Fundamentos de PowerShell

¿ESTÁS INTERESADO EN ALGÚN CURSO O EN ALGO QUE LEES POR AQUÍ?

Pregúntame por LinkedIn y te doy más información sobre profesores y precios, nosotros no enseñamos lo típico que puedes leer en cualquier sitio… Deja de hacer cursos vacuos, confía en nosotros, vas a aprender cosas interesantes de verdad.

https://www.linkedin.com/in/jesuspowershell/

Módulo 1: Iniciando con PowerShell

Lección 1: Resumen y antecedentes de Windows PowerShell y PowerShell Core

Windows PowerShell:

- Windows PowerShell, a menudo denominado simplemente «PowerShell,» fue desarrollado por Microsoft y lanzado por primera vez en 2006. Su objetivo principal era proporcionar a los administradores de sistemas una interfaz de línea de comandos más eficiente y versátil para la administración de sistemas Windows.
- Se basa en el marco de .NET y utiliza cmdlets (comandos de una sola función) para realizar tareas específicas. La estructura de un cmdlet consta de un verbo y un sustantivo, por ejemplo, «Get-Service» para obtener información sobre los servicios del sistema.

- Windows PowerShell se convirtió rápidamente en una herramienta esencial para los profesionales de TI debido a su capacidad para automatizar tareas, gestionar servicios, configurar servidores, entre otros.
- A lo largo de su evolución, se introdujeron múltiples versiones, con la última versión importante siendo PowerShell 5.1 en ese momento.

PowerShell Core:

- PowerShell Core, en contraste, es una versión más reciente que fue anunciada en 2016. A diferencia de Windows PowerShell, es multiplataforma y puede ejecutarse en sistemas no Windows, como macOS y Linux.
- Esta decisión de hacer PowerShell multiplataforma amplió su alcance y lo convirtió en una herramienta esencial para la administración de sistemas en diversos entornos.
- PowerShell Core se basa en el marco .NET Core, lo que le permite ser más ligero y ágil, y está diseñado para admitir el desarrollo y la automatización en entornos heterogéneos.

Importancia y Aplicaciones:

- Ambas versiones de PowerShell son fundamentales para la administración de sistemas modernos. Pueden automatizar tareas repetitivas, gestionar configuraciones de servidores, interactuar con servicios en la nube, y mucho más.
- En la actualidad, PowerShell es una habilidad crucial para profesionales de TI y administradores de sistemas, ya que les permite ser más eficientes y efectivos en la gestión de entornos informáticos complejos.
- PowerShell se ha convertido en una herramienta estándar en la administración de servidores Windows y está ganando popularidad en otros sistemas operativos debido

a la versatilidad de PowerShell Core.

Lección 2: Comprender la sintaxis de comandos

Estructura Básica de un Comando:

- En PowerShell, los comandos siguen una estructura básica compuesta por un verbo y un sustantivo. Esta estructura se utiliza para identificar la acción que se va a realizar y el objetivo al que se aplicará. Por ejemplo, el comando «Get-Process» utiliza el verbo «Get» para indicar que se va a obtener algo y el sustantivo «Process» especifica lo que se va a obtener, en este caso, información sobre procesos en el sistema.

Cmdlets y Funciones:

- En PowerShell, existen dos tipos principales de comandos: cmdlets y funciones. Los cmdlets son comandos incorporados que realizan tareas específicas, como obtener información del sistema o administrar recursos. Las funciones, por otro lado, son comandos personalizados creados por el usuario para realizar tareas específicas.

Parámetros y Argumentos:

- Los comandos en PowerShell a menudo requieren parámetros para modificar su comportamiento. Los parámetros son opciones que se agregan al comando para personalizar su acción. Por ejemplo, al usar el cmdlet «Get-Service,» puedes proporcionar el parámetro «-Name» seguido del nombre de un servicio específico que deseas obtener.
- Los argumentos son los valores que se pasan a los

parámetros. Por ejemplo, en «Get-Service -Name ‘Spooler’,» ‘Spooler’ es el argumento que se pasa al parámetro «-Name.»

