

# Comunicaciones remotas Windows PowerShell

El mecanismo de comunicación a distancia Windows PowerShell de la versión 2 de PowerShell se basa en el protocolo WS-MAN, también llamado «Administración de los servicios Web» o también «Administración remota de Windows (WinRM)».

WinRM es una tecnología que ha surgido recientemente y que apareció con Windows Server 2003 R2. WinRM, que quiere decir Windows Remote Management, permite enviar peticiones de administración a máquinas por red vía los puertos 5985 (HTTP) /5986 (HTTPS). Estos últimos son los nuevos puertos definidos en Windows 7 y Windows Server 2008 R2 como puerto de escucha para las comunicaciones WinRM. Sigue también siendo posible utilizar los puertos «standard» 80 (HTTP) y 443 (HTTPS) siempre que se tenga en cuenta especificarlos (ver más adelante). El protocolo que se esconde detrás de esta tecnología se denomina WS-Management; está basado en SOAP (*Simple Object Access Protocol*).

Al igual que WBEM, WS-Management (o WSMAN) aporta un estándar (esquema XML) orientado a la administración de hardware para facilitar la interoperabilidad entre sistemas heterogéneos en el seno de una infraestructura IT. Más concretamente, WinRM establece una sesión con una máquina remota a través del protocolo WS-Management en lugar de DCOM, como hace WMI en entornos distribuidos. Por consiguiente, el resultado de una petición WS-Management es un flujo de datos XML que transita por los puertos estándar de la Web.

La seguridad de WinRM se basa en métodos estándar para la autenticación y el cifrado de los intercambios; para la autenticación Kerberos es el protocolo utilizado por defecto en el interior de un dominio. Pero pueden utilizarse otros métodos de autenticación como: NTLM, el mapeo de un certificado cliente, o también como siempre la autenticación básica con usuario y contraseña. Por supuesto ésta es altamente desaconsejable en un entorno de producción, a menos que se utilice con HTTPS o IPSEC (*Internet Protocol SECURITY*).

## 1. Instalación de los requisitos previos

Los requisitos previos tanto en la máquina local como en las máquinas remota(s) son los siguientes:

- Windows PowerShell v2 o posterior
- Framework .NET 2.0 o posterior
- Windows Remote Management 2.0

Todos estos compuestos vienen instalados de forma nativa en Windows 7 y Windows Server 2008 R2. Forman también parte del paquete de instalación de PowerShell v2 para las versiones anteriores de Windows

Para estar en condiciones de ejecutar comandos remotos, debe ser miembro del grupo Administradores de la máquina remota o ser capaz de proporcionar información de identificación de un administrador.

Es también necesario iniciar la sesión PowerShell como administrador si desea hacer por lo menos una de las tareas siguientes:

- Conexión local por medio de una conexión remota. Esta operación se denomina «bucle invertido»,
- Administración de la configuración de sesión en el ordenador local,
- Visualización y modificación de los parámetros de Administración de los servicios Web en el ordenador local mediante un proveedor WSMAN.

Para verificar la versión de PowerShell instalada, utilice la variable automática \$PSVersionTable. Ver a continuación:

PS > \$PSVersionTable	
Name	Value
----	-----

CLRVersion	2.0.50727.4927
BuildVersion	6.1.7600.16385
PSVersion	2.0
WSManStackVersion	2.0
PSCompatibleVersions	{1.0, 2.0}
SerializationVersion	1.1.0.1
PSRemotingProtocolVersion	2.1


## 2. Configuración del sistema

Para autorizar a un sistema a recibir comandos remotos es necesario utilizar el commandlet `Enable-PSRemoting`. Observe que para enviar comandos, este paso no es necesario. Es incluso altamente desaconsejado si no es necesario ya que abre el sistema a posibles ataques.


`Enable-PSRemoting` efectúa las operaciones de configuración siguientes:

- Inicio del servicio WinRM si no está iniciado,
- Cambia el tipo de inicio del servicio WinRM a «automático»,
- Crea un componente de escucha WinRM para aceptar las peticiones de cualquier dirección IP,
- Activa una excepción en el firewall para las comunicaciones WS-Man,
- Activa todas las configuraciones de sesión inscritas de forma que puedan recibir ordenes de un ordenador remoto,
- Inscribe la configuración de sesión «Microsoft.PowerShell» (veremos más adelante que es una configuración de sesión),
- Inscribe la configuración de sesión «Microsoft.PowerShell32» si el sistema operativo es de 64 bits,
- Elimina la autorización «Rechazar a Todo el mundo» del descriptor de seguridad para todas las configuraciones de sesión existentes,
- Reinicia el servicio WinRM para aplicar los cambios.

---

 La configuración del servicio WinRM activa la regla del firewall llamada «Windows Remote Management (HTTP-in)» abriendo el puerto TCP 5985.

---

 El nombre de esta regla es «Administración remota de Windows» en las versiones Windows 7 y en las versiones de servidor anteriores a Windows Server 2008.

---

Si el sistema nunca se ha configurado, se debería obtener algo bastante similar a:

```
PS > Enable-PSRemoting
```

### Configuración rápida de WinRM

Se está ejecutando el comando "Set-WSManQuickConfig" para habilitar este equipo para administración remota mediante el servicio WinRM.

