones instaladas en el puesto de trabajo. Todas estas etapas se automatizan completamente: basta con lanzar la operación mediante algunos cmdlets.

Incluido con la versión 5.0 de Windows PowerShell, el administrador de paquetes aparece en los sistemas operativos Windows gracias al módulo PackageManagement (anteriormente OneGet). Este último se ocupa de todo lo relativo al Software Discovery, Installation and Inventory (SDII), y responde a las problemáticas de los profesionales de la informática y de DevOps.

1. Arquitectura

Un paquete es, en realidad, un archivo que contiene tanto la aplicación que se ha de instalar (o descargar desde una URL que se indica en el archivo) como también los procedimientos (scripts) necesarios para la instalación de la aplicación. Estos paquetes se almacenan en repositorios, accesibles generalmente desde Internet. Este es el caso, por ejemplo, de Chocolatey, un repositorio de paquetes para las plataformas de Windows. Este repositorio, basado en NuGet, está gestionado por una comunidad que agrega y mantiene actualizados los paquetes, disponibles para su despliegue en puestos de trabajo.

NuGet es una plataforma de administración de paquetes (del lado servidor). Disponible en open source y bajo licencia Apache License 2.0, tiene la posibilidad de descargarla e instalarla en un servidor. Esta acción le permitirá disponer de su propio repositorio dentro de su empresa, que deberá alimentar, evidentemente, con los paquetes que desee desplegar.

2. El módulo Package Management

El módulo de gestión de paquetes le ofrece tres funciones principales, que veremos en cada una de las siguientes secciones.

Aquí se utilizará el repositorio Chocolatey para llevar a cabo el conjunto de ejemplos que se muestran a continuación, pues provee los paquetes que permiten instalar las aplicaciones en máquinas Windows.

a. Encontrar un paquete

Para poder encontrar el paquete correspondiente a sus necesidades, previamente debe tener acceso a los repositorios de Chocolatey. En efecto, este último no está referenciado en Windows PowerShell, de modo que será necesario instalar este proveedor antes de poder acceder a los paquetes. Esta operación se realiza a través del cmdlet **`Install-PackageProvider`**.

Una vez configurado el acceso al repositorio efectivo, el módulo de gestión de paquetes le permite recorrer y descubrir paquetes dentro de estos repositorios, con el objetivo de encontrar el nombre del paquete correspondiente a la aplicación que desea instalar, incluyendo eventualmente una versión específica. Esta etapa se lleva a cabo con ayuda del cmdlet **`Find-Package`**. Si bien el nombre del paquete es, por lo general, similar al de la aplicación buscada, este no es siempre el caso.

b. Instalar paquetes

La instalación de paquetes consiste en ejecutar el procedimiento de instalación de paquetes indicándolos por su nombre. Windows PowerShell se ocupa entonces de descargar el paquete o los paquetes necesarios (si se indican varios, o incluso en caso de existir dependencias con otros paquetes) para la instalación de la aplicación. Una vez recuperados, la aplicación se instala en el puesto de trabajo de forma totalmente silenciosa: no se requiere ninguna interacción con el usuario durante toda la fase de instalación. Esta etapa se realiza con el cmdlet **`Install-Package`**.

Por ello, la gestión de paquetes con Windows PowerShell se encarga de llevar a cabo todas las operaciones molestas realizadas generalmente por los administradores de sistemas:

- Comprobar que el instalador que se posee corresponde a la última versión de la aplicación en el sitio del fabricante.
- Si no es el caso, descargarla.
- Lanzar la instalación de la aplicación y seguir todas las etapas del asistente de instalación (si no es posible lanzar la instalación silenciosa).

- Eventualmente, empaquetar la aplicación para un despliegue a través de GPO, SCCM o alguna otra herramienta externa.

Así, recurriendo a dos o tres comandos, se automatizan todas estas etapas. Las ventajas de disponer de un administrador de paquetes en un sistema operativo son múltiples: facilita la administración y permite ahorrar un tiempo considerable, sobre todo en comparación con una administración con el ratón de las etapas descritas anteriormente.

Diferencias con Windows Store

Tenga precaución de no confundir el administrador de paquetes con Windows Store. El administrador de paquetes es una herramienta claramente orientada a los administradores del sistema, que presenta todo un conjunto de ventajas, sobre todo en ciertos escenarios concretos.

Otra diferencia importante es la actualización de las aplicaciones. Si bien las apps que provienen de Windows Store se actualizan de manera automática, no ocurre así con las aplicaciones provenientes de repositorios (salvo si la aplicación posee su propio programa de actualizaciones y no se encuentra deshabilitado). Para estas últimas, será necesario ejecutar una línea de comando para ordenar la actualización de la versión de la aplicación.

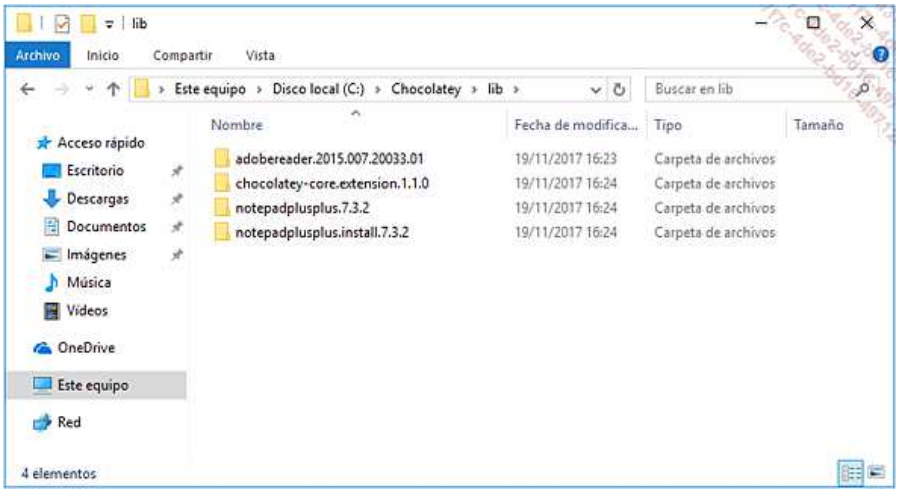
c. Herramienta de inventario

Si el módulo de gestión de paquetes de Windows PowerShell le permite desplegar paquetes en Windows, se trata de un agregador. En efecto, el conjunto de programas instalados en el puesto de trabajo se obtienen a través de este módulo, ya se trate de aplicaciones instaladas por Windows Installer (MSI/MSU), Programas (.exe), o paquetes (PowerShellGet, NuGet…). Además de disponer de una interfaz unificada y coherente, el módulo Package Management puede utilizarse como herramienta de inventario.

El cmdlet **`Get-Package`** le permitirá enumerar todos los programas instalados en el puesto de trabajo.

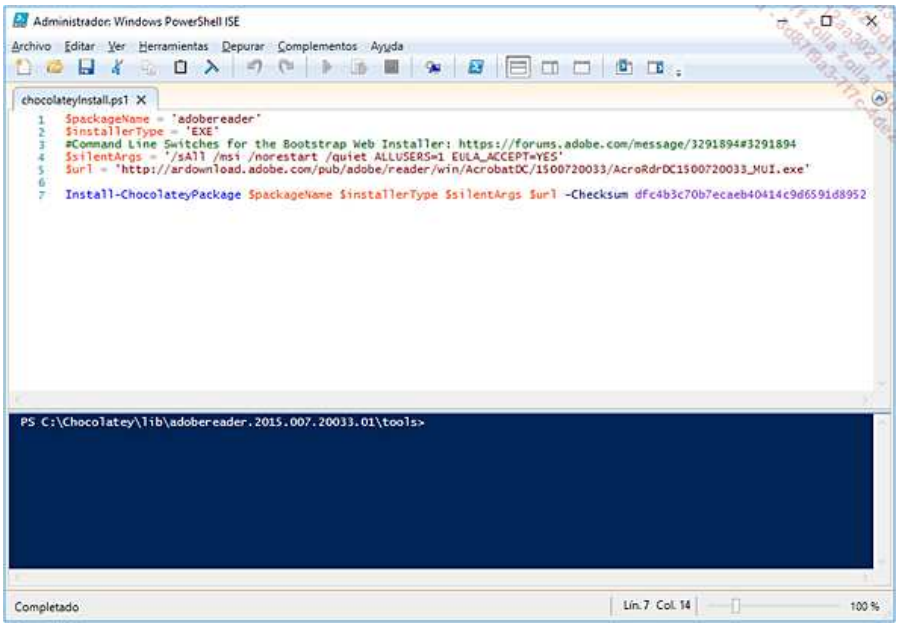
Requisitos previos y funcionamiento

Los paquetes descargados desde el repositorio Chocolatey se guardan en el puesto de trabajo en la carpeta C:\Chocolatey\lib. Cada uno de estos paquetes contiene los recursos que permitirán descargar la aplicación si fuera necesario y ejecutar su instalación en modo silencioso.



Los recursos asociados a los paquetes descargados se guardan en C:\Chocolatey\lib

La gran mayoría de los paquetes contiene scripts de PowerShell que permiten automatizar el conjunto de tareas. Es imprescindible, por lo tanto, que la política de ejecución de scripts en el puesto de trabajo sea, como mínimo, **`RemoteSigned`**. Si la política de ejecución de scripts es **`Restricted`**, los paquetes se descargarán, pero no será posible instalar la aplicación.



Ejemplo de script presente en un paquete de software de Chocolatey

A su vez, es preciso que el puesto de trabajo tenga acceso a Internet para descargar los paquetes desde los repositorios, y también desde los distintos fabricantes. Las aplicaciones se descargarán generalmente desde estos últimos.

Descripción de los cmdlets y demostración

En esta sección veremos el conjunto de cmdlets de Windows PowerShell que permiten interactuar con el administrador de paquetes:

- **`Install-PackageProvider`**: permite registrar un proveedor de paquetes y su repositorio.
- **`Get-PackageProvider`**: enumera el conjunto de proveedores de paquetes registrados.
- **`Find-Package`**: permite buscar un paquete en los proveedores de paquetes inscritos.
- **`Install-Package`**: lanza la instalación de la aplicación o del paquete correspondiente.
- **`Uninstall-Package`**: lanza la desinstalación de la aplicación o del paquete.
- **`Get-Package`**: enumera el conjunto de programas y paquetes instalados.

1. Agregar un proveedor de paquetes

Si bien por defecto el proveedor de paquetes Chocolatey no está instalado en Windows PowerShell, sí está referenciado como tal en una lista que el intérprete de comandos recupera de Internet. Para instalar un proveedor de paquetes a través de Windows PowerShell, hay que usar el cmdlet **`Install-PackageProvider`**. He aquí sus parámetros:

Parámetro	Descripción
`-Force`	Fuerza la instalación del proveedor de paquetes sin solicitar confirmación.
`-Name` `<String[]>`	Nombre del proveedor de paquetes que se ha de registrar.

Ejemplo: inscribir el proveedor de paquetes Chocolatey

```
PS C:\Windows\system32> Install-PackageProvider -Name Chocolatey -Force
```

```
Name                    Version          Source           Summary
----                    -------          ------           -------
chocolatey                      2.8.5.130             https://onege...
ChocolateyProtot
```

Si le proveedor NuGet no está preinstalado en el puesto de trabajo, entonces Windows PowerShell se encargará de hacerlo. Se trata de un requisito previo para la instalación del proveedor Chocolatey.

Windows PowerShell se encarga entonces de descargar los archivos necesarios para registrar el proveedor Chocolatey. Los archivos descargados provienen del sitio oneget.org.

Una vez realizada la operación, puede comprobar con el cmdlet **`Get-PackageProvider`** si está presente el proveedor Chocolatey:

```
PS C:\Windows\system32> Get-PackageProvider

Name            Version     DynamicOptions
----            -------     --------------
Chocolatey         2.8.5.130     SkipDependencies, ContinueOnFailure,
Exclude...
msi             3.0.0.0     AdditionalArguments
msu             3.0.0.0
NuGet               2.8.5.208     Destination, ExcludeVersion, Scope,
SkipDepe...
PowerShellGet   1.0.0.1       PackageManagementProvider, Type, Scope,
Allo...
Programs                  3.0.0.0             IncludeWindowsInstalar,
IncludeSystemComponent
```

2. Buscar un paquete

Ahora que tiene acceso a un repositorio, debe buscar el nombre del paquete de software correspondiente a la aplicación que desea instalar. Si bien en la mayoría de los casos el nombre del paquete es parecido al de la aplicación, igualmente conviene saber cómo proceder. Esta operación se lleva a cabo con ayuda del cmdlet **`Find-Package`**. He aquí los parámetros:

Parámetro	Descripción
`-AllVersions`	Muestra el conjunto de versiones de los paquetes de software disponibles. Por defecto, solo se recupera la versión más reciente.

Parámetro	Descripción
-Filter <String>	Especifica los términos que hay que buscar en los campos Name y Description.
-MaximumVersion <String>	Especifica la versión máxima del paquete de software que quiere encontrar. Por defecto, **Find-Package** buscará la versión más reciente.
-MinimumVersion <String>	Especifica la versión mínima del paquete de software que desea encontrar.
-Name <String[]>	Nombre del paquete o de los paquetes de software que hay que buscar. Está permitido el uso del carácter comodín * (*wildcard*).
-ProviderName <String[]>	Indica el nombre del proveedor de paquetes sobre el que se realizará la búsqueda. Para conocer los proveedores disponibles, ejecute el cmdlet **Get-PackageProvider**.
-**RequiredVersion** <String>	Indica la versión exacta del paquete de software que se va a buscar.
-Source <String[]>	Indica sobre qué fuente (repositorio) se buscarán los paquetes. Por defecto, se consultarán todas las fuentes. Para conocer los repositorios disponibles, ejecute el cmdlet **Get-PackageSource**.

Para buscar específicamente los paquetes provenientes de Chocolatey, se recomienda especificar en la línea de comando el parámetro **-ProviderName** o **-Source** indicando el nombre del proveedor. Si no se especifica nada, el comando **Find-Package** realizará la búsqueda en todos los repositorios y se obtendrán resultados provenientes de PSGallery (proveedor de PowerShellGet).

Ejemplo: buscar una aplicación con ayuda del filtro -Filter

```
PS C:\Windows\system32> Find-Package -Filter paint -ProviderName
Chocolatey

Name                    Version          Source           Summary
----                    -------          ------           -------
paint.net               4.0.6            chocolatey       Paint.NET
is...
krita                    3.1.2.2           chocolatey       Digital
pain...
drawpile                1.0.6            chocolatey       Drawpile
is ...
```

Con ayuda del parámetro **-Filter**, **Find-Package** devuelve también todos los paquetes donde la cadena de caracteres especificada esté presente en los campos nombre (`Name`) o resumen (`Summary`).

Ejemplo: buscar una aplicación con ayuda del parámetro -Name

```
PS C:\Windows\system32> Find-Package -Name notepad* -ProviderName
Chocolatey

Name                                Version          Source
Summary
----                              -------          ------          -----
--
notepadplusplus                     7.3.1              chocolatey
Notepad+...
notepadplusplus.install           7.3.1              chocolatey
Notepad+...
notepadplusplus.commandline     7.3.1              chocolatey
Notepad+...
Notepadplusplus.Settings          1.0.0.20141029     chocolatey
Allows N...
notepad2                           4.2.25.20160422   chocolatey      A
fast a...
notepad2-mod                        4.2.25.98500      chocolatey
Notepad2...
notepadreplacer                     1.1.6              chocolatey
Replace ...
```

El uso del parámetro **-Name** permite buscar únicamente a nivel del nombre. Es imprescindible, entonces, utilizar el carácter comodín * (*wildcard*) para extender la búsqueda y encontrar más fácilmente el paquete deseado.

Ejemplo: buscar todas las versiones de la aplicación Adobe Reader

```
PS C:\Windows\system32> Find-Package -Name adobereader -AllVersions

-ProviderName Chocolatey

Name                    Version          Source           Summary
----                    -------          ------           -------
adobereader               10.0.1             chocolatey        Adobe
```

```
Reader...
adobereader                10.0.1.1           chocolatey         Adobe
Reader...
adobereader                10.1.0             chocolatey         Adobe
Reader...
adobereader                10.1.1             chocolatey         Adobe
Reader...
adobereader                10.1.2             chocolatey         Adobe
Reader...
adobereader                10.1.3             chocolatey         Adobe
Reader...
adobereader                 10.1.3.20120712   chocolatey         Adobe
Reader...
adobereader                11.0.01            chocolatey         Adobe
Reader...
adobereader                 11.0.01.20130318  chocolatey         Adobe
Reader...
adobereader                11.0.02            chocolatey         Adobe
Reader...
adobereader                11.0.03            chocolatey         Adobe
Reader...
adobereader                11.0.04            chocolatey         Adobe
Reader...
adobereader                11.0.06            chocolatey         Adobe
Reader...
adobereader                11.0.07            chocolatey         Adobe
Reader...
adobereader                11.0.08            chocolatey         Adobe
Reader...
adobereader                11.0.09            chocolatey         Adobe
Reader...
adobereader                 11.0.09.20140924  chocolatey         Adobe
Reader...
adobereader                 11.0.09.20140925  chocolatey         Adobe
Reader...
adobereader                11.0.10            chocolatey         Adobe
Reader...
adobereader                 2015.007.20033    chocolatey         Adobe
Reader...
adobereader                 2015.007.2003...  chocolatey         Adobe
Reader...
```

Ejemplo: buscar todas las versiones de la aplicación Adobe Reader comprendidas entre las versiones 11.0.09 y 2015.007.20033

```
PS C:\Windows\system32> Find-Package -Name adobereader -
MinimumVersion
11.0.09 -MaximumVersion 2015.007.20033 -AllVersions -ProviderName
Chocolatey

Name                    Version          Source           Summary
----                    -------          ------           -------
adobereader               11.0.09            chocolatey         Adobe
Reader...
adobereader                11.0.09.20140924  chocolatey         Adobe
Reader...
adobereader                11.0.09.20140925  chocolatey         Adobe
Reader...
adobereader               11.0.10            chocolatey         Adobe
Reader...
```

```
adobereader                2015.007.20033    chocolatey        Adobe
Reader...
```

Nombre de los paquetes de software más comunes

He aquí una lista no exhaustiva de las aplicaciones más comunes y el nombre del paquete de software correspondiente.

Nombre de la aplicación	Nombre del paquete de software
7zip	`7zip`
Adobe Reader	`adobereader`
Avast Antivirus	`avastfreeantivirus`
Chrome	`GoogleChrome`
Excel Viewer	`Excel.Viewer`
FileZilla	`Filezilla`
Firefox	`Firefox`
LibreOffice	`libreoffice`
Malwarebytes	`malwarebytes`
mRemoteNG	`mRemoteNG`
Notepad++	`notepadplusplus`
Office 365 Business	`Office365Business`
Office 365 ProPlus	`Office365ProPlus`
OpenOffice	`OpenOffice`
Paint.NET	`Paint.net`
PowerPoint Viewer	`Powerpoint.Viewer`
Remote Desktop Connection Manager	`rdcman`
TeamViewer	`teamviewer`
Visio Viewer	`Visioviewer`
VLC Media Player	`vlc`
Word Viewer	`Word.Viewer`

3. Instalar un paquete

Para lanzar la instalación de un paquete de software, se utiliza el cmdlet **Install-Package**. Bastará con indicar el nombre del paquete de software que se quiere instalar y, de manera recomendada, el del proveedor de paquetes o de la fuente. He aquí sus parámetros:

Parámetro	Descripción
-Force	Omite las peticiones de confirmación.
-Name <String[]>	Nombre del paquete o de los paquetes de software que se van a instalar.
-ProviderName <String[]>	Indica el nombre del proveedor de paquetes cuyos paquetes se recuperarán. Para conocer los proveedores disponibles, ejecute el cmdlet **Get-PackageProvider**.
-RequiredVersion <String>	Indica la versión exacta del paquete de software que se va a instalar.
-Source <String[]>	Indica desde qué fuente (repositorio) se recuperarán los paquetes. Por defecto, se consultarán todas las fuentes. Para conocer los repositorios disponibles, ejecute el cmdlet **Get-PackageSource**.

Igual que para descubrir paquetes, se recomienda utilizar los parámetros **-ProviderName** o **-Source**. En efecto, un mismo nombre de paquete puede estar presente en varios repositorios, de modo que se recomienda especificar el origen del paquete (Chocolatey).

Tenga también precaución con la política de ejecución de scripts en PowerShell, que debe ser como máximo `RemoteSigned`. Consulte la sección Requisitos previos y funcionamiento para obtener una explicación más detallada.

Ejemplo: instalar un paquete de software

Una vez ejecutado el comando, Windows PowerShell se encarga

```
PS C:\Windows\system32> Install-Package -Name adobereader -
ProviderName
Chocolatey -Force

Name                    Version          Source           Summary
----                    -------          ------           -------
adobereader                 2015.007.2003... chocolatey         Adobe
Reader...
```

de descargar el paquete o los paquetes necesarios para llevar a cabo la instalación. Se muestra una barra de progreso en la parte superior de la consola Windows PowerShell, que muestra el estado de avance de la tarea. Observará, por otro lado, que en este caso la aplicación se descarga directamente desde el sitio del fabricante (lo que ocurre para un gran número de paquetes):

```
Downloading
'http://ardownload.adobe.com/pub/adobe/reader/win/AcrobatDC/1500720
033/
AcroRdrDC1500720033_MUI.exe'
    To C:\Users\Admin\AppData\Local\Temp\chocolatey\adobereader\
adobereaderinstall.EXE

  [ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
    ]
```

Una vez terminada la descarga, se realiza la instalación de la aplicación en el puesto de trabajo. Cuando Windows PowerShell le devuelve el control, la aplicación correspondiente está lista para su uso y también visible en la lista **Programas y características** del Panel de control de Windows. Toda esta etapa se realiza de manera completamente silenciosa.

Dado que no se ha indicado ninguna versión específica, se ha instalado la versión más reciente.

Ejemplo: instalar una versión concreta de un paquete de software

Si por algún motivo concreto es necesario instalar alguna versión concreta de una aplicación, es preciso indicarlo. Esto se hace con ayuda del parámetro **-RequiredVersion**.

```
PS C:\Windows\system32> Find-Package -Name notepadplusplus -
ProviderName
Chocolatey -RequiredVersion 6.8.4 | Install-Package -Force

Name                    Version         Source           Summary
----                    -------         ------           -------
notepadplusplus.install 6.8.4             chocolatey       Notepad++
is...
notepadplusplus          6.8.4            chocolatey       Notepad++
is...
```

Archivos MSI/MSU

También es posible instalar los archivos MSI y MSU con **`Install-Package`**. Sin tener que especificar ningún parámetro, salvo la ruta de acceso al archivo, la aplicación o la actualización se instalará de una manera totalmente silenciosa.

Ejemplo: instalar una aplicación proveniente de un archivo MSI

```
PS D:\> Install-Package D:\AdobeReader\AcroRead.msi

Name                                Version          Source
Summary
----                                -------          ------             --
-----
Adobe Acrobat Reader DC - F... 15.007.20033     D:\AdobeReade...
```

Ejemplo: instalar una actualización proveniente de un archivo MSU

```
PS D:\> Install-Package D:\WindowsTH-RSAT_WS2016-x64.msu -Force
ADVERTENCIA: Reboot is required to complete the installation
operation.
```

4. Actualizar un paquete

Es posible actualizar un paquete de software previamente instalado. Esta operación se realiza también con el cmdlet **`Install-Package`** y con los mismos parámetros.

Ejemplo: actualizar un paquete de software

```
PS C:\Windows\system32> Install-Package -Name notepadplusplus -
ProviderName
Chocolatey -Force

Name                    Version          Source           Summary
----                    -------          ------           -------
notepadplusplus.install 7.3.1             chocolatey       Notepad++
is...
notepadplusplus         7.3.1             chocolatey       Notepad++
is...
```

La aplicación Notepad++ se actualiza en el puesto de trabajo con la última versión disponible.

5. Desinstalar un paquete

Es posible desinstalar un paquete con **`Uninstall-Package`**. La ejecución de este comando eliminará el paquete que ha servido para la instalación de la aplicación (paquete del proveedor

Chocolatey), pero no desinstalará la aplicación del puesto de trabajo.

Es posible recuperar la lista de paquetes provenientes de Chocolatey con ayuda del siguiente comando:

```
PS C:\Windows\system32> Get-Package -ProviderName Chocolatey

Name                           Version          Source
ProviderName
----                           -------          ------             -------
-----
chocolatey-core.extension      1.1.0              C:\Chocolatey\...
Chocolatey
notepadplusplus                7.3.2              C:\Chocolatey\...
Chocolatey
notepadplusplus.install        7.3.2              C:\Chocolatey\...
Chocolatey
```

He aquí un resumen de los parámetros de **`Uninstall-Package`**:

Parámetro	Descripción
`-Force`	Omite las peticiones de confirmación.
`-Name` `<String[]>`	Nombre del paquete o de los paquetes de software que se van a desinstalar.
`-ProviderName` `<String[]>`	Indica el nombre del proveedor de paquetes cuyos paquetes se desinstalarán.