Uso de Comodines:

- En PowerShell, puedes utilizar comodines para realizar búsquedas de patrones en comandos. El asterisco (*) se usa como comodín para representar cualquier número de caracteres, mientras que el signo de interrogación (?) se utiliza para representar un único carácter. Esto es útil para buscar elementos que coincidan con un patrón específico.

Lección 3: Encontrar y ejecutar comandos

Utilizando Get-Command:

- Una de las formas más efectivas de encontrar comandos en PowerShell es utilizando el cmdlet «Get-Command». Este cmdlet permite buscar comandos disponibles en el entorno de PowerShell. Puedes buscar por nombre, verbo, sustantivo o cualquier otro criterio que te ayude a identificar los comandos adecuados para la tarea que deseas realizar.

El Operador de Tubería (Pipeline):

- La tubería, representada por el símbolo «|» en PowerShell, es una característica poderosa que te permite encadenar comandos y procesar datos de manera eficiente. Puedes tomar la salida de un comando y usarla como entrada para otro, lo que te permite realizar tareas más complejas y personalizadas. Por ejemplo, puedes obtener una lista de procesos con «Get-Process» y

luego filtrarlos para encontrar procesos específicos.

Ejecución de Comandos:

- Una vez que hayas encontrado el comando adecuado, es importante saber cómo ejecutarlo. En PowerShell, la sintaxis básica para ejecutar un comando es simplemente escribir el nombre del comando seguido de cualquier parámetro necesario. Por ejemplo, para obtener una lista de servicios, puedes usar «Get-Service».

Redirección de Salida y Manejo de Errores:

- PowerShell ofrece diversas formas de redirigir la salida de los comandos. Por ejemplo, puedes utilizar el operador «>>» para redirigir la salida a un archivo en lugar de mostrarla en la pantalla. Además, es importante comprender cómo manejar los errores que puedan ocurrir al ejecutar comandos. PowerShell proporciona mecanismos para capturar y gestionar errores de manera efectiva.

Módulo 2: Cmdlets para administración

Lección 1: Cmdlets de administración de Active Directory

Active Directory y su Importancia:

- Active Directory es un servicio de directorio de Microsoft utilizado para gestionar y organizar recursos en una red, como usuarios, equipos, impresoras y otros objetos. Juega un papel fundamental en entornos empresariales y es necesario para la gestión de

seguridad, políticas de acceso y la administración de recursos compartidos.

Cmdlets de Active Directory:

- PowerShell proporciona un conjunto completo de cmdlets para administrar Active Directory. Estos cmdlets permiten realizar una amplia variedad de tareas, desde la creación y gestión de usuarios hasta la configuración de políticas de seguridad y la administración de grupos.

Ejemplos de Cmdlets de Administración de Active Directory:

A continuación, se presentan algunos ejemplos con detalle:

1. **Get-ADUser**:
   - Este cmdlet se utiliza para obtener información sobre usuarios en Active Directory. Puedes especificar diversos parámetros para buscar usuarios en función de criterios como el nombre, el apellido, el nombre de inicio de sesión, etc.
   - Ejemplo: Get-ADUser -Filter "Name -like 'John Smith'" buscará y mostrará usuarios cuyos nombres coincidan con «John Smith.»

2. **New-ADUser**:
   - Permite crear nuevos usuarios en Active Directory. Debes proporcionar información como el nombre de usuario, la contraseña y otros detalles obligatorios.
   - Ejemplo: New-ADUser -Name 'Alice' -SamAccountName 'alice.smith' -UserPrincipalName 'alice@contoso.com' -AccountPassword (ConvertTo-SecureString -AsPlainText 'P@ssw0rd' -Force) -Enabled $true crea un nuevo usuario llamado ‘Alice’ con nombre de inicio de sesión

‘alice.smith’ y contraseña ‘P@ssw0rd.’