Esto incluye:

1. Iniciar o reiniciar (si ya está iniciado) el servicio WinRM
2. Establecer el tipo del servicio WinRM en inicio automático
3. Crear una escucha para aceptar solicitudes en cualquier dirección IP
4. Habilitar una excepción de firewall para el tráfico WS-Management

(sólo para http).

#### ¿Desea continuar?

[S] Sí [O] Sí a todo [N] No [T] No a todo [U] Suspendir [?] Ayuda (el valor predeterminado es "S"): S  
WinRM se ha actualizado para recibir solicitudes.  
Se cambió el tipo de servicio WinRM correctamente.  
Servicio WinRM iniciado.

Para verificar la correcta configuración del sistema puede intentar crear una sesión con la ayuda del comando `New-PSSession`, como se muestra a continuación:

```
PS > New-PSSession
```

Id	Name	ComputerName	State	ConfigurationName	Availability
1	Session1	localhost	Opened	Microsoft.PowerShell	Available

Si obtiene el mismo resultado, entonces felicitaciones, su sistema está listo para recibir comandos remotos!

### 3. Administración de la configuración de sesiones remotas

Si usted ha observado con atención el proceso de configuración de la máquina, habrá podido constatar que ha tenido lugar la creación de una configuración de sesión con la ayuda del comando `Register-PSSessionConfiguration`.

Una configuración de sesión es un grupo de parámetros en la máquina local que define el entorno para las sesiones remotas creadas cuando los usuarios remotos se conectan. Es posible crear configuraciones de sesión más o menos restrictivas en términos de seguridad según las necesidades del momento. Por defecto, sólo los miembros del grupo Administradores tienen autorización para conectarse remotamente. Es posible, por ejemplo, en una configuración de sesión, limitar el tamaño de los objetos que el ordenador recibe en la sesión, definir el modo de lenguaje y especificar los commandlets, proveedores y funciones que están disponibles en la sesión.

Las configuraciones de sesión PowerShell se utilizan únicamente cuando usted se conecta remotamente con los comandos `New-PSSession`, `Enter-PSSession` y `Invoke-Command`.

Los comandos de administración de configuración son los siguientes, y siempre se aplican localmente:

Comando	Descripción
<code>Register-PSSessionConfiguration</code>	Crea una configuración de sesión en la máquina local
<code>Unregister-PSSessionConfiguration</code>	Elimina una configuración de sesión
<code>Get-PSSessionConfiguration</code>	Obtiene las configuraciones de sesión inscritas
<code>Set-PSSessionConfiguration</code>	Modifica las propiedades de una configuración de sesión inscrita
<code>Enable-PSSessionConfiguration</code>	Activa las configuraciones de sesión
<code>Disable-PSSessionConfiguration</code>	Rechaza el acceso a las configuraciones de sesión
<code>New-PSSessionOption</code>	Crea un objeto que contiene las opciones avanzadas de una sesión <code>PSSession</code>

#### Configuración de sesión por defecto

Windows PowerShell incluye una configuración de sesión integrada denominada `Microsoft.PowerShell`. En los ordenadores que ejecutan versiones 64 bits de Windows (como en nuestro caso), Windows PowerShell proporciona también una segunda configuración de sesión llamada `Microsoft.PowerShell32`.

Esta configuración de sesión se utiliza por defecto para las sesiones, es decir, cuando un comando permite crear una sesión no incluye el parámetro `-ConfigurationName`.

Es también posible modificar la configuración de sesión por defecto utilizando la variable de preferencia `$PSSessionConfigurationName`.

Observemos nuestra configuración de sesión por defecto:

```
PS > Get-PSSessionConfiguration Microsoft.PowerShell | Format-List

Name                : microsoft.powershell
Filename            : %windir%\system32\pwrshplugin.dll
SDKVersion          : 1
XmlRenderingType    : text
lang                : es-ES
PSVersion           : 2.0
ResourceUri         : http://schemas.microsoft.com/powershell/
microsoft.powershell
SupportsOptions     : true
Capability           : {Shell}
xmlns               : http://schemas.microsoft.com/wbem/wsman/1/
config/PluginConfi...
Uri                 : http://schemas.microsoft.com/powershell/
microsoft.powershell
ExactMatch          : true
SecurityDescriptorSddl : O:NSG:BAD:P(A;;GA;;;BA)S:P(AU;FA;GA;;;WD)
(AU;SA;GXGW;;;WD)
Permission          : BUILTIN\Administrators AccessAllowed
```

Observe que también es posible acceder a esta información con la ayuda de un proveedor WSMAN.

## 4. Crear una sesión remota

Antes de poder enviar órdenes a una ordenador remoto es necesario establecer un sesión remota.