Ejemplo: desinstalar un paquete

```
PS C:\Windows\system32> Uninstall-Package -Name notepadplusplus

Name                           Version      Source           Summary
----                           -------      ------           -------
notepadplusplus                  7.3.1        C:\Chocolatey... Notepad++
is...
notepadplusplus.install          7.3.1        C:\Chocolatey... Notepad++
is...
```

La desinstalación elimina todos los recursos asociados al paquete que ha servido para instalar la aplicación. Recuerde que los recursos están ubicados en la carpeta C:\Chocolatey\lib.

Desinstalación de aplicaciones MSI/MSU

Es posible desinstalar aplicaciones provenientes de instaladores MSI directamente con **`Uninstall-Package`**.

Ejemplo: desinstalar una aplicación cuyo proveedor es MSI

```
PS D:\adobeMSI> Get-Package -ProviderName MSI -Name Adobe*

Name                                   Version          Source
ProviderName
----                                   -------          ------       ------
------
Adobe Acrobat Reader DC - F... 15.7.20033       C:\Progra... msi


PS D:\adobeMSI> Get-Package -ProviderName MSI -Name Adobe* |
Uninstall-Package
ADVERTENCIA: Reboot is required to complete the uninstallation
operation.

Name                                Version          Source
Summary
----                                -------          ------           --
-----
Adobe Acrobat Reader DC - F... 15.7.20033
```

6. Mostrar el conjunto de programas instalados (auditoría)

Es posible obtener la lista de todos los programas instalados en el puesto de trabajo con el cmdlet **`Get-Package`**. El comando no hace sino recuperar los paquetes instalados provenientes de los repositorios, y muestra todo lo que se ha instalado en el equipo, sea cual sea su origen: MSI/MSU (Windows Installer), Programas (archivos ejecutables), paquetes provenientes de un proveedor basado en NuGet o PowerShellGet...

Así, además de que **`Get-Package`** permite visualizar el conjunto de programas instalados para realizar operaciones, también puede utilizarse para auditar los puestos de trabajo de una manera rápida y sencilla. He aquí un resumen de los parámetros de **`Get-Package`**:

Parámetro	Descripción
`-IncludeWindowsInstaller`	Muestra la lista de aplicaciones instaladas (**Programas y características** del Panel de control), incluyendo aquellas que no son visibles.

Parámetro	Descripción
-Name <String[]>	Nombre del paquete o de los paquetes de software que se van a recuperar. Está permitido el uso del carácter comodín * (*wildcard*).
-ProviderName <String[]>	Indica el nombre del proveedor de paquetes cuyos paquetes se recuperarán.

Ejemplo: recuperar todos los paquetes y programas instalados

```
PS C:\Windows\system32> Get-Package

Name                                Version          Source
ProviderName
----                                -------          ------       ------
------
Office 15 Click-to-Run Exte... 15.0.4893.1002               msi
Office 15 Click-to-Run Loca... 15.0.4893.1002               msi
Office 15 Click-to-Run Lice... 15.0.4893.1002               msi
Adobe Refresh Manager          1.8.0            C:\Progra... msi
Adobe Acrobat Reader DC MUI    15.7.20033       C:\Progra... msi
64 Bit HP CIO Components In... 18.2.4                       msi
paint.net                      4.0.13                       msi
paint.net                           4.0.6              C:\Chocol...
Chocolatey
adobereader                            2015.007.2003... C:\Chocol...
Chocolatey
chocolatey-core.extension         1.0.7                 C:\Chocol...
Chocolatey
notepadplusplus                     7.3.1               C:\Chocol...
Chocolatey
notepadplusplus.install            7.3.1                C:\Chocol...
Chocolatey
Notepad++ (64-bit x64)                7.3.1
        Programs
Microsoft Office 365 ProPlu... 15.0.4893.1002
Programs
Microsoft OneDrive                 17.3.6743.1212
Programs
[...]
```

En este ejemplo, **Get-Package** devuelve todos los programas presentes en el puesto de trabajo. La columna ProviderName indica el tipo de instalador o el proveedor de paquetes utilizado.

Ejemplo: ejecutar Get-Package en un equipo remoto

Como el cmdlet **Get-Package** no posee un parámetro **-ComputerName**, es preciso utilizar **Invoke-Command**

especificando el equipo remoto que se utilizará para realizar la consulta.

```
PS C:\Windows\system32> $cred = Get-Credential
PS  C:\Windows\system32>  Invoke-Command  -ComputerName  "LAB-WS01"  -
Credential
$cred -ScriptBlock {Get-Package}


PropertyOfSoftwareIdentity: PropertyOfSoftwareIdentity
PSComputerName            : LAB-WS01
RunspaceId                : 16883f0d-2859-43c4-80f8-84f138ef87ae
FastPackageReference      :
hklm64\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
\Uninstall\
7-Zip
ProviderName              : Programas
Source                    :
Status                    : Installed
[...]
```

Se obtienen todas las propiedades para cada una de las entradas.

Escenarios de uso

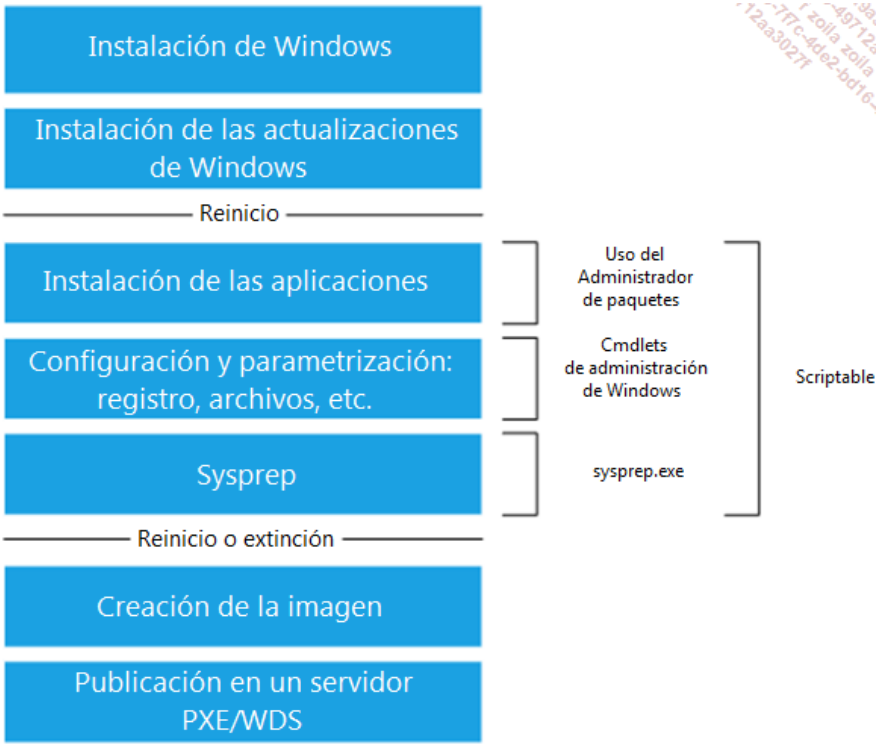
Si bien resulta fácil imaginar la implementación del administrador de paquetes de Windows en un entorno de helpdesk, también puede resultar muy práctico en otro escenario: la creación de imágenes de Windows.

Las imágenes de Windows permiten desplegar fácilmente los sistemas operativos Windows en los puestos de trabajo. Este método de despliegue se realiza generalmente a través de la red, con ayuda de un servidor PXE (*Preboot Execution Environment*) o WDS (*Windows Deployment Services*).

Para optimizar el proceso de preparación de los puestos de trabajo y ahorrar tiempo, las imágenes de Windows contienen, en general, un conjunto de aplicaciones, de configuraciones y de actualizaciones de Windows instaladas previamente antes de la creación de la imagen.

Así, algunas empresas crean una o varias imágenes de Windows en función de sus necesidades o clientes. Sin embargo, la creación de una imagen o su actualización puede resultar molesta. Es aquí donde interviene el administrador de paquetes: toda la parte de descarga de aplicaciones y su instalación puede automatizarse completamente por línea de comandos… ¡y con scripts!

Para hacernos una idea del conjunto de etapas de creación de una imagen de Windows y en qué momento entra en juego el administrador de paquetes, he aquí un esquema:



Proceso global de creación de imágenes de Windows

Introducción

El perímetro de acción de Windows PowerShell no termina en el puesto de trabajo local. En efecto, Windows PowerShell ofrece la posibilidad de administrar de manera remota una o varias máquinas a la vez. Así, el conjunto de cmdlets descritos en los capítulos previos pueden emplearse para administrar el conjunto del parque informático.

Este capítulo describe cómo administrar de manera remota los puestos de trabajo Windows y también las tecnologías empleadas, los requisitos previos, y aporta también una breve descripción del conjunto de cmdlets que permiten la administración remota.

La comunicación remota del Framework .NET

La potencia ofrecida por Windows PowerShell proviene del Framework .NET, que es por otro lado un requisito previo obligatorio para poder instalar el intérprete de comandos. Por lo tanto, este hereda las funcionalidades de ejecución remota del Framework .NET.

Sin duda habrá observado la presencia del parámetro **-ComputerName** en varios cmdlets: permite ejecutar un comando sobre uno o varios puestos de trabajo remotos. Una de las ventajas ofrecidas por la comunicación remota del Framework .NET es que no es necesario que Windows PowerShell esté instalado en los puestos de trabajo remotos, y que se trata de un método sencillo para la administración remota de las máquinas. El único requisito previo es estar conectado a la máquina local con una cuenta que posea permisos de administración en el equipo remoto.

Sin embargo, existe también un inconveniente: la comunicación se basa en el protocolo RPC, que a menudo está bloqueado en los routers de la red y también en los firewalls.

1. Los cmdlets de comunicación remota del Framework .NET

Para identificar los cmdlets que poseen el parámetro **-ComputerName**, basta con introducir la siguiente línea de comando:

```
PS C:\Windows\system32> Get-Command -ParameterName ComputerName
```

Sin embargo, preste atención, pues estos no son en realidad todos los cmdlets que se basan en la comunicación remota del Framework .NET. En efecto, existen cmdlets que se basan en PowerShell Remoting, que veremos más adelante.

La única manera de saber si el parámetro **-ComputerName** utiliza el protocolo RPC es consultar la ayuda en línea del cmdlet que se desea utilizar, con el cmdlet **Get-Help**. He aquí un ejemplo con **Get-WmiObject**:

```
PS C:\Windows\system32> Get-Help Get-WmiObject -Parameter
ComputerName

-ComputerName <String[]>
    Specifies the target computer for the management operation.
    Enter a fully qualified domain name, a NetBIOS name, or an IP
    address. When the remote computer is in a different domain
    than the local computer, the fully qualified domain name is
    required.

    The default is the local computer. To specify the local
    computer, such as in a list of computer names, use
    "localhost", the local computer name, or a dot (.).

    This parameter does not rely on Windows PowerShell remoting,
    which uses WS-Management. You can use the ComputerName
    parameter of Get-WmiObject even if your computer is not
    configured to run WS-Management remote commands.
```

La mención «This parameter does not rely on Windows PowerShell remoting» (en español: «Este parámetro no se basa en la comunicación remota Windows PowerShell») significa que el parámetro **-ComputerName** de este cmdlet se basa efectivamente en los mecanismos de comunicación remota del Framework .NET.

Es importante saber qué tecnología de comunicación se utiliza cuando se hace uso del parámetro **-ComputerName**, pues los requisitos previos para establecer la comunicación con la máquina remota no son los mismos y, por ello, obtendrá comportamientos diferentes entre un cmdlet y otro. He aquí un ejemplo con dos comandos que tienen el mismo fin, pero que no utilizan el mismo protocolo:

```
PS C:\Users\Administrador> Get-Service -Name TermService
-ComputerName PC-01

Status   Name               DisplayName
------   ----               -----------
Running  TermService        Servicios de Escritorio remoto


PS C:\Users\Administrador> Invoke-Command -ComputerName PC-01
-ScriptBlock {Get-Service -Name TermService}
[PC-01] Error de conexión al servidor remoto PC-01. Mensaje de
error: El cliente no puede conectarse con el destino especificado
en la solicitud.
Compruebe que el servicio del destino se esté ejecutando y
que acepte solicitudes. Consulte los registros y la documentación
del servicio WS-Management que se ejecuta en el destino,
normalmente IIS o WinRM.
Si el destino es el servicio WinRM, ejecute el siguiente
comando en el destino para analizar y configurar el servicio
WinRM: "winrm quickconfig".
Para obtener más información, vea el tema de la Ayuda
about_Remote_Troubleshooting.
```

En el ejemplo anterior, el cmdlet **Invoke-Command** no utiliza el mecanismo de comunicación remota del Framework .NET, sino PowerShell Remoting. Dado que este último no está habilitado en el puesto remoto, se obtiene un error.

He aquí una descripción de cada uno de los cmdlets que utilizan la comunicación remota del Framework .NET:

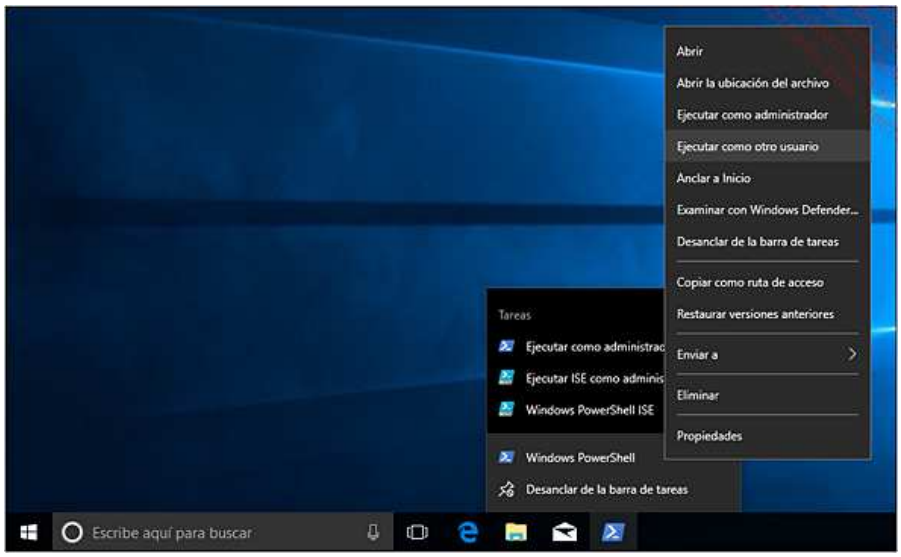
Cmdlet	Descripción
Add-Computer	Agrega el equipo a un dominio o a un grupo de trabajo.
Remove-Computer	Elimina el equipo del dominio.
Rename-Computer	Renombra el equipo (este cmdlet solo renombra un equipo a la vez).
Restart-Computer	Reinicia el sistema operativo.
Stop-Computer	Detiene el equipo.
Clear-EventLog	Elimina todas las entradas de los registros de eventos especificados.
Get-EventLog	Recupera los eventos de un registro de eventos o la lista de registros de eventos.
Limit-EventLog	Limita el tamaño máximo de un registro de eventos y gestiona el tiempo de duración de las entradas.
New-EventLog	Crea un nuevo registro de eventos y un origen de eventos.
Remove-EventLog	Elimina un registro de eventos o anula el registro de un origen de eventos.
Show-EventLog	Abre el Visor de eventos en el equipo local.
Write-EventLog	Escribe un evento en un registro de eventos.
Get-HotFix	Obtiene las revisiones de software que se han aplicado.
Get-Process	Obtiene los procesos que se ejecutan en el equipo.

Cmdlet	Descripción
Get-Service	Obtiene los servicios instalados en el equipo.
Set-Service	Modifica las propiedades de un servicio. Permite también iniciar, detener o interrumpir (suspender) un servicio.
Get-WmiObject	Obtiene las instancias de las clases WMI o información acerca de las clases WMI disponibles.
Remove-WmiObject	Elimina una instancia de una clase WMI existente.
Invoke-WmiMethod	Invoca a métodos WMI.
Register-WmiEvent	Se suscribe a un evento WMI.
Set-WmiInstance	Crea o actualiza una instancia de una clase WMI.
Test-Connection	Envía los paquetes de petición de eco ICMP (equivale al comando **ping** del símbolo del sistema). Este comando no requiere ningún permiso de administración.

2. Algunos ejemplos de envío remoto de comandos

Cuando se usan los cmdlets de comunicación remota del Framework .NET, la cuenta utilizada para la autenticación en la máquina remota es, por defecto, la cuenta del usuario actualmente autenticado en la máquina origen. De modo que debe estar conectado en la máquina local con una cuenta presente en el grupo Administradores del puesto de trabajo remoto. Algunos cmdlets ofrecen la posibilidad de especificar otra cuenta de usuario, pero estos son los menos.

Si desea utilizar otra cuenta para ejecutar un comando, debe abrir una consola de Windows PowerShell utilizando la opción **Ejecutar como otro usuario**. Para acceder a esta opción, haga clic con el botón derecho del ratón en el icono de **Windows PowerShell** pulsando la tecla [Shift].



Abrir una consola de Windows PowerShell como otro usuario

Se le pedirán unas nuevas credenciales. Una vez validadas, se abre la consola de Windows PowerShell, y puede ejecutar comandos destinados a los puestos de trabajo remotos con esta autenticación.

Para verificar con qué cuenta se encuentra autenticado, utilice el comando `whoami`, que devuelve las credenciales de la cuenta:

```
PS C:\WINDOWS\system32> whoami
contoso\julien
```

He aquí algunos ejemplos de envío de comandos a una máquina remota. En el caso de que la consulta falle y obtenga el mensaje de error «El servidor RPC no está disponible», asegúrese de que:

- Está autenticado con una cuenta que posea permisos de administración en la máquina remota.
- No hay ningún firewall bloqueando el puerto de comunicación.

Ejemplo 1: enviar una petición WMI a una máquina remota

El cmdlet **`Get-WmiObject`** que hemos visto en el capítulo Buscar y recopilar información permite enviar peticiones WMI a máquinas remotas.

```
PS C:\Windows\system32> Get-WmiObject -ComputerName PC-01 -Class
Win32_ComputerSystem

Domain              : CONTOSO.LOCAL
Manufacturer        : Microsoft Corporation
```

```
Model              : Virtual Machine
Name               : PC-01
PrimaryOwnerName   : Admin
TotalPhysicalMemory: 2147012608
```

Puede consultar el conjunto de clases WMI y conocer los elementos y la configuración de una máquina remota utilizando este medio.

Ejemplo 2: comprobar el estado de un servicio en una máquina remota e iniciarlo

Para comprobar el estado de un servicio, escriba:

```
PS C:\Windows\system32> Get-Service -Name RemoteRegistry
-ComputerName PC-01

Status   Name               DisplayName
------   ----               -----------
Stopped  RemoteRegistry     Registro remoto
```

El resultado informa de que el servicio Registro remoto está detenido. Para iniciar este servicio, se utiliza el comando **Set-Service**:

```
PS C:\Windows\system32> Set-Service -Name RemoteRegistry
-ComputerName PC-01 -StartupType Manual -Status Running
```

Si intenta iniciar un servicio cuando su tipo de inicio está deshabilitado, entonces Windows PowerShell devuelve un error. Ocurre así con el servicio Registro remoto, que posee esta configuración por defecto: es preciso definir un nuevo tipo de inicio (manual o automático) para poder iniciarlo inmediatamente.

Ejemplo 3: recuperar información de los registros de eventos de una máquina remota

Para recuperar información de los registros de eventos (**Get-EventLog**), previamente es necesario que el servicio Registro remoto presente en la máquina remota pueda iniciar o bien esté iniciado para funcionar. Si no es el caso, se obtiene un error. Ocurre lo mismo para poder recuperar información acerca de los procesos (**Get-Process**).

He aquí un ejemplo de recuperación de una parte del registro de eventos de una máquina remota:

```
PS C:\Users\Administrador> Get-EventLog -LogName System -
ComputerName PC-01
-Newest 3 -EntryType Information
```

Administración remota con Windows PowerShell

Introducida con la versión 2.0 de Windows PowerShell, la tecnología PowerShell Remoting permite administrar de manera remota diversos productos de Microsoft a través de la red.

Basándose en técnicas y protocolos normalizados, PowerShell Remoting responde a las necesidades cada vez más extendidas en la empresa: administrar sistemas operativos (incluyendo los roles y las características) y también servidores (Exchange, etc.) únicamente con Windows PowerShell. Esta es la orientación que toma Microsoft sobre el conjunto de sus productos.

A continuación verá cómo funciona PowerShell Remoting con los puestos de trabajo, cómo utilizarlo, y también los distintos escenarios que pueden presentarse.

1. ¿Qué es PowerShell Remoting?

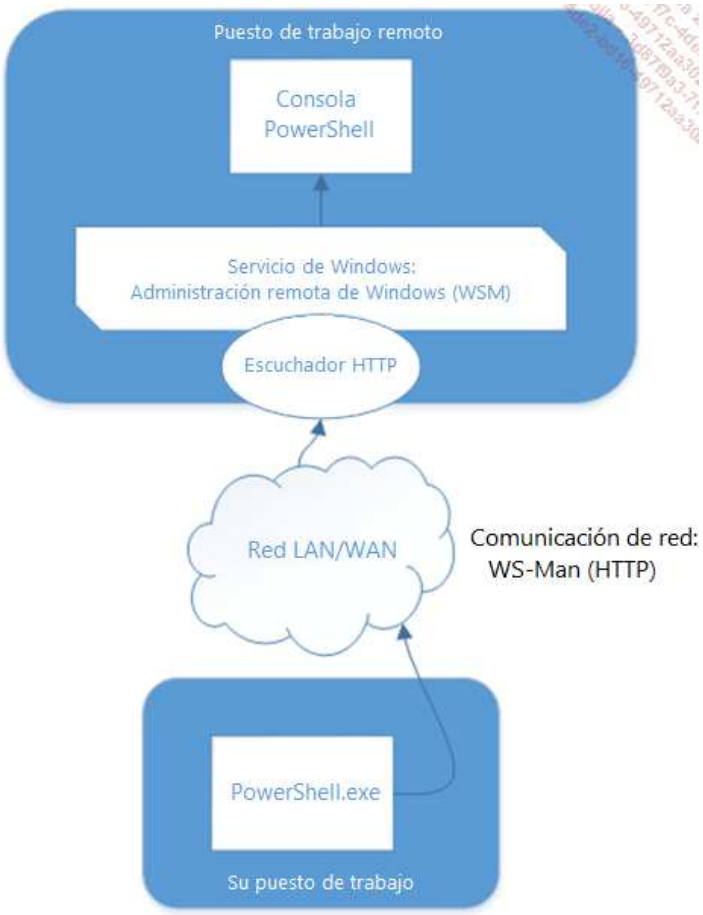
PowerShell Remoting es una funcionalidad que permite la ejecución de líneas de comando y de scripts en una o varias máquinas remotas. Proporciona un entorno genérico que permite la administración remota y la ejecución de cualquier línea de comando. Puede abrir directamente una consola de Windows PowerShell como si estuviera trabajando físicamente en el equipo remoto.

Este mecanismo de comunicación se basa en el protocolo WS-Management (abreviado como WS-Man), que funciona gracias al servicio «Administración remota de Windows (WS-Management)», también llamada WinRM (*Windows Remote Management*). Esta tecnología permite enviar peticiones de gestión a las máquinas por la red a través de los puertos especificados.

Los puestos de trabajo que deben recibir estas peticiones deben estar configurados previamente para escuchar en la red y procesar las peticiones recibidas.

2. Funcionamiento de PowerShell Remoting

Para comprender mejor cuáles son los componentes que se utilizan en una comunicación remota de Windows PowerShell, he aquí un esquema resumen:



Esquema que detalla el funcionamiento de PowerShell Remoting

He aquí una descripción de las distintas etapas del funcionamiento de PowerShell Remoting:

1. Su puesto de trabajo (también llamado cliente) está situado en la parte inferior del esquema. Es aquí donde se encuentra físicamente, y desde esta ubicación se lanzan los comandos de administración remota.

2. La comunicación entre los puestos de trabajo utiliza el protocolo WS-Management. Puede pasar a través de los distintos tipos de redes: LAN, WAN, etc.

3. El puesto de trabajo remoto (también llamado servidor), utiliza el servicio de Windows «Administración remota de Windows (WS-Management)» para implementar uno o varios escuchadores.

Cada uno de estos escuchadores espera las peticiones WS-Man en un puerto, un protocolo (HTTP o HTTPS) y una dirección IP específica que se corresponde con su configuración.