3. **Set-ADUser**:
    - Utilizado para modificar las propiedades de un usuario existente en Active Directory.
    - Ejemplo: Set-ADUser -Identity 'alice.smith' -Office 'HR Department' asigna la ubicación de la oficina «HR Department» al usuario ‘alice.smith.’

4. **Get-ADGroup**:
    - Este cmdlet se emplea para obtener información sobre grupos de seguridad en Active Directory. Puedes filtrar grupos según su nombre, miembros, etc.
    - Ejemplo: Get-ADGroup -Filter "Name -like 'Sales*'" mostrará grupos cuyos nombres comiencen con «Sales.»

5. **New-ADGroup**:
    - Permite la creación de nuevos grupos en Active Directory. Debes especificar el nombre del grupo, el ámbito (seguridad o distribución) y otros detalles.
    - Ejemplo: New-ADGroup -Name 'ITSupport' -GroupScope 'Global' -GroupCategory 'Security' crea un nuevo grupo de seguridad llamado ‘ITSupport’ con ámbito global.

Estos ejemplos representan solo una muestra de los cmdlets de administración de Active Directory en PowerShell. Los cmdlets de Active Directory permiten a los administradores de sistemas realizar una variedad de tareas, desde la creación de usuarios y grupos hasta la gestión de permisos y políticas de seguridad. Es fundamental comprender cómo utilizar estos

cmdlets de manera efectiva para administrar un entorno de Active Directory de manera eficiente y segura.

Lección 2: Cmdlets de configuración de red

Cmdlets para la Configuración de Red:

Los cmdlets para la configuración de red en PowerShell son fundamentales para gestionar adaptadores de red, configurar direcciones IP, administrar puertas de enlace y diagnosticar problemas de red en sistemas Windows. A continuación, se explican en detalle algunos de los cmdlets clave relacionados con la configuración de red:

1. **Get-NetAdapter**:
    - Este cmdlet se utiliza para obtener información sobre los adaptadores de red en el sistema. Proporciona detalles como el nombre del adaptador, el estado, la velocidad, la dirección MAC y la información de conexión.
    - Ejemplo: Get-NetAdapter mostrará una lista de todos los adaptadores de red en el sistema, lo que es útil para identificar qué adaptadores están disponibles.

2. **Set-NetIPInterface**:
    - Permite configurar propiedades de la interfaz de red, como la dirección IP, la máscara de subred y la puerta de enlace predeterminada. Este cmdlet es crucial para la configuración de la red.
    - Ejemplo: Set-NetIPInterface -InterfaceAlias 'Ethernet' -Dhcp Enabled habilitará DHCP en la interfaz de red llamada ‘Ethernet’ para obtener automáticamente una dirección IP.

3. **Test-NetConnection**:
    - Utilizado para realizar pruebas de conectividad a otros dispositivos en la red. Puedes especificar el destino, el puerto y otras opciones para verificar la disponibilidad de un recurso de red.
    - Ejemplo: Test-NetConnection -ComputerName 'server.contoso.com' -Port 80 verificará si el servidor ‘server.contoso.com’ está escuchando en el puerto 80.

4. **Get-NetRoute**:
    - Proporciona información sobre la tabla de enrutamiento del sistema. Puedes ver las rutas IP definidas, las puertas de enlace y las métricas.
    - Ejemplo: Get-NetRoute mostrará una lista de rutas de red configuradas en el sistema, lo que es útil para comprender cómo se enrutan los datos.

Estos cmdlets son esenciales para la configuración y administración de adaptadores de red, direcciones IP y la realización de pruebas de conectividad en sistemas Windows. Los administradores de sistemas los utilizan para garantizar que la red funcione correctamente y para diagnosticar problemas de conectividad. Entender cómo funcionan estos cmdlets y saber cuándo y cómo utilizarlos es crucial en la administración de redes y sistemas.