Una sesión remota puede ser:

- **Temporal:** una sesión temporal dura únicamente el tiempo del envío de un comando con `Invoke-Command` o `Enter-PSSession`. Tal es el caso en el momento de la utilización del parámetro `-ComputerName`.
- **Permanente:** una sesión «permanente» persiste durante el tiempo de la sesión PowerShell. Una sesión permanente es útil en el caso en que se deban ejecutar diversos comandos que compartan datos por medio de variables o funciones. Se crea una sesión permanente cuando se utiliza el parámetro `-session` de los comandos `Invoke-Command` o `Enter-PSSession`.

La creación de una conexión permanente con la máquina local o remota se efectúa con el comando `New-PSSession`. Observemos sus parámetros con la ayuda de la tabla siguiente:

Parámetro	Descripción
<code>AllowRedirection &lt;Switch&gt;</code>	Permite el direccionado de esta conexión hacia un identificador uniforme de recursos URI ( <i>Uniform Resource Identifier</i> ) alternativo.

ApplicationName <String>	Especifica el segmento del nombre de aplicación del identificador URI de la conexión.
Authentication <AuthenticationMechanism>	Especifica el mecanismo que se utiliza para autenticar las credenciales del usuario.
CertificateThumbprint <String>	Especifica el certificado de clave pública digital (X509) de una cuenta de usuario que tiene permiso para realizar esta acción.
ComputerName <String[]>	Crea una conexión persistente (PSSession) con el equipo especificado.
ConfigurationName <String>	Especifica la configuración de sesión que se usa para la nueva sesión de PowerShell.
ConnectionURI <Uri[]>	Especifica un identificador uniforme de recursos (URI) que define el extremo de la conexión.
Credential <PSCredential>	Especifica una cuenta de usuario con permiso para realizar esta acción.
Name <String[]>	Especifica un nombre sencillo para la PSSession.
Port <Int>	Especifica el puerto de red del equipo remoto que se utiliza para este comando. El valor predeterminado es el puerto 80 (puerto HTTP).
Session <PSSession[]>	Utiliza la PSSession especificada como modelo para la nueva PSSession.
SessionOption <PSSessionOption>	Establece opciones avanzadas para la sesión.
ThrottleLimit <Int>	Especifica el número máximo de conexiones simultáneas que se pueden establecer para ejecutar este comando.
UseSSL <Switch>	Utiliza el protocolo Capa de sockets seguros (SSL) sobre HTTPS para establecer la conexión. Por defecto, no se usa SSL.

Veamos algunos ejemplos de la utilización de New-PSSession:

#### Ejemplo 1:

```
PS > $session = New-PSSession
```

Creación de una sesión en el ordenador local y almacenamiento de la referencia del objeto PSSession en la variable \$session.

#### Ejemplo 2:

```
PS > $session = New-PSSession -ComputerName W2K8R2SRV
```

Creación de una sesión en el ordenador remoto «W2K8R2SRV» y almacenamiento de la referencia del objeto PSSession en la variable \$session.

#### Ejemplo 3:

```
PS > $cred = Get-Credential
PS > $session = New-PSSession -ComputerName W2K8R2SRV -Credential $cred
```

Creación de una sesión en el ordenador remoto «W2K8R2SRV» especificando autorizaciones alternativas y almacenamiento de la referencia del objeto PSSession en la variable \$sesion.

#### Ejemplo 4:

```
PS > $maquinas = Get-Content C:\temp\maquinas.txt
PS > $sesiones = New-PSSession -ComputerName $maquinas -ThrottleLimit 50
```

Creación de múltiples sesiones en una lista de ordenadores remotos que especifican un número de 50 conexiones máximas simultáneas.

## 5. Ejecución de comandos remotos

¡Entremos ahora de lleno en el tema! El comando necesario para la ejecución de comandos remotos es `Invoke-Command`. Éste posee numerosos parámetros que veremos un poco más tarde.

Como le decíamos en el apartado anterior, una sesión puede ser temporal o permanente. Se puede elegir una u otra en función de la necesidad del momento. Para el envío puntual de un comando remoto puede ser más fácil utilizar una sesión temporal, mientras que, si usted debe enviar una sucesión de comandos, será más fácil establecer una sesión permanente.

Para ilustrar estas propuestas, nos interesaremos por algunos casos de uso:

#### Ejemplo 1: recuperación de una variable de entorno en un ordenador remoto

```
PS > Invoke-Command -ComputerName W2K8R2SRV -ScriptBlock
{$env:PROCESSOR_IDENTIFIER}
Intel64 Family 6 Model 15 Stepping 11, GenuineIntel
```

No presenta ninguna complicación especial, se especifica mediante el parámetro `-ComputerName` el nombre de la máquina en la que se ejecutará el bloque del script pasado entre llaves al parámetro `-ScriptBlock`.

#### Ejemplo 2: listar los servicios suspendidos en varios ordenadores

```
PS > $cmde = { Get-Service | Where {$_.Status -eq 'paused'} }
PS > Invoke-Command -ComputerName W2K8R2SRV, localhost -ScriptBlock $cmde
```

Status	Name	DisplayName	PSComputerName
-----	----	-----	-----
Paused	vmacthlp	Hyper-V Time Synchronization Service	w2k8r2srv
Paused	Winmgmt	Infraestructura de gestión Windows	localhost

Vea con qué sorprendente facilidad hemos podido ejecutar un pequeño bloque de script en una lista de ordenadores. Observe la aparición de la propiedad `PSComputerName`. Ésta es muy práctica ya que nos indica el nombre de la máquina a la que pertenece el resultado.