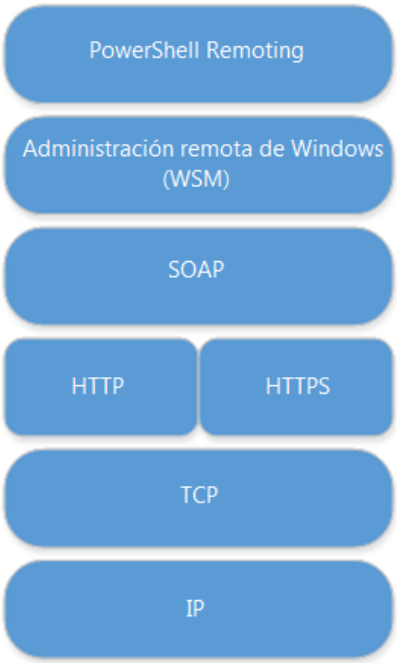
4. El servicio de Windows «Administración remota de Windows (WS-Management)» recibe la información que proviene de los escuchadores y tiene como objetivo organizar las aplicaciones para saber hacia qué aplicaciones distribuirlas.

Muchas otras herramientas pueden invocar a WS-Man y, por tanto, al servicio WinRM. Un puesto de trabajo remoto puede poseer varios escuchadores y otros tantos conectores.

Comprender la relación entre los distintos protocolos

El protocolo de comunicación para la administración de máquinas WS-Management está basado en SOAP (*Simple Object Access Protocol*). Este se basa en HTTP, que tiene la facultad de encapsular toda una variedad de comunicaciones. Cabe destacar que es posible trabajar sobre HTTPS.

Para comprender bien el conjunto de protocolos utilizados y cómo están encapsulados, he aquí un esquema que permite visualizar mejor los distintos protocolos sobre los que se basa PowerShell Remoting:



Capas de protocolos utilizados con PowerShell Remoting

3. Requisitos previos

Para utilizar PowerShell Remoting, los siguientes elementos deben estar instalados en la máquina local y también en el conjunto de máquinas remotas que deben ser administradas:

- Windows PowerShell v2 o una versión superior.
- Framework .NET en versión 2.0 o una versión superior.
- Windows Remote Management 2.0.

Todos estos elementos están instalados nativamente con Windows 7, de modo que no tiene que instalar nada en este sistema operativo, ni en versiones superiores de Windows, que ya poseen estos elementos con versiones más recientes. He aquí, sin embargo, una tabla resumen que agrupa las versiones de estos productos instaladas de manera nativa en función de la versión de Windows:

	Windows PowerShell	**.NET Framework**	**Windows Remote Management**
Windows 7	2.0	3.5	2.0
Windows 8	3.0	4.5	3.0
Windows 8.1	4.0	4.5.1	3.0
Windows 10	5.0	4.6	3.0
Windows 10 (November Update)	5.0	4.6.1	3.0
Windows 10 (Anniversary Update)	5.1	4.6.2	3.0
Windows 10 (Creators Update)	5.1	4.6.2	3.0

4. Activar PowerShell Remoting

Por defecto, los puestos de trabajo Windows no aceptan conexiones ni comandos de PowerShell Remoting. En primer lugar,

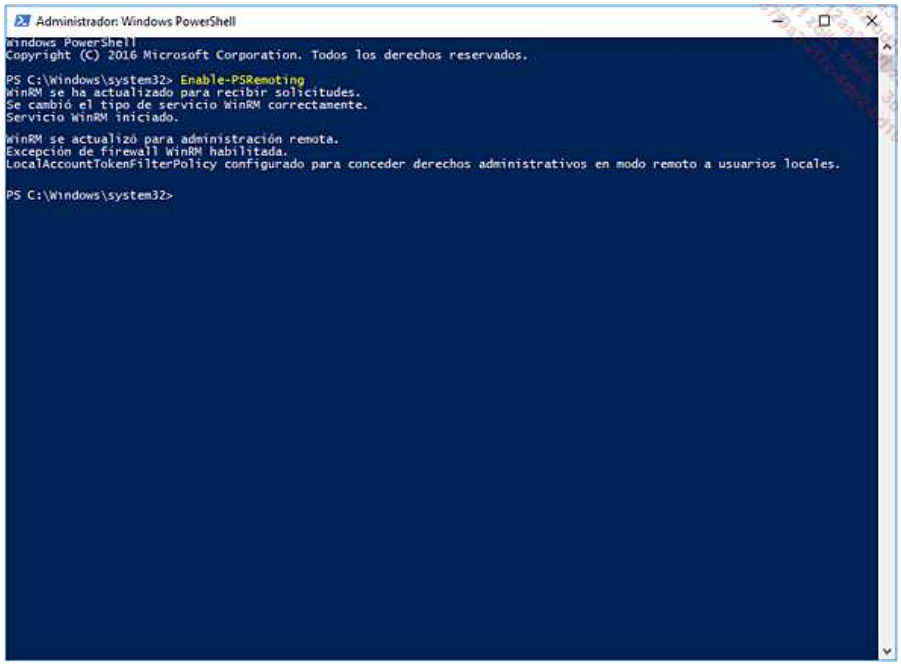
es preciso activarlo en el conjunto de puestos de trabajo que se desea administrar de manera remota.

La activación de PowerShell Remoting consiste en:

- Iniciar el servicio WinRM y configurar el tipo de inicio en «automático».
- Configurar un escuchador HTTP.
- Registrar la configuración de sesión.
- Configurar el Firewall de Windows: agregar una excepción para las comunicaciones WS-Man.

Por supuesto, es posible configurarlo todo manualmente, pero la manera más sencilla y más rápida es utilizar el cmdlet dedicado a la configuración automática de PowerShell Remoting: **`Enable-PSRemoting`**.

Parámetro	Descripción
`-Force`	Elimina todas las peticiones de confirmación.



Activación de PowerShell Remoting en un puesto de trabajo

En función de la versión de Windows PowerShell instalada en el puesto de trabajo, **`Enable-PSRemoting`** puede solicitarle confirmación para cada una de las etapas de configuración

realizadas por el cmdlet. Utilice el parámetro **-Force** para omitir el conjunto de peticiones.

Tan solo queda por comprobar la configuración de PowerShell Remoting intentando crear una sesión en el puesto de trabajo remoto con el cmdlet **New-PSSession** (la información detallada de este cmdlet está disponible en la sección Crear una sesión permanente de manera remota):

```
PS C:\Users\AdminIT> New-PSSession -ComputerName W10

Id  Name         ComputerName      State      ConfigurationName
Availability
--- ----       ------------    -----    -----------------     -------
-----
  1 Session1   W10               Opened    Microsoft.PowerShell
Available
```

Si se establece una sesión, entonces la característica PowerShell Remoting está correctamente configurada en el puesto de trabajo remoto. En caso de que tenga que autenticarse con otra cuenta diferente a la utilizada en su sesión de Windows, previamente hay que crear un objeto `PSCredential` y pasar estas credenciales con ayuda del parámetro **-Credential**:

```
PS C:\Users\AdminIT> $cred = Get-Credential
PS  C:\Users\AdminIT>  New-PSSession  -ComputerName  W10  -Credential
$cred

Id  Name            ComputerName     State       ConfigurationName
 Availability
--- ----        ------------   -----    -----------------    --------
----
12  Session12   W10                Opened    Microsoft.PowerShell
 Available
```

Por defecto, solo los miembros del grupo Administradores de la máquina remota pueden utilizar PowerShell Remoting para establecer una sesión sobre ella.

En los sistemas operativos Windows 8, 8.1 y 10, todos los miembros del grupo Administradores, pero también Usuarios de administración remota, pueden conectarse a la máquina remota.

5. Los métodos de autenticación

Hablar de administración remota es hablar de seguridad. En efecto, no conviene que cualquier usuario pueda ejecutar líneas de comando PowerShell en cualquier puesto de trabajo o servidor de la empresa. De modo que es necesario un método de autenticación para poder utilizar PowerShell Remoting.

Autenticación Kerberos

Kerberos es un protocolo de autenticación desarrollado por el MIT (*Massachusetts Institute of Technology*) a finales de los años 1980. Basado en un sistema de claves privadas y el uso de tickets, presenta la ventaja de que no envía las contraseñas de los usuarios a través de la red.

Microsoft ha implementado Kerberos desde el sistema operativo Windows 2000, y se trata del protocolo por defecto en un dominio Active Directory.

Para poder utilizar PowerShell Remoting con Kerberos, los puestos de trabajo, y también los servidores, deben formar parte de un dominio Active Directory. Así, cuando se realiza una petición de inicio de sesión PowerShell remota, se realiza una autenticación de tipo Kerberos y se comprueba si se poseen permisos en la máquina remota.

Los accesos se autorizan si se poseen los siguientes permisos:

- Ser miembro del grupo Administradores de la máquina remota.
- Ser miembro del grupo Usuarios de administración remota (existente desde Windows 8 y en todas las versiones posteriores). Los miembros de este grupo pueden abrir una sesión remota; sin embargo, tienen un acceso muy limitado: no pueden realizar tareas que requieren permisos de administración.

Otro tipo de autenticación

Pero PowerShell Remoting también puede utilizarse en escenarios muy diferentes, como por ejemplo:

- En puestos de trabajo presentes en varios dominios AD diferentes (sin relación de confianza).
- En puestos de trabajo que pueden estar en grupos de trabajo (*workgroup*) y otros en un dominio AD.

- En puestos de trabajo que están únicamente en grupos de trabajo (*workgroup*).

Para estos escenarios, debe registrar el nombre de la máquina o las máquinas remotas que se han de administrar en el valor de la propiedad **`TrustedHosts`** presente en WSMan:\localhost\Client. También se recomienda encarecidamente utilizar el protocolo HTTPS para llevar a cabo la comunicación con las máquinas remotas: esto implica la configuración del escuchador, y también la compra de un certificado o la creación de un certificado autofirmado que debe desplegarse en el conjunto de máquinas remotas para administrar.

Cada escenario posee sus requisitos previos, que deben cumplirse para permitir la autenticación de una cuenta de usuario que utilice PowerShell Remoting. Windows PowerShell es muy exigente en términos de seguridad (consulte también el capítulo Scripting, sección Las directivas de ejecución), lo cual es del todo normal para una herramienta que controla y automatiza la administración de sistemas operativos Windows.

Administración remota

Es el momento de entrar en materia con PowerShell Remoting: la administración remota. Esta sección explica cómo enviar comandos de administración a los puestos de trabajo remotos y también cómo enviar scripts.

Los cmdlets detallados son los siguientes:

- **`New-PSSession`**: crea una nueva sesión permanente con un equipo remoto.
- **`Get-PSSession`**: recupera la lista de sesiones permanentes abiertas.
- **`Remove-PSSession`**: elimina una sesión permanente.
- **`Enter-PSSession`**: inicia una sesión interactiva con el equipo remoto.
- **`Exit-PSSession`**: sale de la sesión interactiva.
- **`Invoke-Command`**: ejecuta un comando PowerShell en un equipo remoto. También permite enviar un script.

1. Los distintos tipos de sesiones remotas

Existen dos tipos de sesiones que permiten ejecutar comandos Windows PowerShell a través de PowerShell Remoting, y cada tipo posee sus ventajas e inconvenientes:

- La sesión **temporal**: se establece mediante los cmdlets **`Invoke-Command`** y **`Enter-PSSession`**. Una sesión temporal dura el tiempo de ejecución de un comando en el caso de **`Invoke-Command`**, mientras que para **`Enter-PSSession`** esta dura hasta que el administrador cierra la sesión interactiva.
- La sesión **permanente** (creada con **`New-PSSession`**): dura el tiempo de una sesión PowerShell, a menos que el administrador la elimine voluntariamente. Establecer una o varias sesiones permanentes puede resultar útil cuando deben compartirse los datos. Para utilizar estas sesiones permanentes, hay que recurrir al parámetro **`-Session`** de

los cmdlets **Invoke-Command** o **Enter-PSSession** y especificar la sesión deseada.

2. Las sesiones permanentes

Las sesiones permanentes presentan una gran ventaja: permiten compartir los datos entre su equipo y el puesto de trabajo remoto (vea el ejemplo 3 de la sección Ejecución remota de un comando).

a. Crear una sesión permanente de manera remota

Para crear una sesión permanente, se utiliza el cmdlet **New-PSSession**. He aquí los parámetros utilizados con más frecuencia para este cmdlet:

Parámetro	Descripción
-ComputerName <String[]>	Nombre del equipo remoto sobre el que se inicia una sesión Windows PowerShell de manera permanente. Si se indican varios nombres de equipo, **New-PSSession** crea una sesión en cada equipo.
-Credential <PSCredential>	Credenciales del usuario que interactúa con el equipo remoto. Por defecto, se trata del usuario en curso.
-Name <String[]>	Asigna el nombre deseado a esta sesión PSSession.

Ejemplo 1: conexión con la cuenta de usuario en curso

La cuenta utilizada para iniciar una sesión Windows PowerShell en la máquina remota es la cuenta con la que está autenticado actualmente en la máquina cliente, salvo si se ha abierto con anterioridad la consola Windows PowerShell utilizando la opción **Ejecutar como otro usuario**.

```
PS C:\Users\AdminIT> New-PSSession -ComputerName W10

Id Name            ComputerName    State         ConfigurationName
Availability
-- ----            ------------    -----         -----------------
```

```
-------
 2 Session2    W10              Opened    Microsoft.PowerShell
Available
```

Ejemplo 2: conexión con otra cuenta

Es posible iniciar una sesión Windows PowerShell en un puesto de trabajo remoto autenticándose con una cuenta diferente a la utilizada en la sesión de Windows gracias al cmdlet **Get-Credential**.

```
PS C:\Users\AdminIT> $cred = Get-Credential
PS C:\Users\AdminIT> New-PSSession -ComputerName W7 -Credential
$cred

Id Name          ComputerName      State      ConfigurationName
Availability
-- ----        ------------    -----    -----------------     -----
-------
 3 Session2    W7               Opened    Microsoft.PowerShell
Available
```

Cada sesión abierta devuelve un número de identificador (**Id**) y un nombre (**Name**). Estos datos son necesarios a continuación para poder interactuar con los demás cmdlets disponibles.

b. Obtener la lista de sesiones permanentes abiertas

Para conocer las sesiones permanentes en curso, se utiliza el cmdlet **Get-PSSession**. Permite recuperar la información de cada una de las sesiones en caso necesario.

Parámetros	Descripción
-ComputerName <String[]>	Recupera la lista de sesiones abiertas en función del nombre o de los nombres de equipo indicados.
-Id <Int32[]>	Filtra la lista de sesiones abiertas con el número o los números de identificador indicados.
-Name <String[]>	Obtiene la lista de sesiones abiertas en función del nombre o de los nombres de sesión indicados.

Ejemplo: recupera la lista de sesiones permanentes en curso

Sin indicar ningún parámetro, **Get-PSSession** devuelve todas las sesiones abiertas.

```
PS C:\Users\AdminIT> Get-PSSession

Id Name          ComputerName      State      ConfigurationName
Availability
-- ----          ------------      -----      -----------------      -----
-------
 2 Session2      W10               Opened     Microsoft.PowerShell     Avai
lable
 3 Session2      W7                Opened     Microsoft.PowerShell     Avai
lable
```

c. Eliminar una sesión permanente de manera remota

Es posible cerrar una sesión permanente abierta con **New-PSSession** utilizando el comando **Remove-PSSession**. Para ello basta con indicar, utilizando alguno de los parámetros de la siguiente tabla, la sesión que se va a terminar.

Parámetro	Descripción
-ComputerName <String[]>	Nombre del equipo remoto en el que se cerrará la sesión de Windows PowerShell. Si hay abiertas varias sesiones en un mismo equipo, entonces se cerrarán todas.
-Id <Int32[]>	Número de identificador de la sesión. Pueden indicarse varios identificadores (separados por comas).
-Name <String[]>	Nombre común de la sesión.
-Session <PSSession[]>	Indica el objeto sesión PSSession.

Partamos de la siguiente sesión:

```
PS C:\Windows\system32> $session = New-PSSession -ComputerName
W7
PS C:\Windows\system32> $session

Id Name            ComputerName      State      ConfigurationName
 Availability
-- ----            ------------      -----      -----------------      --------
----
 5 Session5        W7                 Opened     Microsoft.PowerShell
 Available
```

Ejemplo 1: elimina la sesión utilizando el parámetro -Session

```
PS C:\Windows\system32> Remove-PSSession -Session $session
```

Ejemplo 2: elimina la sesión utilizando el parámetro -Id

```
PS C:\Windows\system32> Remove-PSSession -Id 5
```

3. Abrir una sesión interactiva de PowerShell remota

El cmdlet **Enter-PSSession** permite abrir una sesión interactiva de Windows PowerShell en un puesto de trabajo remoto. Este cmdlet posee varios parámetros, pero hay tres que son indispensables:

Parámetros	Descripción
-ComputerName <String>	Inicia una sesión interactiva en el equipo remoto especificado. Solo puede indicarse un nombre de equipo.
-Credential <PSCredential>	Indica la cuenta de usuario que interactuará con el equipo remoto. Por defecto, se trata del usuario en curso.
-Session <PSSession>	Indica el objeto de sesión PSSession.

Una vez establecida la sesión interactiva con el equipo remoto, se registra su nombre en la línea de comandos. Este indicador, que aparece en cada línea, nos recuerda que estamos conectados al puesto de trabajo remoto y que todos los comandos de PowerShell ejecutados tendrán efecto en él.

Respecto a la autenticación, si se omite el parámetro **-Credential**, **Enter-PSSession** intenta realizar una autenticación en el puesto de trabajo remoto con la cuenta con la que se ha abierto la consola de Windows PowerShell. También puede autenticarse con otra cuenta: para ello, es obligatorio utilizar el parámetro **-Credential**, al que habrá que pasar un objeto `PSCredential` con las credenciales de otra cuenta de usuario.

Para salir de la sesión interactiva con el equipo remoto, basta con ejecutar el cmdlet **`Exit-PSSession`**, o simplemente **`Exit`**.

Ejemplo 1: iniciar una sesión interactiva de Windows PowerShell remota

```
PS C:\Windows\system32> Enter-PSSession -ComputerName W10
[W10]: PS C:\Users\Julien\Documents>
```

Ejemplo 2: lo mismo, pero utilizando el parámetro -Session

```
PS C:\Windows\system32> $session = New-PSSession -ComputerName
W10
PS C:\Windows\system32> Enter-PSSession -Session $session
[W10]: PS C:\Users\Julien\Documents>
```

Observe también que, cuando la sesión interactiva está operativa, la consola de Windows PowerShell funciona con la versión instalada en el equipo remoto. De modo que algunas funcionalidades o parámetros pueden no estar disponibles.

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

Name                           Value
----                           -----
PSVersion                      5.1.14393.576
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.14393.576
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1


PS C:\Windows\system32> Enter-PSSession -ComputerName W7
[W7]: PS C:\Users\Julien\Documents> $PSVersionTable

Name                           Value
----                           -----
CLRVersion                     2.0.50727.5420
BuildVersion                   6.1.7601.17514
PSVersion                      2.0
WSManStackVersion              2.0
PSCompatibleVersions           {1.0, 2.0}
SerializationVersion           1.1.0.1
PSRemotingProtocolVersion      2.1
```

4. Ejecución remota de un comando

Cuando se desea enviar un único comando a una o varias máquinas remotas, se abre una sesión de PowerShell Remoting temporalmente entre la máquina cliente y la máquina o las máquinas remotas durante la ejecución del comando. Esto resulta

muy práctico en el caso de tener que enviar puntualmente un comando.

Pero en el caso de tener que enviar varios comandos sucesivos, y sobre todo si se utilizan variables, entonces es preferible abrir una sesión permanente, indicando el parámetro **-Session** con el objeto `PSSession` adecuado. No olvide que el cierre de una sesión supone la destrucción de las variables y que por consiguiente se perderán los valores contenidos en ellas.

Antes de ver algunos ejemplos, he aquí los parámetros más útiles de **Invoke-Command**:

Parámetro	Descripción
-ComputerName <String[]>	Especifica uno o varios nombres de equipos en los que se ejecutará el comando.
-Credential <PSCredential>	Especifica una cuenta de usuario que interactuará con el equipo o los equipos remotos. Por defecto, se trata del usuario en curso.
-Session <PSSession[]>	Sesión de Windows PowerShell (PSSession) que se utiliza para la ejecución del comando. Es posible indicar varias sesiones.
-ScriptBlock <ScriptBlock>	Especifica el comando o los comandos que se ejecutan en la máquina o las máquinas remotas. El código debe incluirse entre llaves: { *code* }.

Ejemplo 1: recuperar un valor del registro

```
PS C:\Windows\system32> Invoke-Command -ComputerName W10
-ScriptBlock {(Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\
Windows NT\CurrentVersion').ProductName}
Windows 10 Pro
```

Ejemplo 2: recuperar un servicio en varias máquinas remotas

```
PS C:\Windows\system32> $command = {Get-Service -Name wuauserv}
PS C:\Windows\system32> $result = Invoke-Command -ComputerName
W7,W10
```

```
-ScriptBlock $command
PS C:\Windows\system32> $result

Status     Name                        DisplayName
 PSComputerName
------   ----                -----------                 ------------
--
Running  wuauserv           Windows Update              W7
Running  wuauserv           Windows Update              W10
```

Observe la presencia de la columna `PSComputerName`, que indica el nombre de la máquina a la que pertenece el resultado.

Ejemplo 3: terminar un proceso remoto

En este ejemplo, va a utilizar una variable para manipular un proceso y realizar la manipulación en varias consolas de comandos (tantas como desee). Para guardar esta variable en memoria durante los intercambios, hay que crear una sesión permanente:

```
PS  C:\Windows\system32>  $session  =  New-PSSession  -ComputerName
W10
PS C:\Windows\system32> $session

Id  Name          ComputerName      State      ConfigurationName
Availability
--  ----        ------------    -----    -----------------     ------
------
18  Session18   W10                 Opened    Microsoft.PowerShell
Available
```

Una vez creada la sesión, recupere el conjunto de procesos powershell.exe (si es necesario, abra una consola Windows PowerShell en la máquina remota):

```
PS   C:\Windows\system32>   Invoke-Command   -Session   $session   -
ScriptBlock {$p
= Get-Process -Name powershell}
PS   C:\Windows\system32>   Invoke-Command   -Session   $session   -
ScriptBlock {$p}

Handles  NPM(K)  PM(K)  WS(K)    CPU(s)       Id    SI  ProcessName
PSComputerName
-------  ------  -----  -----   ------     --   -- -----------   --------
------
    439      28 56244 54532    0,52  6624   4 powershell    W10
    515      32 59220 43344    1,48  6968   5 powershell    W10
```

Llegados a este punto, la variable `$p` contiene toda la información de un objeto proceso. Es posible conocer el conjunto de datos que contiene:

```
PS   C:\Windows\system32>   Invoke-Command   -Session   $session   -
ScriptBlock
{$p | fl *}
```

```
Name                          : powershell
Id                            : 6624
PriorityClass                 : Normal
FileVersion                   : 10.0.14393.0 (rs1_release.160715-1616)
HandleCount                   : 439
WorkingSet                    : 55840768
PagedMemorySize               : 57593856
PrivateMemorySize             : 57593856
VirtualMemorySize             : 670781440
TotalProcessorTime            : 00:00:00.5156250
SI                            : 4
Handles                       : 439
VM                            : 2199694036992
WS                            : 55840768
PM                            : 57593856
NPM                           : 28408
Path                          :
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Company                       : Microsoft Corporation
CPU                           : 0,515625
ProductVersion                : 10.0.14393.0
Description                   : Windows PowerShell
Product                       : Sistema operativo Microsoft® Windows®
[...]
```

A continuación, fuerce la parada del proceso y compruebe que ya no existe:

```
PS C:\Windows\system32> Invoke-Command -Session $session
-ScriptBlock {Stop-Process $p -Force}
PS C:\Windows\system32> Invoke-Command -Session $session
-ScriptBlock {Get-Process -Name powershell}
No se encuentra ningún proceso con el nombre "powershell".
Compruebe
el nombre del proceso y ejecute de nuevo el cmdlet.
```

Si no se hubiera creado la sesión permanente, entonces la variable `$p` no habría podido utilizarse desde el segundo comando, pues la sesión temporal abierta con el primer comando se habría cerrado tras su ejecución. La situación es la misma en su puesto local cuando cierra la consola de Windows PowerShell: todas las variables, funciones y demás datos se pierden.

5. Ejecutar scripts de manera remota

¿Qué puede resultar más interesante que ejecutar scripts de manera remota? ¡Y además en varias máquinas con una única línea de comando!

El cmdlet **`Invoke-Command`** permite ejecutar scripts de manera remota. Pero esta vez no se utiliza el parámetro **`-ScriptBlock`**, sino **`-FilePath`**.

Parámetro	Descripción
-ArgumentList <Object[]>	Provee los argumentos al script indicado por el parámetro **-FilePath**. Los argumentos pueden ser variables locales que se reemplazarán por sus valores antes de la ejecución del comando en el equipo remoto.
-ComputerName <String[]>	Especifica uno o varios nombres de equipos en los que se ejecutará el comando.
-Credential <PSCredential>	Especifica una cuenta de usuario que interactuará con el equipo o los equipos remotos. Por defecto, se utiliza el usuario en curso.
-Session <PSSession[]>	Sesión de Windows PowerShell (PSSession) que se utiliza durante la ejecución del comando. Es posible indicar varias sesiones.
-FilePath <String>	Indica la ruta de acceso al script que se ejecutará en uno o varios equipos remotos.

El parámetro **-FilePath** debe indicar la ruta de acceso al archivo de script, que debe encontrarse en el equipo local o en una carpeta compartida a la que tenga acceso el usuario autenticado.

Durante la ejecución de este comando, Windows PowerShell toma el contenido del script, lo transforma en un bloque de script y, a continuación, lo envía al puesto de trabajo remoto que tendrá que ejecutarlo. El script se ejecuta como si fuera de manera local.

Es posible pasar argumentos al script utilizando el parámetro –**ArgumentList** y especificando el valor o los valores que se desean enviar.

Ejemplo: ejecutar un script de PowerShell en varias máquinas remotas

¿Desea conocer la versión de la BIOS de varias máquinas que componen su parque informático para saber cuáles debe actualizar? He aquí el detalle del script que se enviará mediante el cmdlet **Invoke-Command** a varios equipos:

```
# Recupera la información correspondiente al equipo, su placa base
y la versión de la BIOS
[pscustomobject]@{

         "Computer   manufacturer"      =   (Get-WmiObject   -Class
Win32_ComputerSystem).Manufacturer
       "Computer  model"                  =  (Get-WmiObject  -Class
Win32_ComputerSystem).Model
          "Baseboard   manufacturer"   =   (Get-WmiObject   -Class
Win32_Baseboard).Manufacturer
        "Baseboard  product"             =  (Get-WmiObject  -Class
Win32_Baseboard).Product
      "Bios  version"                     =  (Get-WmiObject  -Class
Win32_Bios).SMBIOSBIOSVersion
}
```

La información se recupera a través de peticiones WMI, y se muestra a continuación mediante una tabla de hash (de tipo PSObject). Se recuperan cinco elementos:

- Fabricante del equipo.
- Modelo del equipo.
- Fabricante de la placa base.
- Nombre de la placa base.
- Número de versión de la BIOS.

He aquí el resultado de la ejecución del script con **Invoke-Command** en tres máquinas:

```
PS C:\Windows\system32> Invoke-Command -ComputerName
W7,W10,$env:COMPUTERNAME -FilePath C:\Scripts\Bios.ps1


Computer manufacturer : System manufacturer
Computer model        : System Product Name
Baseboard manufacturer: ASUSTeK Computer INC.
Baseboard product     : P7H55D-M EVO
Bios version          : 1604
PSComputerName        : CONTOSO-DC01
RunspaceId            : b5a299d1-5a55-4a92-a2e6-69906e97deea

Computer manufacturer : Microsoft Corporation
Computer model        : Virtual Machine
Baseboard manufacturer: Microsoft Corporation
```

```
Baseboard product     : Virtual Machine
Bios version          : Hyper-V UEFI Release v1.0
PSComputerName        : W10
RunspaceId            : 534cf97a-c083-4a5c-9e06-bea61654cfd9

Computer manufacturer : Dell Inc.
Computer model        : Optiplex 7010
Baseboard manufacturer: Dell Inc.
Baseboard product     : 0773VG
Bios version          : A01
PSComputerName        : W7
RunspaceId            : b5a299d1-5a55-4a92-a2e6-69906e97deea
```

- Incluso aunque la política de ejecución de scripts sea «Restricted» en los puestos de trabajo remotos, la ejecución del código no se rechaza. En efecto, Windows PowerShell convierte el script en un bloque de script, el cual se recibe en los equipos como tal. De modo que no se considera como un script.

Además de recuperar y mostrar la información correctamente, aparecen dos nuevas propiedades suplementarias: `PSComputerName` y `RunspaceId`.

Como hemos visto antes, `PSComputerName` indica el nombre del equipo al que pertenece el resultado devuelto. `RunspaceId` indica literalmente un «espacio de ejecución», que se corresponde con la sesión de PowerShell Remoting abierta. El identificador de tipo GUID cambia sistemáticamente con cada nueva sesión creada (temporal o permanente). De forma permanente, Windows PowerShell utiliza estas propiedades cuando usa mecanismos de comunicación remota, aunque no se muestran sistemáticamente.

Seguridad

1. ¿Por qué es importante la seguridad?

Con el desarrollo de las redes de área local, y más adelante de Internet, la seguridad en el conjunto de equipos se ha convertido en un aspecto esencial. No basta con atar su equipo de trabajo con un cable de seguridad e implementar algún método de identificación en el inicio de Windows. En la actualidad, los fallos de seguridad provienen principalmente de las redes (programas malintencionados, personas malintencionadas o también incompetentes), de los dispositivos de almacenamiento (llaves USB y discos duros externos, smartphones), etc.

Por este motivo es importante implementar políticas de seguridad en los puestos de trabajo, principalmente en aquellos dedicados a los colaboradores de la empresa. Pero conviene también tener precaución cuando se es administrador: en cualquier momento puede aparecer un error...

2. ¿Cuáles son los riesgos?

Ha visto a lo largo de los distintos capítulos del libro las posibilidades que ofrece Windows PowerShell para la administración de los puestos de trabajo en Windows. Imagine un script PowerShell malintencionado que circule por la empresa: el resultado podría tener graves consecuencias. De modo que hay que estar atento principalmente a los dos aspectos siguientes:

- El origen del script: puede provenir de Internet o haber sido desarrollado por alguna persona malintencionada.
- El método de difusión: e-mails, dispositivos de almacenamiento o las redes son los métodos de difusión más comunes.

Conviene, por tanto, implementar una política de seguridad sobre la ejecución de scripts PowerShell en los puestos de trabajo, y también sobre las interfaces por las que pueden transitar los scripts.

Por supuesto, la elección de la política de seguridad depende en gran medida del contexto de la empresa, de las restricciones de

seguridad que pueden imponer los clientes, etc.

3. Optimizar la seguridad de ejecución de los scripts

Por defecto, Windows PowerShell está configurado de manera segura. Es necesario comprender bien los dos siguientes aspectos que describen la seguridad en Windows PowerShell.

En primer lugar, el programa por defecto asociado a los archivos con extensión .ps1 (script Windows PowerShell) juega un papel importante en la seguridad. En efecto, ¿qué ocurre si se hace doble clic con el ratón sobre este tipo de archivo?

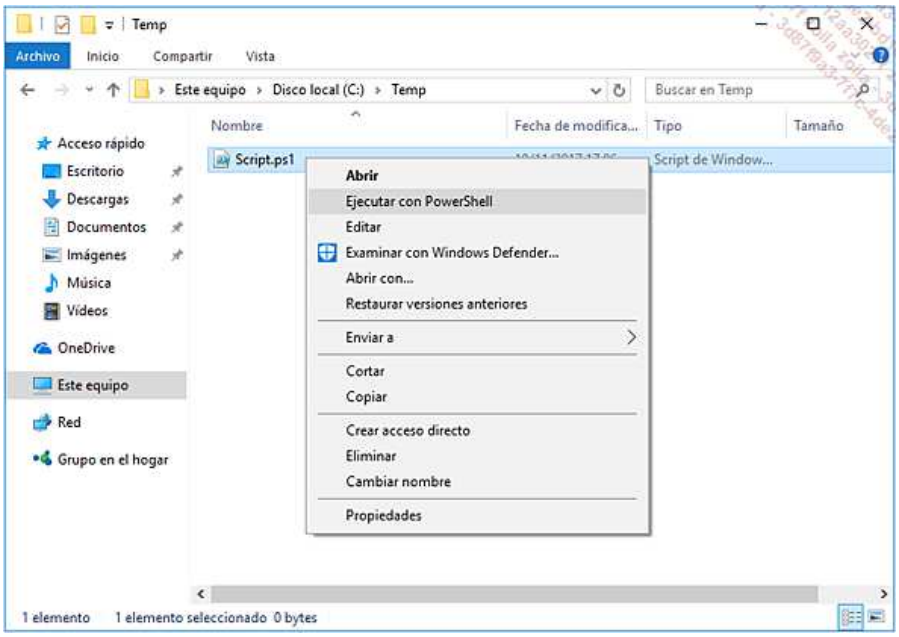
En segundo lugar, la política de ejecución de los scripts en Windows PowerShell permite definir un nivel de seguridad en su ejecución. En función de su origen, ¿qué scripts tendrán autorizada su ejecución?

a. Los archivos de script Windows PowerShell (.ps1)

Los archivos con extensión .ps1 son scripts de Windows PowerShell. Por defecto, Windows asocia este tipo de archivo a la aplicación Bloc de notas (o Notepad), y no a la aplicación (y motor de script) Windows PowerShell. Esto hace que se abra el archivo en modo de edición en lugar de su ejecución (potencialmente peligrosa, sobre todo si se manipula de forma incorrecta).

Esto no ocurre con los archivos VBS, que están asociados directamente al motor de script Windows Script Host cuando se hace doble clic sobre ellos. ¡Esto provoca directamente la ejecución del script!

Las buenas prácticas exigen que, durante la manipulación de scripts, se use el clic con el botón derecho del ratón para escoger la acción deseada en el menú contextual.



Menú contextual de los archivos Windows PowerShell (.ps1)

b. Las directivas de ejecución

Windows PowerShell implementa un concepto de seguridad basado en directivas de ejecución (*execution policy*). Son un total de cinco; cada una de estas directivas de ejecución adopta un comportamiento distinto durante la ejecución de scripts de PowerShell:

- `Restricted`: se trata de la directiva de ejecución por defecto. Está prohibida la ejecución de cualquier script. Windows PowerShell acepta únicamente las instrucciones por línea de comandos. Esta directiva es la más restrictiva de las cinco y protege de toda ejecución involuntaria de un script PowerShell .ps1.

  Así, si intenta ejecutar un script con esta directiva, obtendrá el siguiente mensaje:

```
No se puede cargar el archivo C:\Temp\Script.ps1 porque
en el sistema está deshabilitada la ejecución de scripts.
```

- `AllSigned`: autoriza únicamente los scripts firmados digitalmente. La firma digital debe provenir de una entidad de certificación (CA), y exige poseer los certificados correspondientes.
- `RemoteSigned`: autoriza la ejecución de scripts creados y registrados de manera local, sin estar firmados digitalmente.

Sin embargo, esta directiva de ejecución bloqueará cualquier script proveniente de Internet y que no esté firmado digitalmente.

- `Unrestricted`: autoriza la ejecución de todos los scripts. Se trata de la política menos restrictiva, pero también de la menos segura. Esta directiva de ejecución es la más expuesta al riesgo de ejecución de scripts malintencionados.

  Sin embargo, se muestra un mensaje de seguridad que le advierte de que se está ejecutando un script proveniente de Internet:

```
PS C:\Temp> .\scriptInternet.ps1

Advertencia de seguridad
Ejecute solo los scripts de confianza. Los scripts
procedentes de Internet pueden ser útiles, pero este
script podría dañar su equipo. Si confía en este script,
use el cmdlet Unblock-File para permitir que se ejecute
sin este mensaje de advertencia. ¿Desea ejecutar
C:\Temp\scriptInternet.ps1?
[N] No ejecutar  [S] Ejecutar una vez  [U] Suspender  [?]
Ayuda (el valor predeterminado es "N"):
```

- `Bypass`: similar a `Unrestricted`, pero sin mostrar ningún mensaje de advertencia. Esta directiva de ejecución está destinada en particular a las aplicaciones de terceros que pueden ejecutar Windows PowerShell. Así, la aplicación puede esquivar la configuración de Windows PowerShell y ejecutar los scripts no firmados.

c. Conocer la directiva de ejecución en curso

Para conocer la directiva de ejecución que está activa en un puesto de trabajo, puede utilizar el cmdlet **`Get-ExecutionPolicy`**:

```
PS C:\Windows\system32> Get-ExecutionPolicy
Restricted
```

d. Cambiar de directiva de ejecución

Por defecto, está aplicada la directiva de ejecución Restricted. Pero esta no permite ejecutar scripts, de modo que puede no resultar conveniente en ciertas situaciones.

Es posible cambiar de directiva de ejecución mediante el cmdlet **`Set-ExecutionPolicy`**, seguido del nuevo modo seleccionado:

```
PS C:\Windows\system32> Set-ExecutionPolicy RemoteSigned

Cambio de directiva de ejecución
La directiva de ejecución le ayuda a protegerse
de scripts en los que no confía. Si cambia dicha
directiva podría exponerse a los riesgos de seguridad
descritos en el tema de la Ayuda about_Execution_Policies
en http://go.microsoft.com/fwlink/?LinkID=135170.
¿Desea cambiar la directiva de ejecución?
[S]  Sí     [N]  No     [U]  Suspender     [?]  Ayuda  (el  valor
predeterminado
es "S"):
```

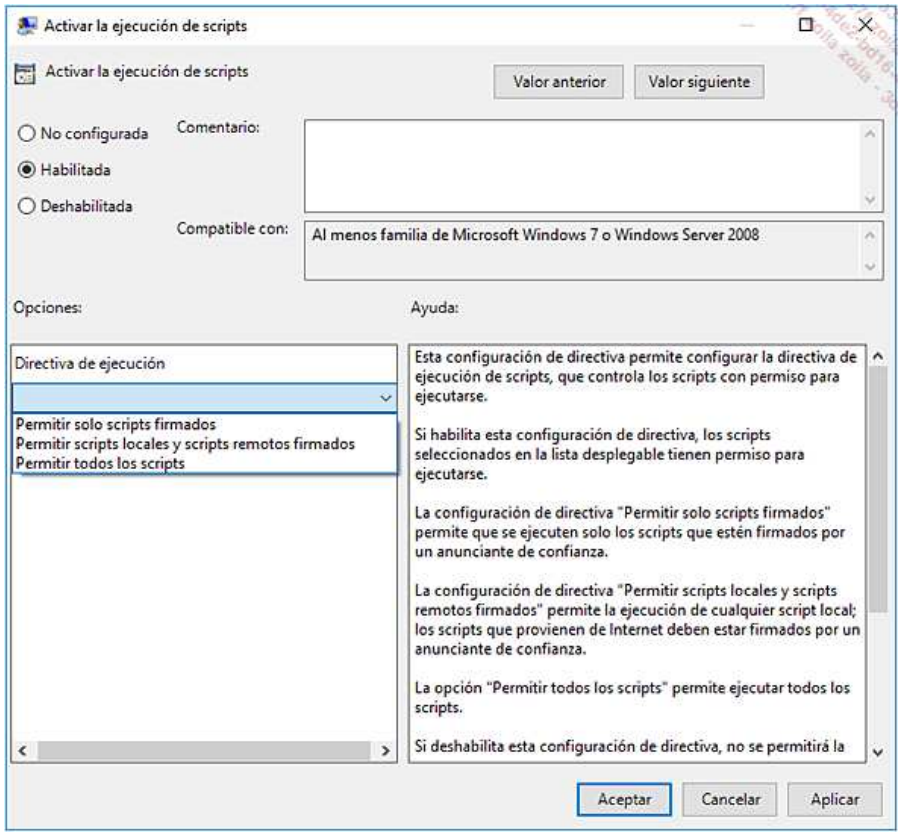
Tras la validación, se activa la nueva directiva de ejecución.

4. Implementar las directivas de ejecución por GPO

Para implementar la misma política de ejecución en el conjunto de puestos de trabajo de una empresa, la mejor forma es la implementación de un GPO (*Group Policy Object*).

En este sentido, vaya a la **Administración de directivas de grupo** de un controlador de dominio. Edite un objeto de directiva de grupo (o cree uno nuevo) y, a continuación, seleccione: **Configuración del equipo - Plantillas administrativas - Componentes de Windows - Windows PowerShell**. Haga doble clic en la directiva **Activar la ejecución de scripts** para modificarla.

Se abre la siguiente ventana:



*Modificación de la directiva **Activar la ejecución de scripts***

Escoja la directiva de ejecución de los scripts Windows PowerShell deseada:

- **Habilitada**
  - **Permitir solo scripts firmados**: corresponde a AllSigned.
  - **Permitir scripts locales y scripts remotos firmados**: corresponde a RemoteSigned.
  - **Permitir todos los scripts**: corresponde a Unrestricted.
- **Deshabilitada**: corresponde a Restricted.

Haga clic en **Aceptar** para validar la modificación de la directiva.

Si no está trabajando en la red de una empresa, siempre tiene la posibilidad de definir esta directiva de manera local, abriendo la consola MMC (mmc.exe). Una vez abierta, haga clic en **Archivo** y, a continuación, en **Agregar/Eliminar complemento...** Agregue el complemento **Editor de objetos de directiva de grupo** (Equipo local) y, a continuación, haga clic en **Aceptar**.

Solo queda por encontrar la directiva **Activar la ejecución de scripts**, como se ha descrito anteriormente, y realizar la modificación deseada.

Windows PowerShell ISE

El editor de scripts Windows PowerShell ISE provee un entorno completo para el desarrollo de scripts de Windows PowerShell. El autocompletado de cmdlets y parámetros o la depuración son funcionalidades que le permitirán ahorrar tiempo en la escritura de scripts.

1. Las buenas prácticas del scripting

Para garantizar un buen seguimiento del desarrollo de un script, es conveniente respetar algunas buenas prácticas.

La legibilidad es un aspecto esencial en la escritura de scripts. Permite una lectura y una comprensión más rápida en caso de que otro administrador deba retomar un script, y también a uno mismo si ha olvidado el contenido del script. Retomar un script al cabo de meses o años es algo habitual, y para facilitar esta tarea no hay nada mejor que respetar los siguientes puntos:

- Incluir comentarios:
  - Comentarios monolínea: se incluye un carácter **#** al comienzo de la línea.
  - Comentario multilínea: para comentar un bloque completo, se abre con los caracteres **<#** y se cierra a continuación con los caracteres **#>**.
- Asignar nombres explicativos a las variables; por ejemplo, **`$tabUsuarios`** para una tabla que represente una lista de usuarios.
- Insertar tabulaciones: cuando se utilizan funciones, bucles, condiciones, etc., es conveniente incluir tabulaciones para marcar bien a qué bloque pertenecen los comandos. He aquí un ejemplo:

```
Function abc ($a, $b, $c)
{
    If ($a -eq $b)
    {
        Write-Host "¡Las 2 variables son idénticas!"
    }
    Else
    {
```

```
            $a = $c
        }
}
```

- Utilizar funciones: estas resultan prácticas para ejecutar tareas repetitivas, y también para estructurar el código y facilitar su comprensión.

Por último, cuando tenga que realizar modificaciones en un script, asegúrese de:

- Realizar de una copia del script antes de su modificación.
- Copiar y pegar la línea o el bloque de script que se va a modificar e incluirlo dentro de un comentario.

Estas recomendaciones permiten la marcha atrás fácilmente en caso de que las modificaciones tengan algún error, o consultar de una manera más sencilla las líneas de comandos que se han alterado. No obvie estos consejos, ¡le ahorrarán mucho tiempo!

2. Paso de parámetros

Cuando se ejecuta un script, es posible pasarle parámetros. Estos parámetros pueden recuperarse como variables y, a continuación, manipularse en el desarrollo del script. Ocurre lo mismo con las funciones.

a. Los parámetros de los scripts

Para pasar parámetros cuando ejecute un script Windows PowerShell, basta con indicar los valores en la misma línea de comando tras el nombre del script; estos valores deben estar separados por un espacio. En el siguiente ejemplo, se pasan dos parámetros:

```
PS C:\Windows\system32> script.ps1 SERVIDOR1 1024
```

El primer parámetro es una cadena de caracteres cuyo valor es `SERVIDOR1`. El segundo parámetro es un entero cuyo valor es `1024`. Para recuperar estos valores en la ejecución del script, se utiliza el comando **`Param`** y se incluyen respectivamente en el orden de los parámetros pasados las variables que van a recuperar estos valores:

```
Param ([string]$nombreServidor, [int]$cifra)
```

Si tenemos el siguiente script test.ps1:

```
Param ([string]$nombreServidor, [int]$cifra)

$cifra = $cifra * 2
Write-Output $nombreServidor
Write-Output $cifra
```

El resultado es el siguiente:

```
PS C:\Temp> .\test.ps1 SERVIDOR1 1024
SERVIDOR1
2048
```

El hecho de implementar esta funcionalidad permite trabajar con un único script con la capacidad de producir diferentes resultados en función de los parámetros y de los valores pasados durante su ejecución.

b. Los parámetros de las funciones

Para pasar parámetros a las funciones, el principio es el mismo que con un script Windows PowerShell. La llamada a la función se realiza con los parámetros, y a continuación la función recupera estos datos con ayuda de las variables que se hayan indicado, siempre respetando el orden.

He aquí, por ejemplo, la siguiente función:

```
Function Visualizar ([string]$mensaje, [string]$color)
{
    Write-Host $mensaje -ForegroundColor $color
}
```

La llamada a la función se realiza de la siguiente manera:

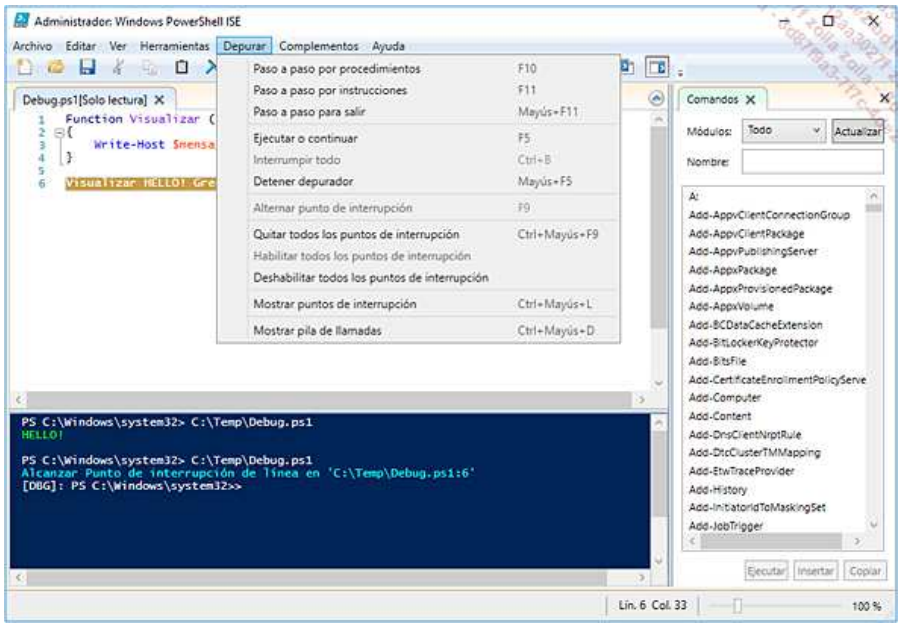
```
Visualizar HELLO! Green
```

El resultado muestra HELLO! en verde por pantalla.

3. Depurar

Windows PowerShell ISE incluye un depurador. Este resulta muy útil para comprender y corregir los problemas o los errores en los scripts. Saber utilizar el depurador es una ventaja considerable en el desarrollo de scripts: localización rápida del problema, ahorro de tiempo para encontrar la solución, etc.

Las herramientas que sirven para llevar a cabo la depuración están presentes en el menú **Depurar** de Windows PowerShell ISE:



Menú Depurar de Windows PowerShell ISE

Este menú contiene los siguientes comandos:

- **Ejecutar o continuar** (tecla [F5]): lanza la ejecución del script desde el principio. En caso de alcanzar un punto de interrupción en la mitad del script, es posible continuar la ejecución del script pulsando de nuevo en la tecla [F5].
- **Alternar punto de interrupción** (tecla [F9]): permite implementar un punto de interrupción en una línea de comando dentro del script. Así, durante la ejecución del script, se desarrollará hasta el siguiente punto de interrupción encontrado. Una vez puesto el script en «pausa», es posible consultar los valores de las variables y, a continuación, proseguir con una ejecución paso a paso.
- **Paso a paso** (teclas [F10] y [F11]): permite ejecutar paso a paso un script (pasando por cada línea de comando).

Cuando se detiene el script en el punto de interrupción, y durante la ejecución paso a paso, tiene la posibilidad de consultar en todo momento los valores contenidos en las variables del script.

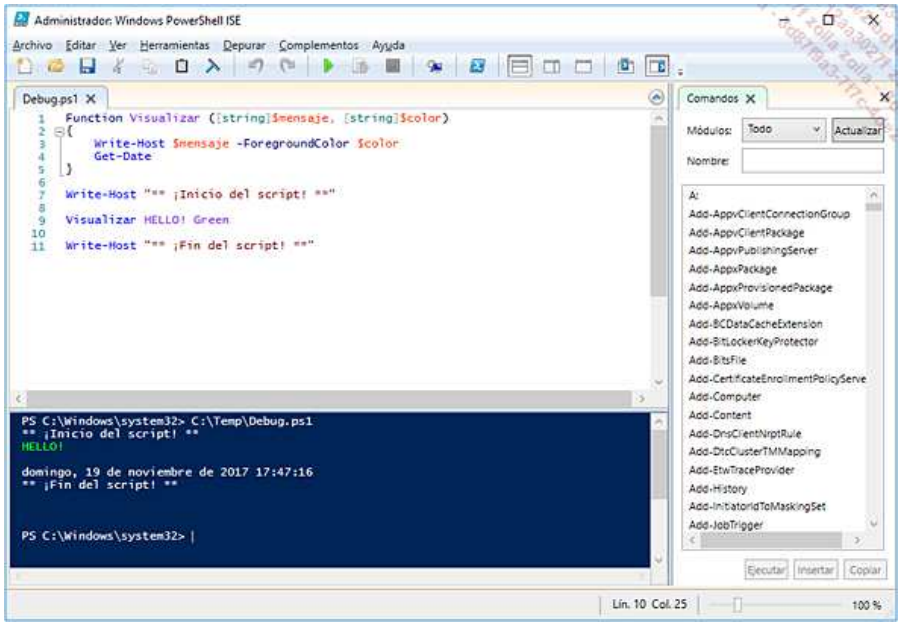
Para poder utilizar las herramientas de depuración, el script sobre el que está trabajando debe estar guardado previamente en su disco duro, pues en caso contrario estas opciones no estarán disponibles. Además, cuando lance la ejecución del script con Windows PowerShell ISE, este entorno se asegurará de que el script está guardado correctamente con sus últimos cambios. Si

estos cambios no se hubieran guardado, entonces el editor de scripts le mostrará un cuadro de diálogo informándole de que se guardará previamente el script que desea ejecutar.

He aquí un ejemplo de uso del depurador con el siguiente script:

```
Function Visualizar ([string]$mensaje, [string]$color)
{
    Write-Host $mensaje -ForegroundColor $color
    Get-Date
}

Write-Host "** ¡Inicio del script! **"

Visualizar "Hello World!" Green

Write-Host "** ¡Fin del script! **"
```

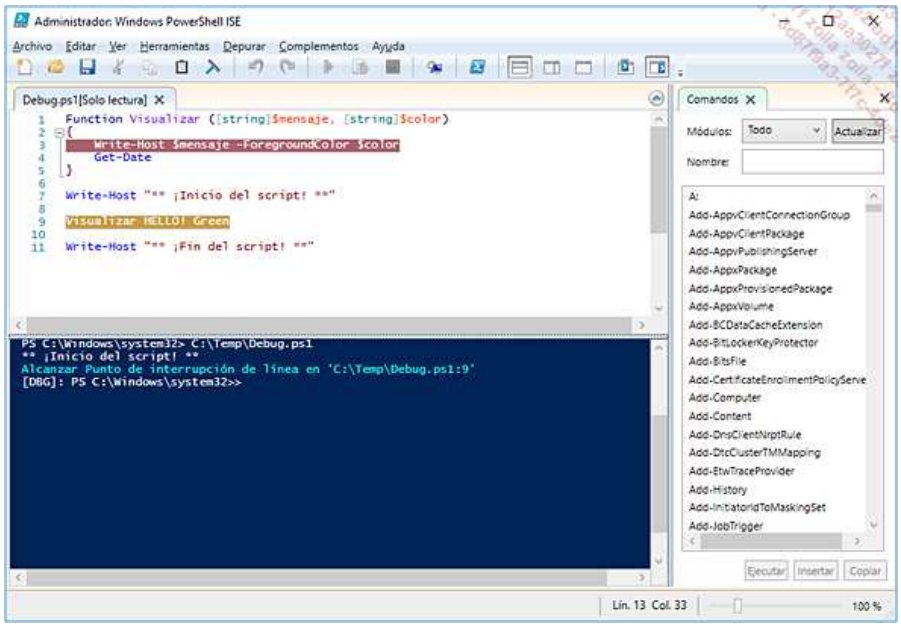
Al pulsar la tecla [F5] se lanza con normalidad la ejecución del script. El resultado es el siguiente:



La consola (abajo a la izquierda) muestra el resultado de la ejecución del script

Para seleccionar un punto de interrupción, haga clic en una línea (no vacía) y pulse la tecla [F9]. La línea con el comando pasa a tener fondo rojo. Puede configurar varios puntos de interrupción si lo desea.

En el siguiente ejemplo, se configuran dos puntos de interrupción en las líneas 3 y 9 y, a continuación, se lanza la ejecución del script:



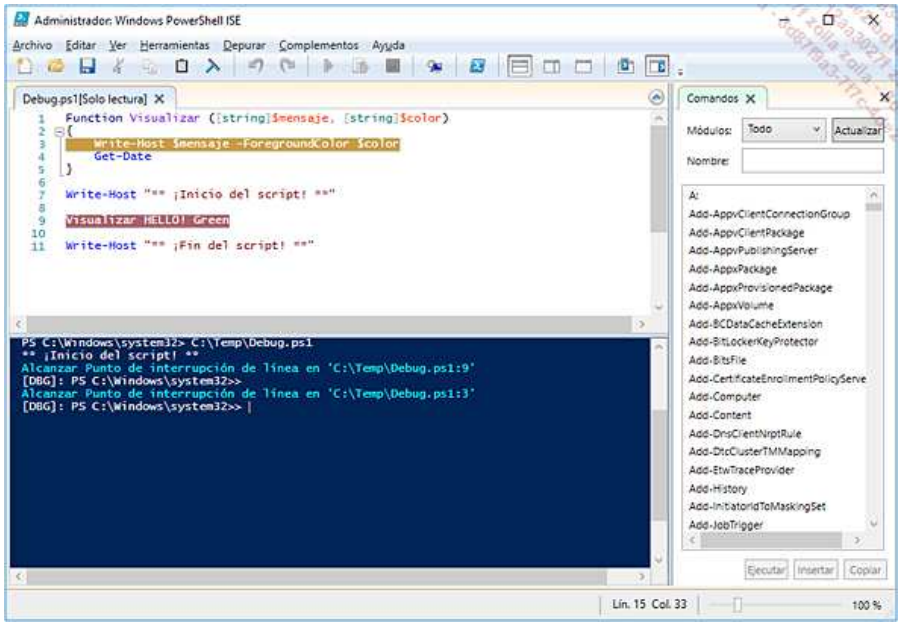
La ejecución del script se encuentra en el primer punto de interrupción

El script se ejecuta de manera normal hasta alcanzar el primer punto de interrupción configurado en la línea 9. Esta pasa a tener fondo amarillo, y la ejecución del script se pone en pausa. Llegados a este punto, es posible:

- **Detener la depuración** (teclas [Mayús]+[F5]).
- **Continuar con la ejecución del script** hasta el siguiente punto de interrupción (tecla [F5]).
- Proseguir paso a paso:
  - El **paso a paso principal** (tecla [F10]) ejecuta la línea de comando actual y, a continuación, se detiene en la siguiente instrucción. Sin embargo, si la línea de comando actual es una llamada a una función o script, el depurador ejecuta la función o el script completos y se detiene en la siguiente instrucción.
  - El **paso a paso detallado** (tecla [F11]) ejecuta el código instrucción a instrucción. Si la línea de comando actual es una llamada a una función o script, el depurador se detendrá en la siguiente instrucción incluida en dicha función o dicho script.

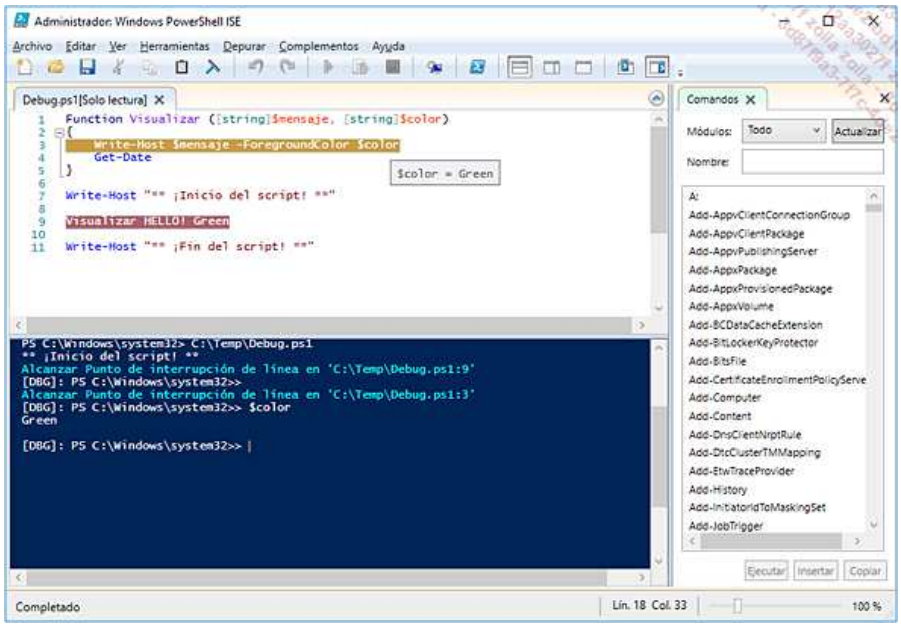
Para seguir con este ejemplo, pulse la tecla [F5] o [F10]. Dado que se ha configurado un punto de interrupción en la función

`Visualizar`, el depurador detiene la ejecución del script en la línea 3:



La ejecución del script está en el segundo punto de interrupción

Otro aspecto importante del depurador es poder conocer el valor contenido en las variables cuando la ejecución del script encuentra un punto de interrupción, y en cada etapa durante la ejecución paso a paso del script. Para ello, basta simplemente con pasar el ratón por encima de la variable (se muestra la información en un pequeño cuadro informativo), o bien introducir la variable en el cuadro de comandos para conocer su contenido.



El contenido de la variable `$color` se muestra en el cuadro informativo

Por otro lado, cuando el depurador se encuentra en un punto de interrupción o en modo de ejecución paso a paso, en cualquier momento puede introducir comandos en el cuadro de comandos para ejecutar instrucciones que no están inicialmente presentes en el script. Esto permite, por ejemplo, probar rápidamente una instrucción en un lugar concreto, sin tener que modificar el script.

Puede proseguir con la ejecución del script hasta el final con las teclas descritas previamente ([F5] o [F10] o [F11]). Sin embargo, es interesante conocer el paso a paso saliente (teclas [Mayús]+ [F11]). Este tipo de paso a paso ejecuta el resto de la función en curso para salir de ella y se detiene automáticamente en el primer comando del nivel superior. Si el depurador se encuentra en el cuerpo principal del script, entonces este se ejecuta hasta el final o hasta el siguiente punto de interrupción.

No descuide el aprendizaje del depurador de Windows PowerShell ISE. Esta herramienta resulta imprescindible para encontrar el origen de los problemas en los scripts.

Ejecutar y desplegar los scripts

El desarrollo de un script PowerShell puede ser largo y tedioso, pero a cambio se verá recompensado ampliamente con el tiempo que se ahorrará con cada ejecución del script. Por regla general, cuando un administrador escribe un script, este tiene el objetivo de ejecutarse varias veces (tarea repetitiva) o bien desplegarse y ejecutarse sobre un número indefinido de puestos de trabajo, o una combinación de ambos.

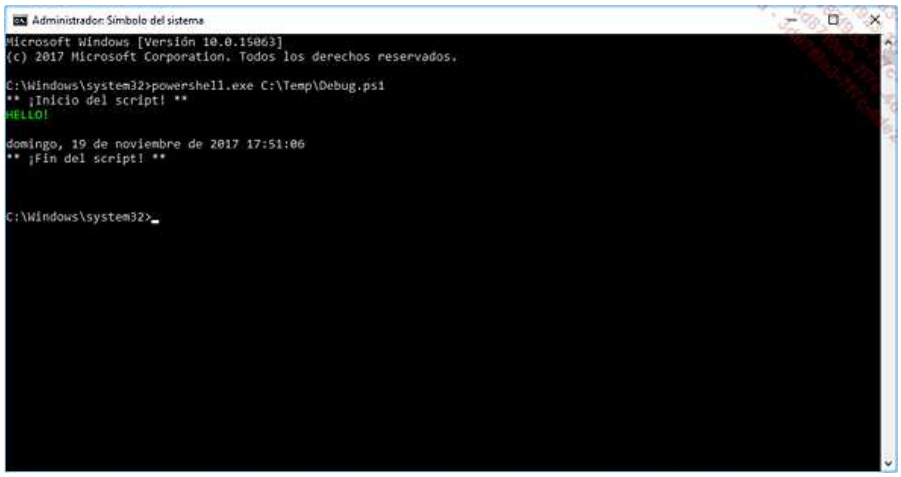
Así, en esta sección veremos cómo ejecutar un script simplemente a través de la consola de Windows PowerShell, y también un método de despliegue de scripts a través de GPO.

1. Por línea de comandos

Es posible ejecutar un script PowerShell por línea de comandos simplemente escribiendo la ruta de acceso absoluta o relativa al archivo del script Windows PowerShell (.ps1):

```
PS C:\Temp> .\debug.ps1
** ¡Inicio del script! **
Hello World!

domingo 19 de octubre de 2014 15:19:03
** ¡Fin del script! **
```

Si debe ejecutar un script desde el símbolo del sistema, debe precisar el motor de script Windows PowerShell, que no es sino powershell.exe, seguido de la ruta absoluta o relativa al archivo del script.



Ejecución de un script PowerShell desde la consola de comandos

En ambos casos, tiene la posibilidad de introducir parámetros indicándolos tras el nombre del script, separados por un espacio. Por ejemplo:

```
PS C:\Temp> .\myScript.ps1 myParameter1 myParameter2
```

2. Por GPO

El despliegue de un script PowerShell por GPO es la manera más práctica de desplegar un script en la empresa. Esto no requiere más que un controlador de dominio (DC) y que los puestos de trabajo estén vinculados al dominio de la empresa.

Así, desde un DC, puede configurar la ejecución de un script PowerShell en el conjunto de puestos de trabajo que haya determinado en función de la ubicación de la directiva de grupo.

La ejecución de un script PowerShell por GPO puede hacerse:

- En el arranque de Windows.
- En la parada de Windows.
- En el inicio de una sesión de usuario.
- En el cierre de una sesión de usuario.

En el caso de que un script PowerShell interactúe con una sesión de usuario (con las claves de registro HKCU o con el perfil del usuario, por ejemplo), conviene lanzar el script durante el inicio o el cierre de una sesión de usuario. Pues durante el arranque o la parada de Windows no hay abierta ninguna sesión y, por consiguiente, no hay accesible ningún dato relativo al usuario.

Por último, un aspecto importante es que los puestos de trabajo que reciben la orden de ejecutar los scripts PowerShell desplegados por GPO no tienen por qué tener necesariamente una directiva de ejecución particular. Así, incluso aunque los puestos de trabajo tengan configurada una directiva de ejecución **Restricted**, no se bloqueará la ejecución de los scripts.

a. Requisitos previos

Antes de desplegar un script PowerShell, conviene probarlo de manera local en un puesto de trabajo o en un entorno de certificación y comprobar su correcto funcionamiento. El despliegue de un script no certificado en varios miles de equipos puede tener consecuencias desastrosas, de modo que no dude en probar el script en varias máquinas y en varias ocasiones. No escatime en los medios y el tiempo necesarios para llevar a cabo esta fase de certificación; se trata de un factor clave en el éxito de su tarea administrativa.

Para probar un script de manera local, consulte la sección Por línea de comandos de este capítulo, que explica cómo lanzar un script PowerShell por línea de comandos. También puede utilizar el complemento **Editor de objetos de directiva de grupo local** (gpedit.msc) para implementar un script como si se hubiera desplegado a través de GPO.

A su vez, es importante saber para qué sirve el script. ¿Debe ejecutarse en todos los puestos de trabajo de la empresa? ¿Únicamente en ciertos modelos de equipos? ¿O bien sobre una versión específica de Windows?

En función de las respuestas, habrá que vincular la directiva de grupo a la ubicación correspondiente en Active Directory y asociarle un filtro si fuera necesario. No olvide que el directorio AD contiene el conjunto de puestos de trabajo vinculados al dominio, y no solo estos: los servidores también están presentes.

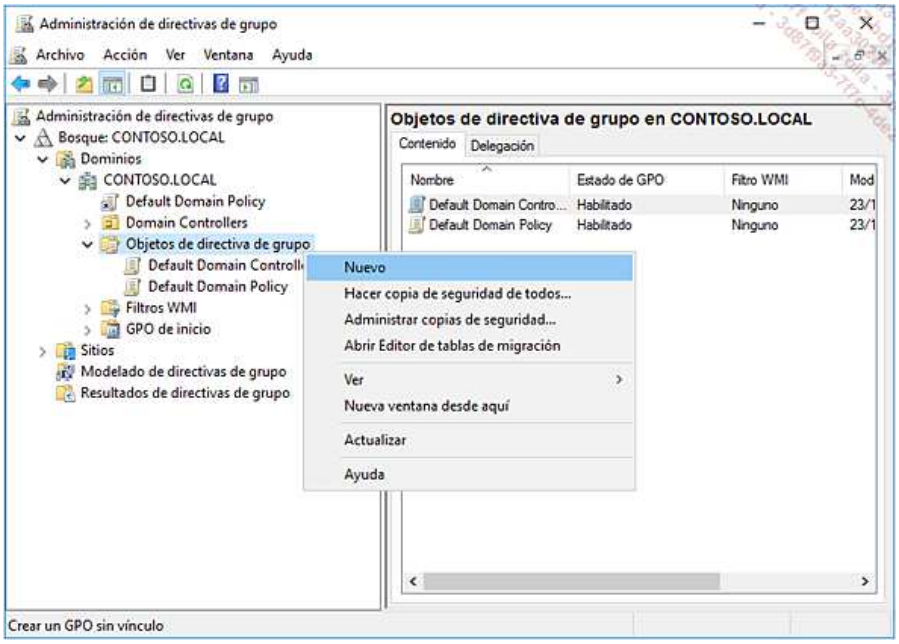
b. Configurar la directiva de grupo

He aquí las etapas para implementar el despliegue y la ejecución de un script PowerShell por GPO. Estas etapas incluyen la creación de un nuevo objeto de directiva de grupo, aunque también es posible utilizar una directiva de grupo existente y vincularle el script que se ha de desplegar.

Crear una nueva directiva de grupo

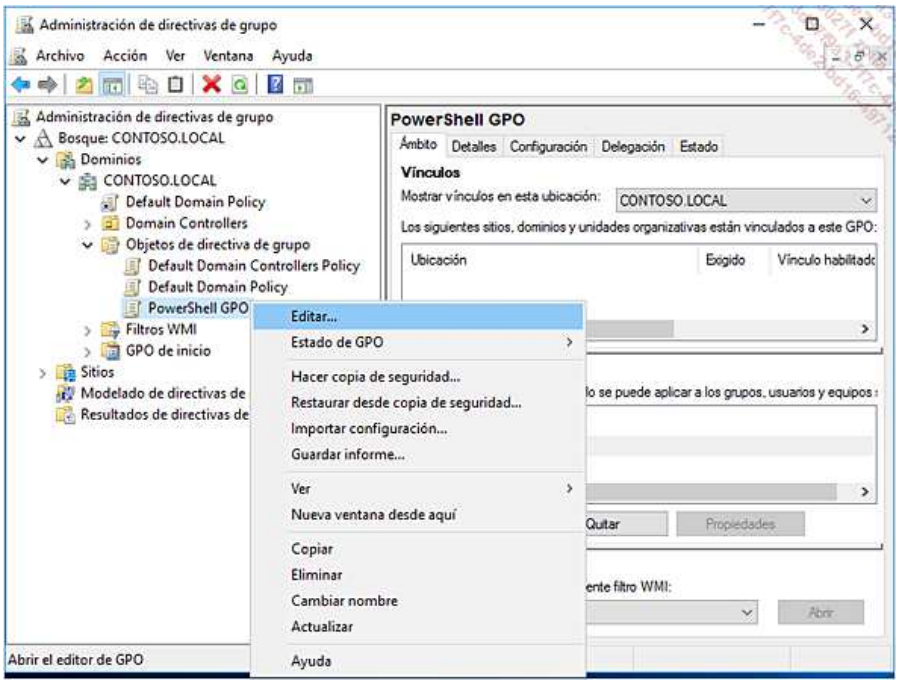
En primer lugar, abra la herramienta **Administración de directivas de grupo** accesible en un servidor DC. Esta herramienta permite administrar y desplegar las directivas de grupo definidas.

Para crear una nueva directiva de grupo, haga clic con el botón derecho del ratón en **Objetos de directiva de grupo** y seleccione **Nuevo**. Asígnele un nombre y, a continuación, haga clic en **Aceptar**.



Crear un nuevo objeto de directiva de grupo

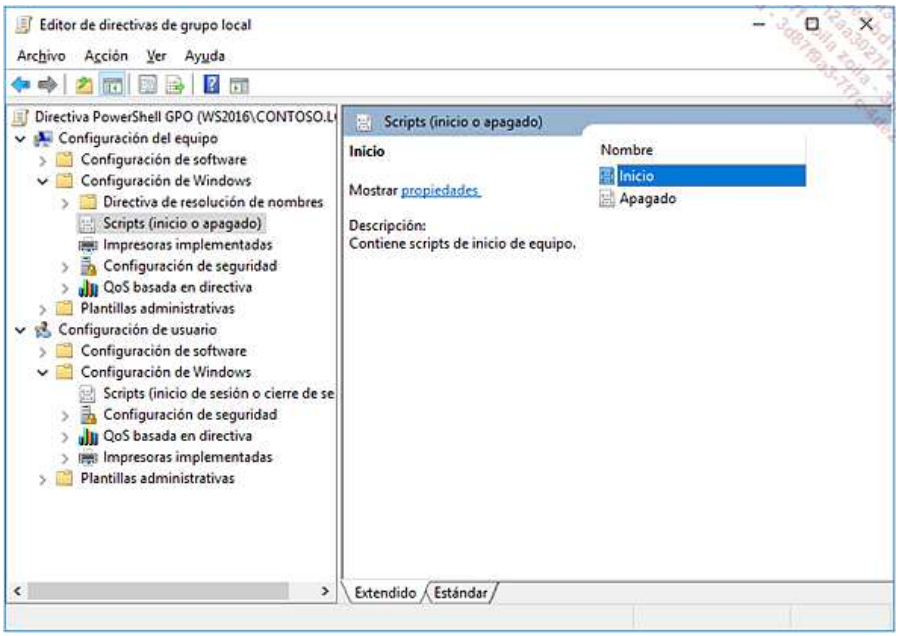
Para editar la directiva de grupo, haga clic con el botón derecho en ella y seleccione **Editar...**.



Editar un objeto de directiva de grupo

Desplegar un script durante el arranque o la parada del equipo

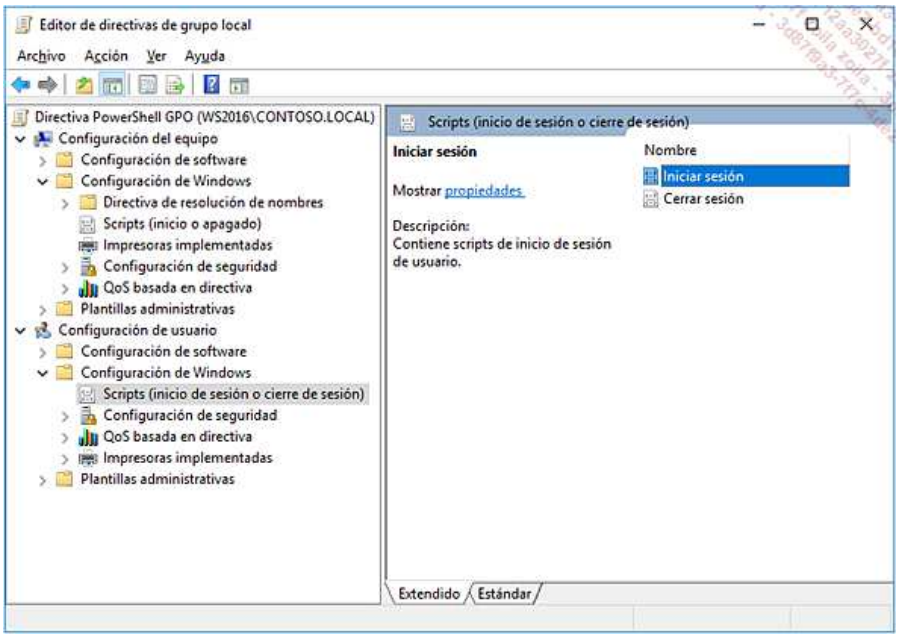
Para configurar la ejecución de un script PowerShell durante el arranque o la parada de Windows, hay que ir a: **Configuración del equipo - Configuración de Windows - Scripts (inicio o apagado)**.



Ejecutar scripts durante el arranque o la parada del sistema

Desplegar un script durante el inicio o el cierre de una sesión de usuario

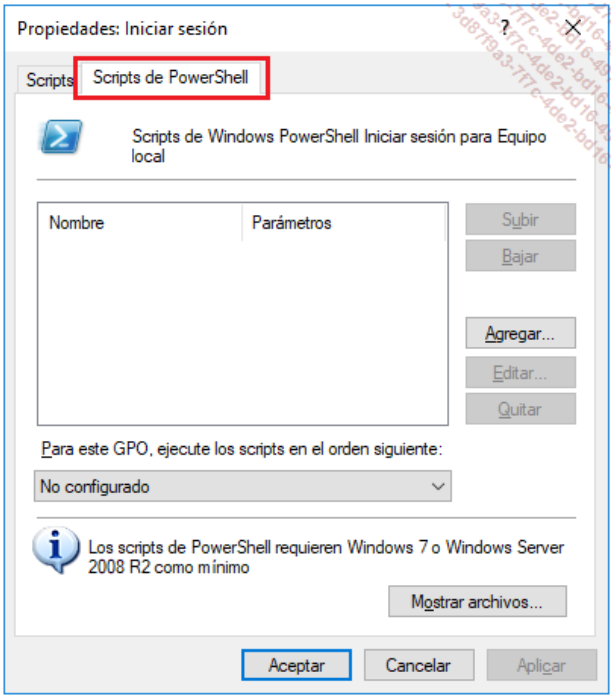
Para configurar la ejecución de un script PowerShell durante el inicio o el cierre de una sesión, hay que ir a: **Configuración de usuario - Configuración de Windows - Scripts (inicio de sesión o cierre de sesión)**.



Ejecutar scripts durante el inicio o el cierre de una sesión de usuario

Agregar el script en la directiva de grupo

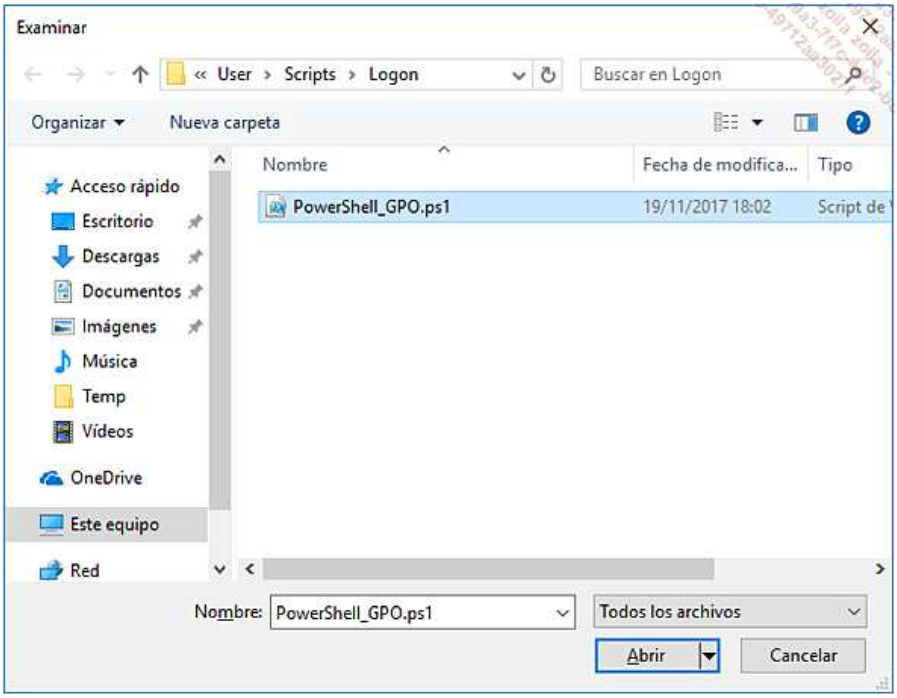
Para agregar el script que hay que ejecutar independientemente del tipo de evento deseado, haga doble clic para abrir la ventana **Propiedades** y, a continuación, haga clic en la pestaña **Scripts de PowerShell**.



Ventana de configuración de los scripts de PowerShell

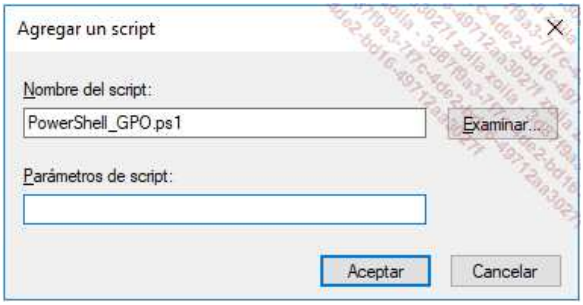
Haga clic en el botón **Agregar...** y, a continuación, en el botón **Examinar...**. Se abre la ventana **Examinar** directamente sobre una carpeta específica: SysVol (*System Volume*). Esta carpeta compartida se replica automáticamente en el conjunto de controladores del dominio y contiene los archivos públicos.

Copie o mueva el script PowerShell en esta carpeta, pues en caso contrario el script no estará accesible para los puestos de trabajo, ni para los usuarios miembros del dominio. Una vez ubicado el archivo en esta carpeta, selecciónelo y haga clic en **Abrir**.



Copiar el script en la carpeta SysVol

Si su script debe ejecutarse con parámetros, hay que introducirlos en la zona de texto **Parámetros de script**. Para validar el script de PowerShell, haga clic en el botón **Aceptar**.

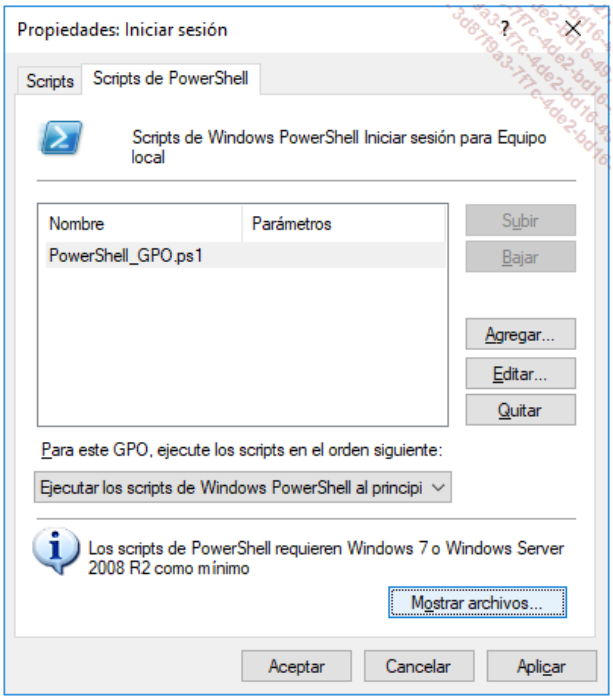


Agregar un script de PowerShell en la directiva de grupo

La ventana **Propiedades** muestra a continuación el script que se acaba de agregar. Puede agregar otros, modificarlos y también eliminarlos.

El botón **Mostrar archivos...** permite abrir la carpeta correspondiente a la directiva de grupo en SysVol. De este modo, tiene la posibilidad de agregar otros scripts o actualizarlos.

El orden de ejecución de los scripts también puede ser importante. Por defecto, los scripts que no son de Windows PowerShell se ejecutan antes que los scripts de Windows PowerShell. Para que los scripts de Windows PowerShell se ejecuten primero, seleccione **Ejecutar los scripts de Windows PowerShell al principio**. Este parámetro se configura para cada uno de los cuatro modos de ejecución: arranque, parada, inicio de sesión, cierre de sesión.



Ventana de configuración de los scripts de PowerShell

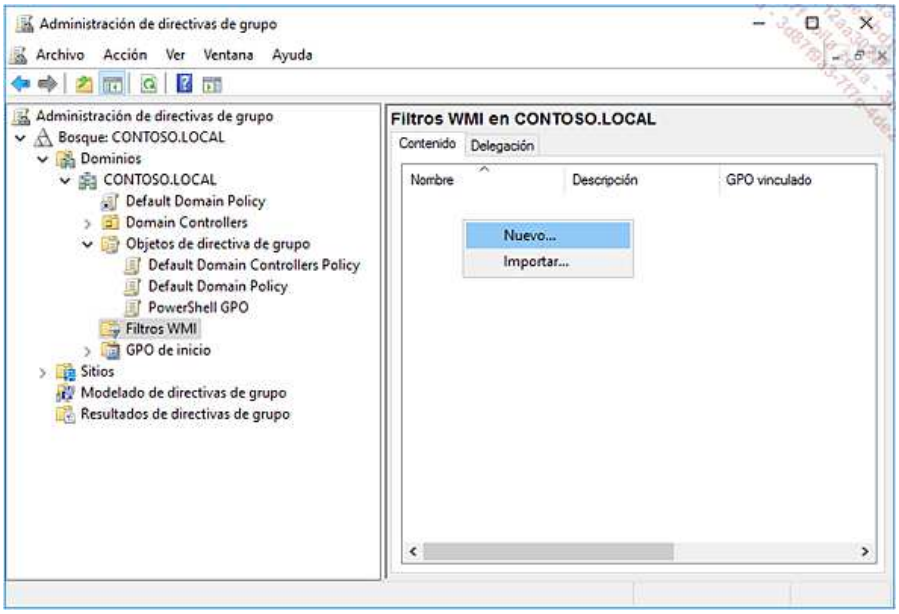
Haga clic en **Aceptar** para confirmar y, a continuación, cierre el Editor de directiva de grupo local.

c. Implementar un filtro WMI

La implementación de un filtro sobre una directiva de grupo permite filtrar su ejecución en función de alguna condición (en forma de consulta WMI) que se evalúa en el equipo que recibe la directiva de grupo. Si se cumple la condición, la directiva de grupo se aplica al puesto de trabajo; en caso contrario, no se ejecuta. Por defecto, si no se configura ningún filtro, la directiva de grupo se aplica a todos los objetos (puestos de trabajo, servidores, etc.) presentes en el contenedor donde está vinculada la directiva de grupo.

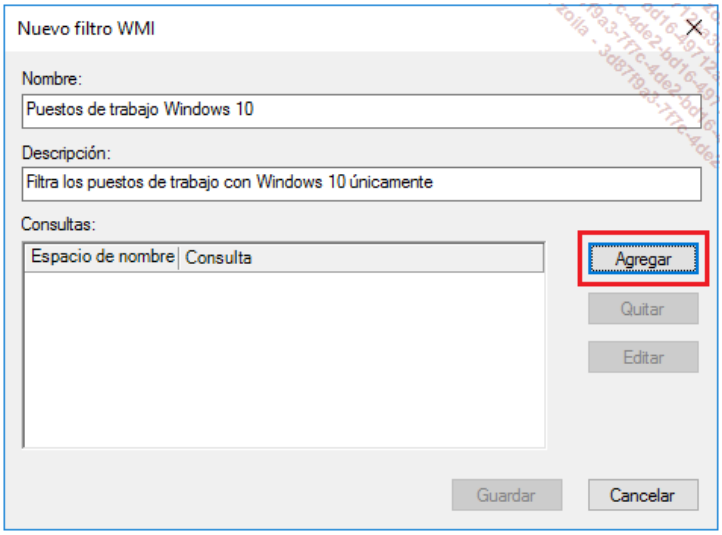
Esto es válido también a nivel de las versiones de Windows: si el script se ha homologado para desplegarse y ejecutarse en puestos Windows 10, no funcionará en Windows 7, sobre todo si se utilizan en el script cmdlets que no existen en esta versión de Windows. Por lo tanto, ¡conviene prestar atención e implementar el filtro correcto!

Para crear un filtro, seleccione **Filtros WMI** en el árbol de la **Administración de directivas de grupo** y a continuación, en la parte derecha, haga clic con el botón derecho y seleccione **Nuevo...**.



Crear un nuevo filtro WMI

Asigne un nombre a este nuevo filtro e introduzca una descripción. A continuación hay que agregar una o varias consultas WMI que servirán de filtro. Para ello, haga clic en el botón **Agregar**.



Ventana de configuración del nuevo filtro WMI

Las consultas deben tener el formato WQL (*WMI Query Language*), que es un lenguaje específico para las consultas WMI similar al lenguaje SQL. He aquí varios ejemplos de consultas WMI, que le permitirán hacerse una idea, y a partir de las cuales podrá construir sus propias consultas. También puede consultar la sección Obtener información gracias a las clases WMI del capítulo Buscar y recopilar información, que le ayudará a encontrar algunas propiedades para realizar comprobaciones.

Esta consulta selecciona únicamente los puestos de trabajo Windows (`ProductType = "1"`). De este modo, se excluyen los servidores:

```
Select * from Win32_OperatingSystem where ProductType = "1"
```

El siguiente filtro selecciona únicamente los puestos de trabajo con Windows 7 instalado (`Version like "6.1%"`). La comprobación sobre la propiedad `ProductType` es obligatoria, pues en caso contrario se incluirían también las máquinas con Windows Server 2008 R2:

```
Select * from Win32_OperatingSystem where Version like "6.1%"
and ProductType = "1"
```

Igual que con la consulta anterior, pero esta vez incluyendo las versiones 8 y 8.1 de Windows:

```
Select * from Win32_OperatingSystem where (Version like "6.1%"
or
Version like "6.2%" or Version like "6.3%") and ProductType = "1"
```

El número de versión de Windows 10 es 10.0.xxxxx, de modo que la consulta para filtrar únicamente los puestos de trabajo con esta versión es la siguiente:

```
Select * from Win32_OperatingSystem where (Version like "10.0%")

and ProductType = "1"
```

Esta consulta filtra únicamente los equipos de la marca DELL:

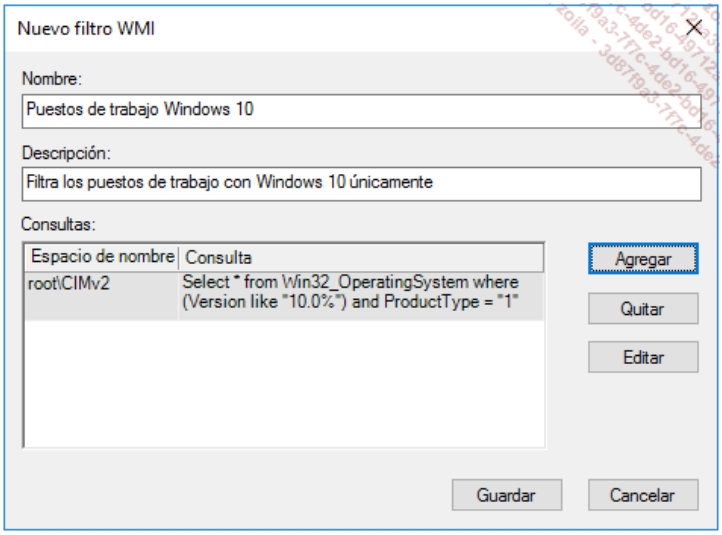
```
Select * from Win32_ComputerSystem where Manufacturer = "DELL Inc."
```

Esta consulta filtra únicamente los modelos de puestos de trabajo Optiplex 7010:

```
Select * from Win32_ComputerSystem where Model = "Optiplex 7010"
```

Tiene la posibilidad de incluir distintas consultas en un único filtro. En este caso, si se cumplen todas las condiciones, entonces el filtro deja que se aplique la directiva de grupo en los puestos de trabajo.

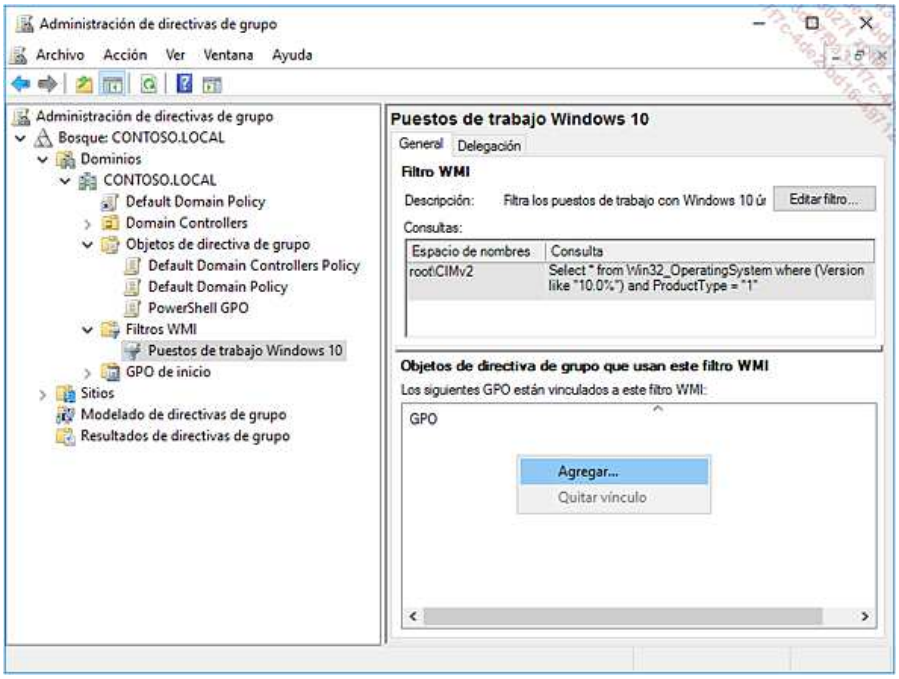
Para terminar con la creación del filtro, haga clic en **Guardar**.



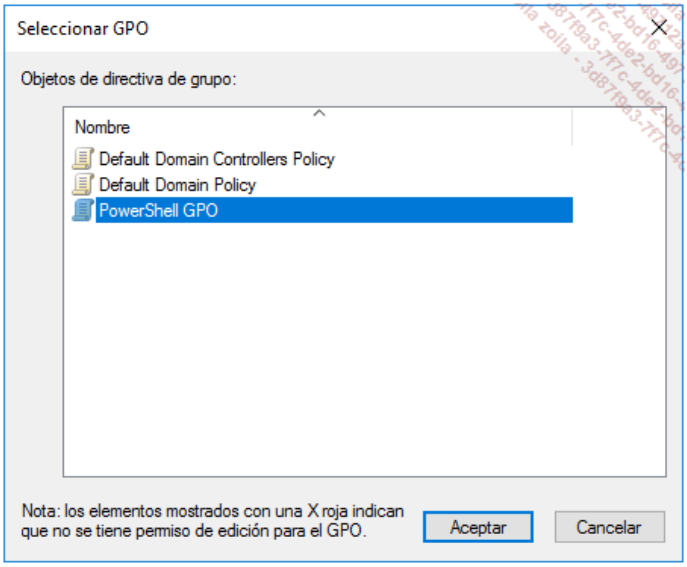
Ventana de configuración del nuevo filtro WMI completo

Una vez creado el filtro, hay que asociarlo a la directiva de grupo que va a desplegar el script de Windows PowerShell. Para ello, existen dos posibilidades:

La primera, desde el filtro WMI: desde el filtro WMI creado previamente, haga clic con el botón derecho en la lista **Objetos de directiva de grupo que usan este filtro WMI** y haga clic en **Agregar...**. A continuación, basta con seleccionar el filtro correspondiente y finalizar la operación haciendo clic en **Aceptar**.

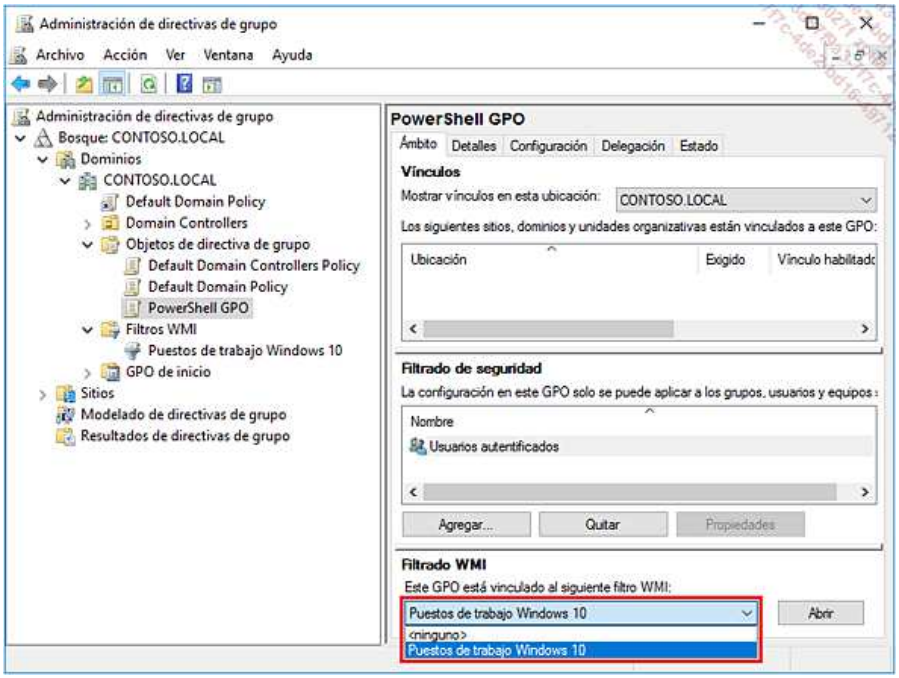


Agregar un filtro a un objeto GPO



Ventana de selección de los objetos GPO

La segunda posibilidad es a nivel del objeto GPO: vuelva a la directiva de grupo y seleccione el filtro correspondiente en la sección **Filtrado WMI**.

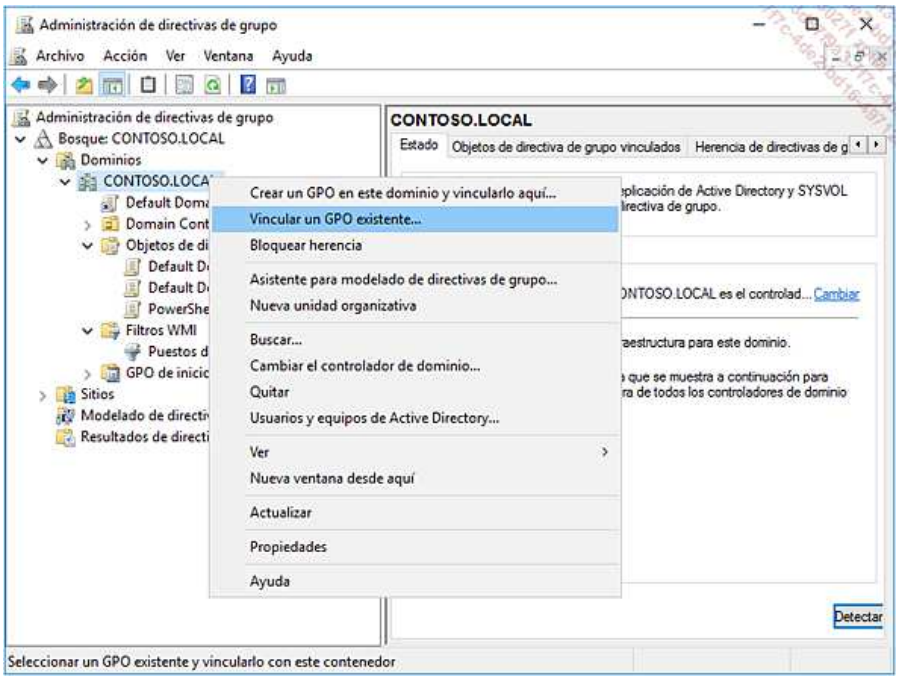


Selección del filtro WMI que se va a asociar al objeto GPO

d. Vincular la directiva de grupo a un objeto

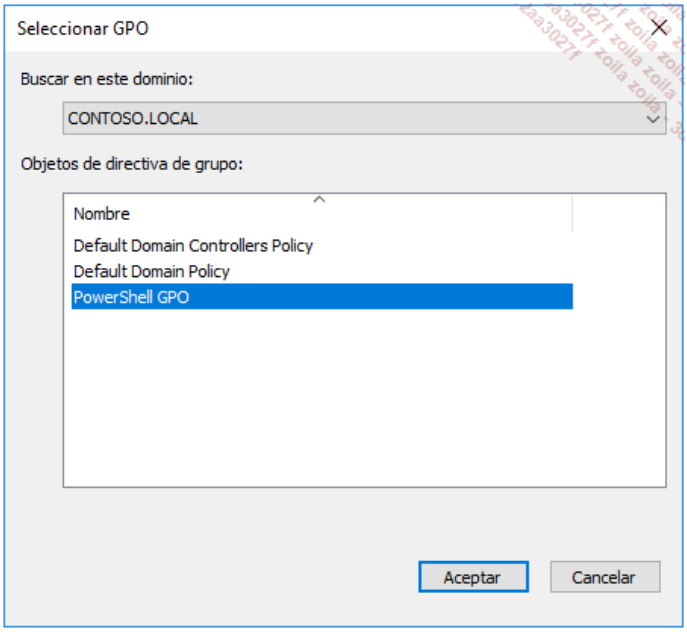
Por último, la última etapa para hacer que la nueva directiva de grupo sea efectiva es vincularla a uno o varios contenedores de Active Directory (dominio, sitios o unidades organizativas). Esto depende, por supuesto, de la arquitectura de su AD.

Para vincular la nueva directiva de grupo, seleccione la ubicación donde desea vincularla, haga clic con el botón derecho y seleccione **Vincular un GPO existente...**.



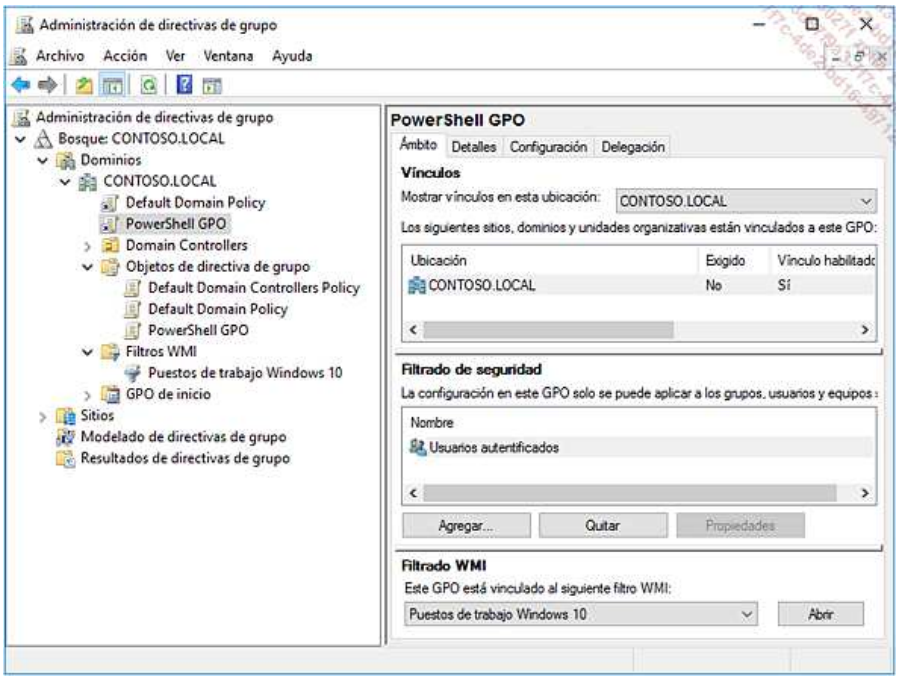
Vincular un objeto de directiva de grupo existente

A continuación, en la ventana de selección, haga clic en la directiva de grupo que desea vincular. Finalice la operación haciendo clic en **Aceptar**.



Selección de un objeto GPO para vincular

El resultado final es el siguiente:



El objeto GPO «PowerShell GPO» está vinculado al dominio CONTOSO.LOCAL

Ahora, la directiva de grupo está operativa. En función de la arquitectura de su Active Directory, se requerirá cierto tiempo de replicación para que se tengan en cuenta las últimas modificaciones realizadas.

Para forzar a los puestos de trabajo a recuperar la última versión de los objetos de directiva de grupo, puede ejecutar el comando **`gpupdate /force`** en los puestos cliente Windows. Como respuesta, este comando le indicará si se han actualizado correctamente las directivas.