Configuración de Direcciones IP:

La configuración de direcciones IP es fundamental en la administración de redes, ya que permite que los dispositivos se comuniquen de manera efectiva en una red. En PowerShell, puedes utilizar cmdlets específicos para configurar direcciones IP en adaptadores de red. Aquí tienes una explicación más detallada:

- **Get-NetIPAddress**: Este cmdlet te permite ver las direcciones IP configuradas en el sistema. Puedes obtener información sobre la dirección IP, la máscara de subred, la interfaz de red y otros detalles.
    - Ejemplo: Get-NetIPAddress mostrará una lista de todas las direcciones IP configuradas en el sistema, lo que es útil para conocer las asignaciones actuales.

- **New-NetIPAddress**: Con este cmdlet, puedes asignar una nueva dirección IP a una interfaz de red. Debes especificar la dirección IP, la máscara de subred y la interfaz de destino.
    - Ejemplo: New-NetIPAddress -InterfaceAlias 'Ethernet' -IPAddress '192.168.1.100' -PrefixLength 24 asignará la dirección IP '192.168.1.100' a la interfaz 'Ethernet' con una máscara de subred de 255.255.255.0 (prefix length 24).

- **Set-NetIPAddress**: Este cmdlet se utiliza para modificar una dirección IP existente en una interfaz de red.
    - Ejemplo: Set-NetIPAddress -InterfaceAlias 'Wi-Fi' -IPAddress '192.168.1.101' cambiará la dirección IP de la interfaz 'Wi-Fi' a '192.168.1.101'.

Diagnóstico de Red:

El diagnóstico de red es esencial para identificar y solucionar problemas de conectividad. PowerShell proporciona cmdlets que te ayudan a realizar estas tareas. Aquí se explican algunos de ellos:

- **Test-NetConnection**: Este cmdlet se utiliza para probar

la conectividad a un servidor o recurso en la red. Puedes especificar el nombre del servidor y el puerto que deseas probar.
    - Ejemplo: Test-NetConnection -ComputerName 'servidor.contoso.com' -Port 80 verificará si el servidor ‘servidor.contoso.com’ responde en el puerto 80.

- **Test-Connection**: Este cmdlet permite realizar pruebas de ping a dispositivos en la red para verificar la conectividad. Puedes especificar el nombre del dispositivo y la cantidad de intentos.
    - Ejemplo: Test-Connection -ComputerName 'router.contoso.com' -Count 4 realizará 4 pruebas de ping al router ‘router.contoso.com’ y mostrará los resultados.

Políticas de Firewall:

Las políticas de firewall son esenciales para la seguridad de la red. PowerShell permite administrar las reglas de firewall en sistemas Windows. Aquí tienes información detallada sobre cómo hacerlo:

- **Get-NetFirewallRule**: Utiliza este cmdlet para ver una lista de todas las reglas de firewall configuradas en el sistema. Puedes obtener detalles sobre las reglas existentes, como los puertos permitidos o los programas autorizados.
    - Ejemplo: Get-NetFirewallRule mostrará una lista de todas las reglas de firewall en el sistema, lo que es útil para comprender la configuración actual.

- **New-NetFirewallRule**: Este cmdlet se utiliza para crear

nuevas reglas de firewall. Debes especificar detalles como el nombre de la regla, el puerto y si permites o bloqueas el tráfico.
    - Ejemplo: New-NetFirewallRule -Name 'AllowHTTP' -DisplayName 'Allow HTTP Traffic' -Enabled True -Action Allow -Protocol TCP -Direction Inbound -LocalPort 80 creará una regla que permite el tráfico HTTP entrante en el puerto 80.

- **Set-NetFirewallRule**: Permite modificar una regla de firewall existente, lo que es útil para ajustar la configuración de seguridad.
    - Ejemplo: Set-NetFirewallRule -Name 'AllowHTTP' -Enabled False deshabilitará la regla 'AllowHTTP' para bloquear el tráfico HTTP entrante.