Establecemos ahora una sesión permanente en la máquina «W2K8R2SRV» y veamos las otras cosas que se pueden hacer en una sesión temporal.

#### Ejemplo 3: cambiar el estado de un servicio del estado «Paused» al estado «Running»

En primer lugar creamos una sesión remota:

```
PS > $psession = New-PSSession -ComputerName W2K8R2SRV
PS > $psession
```

Id	Name	ComputerName	State	ConfigurationName	Availability
1	Session1	w2k8r2srv	Opened	Microsoft.PowerShell	Available

En segundo lugar, utilizamos la sesión para conectarnos a la máquina remota:

```
PS > Invoke-Command -Session $psession -ScriptBlock {
$s = Get-Service vmictimesync}
```

Ahora hemos recuperado en la variable `$s` el objeto correspondiente al servicio «vmictimesync».

Tratemos de manipular este servicio con el fin de modificar su estado de pausa:

```
PS > Invoke-Command -Session $psession -ScriptBlock {$s}
```

Status	Name	DisplayName	PSComputerName
-----	----	-----	-----
Paused	vmictimesync	Hyper-V Time Synchronization Service	w2k8r2srv

Hagamos una llamada al método `Continue` para modificar el estado del servicio:

```
PS > Invoke-Command -Session $psession -ScriptBlock {$s.continue()}
```

Después un pequeño `Refresh` del estado para ver si ha pasado alguna cosa:

```
PS > Invoke-Command -Session $psession -ScriptBlock {$s.refresh()}
```

Recordamos la variable `$s` para ver su contenido:

Status	Name	DisplayName	PSComputerName
-----	----	-----	-----
Running	vmictimesync	Hyper-V Time Synchronization Service	w2k8r2vm

¡Perfecto, ha funcionado!

Ahora intentemos de ver si podemos hacer lo mismo utilizando una sesión temporal.

```
PS > Invoke-Command -ComputerName W2K8R2SRV -ScriptBlock {
$s = Get-Service vmictimesync}
PS > Invoke-Command -ComputerName W2K8R2SRV -ScriptBlock {$s}
```


No pasa nada por la sencilla razón de que no hay persistencia de datos entre sesiones. Cuando se utiliza una sesión temporal, es como si cada vez abriésemos y cerrásemos una consola PowerShell remota. Por consiguiente, todas las variables se destruyen sistemáticamente al final de la sesión.

## 6. Ejecución de scripts remotos

La ejecución de scripts remotos se hace de la misma forma que la ejecución de un bloque de script, salvo que el parámetro a utilizar en lugar de `-ScriptBlock` se denomina `-FilePath`.

La ruta indicada por el parámetro `-FilePath` es la ruta del script situado en la máquina local. Con este parámetro, PowerShell convierte el contenido de un archivo de script en un bloque de script, transmitiendo el bloque al ordenador remoto para después ejecutarlo en el ordenador remoto.

Observe sin embargo que para especificar valores de parámetro al script, es preciso utilizar el parámetro `-ArgumentList` para especificarlos.

 Puesto que PowerShell efectúa una conversión script -> bloque de script y luego transmite el bloque de script a los ordenadores remotos, no existe riesgo de que la ejecución del bloque de script sea rechazada por las políticas de ejecución de las máquinas remotas. En efecto, incluso en modo «restricted» la ejecución de comandos es posible; lo que no es el caso de los scripts, ejecutados localmente.

### Ejemplo 1:

*Ejecución de un script de recuperación del espacio de disco restante (este script lo trataremos al final de este apartado)*

*Tenemos un pequeño script llamado Get-FreeSpace.ps1 que, como su nombre indica, devuelve el espacio de disco libre de todas las unidades.*

*Veamos que obtenemos de su ejecución local:*

```
PS > C:\scripts\get-freespace.ps1
```

Sistema	Disco	Disponible (Gb)	% restante
WIN7_US_X64	C:	1,2	10

Ahora, ejecutamos este script en una máquina remota y vemos el resultado:

```
PS > Invoke-Command -ComputerName w2K8R2SRV -FilePath
C:\scripts\get-freespace.ps1
```

Sistema : W2K8R2SRV  
Disco : C:  
Disponible (Gb) : 118,4  
% restante : 93  
PSComputerName : w2k8r2srv  
RunspaceId : a145d985-e35c-42a8-816c-62fala1bc8a5  
PSShowComputerName : True

La presentación del resultado en modo lista en lugar del modo tabla es normal en la medida en que hay más de cuatro propiedades a mostrar (véase el Capítulo Descubra PowerShell - Formateo de la vista). Pero esto no es lo más importante... En efecto, podemos constatar que aparecen propiedades adicionales `PSComputerName`, `RunspaceID` y `PSShowComputerName`. En realidad estas propiedades están siempre presentes cuando se trabaja con los mecanismos de comunicación remota PowerShell, pero no siempre están visibles.