```
PS C:\Windows\system32> gpupdate /force
Actualizando directiva...

La actualización de la directiva de equipo se completó correctamente.
Se completó correctamente la actualización de directiva de usuario.
```

Si, por el contrario, se obtiene un error, hay que investigar la causa y aplicar las correcciones necesarias.

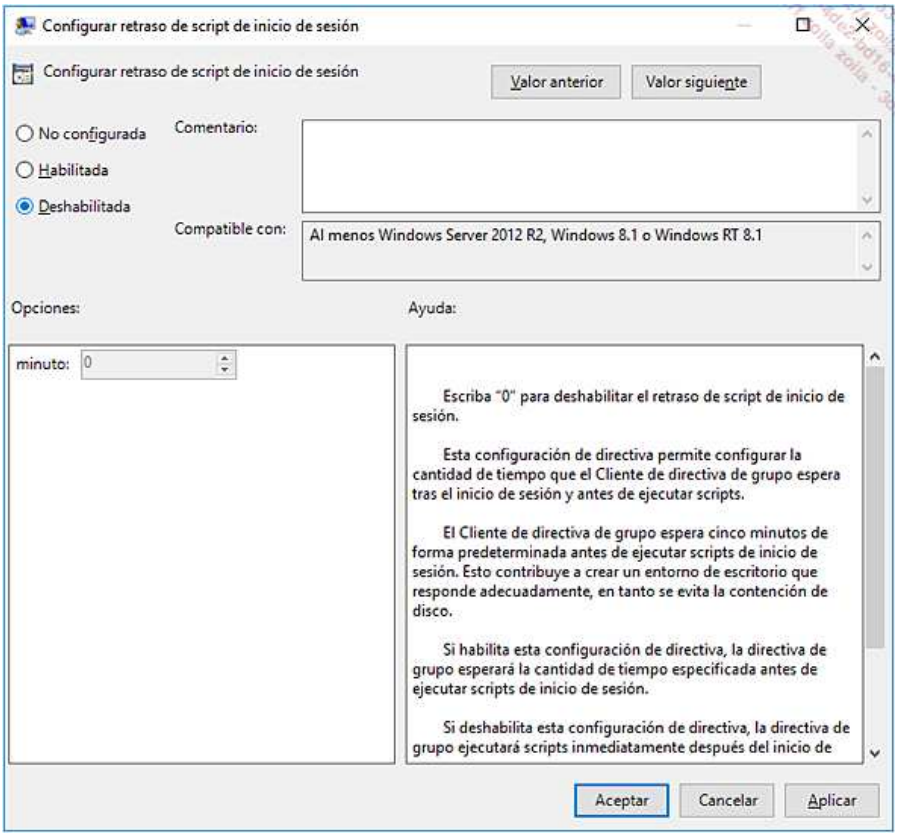
e. Comprobar la aplicación del objeto GPO en los puestos cliente

Para asegurarse de que el script de Windows PowerShell se ha desplegado y ejecutado correctamente, tiene la posibilidad, desde el puesto de trabajo, de ejecutar el comando **`gpresult /v`**. Este comando devuelve un informe completo acerca de las directivas de grupo que se han aplicado o no en el equipo.

```
PS C:\Users\Julien> gpresult /v

Herramienta de resultados para la directiva de grupos del sistema
operativo
Microsoft (R) Windows (R) v2.0
© 2016 Microsoft Corporation. Todos los derechos reservados.

Creado en 31/01/2017 a 23:46:01


RSOP datos para CONTOSO\julien en W10: modo de inicio de sesión
-----------------------------------------------------------------

Configuración del sistema operativo:          Estación de trabajo
miembro
Versión del sistema operativo:              10.0.14393
Nombre del sitio:                           N/A
Perfil móvil:                               N/A
Perfil local:                                C:\Users\Julien
¿Conectado a un vínculo de baja velocidad?: No


CONFIGURACIÓN DE USUARIO
------------------------
    CN=Julien,OU=Contoso,DC=contoso,DC=local
    Última vez que se aplicó la Directiva de grupo: 31/01/2017
a 23:45:48
        Directivas de grupo aplicadas desde:        CONTOSO-
DC01.contoso.local
     Umbral del vínculo de baja velocidad de las directivas de
grupo:   500 kbps
    Nombre de dominio:                     CONTOSO
     Tipo de dominio:                              Windows 2008 o
superior

    Objetos de directiva de grupo aplicados
    ---------------------------------------
        PowerShell GPO

      Los objetos GPO siguientes no se aplicaron porque fueron
filtrados
     ---------------------------------------------------------------
------
        Directiva de grupo local
          Filtrar:  No aplicado (vacío)

[...]

    Conjunto resultante de directivas para el usuario
    -------------------------------------------------

        Instalaciones de software
        -------------------------
            N/A
```

```
        Scripts de inicio de sesión
        -----------------------------
                GPO: PowerShell GPO
                Nombre:         PowerShell_GPO.ps1
                Parámetros:
                Último_ejecutado: 22:45:53

        Scripts de cierre de sesión
        -------------------------------
[...]
```

Preste atención: por defecto, con Windows 8.1 los scripts de inicio de sesión se ejecutan 5 minutos después del inicio de sesión, con el objetivo de hacer que el Escritorio esté disponible. Para lograr que la ejecución del script sea inmediata tras el inicio de sesión, hay que editar la directiva de grupo y modificar el parámetro **Configurar retraso de script de inicio de sesión** presente en la siguiente ubicación: **Configuración del equipo - Plantillas administrativas - Sistema - Directiva de grupo**. Hay que deshabilitar este parámetro para que los scripts se ejecuten inmediatamente tras el inicio de sesión.



Directiva «Configurar retraso de script de inicio de sesión»

De manera inversa, si desea aumentar el retraso en la ejecución de los scripts tras el inicio de sesión, haga clic en **Habilitada** y defina el retraso deseado (en minutos). Este parámetro de directiva es válido para Windows 8.1 y Windows 10.

f. Desplegar y ejecutar un script una única vez

Los puestos de trabajo se arrancan y detienen varias veces durante su ciclo de vida. Configurando el despliegue de un script de Windows PowerShell mediante una directiva de grupo, este script se ejecutará varias veces. Pero no todos los scripts necesitan ejecutarse periódicamente, de modo que conviene saber cómo hacer que la ejecución sea única.

Para ello, puede utilizarse un indicador (o flag, en inglés). Puede tratarse de un archivo o una carpeta, o también una clave de registro, creados intencionadamente o no (instalación de un programa, por ejemplo) durante la primera ejecución del script.

Una vez seleccionado el indicador, basta con incluir en el inicio del script una condición que compruebe su presencia. En caso negativo, se ejecutará el script completo y se configurará un indicador en caso necesario. Así, cuando la directiva de grupo vuelva a ejecutar el script, la condición será positiva y se producirá la salida del script.

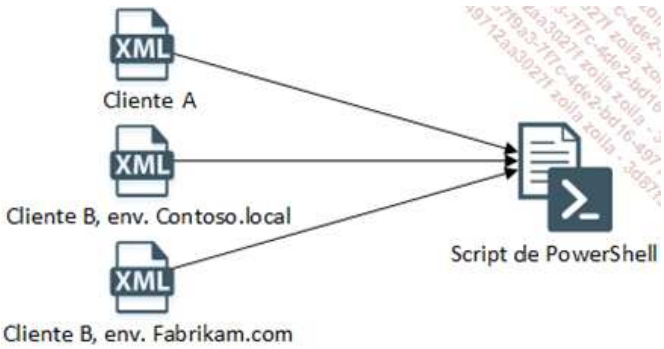
He aquí un ejemplo:

```
Function flag
{
    If (Test-Path HKLM:\Software\myCompany\Appli1)
    {
        # Aplicación instalada o script ya ejecutado
        Exit
    }
}

# Inicio del script
flag
```

3. Uso de un archivo XML para sus scripts

En ciertos casos, cuando se desarrolla un script, se quiere que pueda migrarse fácilmente de un entorno a otro. Los ejemplos son múltiples: dominios diferentes, sufijos de red diferentes, arquitecturas de sistema diferentes en función de cliente, etc.

En lugar de tener un script PowerShell para cada caso, lo cual resulta complejo de mantener, es posible exportar el conjunto de variables y valores definidos en estos scripts en un archivo XML. De este modo, tendrá que mantener un único script PowerShell, el cual buscará la información correspondiente al entorno en el archivo XML especificado en los argumentos durante la ejecución del script.



Uso de archivos XML con un script de PowerShell

a. Explicación y escenario de ejemplo

Si partimos del archivo XML siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <DomainInformation>
    <Name>contoso.local</Name>
    <Credential>
      <Login>Administrator</Login>
      <Password>
01000000d08c9ddf0115d1118c7a00c04fc297eb010000006f83c5cd643b914894
fa4
c422d15530e0000000020000000000003660000c0000001000000004a4be8eedc
4fc
1c5820250c2cd3418f0000000004800000a000000010000000e2401c201f7a1d46
006
5dfc80076f9144000000050d7b4205d8567e3bbeff5bac617ff2977bef5a0e019d
88a
44e59f5c748e8621349de591629c5728a05291d46d29549bf83dce62b415d82f1b
460
8c9f93c1e8214000000f10d29ebc04ae9427d51234faac11976357c1d78</Passw
ord>
    </Credential>
  </DomainInformation>
  <PackageInstallation>
    <Provider>Chocolatey</Provider>
    <Package>
      <Name>notepadplusplus</Name>
      <Name>adobereader</Name>
      <Name>paint.net</Name>
      <Name>visioviewer</Name>
    </Package>
```

```
  </PackageInstallation>
</Configuration>
```

El archivo XML contiene un cierto número de elementos, referenciados de manera personalizada, en función de los datos necesarios para la ejecución del script. En el caso actual, el script va a agregar el puesto de trabajo en un dominio, y a continuación instalará los paquetes de software a través del módulo PackageManagement de Windows PowerShell.

Supongamos que tenemos el script PowerShell siguiente, que se ejecutará pasándole como parámetro la ruta de acceso al archivo XML:

```
# Inicio del script

If ($args[0] -eq $null)
{
    Write-Host "XML file missing !!" -ForegroundColor Red
-BackgroundColor Yellow
    Write-Host Press any key to continue...
    $null = $Host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")
    Exit
}

# Carga del archivo XML
[xml]$xmlInput = (Get-Content $args[0]) -replace ("localhost",
$env:COMPUTERNAME)

If ((Get-ComputerInfo).CsDomainRole -ne "MemberWorkstation")
{
    # El equipo no es miembro de un dominio
    Write-Output "Agregar el puesto de trabajo en el dominio $
($xmlInput.Configuration.DomainInformation.Name)"
                                                $user            =
$xmlInput.Configuration.DomainInformation.Credential.Login
                                                $pwd             =
$xmlInput.Configuration.DomainInformation.Credential.Password
                        $cred      =      New-Object      -TypeName
System.Management.Automation.PSCredential
-ArgumentList $user, ($pwd | ConvertTo-SecureString)
                                    Add-Computer         -DomainName
$xmlInput.Configuration.DomainInformation.Name
-Credential $cred
}

# Instalación de los paquetes de software
If ($PSVersionTable.PSVersion.Major -ge 5)
{
     Write-Output "Instalación de los paquetes de software en el
puesto
de trabajo"
    Install-PackageProvider -Name
$xmlInput.Configuration.PackageInstallation.Provider
      $xmlInput.Configuration.PackageInstallation.Package.Name | %
{
        Install-Package -Name $_ -Force
    }
}
```

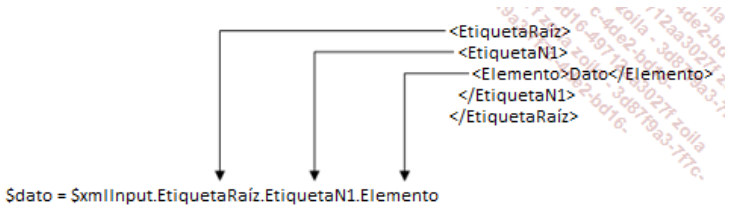
```
Write-Output "¡Ha terminado la ejecución del script!"
# Fin del script
```

El script de PowerShell va a cargar en una variable el contenido completo del archivo XML. Esta operación se lleva a cabo mediante la siguiente línea de comando (`$args[0]` representa la ruta de acceso al archivo XML):

```
[xml]$xmlInput = (Get-Content $args[0])
```

Aquí es importante precisar bien el tipo de datos `[xml]` a la variable que recibirá la información de **`Get-Content`**. En efecto, si se omite el tipo, recibirá una cadena de caracteres `[string]`, y no será posible recuperar los datos.

De este modo, cuando tenga que acceder a un dato incluido en el archivo XML, debe utilizar esta variable agregando las etiquetas correspondientes:



Recuperar un valor del archivo XML

Una vez recuperado el dato, podrá manipularlo según sea necesario en el script: realizar un procesamiento basándose en él y almacenar el resultado en otra variable, mostrar alguna información por pantalla, etc.

Si se invoca un elemento con el mismo nombre referenciado varias veces en una etiqueta, el resultado devuelto es un array de valores que contiene el conjunto de datos.

b. Datos encriptados en el archivo XML

Para evitar tener que escribir sin cifrar ciertos datos en los archivos XML (como por ejemplo, las contraseñas), hay que convertir las cadenas de caracteres en una cadena de caracteres encriptada. Esta operación se realiza mediante los cmdlets **`ConvertTo-SecureString`** y **`ConvertFrom-SecureString`**.

ConvertTo-SecureString

El rol de **ConvertTo-SecureString** es convertir una cadena de caracteres o una cadena de caracteres encriptada en un objeto de tipo `SecureString`. Este objeto podrá utilizarse a continuación con los comandos que requieran este tipo de objeto, como la creación de un objeto de tipo `PSCredential`.

He aquí una demostración donde se convierte una cadena de caracteres bruta en un objeto de tipo `SecureString`:

```
PS C:\Windows\system32> $pwd = "P@$$word!" | ConvertTo-
SecureString
-AsPlainText -Force
PS C:\Windows\system32> $pwd
System.Security.SecureString
```

Así, la variable `$pwd` es un objeto de tipo `SecureString`, que es como una caja fuerte que contiene la contraseña. A continuación, es posible crear directamente un nuevo objeto de tipo `PSCredential` que contendrá las credenciales de una cuenta de usuario, es decir, el nombre de usuario y la contraseña (consulte el ejemplo en el script Explicación y escenario de ejemplo), y proveer este tipo de objeto para realizar una autenticación.

Pero en este estado, un objeto de tipo `SecureString` no puede exportarse a un archivo de texto. Para ello, hay que convertir este objeto en una cadena de caracteres encriptada, operación que se realiza con el cmdlet **ConvertFrom-SecureString**.

ConvertFrom-SecureString

ConvertFrom-SecureString permite convertir un objeto de tipo `SecureString` en una cadena de caracteres encriptada, y así exportarla en un archivo de texto para su uso futuro. La cadena de caracteres encriptada podrá volver a convertirse, de nuevo, en un objeto `SecureString` con el cmdlet **ConvertTo-SecureString**.

He aquí un ejemplo, partiendo de una cadena de caracteres bruta:

```
PS C:\Windows\system32> $pwd = "P@$$word!" | ConvertTo-
SecureString
-AsPlainText -Force | ConvertFrom-SecureString
PS C:\WINDOWS\system32> $pwd
```

```
01000000d08c9ddf0115d1118c7a00c04fc297eb010000004cb962149752334c8c
4
08dd69ea5565d000000002000000000010660000001000020000000e2fe68f19
9
316d0aa9d1c1b655712087f260bcf848d3e78f259245886a15dda8000000000e80
0
000000200002000000082fe7d97cd10a15b2e3385c0c417f58d446b21e6911d643
5
907102bdb367ddd3200000008807380953ec1702f82760f76f62f6fbc2e5c69c8d
7
83bc65acc193caea7ca7740000000780d55016d0b58b2435ed9ba1da71446b6772
3
93b87880c06512143b85308f3c495add347f49d31f1ebde155dcb3f477f58a4e2f
5
9430db959f15c521e6379b
```

A continuación, esta cadena de caracteres podrá exportarse directamente en un archivo de texto:

```
PS C:\Windows\system32> $pwd = "P@$$word!" | ConvertTo-
SecureString
-AsPlainText -Force | ConvertFrom-SecureString | Out-File -
FilePath
C:\Temp\EncryptedPwd.txt
```

La cadena de caracteres encriptada puede insertarse a continuación en el archivo XML y evitar así el envío de datos sin cifrar.

Si alguna persona malintencionada recupera esta cadena encriptada y utiliza **`ConvertTo-SecureString`**, recuperará un objeto `SecureString`, pero no podrá obtener su contenido en una cadena de caracteres bruta.

```
PS C:\WINDOWS\system32> $encrypStr =
"01000000d08c9ddf0115d1118c7a00c04fc297eb010000004cb962149752334c8
c
408dd69ea5565d000000002000000000010660000001000020000000e2fe68f1
9
9316d0aa9d1c1b655712087f260bcf848d3e78f259245886a15dda8000000000e8
0
000000200002000000082fe7d97cd10a15b2e3385c0c417f58d446b21e6911d64
3
5907102bdb367ddd3200000008807380953ec1702f82760f76f62f6fbc2e5c69c8
d
783bc65acc193caea7ca7740000000780d55016d0b58b2435ed9ba1da71446b677
2
3293b87880c06512143b85308f3c495add347f49d31f1ebde155dcb3f477f58a4e
2f59430
db959f15c521e6379b"
PS C:\WINDOWS\system32> $encrypStr | ConvertTo-SecureString
System.Security.SecureString
```

Sin embargo, esto no detendrá a alguien empeñado en utilizar esta cadena de caracteres para comprometer la seguridad de una empresa y usurpar una identidad. Si bien esta protección es bastante buena, conviene ser precavido en cuanto a la

distribución del archivo o de los archivos que poseen alguna cadena de caracteres encriptada.

Conectar una unidad de red

He aquí un ejemplo de script para desplegar mediante un GPO que debe ejecutarse tras el inicio de sesión de los usuarios. Además de conectar una unidad de red personal, este script presenta la particularidad de conectar las unidades de red en función de los grupos de Active Directory de los que sea miembro el usuario.

Conviene, por tanto, disponer de una red empresarial con un dominio Active Directory y puestos de trabajo vinculados a este dominio.

Requisitos previos

Para conectar la unidad de red personal, hay que crear un recurso compartido en el servidor, llamado *users*. Esta carpeta compartida debe contener subcarpetas cuyo nombre sea el login de los usuarios. No olvide asignar los permisos correspondientes en función de las carpetas. Asegúrese de que el usuario posee permisos de lectura/escritura en su carpeta personal, y de que no pueda acceder a las carpetas de sus colegas.

Respecto a los recursos compartidos en función de los grupos AD de los que es miembro el usuario, siéntase libre de crear los nombres de recursos compartidos que desee. Hay que verificar a su vez que los accesos se han definido correctamente. En el script, hay que declarar el nombre del grupo al que está asociado el nombre del recurso compartido, así como la letra de la unidad de red que se utilizará en el puesto cliente para acceder al recurso compartido.

Esta declaración debe realizarse de la siguiente manera:

```
"nombre_del_grupo"=@("nombre_del_recurso_compartido",
"letra_de_la_unidad")
```

En el script se proporcionan ejemplos de declaraciones que le permiten hacerse una idea de cómo proceder. Modifíquelas en función de su configuración.

Por último, la variable `$server` designa el servidor donde se encuentran las carpetas compartidas.

El script

```
# Conectar unidades de red en función de los grupos AD
# de los que es miembro el usuario
# Script a ejecutar tras el inicio de sesión del usuario

#-----------------------------------------------------#
# Funciones
#-----------------------------------------------------#
#   Determina   si   el   usuario   en   curso   pertenece   al   grupo
correspondiente
Function inGroup([string] $groupName) {
    $insideGroup = $false

    $ldapQuery = "(samaccountname="+$env:USERNAME+")"
                                $objDomain        =        New-Object
System.DirectoryServices.DirectoryEntry
                               $objSearcher       =        New-Object
System.DirectoryServices.DirectorySearcher
-ArgumentList $objDomain,$ldapQuery

    $users = $objSearcher.FindAll()
    $user = $users[0]

    Foreach ($group in $user.properties.memberof) {
         If (($group.ToLower()).IndexOf("cn="+$groupName.ToLower()) -
ge 0)
        {
            $insideGroup = $true
            Break
        }
    }
    return $insideGroup
}

#-----------------------------------------------------#
# Variables
#-----------------------------------------------------#
# Servidor de archivos
$server = "WS2012R2.CONTOSO.LOCAL"

# Lista de grupos
#
"nombre_del_grupo"=@("nombre_del_recurso_compartido","letra_de_la_un
idad")
$groups = @{
    "GG_COMMUNICATION"=@("COMM","V:")
    "GG_ISO"=@("Share$","W:")
    "GG_COMMERCIAL"=@("CLIENTES","X:")
    "GG_DSI"=@("SISTEMAS Y REDES","Y:")
    "GG_BACKUP"=@("BACKUP","Z:")
}

#-----------------------------------------------------#

### Inicio del script
$obj = New-Object -ComObject Wscript.Network


# Conexión a la carpeta de red personal
$perNetworkShare = "\\"+$server+"\users\"+$env:USERNAME
If (Test-Path $perNetworkShare)
{
                                          $obj.MapNetworkDrive("U:",
```

```
"\\"+$server+"\users\"+$env:USERNAME)
}


# Conexión a las unidades de red en función de los grupos
$groups.GetEnumerator() | Foreach {
    If (inGroup($_.Name))
    {
            Write-Host $env:USERNAME " es miembro del grupo: "
$_.Value[0]
                                        $obj.MapNetworkDrive($_.Value[1],
"\\"+$server+"\"+$_.Value[0])
    }
}
```

En este script, se crea y se utiliza un objeto COM **`WScript.Network`** para conectar las unidades de red. La conexión a la unidad se realiza con el método `MapNetworkDrive`.