Lección 3: Otros cmdlets de administración de servidores

Cmdlets para la Administración de Servidores:

En esta unidad, exploraremos los cmdlets de PowerShell que son fundamentales para la administración de servidores en un entorno de Windows. A través de estos cmdlets, los administradores de sistemas pueden gestionar los servicios, aplicaciones y configuraciones del servidor de manera eficiente. Veamos en detalle cómo se utilizan algunos de los cmdlets clave:

- **Get-Service**: Este cmdlet permite a los administradores obtener información sobre los servicios que se están ejecutando en el servidor. No solo proporciona una lista de servicios, sino que también muestra su estado actual, lo que es fundamental para la supervisión y el

mantenimiento del servidor.
    - Ejemplo: Get-Service mostrará una lista de servicios en el servidor junto con su estado actual, lo que es útil para identificar problemas.

- **Restart-Service**: Cuando se requiere reiniciar un servicio específico, el cmdlet «Restart-Service» se convierte en una herramienta valiosa. Permite reiniciar un servicio sin necesidad de interrumpir su funcionamiento, lo que es esencial para evitar interrupciones en la operatividad del servidor.
    - Ejemplo: Restart-Service -Name 'Spooler' reiniciará el servicio de impresión (‘Spooler’) sin afectar a otros servicios en ejecución.

- **Install-WindowsFeature**: Este cmdlet se utiliza para instalar roles y características en un servidor Windows. Esto es crucial para personalizar la funcionalidad del servidor según las necesidades específicas de la organización. La capacidad de añadir o quitar funcionalidades es fundamental para adaptar el servidor a tareas específicas.
    - Ejemplo: Install-WindowsFeature -Name 'Web-Server' instalará el rol de servidor web en el servidor, lo que es esencial para alojar sitios web.

- **Get-Process**: Para supervisar y gestionar procesos en el servidor, el cmdlet «Get-Process» proporciona información detallada sobre los procesos en ejecución. Esto incluye detalles como el nombre del proceso, el ID, el uso de CPU y memoria, lo que es esencial para controlar y optimizar el rendimiento del servidor.
    - Ejemplo: Get-Process mostrará una lista de todos los procesos en ejecución, lo que es útil para identificar aquellos que pueden estar consumiendo recursos en exceso.

Administración de Roles y Características:

Esta sección se centra en cómo utilizar cmdlets para administrar roles y características en servidores Windows. A través de estos cmdlets, los administradores pueden personalizar la funcionalidad del servidor según las necesidades específicas de la organización. Algunos puntos clave incluyen:

- **Instalación de Roles y Características**: Los cmdlets permiten instalar roles y características adicionales en el servidor según sea necesario. Esto garantiza que el servidor cumpla con sus funciones específicas, como servir como controlador de dominio, servidor web, servidor de base de datos, entre otros.

Monitorización del Sistema:

La monitorización del sistema es esencial para garantizar que el servidor funcione sin problemas y cumpla con los niveles de rendimiento requeridos. En esta sección, se explorarán cmdlets que ayudan a recopilar datos de rendimiento y supervisar el sistema:

- **Supervisión de Rendimiento**: Los cmdlets permiten obtener información sobre el uso de recursos del servidor, como la carga de CPU, el uso de memoria, el almacenamiento y otros parámetros de rendimiento. Esto es crucial para identificar cuellos de botella y optimizar el rendimiento del servidor.

Administración de Procesos:

La administración de procesos es esencial para asegurarse de que el servidor funcione sin problemas y de manera eficiente. En esta sección, se explicará cómo utilizar cmdlets para administrar procesos:

- **Gestión de Procesos**: Los cmdlets permiten a los administradores ver, detener y gestionar procesos en el servidor. Esto es fundamental para mantener el servidor funcionando sin problemas y evitar problemas de rendimiento.

Resolución de Problemas:

Esta parte de la unidad se centrará en cómo los cmdlets de PowerShell pueden utilizarse para diagnosticar y solucionar problemas comunes en servidores:

- **Diagnóstico de Problemas**: Los cmdlets son útiles para identificar y solucionar problemas de servidor comunes, como cuellos de botella de rendimiento, problemas de red y errores de aplicación. Los administradores aprenderán a utilizar estas herramientas para garantizar la estabilidad y la disponibilidad del servidor.