El `RunspaceID` corresponde a una especie de «burbuja» en la cual se ejecuta la sesión remota PowerShell. Cuando se trabaja con sesiones temporales, el `RunspaceID` cambia a cada nuevo comando invocado por `Invoke-Command`; este no es el caso cuando se trabaja con sesiones permanentes.

He aquí los parámetros del comando `Invoke-Command`:

Parámetro	Descripción
<code>AllowRedirection &lt;Switch&gt;</code>	Permite el direccionado de esta conexión a un identificador URI ( <i>Uniform Resource Identifier</i> ) alternativo.
<code>ApplicationName &lt;String&gt;</code>	Especifica el segmento del nombre de aplicación del identificador URI de la conexión.
<code>ArgumentList &lt;Object[]&gt;</code>	Proporciona los valores de las variables locales en el comando. Las variables en el comando se reemplazan con estos valores antes de que se ejecute el comando en el equipo remoto. Escriba los valores en una lista separada por comas. Los valores se asocian a las variables



	en el orden en que aparecen en la lista.
AsJob <Switch>	Ejecuta el comando como un trabajo en segundo plano en un equipo remoto. Use este parámetro para ejecutar comandos que tardan mucho tiempo en completarse.
Authentication <AuthenticationMechanism>	Especifica el mecanismo que se utiliza para autenticar las credenciales del usuario.
CertificateThumbprint <String>	Especifica el certificado de clave pública digital (X509) de una cuenta de usuario que tiene permiso para realizar esta acción.
ComputerName <String[]>	Especifica los equipos en los que se ejecuta el comando. El valor predeterminado es el equipo local.
ConfigurationName <String>	Especifica la configuración de sesión que se usa para la nueva sesión de PowerShell.
ConnectionURI <Uri[]>	Especifica un URI que define el extremo de la conexión.
Credential <PSCredential>	Especifica una cuenta de usuario con permiso para realizar esta acción.
FilePath <String[]>	Ruta local del script a ejecutar en remoto.
HideComputerName <Switch>	Omite en la presentación de resultados el nombre de equipo de cada objeto (propiedad PSComputerName). De forma predeterminada, se muestra en la pantalla el nombre del equipo que generó el objeto.
InputObject <psobject>	Especifica la entrada al comando. Especifique la variable que contiene los objetos o escriba un comando o una expresión que los obtenga.
JobName <String>	Especifica un nombre sencillo para el trabajo en segundo plano. De forma predeterminada, los trabajos se denominan "Trabajo<n>", donde <n> es un número ordinal. Este parámetro es válido únicamente con el parámetro AsJob.
Port <Int>	Especifica el puerto de red en el equipo remoto utilizado para este comando. El valor predeterminado es el puerto 80 (puerto HTTP).
ScriptBlock <scriptblock>	Especifica los comandos que se han de ejecutar. Incluya los comandos entre llaves ({}) para crear un bloque de script. Este parámetro es obligatorio.
Session <PSSession[]>	Ejecuta el comando en las sesiones de Windows PowerShell (PSSessions) especificadas. Escriba una variable que contenga las PSSessions o un comando que las cree u obtenga, como un comando New-PSSession o Get-PSSession.
SessionOption <PSSessionOption>	Establece opciones avanzadas para la sesión.
ThrottleLimit <Int>	Especifica el número máximo de conexiones simultáneas que se pueden establecer para ejecutar este comando. Si omite este parámetro o escribe un valor 0, se utilizará el valor predeterminado 32.
UseSSL <Switch>	Utiliza el protocolo Secure Socket Layer sobre HTTPS para establecer la conexión. Por defecto, no se usa SSL.

#### Script Get-FreeSpace.ps1:

```
# get-freespace.ps1
param ($computer = '.')

# recupera todos los discos lógicos del ordenador:

Get-WmiObject -Computer $computer Win32_LogicalDisk |
Where {$_.drivetype -eq 3} |
Select-Object @{e={$_.systemname};n='Sistema'},
    @{e={$_.name};n='Disco'},
    @{e=[math]::Round($_.freespace/1GB,1)};n='Disponibile (Gb)'},
    @{e=[math]::Round((([int64]$_.freespace/[int64]$_.size*100),0)};
    n='% restante'}
```



Podrá observar en este ejemplo una anotación un poco particular utilizada con el comando `Select-Object`. Le hemos pasado realmente una tabla hash. Se trata de una sugerencia muy útil, que permite: efectuar cálculos de las propiedades de objetos; definir el nombre de las propiedades del objeto resultantes. El objeto que resulta de tal control es un objeto personalizado de tipo `PSObject`.

## 7. Apertura de una consola PowerShell en remoto

El último punto a abordar antes de cerrar este apartado de las comunicaciones remotas PowerShell, se refiere a la posibilidad de ejecutar comandos en modo interactivo en una máquina remota; es decir que todos los comandos escritos en la consola se ejecutarán en una máquina remota. Se trata de un funcionamiento similar al comando Telnet; pero por supuesto, imás seguro!

Para ello, tenemos dos posibilidades: la primera consiste en abrir una consola clásica con un comando ad-hoc, y la segunda se basa en la consola PowerShell en modo gráfico (PowerShell ISE).