Para saber si el usuario es miembro de alguno de los grupos enumerados en el script, se utilizan dos objetos .NET para poder llevar a cabo una búsqueda en el AD:

- **`System.DirectoryServices.DirectoryEntry`**: permite acceder a los objetos presentes en el servicio de directorio Active Directory.
- **`System.DirectoryServices.DirectorySearcher`**: permite ejecutar consultas de búsqueda en el servicio de directorio Active Directory.

La ejecución de la consulta se realiza con el método **`FindAll()`** de la clase **`DirectorySearcher`**.

Si el script valida el hecho de que un usuario sea miembro de un grupo, entonces la variable `$insideGroup` pasa a valer `True`, lo que provoca la conexión de una unidad de red en función de los valores asociados a ella (letra de la unidad, nombre del recurso compartido).

Mantenimiento del script

Su principal ventaja es la simplicidad de su actualización. Así, si se crea un nuevo grupo o recurso compartido, basta con agregar una única línea en el script, lo que permite tener en cuenta esta actualización. La variable `$groups` contendrá el nuevo dato, y la búsqueda de pertenencia a este nuevo grupo se realizará automáticamente.

Ocurre lo mismo en el caso de tener que eliminar un grupo.

El único punto al que hay que prestar atención es no utilizar dos veces la misma letra de unidad, salvo si está seguro de que un usuario no puede pertenecer a dos grupos que utilicen la misma letra de unidad.

Posibles evoluciones

En términos de evolución puede, en primer lugar, modificar el script para incluir dos nombres de servidores: un servidor de archivos específico que contendrá únicamente las carpetas personales y un segundo servidor que contendrá únicamente los recursos compartidos asociados a los grupos AD.

La segunda evolución posible es llevar un poco más lejos la idea: especificar el nombre del servidor para cada uno de los grupos AD. Puede, así, utilizar un servidor de archivos completamente dedicado al backup y otro dedicado a uno u otro grupo, etc.

Por supuesto, es posible realizar otras evoluciones: ideje volar su imaginación!

Envío de e-mails

Otra funcionalidad interesante de implementar según los objetivos de sus scripts es la posibilidad de enviar e-mails. Así, por ejemplo, un script ejecutado regularmente mediante una tarea planificada puede advertirle con un mensaje en su buzón de correo electrónico y proporcionarle información acerca de algún problema que se haya detectado.

La función del siguiente script es comprobar, entre una lista de servidores, si el servicio **Servidor de Servicios de implementación de Windows** (WDSServer) está correctamente iniciado. En caso negativo, o si se ha detectado algún otro problema, se envía un correo electrónico.

Requisitos previos

Una lista de servidores de Windows con el rol **Servicios de implementación de Windows** instalado. El nombre de estos servidores debe indicarse en la variable `$arrayServidores`.

Las variables `$login`, `$pwd` y `$emailFrom` contienen las credenciales y la dirección de e-mail de la cuenta con la que se enviará el correo electrónico. Si se trata de una cuenta genérica, no olvide que esta debe poseer un buzón de correo electrónico.

La variable `$emailTo` contiene la dirección o las direcciones de e-mail de los destinatarios a los que se enviará el mensaje. Y, por último, la variable `$smtpServer` debe contener la dirección del servidor SMTP.

El script

```
# Comprueba si el servidor está en línea y el estado del servicio
WDS
# Se envía un e-mail con una explicación del problema detectado
# Script a incluir en una tarea planificada

#-----------------------------------------------------#
# Funciones
#-----------------------------------------------------#
# Function eMail (envío de e-mails)
Function eMail ([string]$serverFQDN, [string]$reason) {
    $servername = $serverFQDN.Split(('.'))

    If ($reason -eq "offline")
```

```
    {
            $subject = $servername[0] + ": ¡el servidor no está en
línea!"
          $body = "Comprobar si el servidor " + $servername[0] + "
está
arrancado y accesible."
    }
    ElseIf ($reason -eq "notStarted")
    {
           $subject = $servername[0] + ": ¡el servicio WDS no está
iniciado!"
        $body = "El servicio WDS no está iniciado en el servidor " +

$servername[0] + ".`r`n"
          $body += "Iniciar el servicio Windows Deployment Services
Server
en este servidor."
    }
    ElseIf ($reason -eq "notExist")
    {
              $subject = $servername[0] + ": ¡el servicio WDS no
existe!"
        $body = "El servicio WDS no existe en el servidor " +
$servername[0] + ".`r`n"
         $body += "Comprobar este punto y retirar el servidor de la
lista
si fuera necesario. "
    }

    # Construcción del objeto MailMessage + envío
      $mail = New-Object System.Net.Mail.MailMessage($exp, $emailTo,
$subject,
$body)
    $smtp.Send($mail)
}


# Function checkWDS (comprueba si el servicio WDS está iniciado)
Function checkWDS ([string]$server)
{
    $wds = Get-Service -ComputerName $server -Name WDSServer
-ErrorActionSilentlyContinue
    If (($wds.Status -ne "Running") -and ($wds -ne $null))
    {
        eMail $server "notStarted"
    }
    ElseIf ($wds -eq $null)
    {
        eMail $server "notExist"
    }
}

#-----------------------------------------------------#
# Variables
#-----------------------------------------------------#
# Variables para el envío de e-mails
Set-Variable -Name smtpServer       -Value "exch.contoso.local"
Set-Variable -Name login            -Value "WimAdmin"
Set-Variable -Name pwd              -Value 'P@$$w0rd'
Set-Variable -Name emailFrom        -Value "wimadmin@contoso.com"
Set-Variable -Name emailTo          -Value "wks.admin@contoso.com"

# Lista de servidores
```

```
$arrayServidores = @(
    "WDS-ANNECY.contoso.local"
    "WDS-PAU.contoso.local"
    )

#-----------------------------------------------------#

### Inicio del script
$smtp = New-Object Net.Mail.SmtpClient($smtpServer)
$exp = New-Object System.Net.Mail.MailAddress($emailFrom)
$smtp.Credentials = new-object System.Net.NetworkCredential($login,
$pwd)


# Comprueba si puede alcanzarse el servidor
$arrayServidores | % {

    # Ping del servidor de destino
    $ping = New-Object System.Net.NetworkInformation.Ping
    $reply = $ping.send($_)

    If ($reply.status -eq "Success")
    {
        # Servidor en línea
        checkWDS $_
    }
    Else
    {
        # Servidor fuera de línea
        eMail $_ "offline"
    }
}
```

Explicaciones

Al principio, se crean tres objetos .NET para permitir enviar e-mails a un servidor SMTP.

A continuación, el script toma cada uno de los servidores indicados en la variable `$arrayServidores`, y realiza una primera comprobación: si el servidor está en línea:

- En caso afirmativo, el script continúa y verifica el estado del servicio WDS (función **`checkWDS`**) en el servidor remoto.
- En caso negativo, se invoca la función **`eMail`** pasándole como argumentos el nombre del servidor con el que no ha sido posible conectar y una cadena de caracteres que identifica la causa del problema, "`offline`" en este caso.

En la función **`checkWDS`**, pueden darse tres casos:

- El servicio WDS no existe en el servidor remoto. En este caso, el mensaje de correo electrónico se preparará indicando el origen el problema "`notExist`".

- El servicio WDS está presente, pero no está iniciado. Se preparará entonces un e-mail con el origen del problema "`notStarted`".
- Por último, el servicio está presente y está iniciado. En este caso, no es necesario enviar ningún mensaje.

De modo que existen tres casos en los que puede invocarse la función **`eMail`**, y esta recibe como argumentos el nombre del servidor donde se ha detectado algún problema y la cadena de caracteres que representa la causa del problema. Gracias a esta causa podrán enviarse mensajes diferentes, con un mensaje más explícito para cada caso.

Para terminar, el objeto .NET **`System.Net.MailMessage`** crea el correo electrónico que contiene todos estos elementos: el remitente, el destinatario, el asunto del correo y el mensaje completo. Una vez creado el objeto, se envía al servidor SMTP.

Mantenimiento del script

Si se instala un nuevo servidor WDS en su empresa, basta con agregarlo en la lista dentro de la variable `$arrayServidores` para que sea tenido en cuenta en la próxima ejecución del script.

Posibles modificaciones

Puede modificar la función **`checkWDS`** para que el script compruebe el estado de otro servicio. También tiene la posibilidad de cambiar el tipo de objeto que va a comprobar el script: la presencia de un proceso, por ejemplo.

En lo relativo a la función **`eMail`**, puede variar los mensajes que se envían.

Implementar carpetas compartidas

Esta sección le mostrará cómo implementar carpetas compartidas y cómo administrar los permisos de compartición, abordando también la seguridad NTFS. El script propuesto a continuación no funcionará en Windows 7, pues se utilizan los cmdlets **`*-SmbShare`** y **`*-SmbShareAccess`**. Debe disponer, por lo tanto, de una versión de Windows superior.

Requisitos previos

No existen requisitos previos particulares para este script. Sin embargo, conviene ejecutarlo como administrador.

Debe definir la ruta de acceso a la carpeta que se compartirá (variable `$folderPath`), y a continuación el nombre del recurso compartido (variable `$shareName`), con las siguientes líneas de comando:

```
$folderPath = $env:SystemDrive + "\Public"
$shareName = "Public"
```

Por último, debe definir los permisos del recurso compartido y los permisos NTFS. Para ello, debe indicar las cuentas AD y el nivel de permisos asociado.

Los permisos NTFS aplicados sobre la carpeta se definen en la siguiente línea de comando, en el array que se pasa como segundo parámetro en la llamada a la función **`Security`**:

```
# Agrega los permisos NTFS (pestaña Seguridad)
Security $folderPath @{"CONTOSO\Julien"="FullControl";"CONTOSO\
Benjamin"="ReadAndExecute"}
```

Para los permisos sobre el recurso compartido, hay que introducir también la información presente en la siguiente línea de código, durante la llamada a la función **`Sharing`**:

```
# Compartir la carpeta
Sharing $folderPath $shareName @{"CONTOSO\Julien"="Full";
"CONTOSO\Benjamin"="Read"}
```

El script

```
# Activa el recurso compartido de una carpeta
#  Configura los permisos del recurso compartido y los permisos
NTFS

#-----------------------------------------------------#
# Funciones
#-----------------------------------------------------#
# Configura los permisos NTFS en la carpeta
Function Security ($folder, $arraySecurity)
{
    If ($arraySecurity.Count -ne 0)
    {
        $arraySecurity.GetEnumerator() | % {
            $acl = Get-Acl $folder
            $rule = New-Object System.Security.AccessControl.
FileSystemAccessRule($_.Name,$arraySecurity[$_.Name],
"ContainerInherit,
ObjectInherit", "None", "Allow")
            $acl.AddAccessRule($rule)
            Set-Acl $folder $acl
        }
    }
}


# Creación del recurso compartido + permisos
Function Sharing ($folder, $shareName, $arrayPermisos)
{
    New-SmbShare -Path $folder -Name $shareName

        If  (($arrayPermisos.Count  -ne  0)  -or  ($arrayPermisos  -eq
$null))
    {
        Revoke-SmbShareAccess -Name $shareName -AccountName
"Todos" -Force
    }

    $arrayPermisos.GetEnumerator() | % {
        Grant-SmbShareAccess -Name $shareName -AccessRight
$_.Value -AccountName $_.Name -Force
    }
}

# Comprueba si Windows PowerShell se ejecuta como Administrador
Function Test-Administrator
{
    $user = [Security.Principal.WindowsIdentity]::GetCurrent()
    (New-Object Security.Principal.WindowsPrincipal $user).IsInRole
([Security.Principal.WindowsBuiltinRole]::Administrator)
}

#-----------------------------------------------------#
# Variables
#-----------------------------------------------------#
# Carpetas
$folderPath = $env:SystemDrive + "\Public"
$shareName = "Public"

#-----------------------------------------------------#

# Ejemplos de seguridad NTFS:
# - FullControl    (-> Control total)
```

```
# - Modify          (-> Modificación)
# - ReadAndExecute  (-> Lectura y ejecución)
# - ListDirectory   (-> Mostrar el contenido de la carpeta)
# - Read            (-> Lectura)
# - Write           (-> Escritura)

# Permisos del recurso compartido:
# - Full    (-> Control total)
# - Change  (-> Modificar)
# - Read    (-> Lectura)

### Inicio del script
If (!(Test-Administrator))
{
    Write-Host "¡Debe ejecutar el script en modo Administrador!"
-ForegroundColorRed
    Exit
}

If (!(Test-Path $folderPath) -and !(Get-SmbShare -Name $shareName
-ErrorActionSilentlyContinue))
{
    # Creación de la carpeta
    New-Item $folderPath -ItemType Directory -Force

    # Agrega los permisos NTFS (pestaña Seguridad)
    Security $folderPath @{"CONTOSO\Julien"="FullControl";"CONTOSO\
Benjamin"="ReadAndExecute"}

    # Compartir la carpeta
                              Sharing       $folderPath       $shareName
@{"CONTOSO\Julien"="Full";"CONTOSO\
Benjamin"="Read"}
}
Else
{
     Write-Host "¡La carpeta o el nombre del recuso compartido ya
existen!"
-ForegroundColor Yellow
}
```

Explicaciones

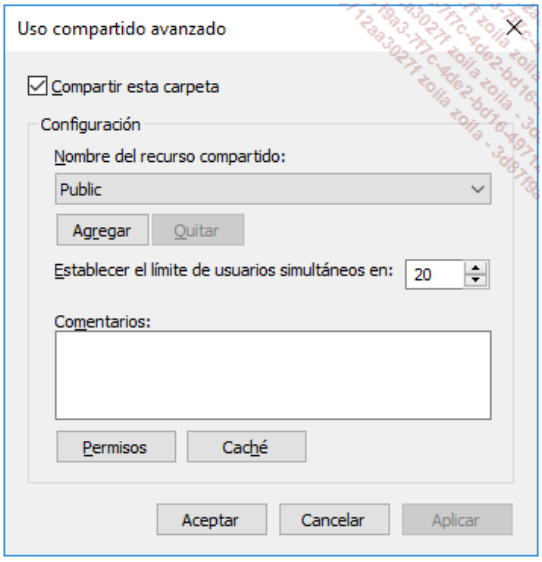
El script realiza, en primer lugar, una prueba para verificar que Windows PowerShell se ha ejecutado en modo administrador, y a continuación evalúa una segunda condición para saber si la carpeta que desea compartir y el nombre del recurso compartido no han sido creados previamente.

Si se cumplen estas condiciones, entonces se crea la carpeta compartida, y a continuación se invoca la función **`Security`**. El rol de esta función es implementar los permisos NTFS sobre la carpeta correspondiente indicada en la variable `$folderPath`. Los nombres de los usuarios y el nivel de permisos se indican a la función, que se encargará de aplicarlos.

Una vez realizada esta parte, el script invoca una segunda función: **`Sharing`**. Su rol es activar la compartición de la carpeta, y a continuación configurar los permisos del recurso compartido, según se haya definido en el array que se pasa a la función. Observe que se incluye una línea de comando que retira el permiso de lectura a «Todos» (permiso que se otorga por defecto cuando se implementa una carpeta compartida).

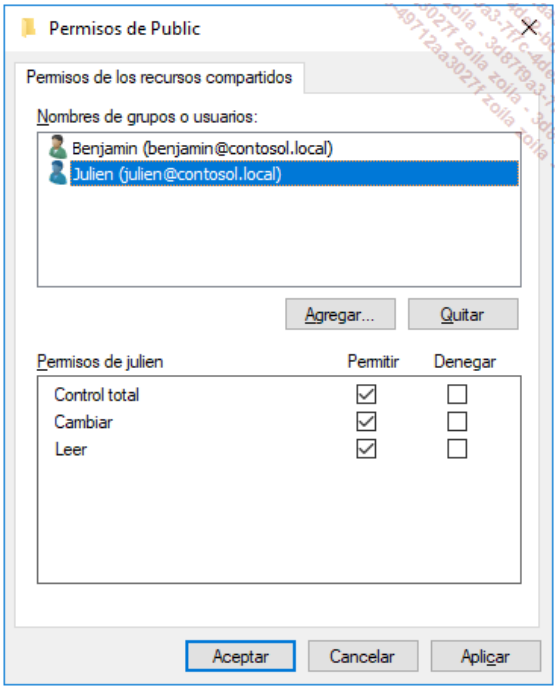
Consulte la sección Compartición de carpetas del capítulo Seguridad y protección para obtener más información acerca del uso de los cmdlets **`*-SmbShare`** y **`*-SmbShareAccess`**.

Tras ejecutar el script, la carpeta se ha compartido, con el nombre del recurso compartido «Public».



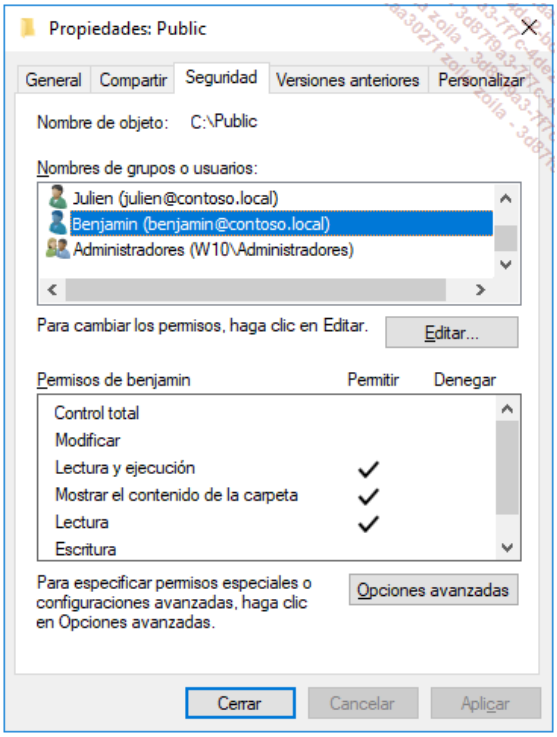
La compartición de la carpeta se ha activado correctamente, con el nombre del recurso compartido «Public»

Los permisos del recurso compartido se han aplicado correctamente, con el nivel de permiso adecuado especificado para cada uno de los usuarios o grupos:



Permisos del recurso compartido para la carpeta compartida

Por último, a nivel de los permisos NTFS, los usuarios o grupos están bien informados, siempre con el mismo nivel de permisos que se les haya asociado:



Permisos NTFS sobre la carpeta compartida

Conclusión

Ha podido ver, con la lectura de este libro, lo que Windows PowerShell es capaz de ofrecer para administrar puestos de trabajo… ¡y esto no es todo! Como Windows 10 se basa en un ciclo de desarrollo continuo, se aportan nuevas características con regularidad, y por consiguiente Windows PowerShell evoluciona también. Esto le proporciona nuevas posibilidades en la gestión de su parque informático.

Sin embargo, y como habrá podido comprobar, Windows PowerShell es un lenguaje de script orientado a objetos y, en comparación con los scripts .bat o .vbs, ofrece una flexibilidad y una agilidad innegables en el desarrollo de scripts y la administración por línea de comandos. Este es el motivo, por otro lado, de que PowerShell se haya convertido con el paso de los años en una herramienta imprescindible en la administración de entornos Microsoft.

Si bien Windows PowerShell nació con Windows y Windows Server, en la actualidad está presente en todo el ecosistema Microsoft: Exchange, SharePoint, SCCM, etc., y también en la cloud con Office 365 y Microsoft Azure. Windows PowerShell le ofrece una mejora increíble en la productividad de la administración del conjunto de aplicaciones del fabricante de Redmond.

Pero el alcance de PowerShell no se detiene aquí: numerosos fabricantes desarrollan y disponen de sus propios módulos de PowerShell, lo que permite interactuar con sus productos. Esto se va a acentuar con la publicación de PowerShell bajo licencia open source (licencia MIT). Sin duda, aprender y profundizar en este lenguaje de script le permitirá ahorrar un tiempo precioso en el conjunto de tareas administrativas que debe realizar de manera cotidiana y en el futuro.

¡Y ahora le toca a usted construir scripts! Los cmdlets y los parámetros que ha visto a través de los distintos capítulos de este libro no son más que una pequeña parte de lo que es posible realizar por línea de comandos. No dude en consultar la documentación de los cmdlets, bien utilizando **`Get-Help`**, o bien consultando el sitio TechNet de Microsoft para obtener su información completa. Además, Windows PowerShell ISE ofrece un entorno de desarrollo de scripts que le ayudará a alcanzar sus objetivos.

El uso de scripts de PowerShell para administrar puestos de trabajo puede hacerse a varios niveles: GPO, despliegue de un nuevo puesto de trabajo, o también para la supervisión, el mantenimiento, la migración, o para realizar cualquier otra operación, ya sea puntual o repetitiva.

Como habrá observado, el scripting es imprescindible para llevar a cabo la gestión y la automatización de las tareas administrativas en informática. Desde el despliegue de Windows hasta dar de baja un puesto de trabajo, Windows PowerShell no presenta límites a lo largo del ciclo de vida de la máquina. Para no perder más el tiempo repitiendo una tarea manualmente, ¡haga clic y arranque Windows PowerShell!

Lista de cmdlets

He aquí la lista de comandos disponibles con Windows PowerShell versión 5.1 en Windows 10.

Los comandos están agrupados por módulo y ordenados alfabéticamente, lo que permite orientarse mejor entre el conjunto de comandos que abordan un asunto particular.

De este modo, si conoce un comando PowerShell pero ha olvidado los demás comandos que pertenecen a la misma familia, puede encontrar el nombre del módulo con **`Get-Command`**:

```
PS C:\WINDOWS\system32> Get-Command New-ScheduledTask

CommandType     Name                               Version    Source
-----------     ----                               -------    ------
Function            New-ScheduledTask                         1.0.0.0
ScheduledTasks
```

Para conocer el nombre del módulo, consulte la columna `Source` o `ModuleName`. El cmdlet **`New-ScheduledTask`** forma parte, por tanto, del módulo ScheduledTasks.

La lista de comandos que se ofrece a continuación corresponde únicamente a los cmdlets disponibles tras una instalación de Windows 10 por defecto. Por ello, los siguientes módulos no están incluidos en esta lista:

- Hyper-V.
- IISAdministration.
- NetWNV.
- WebAdministration.

Cmdlets de Windows PowerShell disponibles en Windows 10

```
AppBackgroundTask
   -----------------

Disable-AppBackgroundTaskDiagnosticLog
Enable-AppBackgroundTaskDiagnosticLog
Get-AppBackgroundTask
Set-AppBackgroundTaskResourcePolicy
Start-AppBackgroundTask
Unregister-AppBackgroundTask
```

Lista de características disponibles en Windows 10

La siguiente lista contiene el nombre de todas las características disponibles en Windows 10 que es posible activar o desactivar utilizando respectivamente los cmdlets **Enable-WindowsOptionalFeature** y **Disable-WindowsOptionalFeature**. Para obtener más detalles acerca del uso de estos cmdlets, consulte las páginas indicadas en el índice.

Nombre de las características (FeatureName) que pueden activarse en Windows 10

```
Microsoft-Windows-Subsystem-Linux
LegacyComponents
DirectPlay
SimpleTCP
SNMP
WMISnmpProvider
MicrosoftWindowsPowerShellV2Root
MicrosoftWindowsPowerShellV2
Windows-Identity-Foundation
Internet-Explorer-Optional-amd64
NetFx3
IIS-WebServerRole
IIS-WebServer
IIS-CommonHttpFeatures
IIS-HttpErrors
IIS-HttpRedirect
IIS-ApplicationDevelopment
IIS-NetFxExtensibility
IIS-NetFxExtensibility45
IIS-HealthAndDiagnostics
IIS-HttpLogging
IIS-LoggingLibraries
IIS-RequestMonitor
IIS-HttpTracing
IIS-Security
IIS-URLAuthorization
IIS-RequestFiltering
IIS-IPSecurity
IIS-Performance
IIS-HttpCompressionDynamic
IIS-WebServerManagementTools
IIS-ManagementScriptingTools
IIS-IIS6ManagementCompatibility
IIS-Metabase
WAS-WindowsActivationService
WAS-ProcessModel
WAS-NetFxEnvironment
WAS-ConfigurationAPI
IIS-HostableWebCore
```