### a. Enter-PSSession

El comando `Enter-PSSession` inicia una sesión interactiva con un ordenador remoto único. Sólo puede estar abierta una única sesión interactiva a la vez. Los posibles perfiles PowerShell presentes en la máquina remota no están cargados.

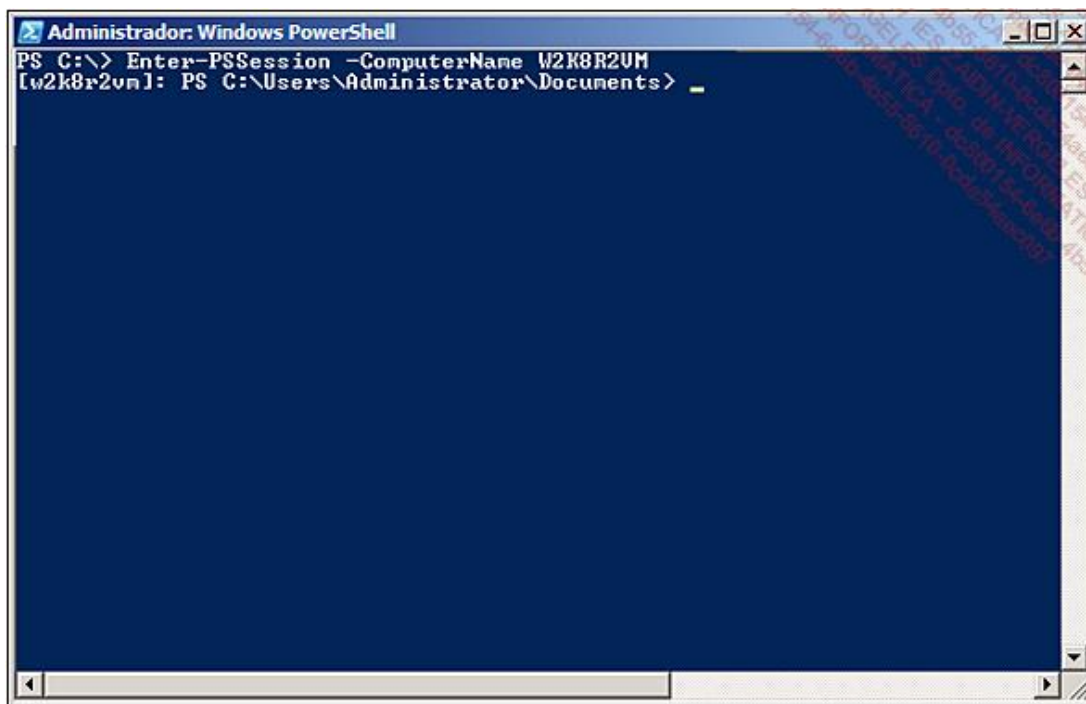
Una vez finaliza la sesión, teclee `Exit-PSSession` o simplemente `Exit` para desconectarse. Para iniciar una sesión PowerShell remota, debe iniciar PowerShell como Administrador.

Generalmente con `Enter-PSSession` se utiliza el parámetro `-ComputerName` para especificar el nombre del ordenador remoto, pero puede también pasar, si lo desea, un objeto de tipo `PSSession` al parámetro `-session`.

#### Ejemplo:

```
PS > Enter-PSSession -ComputerName W2K8R2VM
```

El resultado de esta línea de comandos es que se obtiene un prompt diferente. En este último, se encuentra el nombre del ordenador remoto como se muestra en la captura de pantalla siguiente:



*Apertura de una consola PowerShell remota*

Los parámetros de `Enter-PSSession` son los siguientes:

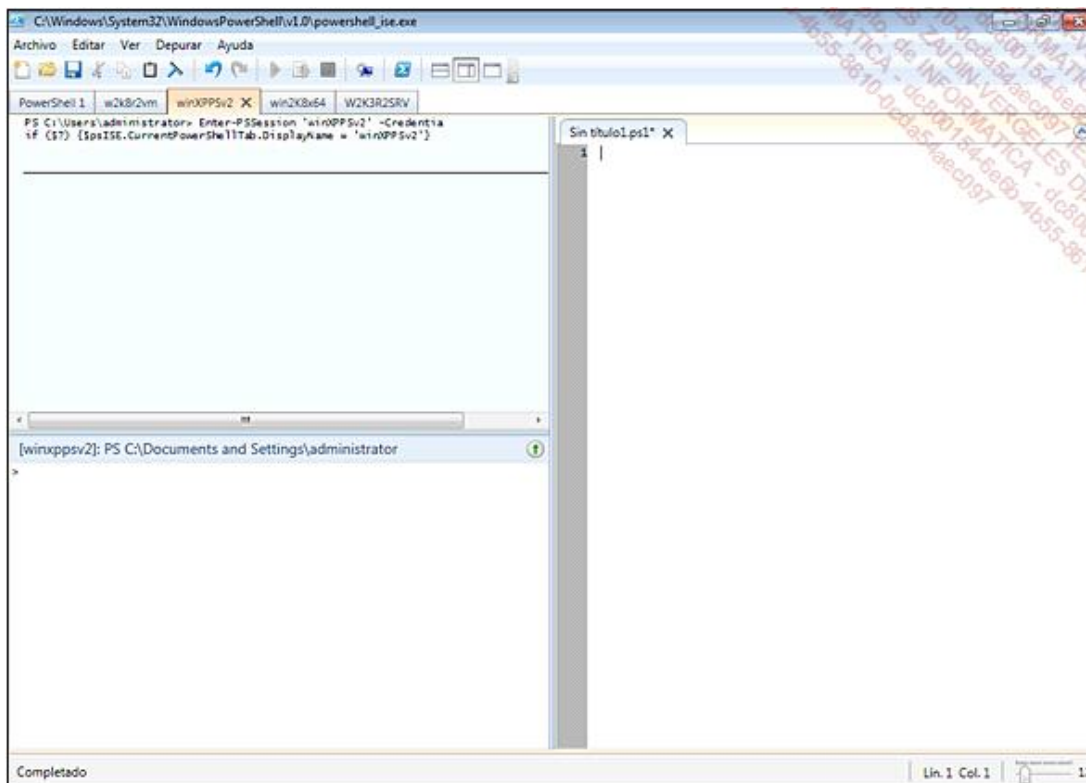
Parámetro	Descripción
<code>AllowRedirection &lt;Switch&gt;</code>	Permite el direccionado de esta conexión a un identificador uniforme de recursos URI ( <i>Uniform Resource Identifier</i> ) alternativo.
<code>ApplicationName &lt;String&gt;</code>	Especifica el segmento del nombre de aplicación del identificador URI de la conexión.
<code>Authentication &lt;AuthenticationMechanism&gt;</code>	Especifica el mecanismo que se utiliza para autenticar las credenciales del usuario.
<code>CertificateThumbprint &lt;String&gt;</code>	Especifica el certificado de clave pública digital (X509) de una cuenta de usuario que tiene permiso para realizar esta acción.
<code>ComputerName &lt;String&gt;</code>	Inicia una sesión interactiva con el equipo remoto especificado. Escriba solamente un nombre de equipo. El valor predeterminado es el equipo local.
<code>ConfigurationName &lt;String&gt;</code>	Especifica la configuración de sesión que se usa para la sesión interactiva.
<code>ConnectionURI &lt;Uri[]&gt;</code>	Especifica un identificador uniforme de recursos (URI) que define el extremo de la conexión de la sesión interactiva.
<code>Credential &lt;PSCredential&gt;</code>	Especifica una cuenta de usuario con permiso para realizar esta acción. El valor predeterminado es el usuario actual.
<code>Id &lt;int&gt;</code>	Especifica el identificador de una sesión existente. <code>Enter-PSSession</code> utiliza la sesión especificada para la sesión interactiva. Para obtener el identificador de una sesión, se utiliza el cmdlet <code>Get-PSSession</code> .

InstanceId <Guid>	Especifica el identificador de instancia de una sesión existente. Enter-PSSession utiliza la sesión especificada para la sesión interactiva.
Name <String[]>	Especifica un nombre sencillo de una sesión existente.
Port <Int>	Especifica el puerto de red en el equipo remoto utilizado para este comando. El valor predeterminado es el puerto 80 (puerto HTTP).
Session <PSSession[]>	Especifica la sesión de Windows PowerShell (PSSession) que se ha de utilizar para la sesión interactiva. Este parámetro toma un objeto de sesión. También puede utilizar los parámetros Name, InstanceID o ID para especificar una PSSession.
SessionOption <PSSessionOption>	Establece opciones avanzadas para la sesión. Se debe especificar un objeto SessionOption creado mediante el cmdlet New-PSSessionOption.
UseSSL <Switch>	Utiliza el protocolo Secure Socket Layer sobre HTTPS para establecer la conexión. Por defecto, no se usa SSL.

## b. Powershell ISE (Integrated Scripting Environment)

Esta nueva interfaz comprende una funcionalidad que permite abrir múltiples consolas PowerShell remotas. Esto resulta especialmente práctico.

Los diferentes sesiones remotas se abren cada una en una pestaña distinta; véase la figura siguiente:



*PowerShell ISE puede conectarse a diferentes ordenadores simultáneamente*

Para abrir una consola remota en ISE, hay que ir al menú **Archivo - Nueva ficha de PowerShell en remoto....** Entonces se le pide que se autentique, y posteriormente aparecerá una nueva pestaña. En el ejemplo de la figura superior, tenemos la pestaña **PowerShell 1** que corresponde a la primera instancia de la consola abierta localmente, después

tenemos una pestaña por cada máquina remota.

Este ejemplo, iniciado a partir de un ordenador con Windows 7, muestra una consola PowerShell abierta en remoto en cuatro máquinas con diferente sistema operativo: Windows Server 2008 R2, Windows XP, Windows Server 2008 y Windows Server 2003 R2, respectivamente.

## 8. Importación de comandos en remoto

Supongamos que en la máquina remota tenemos Snap-Ins o módulos que quisiéramos utilizar en nuestra máquina local. Hasta aquí, podemos decir que tendríamos que utilizar el comando visto anteriormente `Enter-PSSession`. Ciertamente esto funciona... Sin embargo, hemos visto que `Enter-PSSession` no carga los perfiles PowerShell que podrían encontrarse en la máquina remota.

¿Qué haremos por lo tanto si tenemos múltiples funciones indispensables en la máquina remota y deseamos utilizarlas en nuestra máquina local? En este caso, lo más sencillo será importar en la sesión PowerShell actual los comandos de la máquina remota. De este modo nos beneficiaremos de nuestro perfil y del conjunto de comandos ampliado de la máquina remota.

Es por esto que el commandlet `Import-PSSession` tiene su razón de ser. Pero debe haber otros muchos escenarios de uso en los que no hemos pensado...

El comando `Import-PSSession` posee los parámetros siguientes:

Parámetro	Descripción
<code>AllowClobber &lt;Switch&gt;</code>	Importa los comandos especificados, incluso si tienen el mismo nombre que los comandos de la sesión actual.
<code>ArgumentList &lt;Object[]&gt;</code>	Importa la variante del comando que resulta de usar los argumentos especificados (valores de parámetro).
<code>CommandName &lt;String[]&gt;</code>	Importa sólo los comandos con los nombres o patrones de nombre especificados. Se permite el uso de caracteres comodín.
<code>CommandType &lt;CommandTypes&gt;</code>	Importa únicamente los tipos de objetos de comando especificados. El valor predeterminado es <code>Cmdlet</code> .
<code>FormatTypeName &lt;String[]&gt;</code>	Importa las instrucciones de formato de los tipos de Microsoft .NET Framework especificados.
<code>Module &lt;String[]&gt;</code>	Importa sólo los comandos de los complementos (Snap-Ins) y módulos especificados de Windows PowerShell.
<code>Prefix &lt;String&gt;</code>	Agrega el prefijo especificado a los sustantivos en los nombres de los comandos importados. Utilice este parámetro para evitar conflictos entre nombres que podrían producirse cuando diferentes comandos de la sesión tienen el mismo nombre.
<code>Session &lt;PSSession&gt;</code>	Especifica la <code>PSSession</code> desde la que se importan los cmdlets.

Tomemos, por ejemplo, el caso de un servidor controlador de dominio Windows 2008 R2 que tenga instalado el rol «Active Directory Domain Services». Este último tiene por tanto la posibilidad de poseer el módulo `ActiveDirectory` (véase el capítulo Módulo Active Directory de Windows Server 2008 R2). Este módulo brinda numerosos commandlets para la administración de los objetos usuarios, máquinas, grupos, OU, etc. Queremos por tanto importar este módulo a la sesión actual, o en otras palabras, a nuestra consola.

La primera cosa a realizar consistirá en establecer una sesión con una máquina remota, del modo siguiente:

```
PS > $s = New-PSSession -ComputerName W2K8R2VM
```

Ahora, es como si hubiésemos abierto una consola PowerShell en la máquina remota. Será preciso, pues, en esta última, cargar el módulo Active Directory; ya que por el momento únicamente posee los comandos básicos.

```
PS > Invoke-Command -Session $s -ScriptBlock {Import-Module ActiveDirectory}
```

Se puede observar que este comando tarda algunos segundos en ejecutarse y que durante este lapso de tiempo se muestra una barra de progresión en nuestra consola.

Ahora que el módulo está cargado en la máquina remota, vamos a poder importar los comandos que la componen:

```
PS > Import-PSSession -Session $s -Module ActiveDirectory
```

ModuleType	Name	ExportedCommands
Script	tmp_3325e847-774e-4891...	{Set-ADOrganizationalUnit, Get-...

¡Hecho! Para verificar que los comandos están disponibles en la sesión actual, si conocemos sus nombres, podemos realizar el comando siguiente:

```
PS > Get-Command *-AD*
```

CommandType	Name	Definition
Function	Add-ADComputerServiceAccount	...
Function	Add-ADDomainControllerPasswordReplicationPolicy	...
Function	Add-ADFineGrainedPasswordPolicySubject	...
Function	Add-ADGroupMember	...
Function	Add-ADPrincipalGroupMembership	...
...		

O este otro comando si no conocemos los comandos presentes en el módulo:

```
PS > Get-Command -Module tmp*
```

Este último comando lista los comandos del módulo llamado «tmp\*». Cuando importamos un módulo con `Import-PSSession`, se crea localmente un módulo temporal que contiene los comandos importados. Con este comando, pedimos a `Get-Command` que nos proporcione la lista de comandos del módulo temporal cuyo nombre empiece por «tmp».

➤ En la importación de comandos, estos últimos se transforman en funciones. Además, tenga en cuenta que, incluso si los comandos importados parecen locales por su comportamiento, se ejecutan en la máquina remota. Será preciso por tanto vigilar de mantener la sesión PSSession abierta durante el tiempo necesario.

➤ Hemos tomado como ejemplo el módulo Active Directory, pero sepa que hubiese sido lo mismo con el módulo Exchange 2010, por ejemplo.