Módulo 3: Scripting Básico

Lección 1: Introducción a la creación de scripts

1. **Importancia de la Creación de Scripts:**
    - La creación de scripts en PowerShell es fundamental en la administración de sistemas, ya que permite automatizar tareas, ahorrar tiempo y reducir errores. Los scripts son valiosos para gestionar servidores, realizar tareas repetitivas y mantener la coherencia en la configuración del sistema.

2. **Introducción a PowerShell:**

- PowerShell es un lenguaje de scripting y una interfaz de línea de comandos desarrollada por Microsoft. Combina la potencia de un lenguaje de programación con la accesibilidad de los comandos de consola. Se utiliza para administrar sistemas Windows y realizar una variedad de tareas, desde la gestión de usuarios hasta la configuración de servidores.

3. **Sintaxis Básica de PowerShell:**
    - La sintaxis en PowerShell se basa en el uso de cmdlets, que son comandos predefinidos para realizar acciones específicas. Los cmdlets siguen una estructura verbo-sustantivo, y los parámetros se utilizan para personalizar su comportamiento. Además, PowerShell maneja objetos en lugar de texto simple, lo que facilita el procesamiento de datos.

4. **Prácticas Recomendadas para la Creación de Scripts:**
    - Al escribir scripts en PowerShell, es importante seguir prácticas recomendadas, como proporcionar comentarios descriptivos para documentar el código, utilizar nombres de variables significativos, manejar errores de manera adecuada y estructurar los scripts de manera coherente. Estas prácticas mejoran la legibilidad y la mantenibilidad de los scripts.

Lección 2: Construcción de scripts

1. **Definición de Objetivos y Requisitos del Script:**
    - Antes de comenzar a escribir un script, es fundamental definir claramente los objetivos que

se desean lograr y los requisitos del script. Esto implica comprender la tarea que se automatizará y los resultados esperados.

2. **Planificación del Flujo de Trabajo:**
    - En esta etapa, se planifica el flujo de trabajo del script. Se determina la secuencia de acciones que el script llevará a cabo para lograr los objetivos definidos. Esto implica identificar las tareas intermedias y las decisiones lógicas que el script debe tomar.

3. **Selección de Cmdlets y Parámetros:**
    - Con el flujo de trabajo definido, se seleccionan los cmdlets apropiados que realizarán las acciones necesarias. Se eligen los parámetros adecuados para personalizar el comportamiento de los cmdlets según los requisitos.

4. **Escritura del Código:**
    - En esta etapa, se procede a escribir el código del script en PowerShell. Se utilizan los cmdlets seleccionados y se configuran los parámetros. Además, se aplican las mejores prácticas de escritura de scripts, como el uso de comentarios, la asignación de nombres de variables descriptivos y la estructuración coherente del código.

5. **Depuración del Script:**
    - La depuración es un proceso crucial en la construcción de scripts. Los estudiantes aprenderán a utilizar herramientas de depuración de PowerShell para identificar y corregir errores en el código. Se abordarán conceptos como puntos de interrupción, seguimiento y visualización de

variables.

6. **Pruebas del Script:**
   - Antes de implementar un script en un entorno de producción, es esencial realizar pruebas exhaustivas. Los estudiantes aprenderán a crear casos de prueba, ejecutar el script en un entorno de prueba y verificar que produce los resultados esperados.

7. **Optimización del Rendimiento:**
   - La optimización del rendimiento es un aspecto crítico de la construcción de scripts. Se discutirán estrategias para mejorar la eficiencia del script, como la reducción de bucles innecesarios, la gestión de recursos y la implementación de técnicas de almacenamiento en caché.

8. **Documentación del Script:**
   - La documentación es esencial para garantizar la comprensión y el mantenimiento a largo plazo del script. Los estudiantes aprenderán a crear documentación clara que incluya una descripción de la función del script, ejemplos de uso y detalles sobre los parámetros y las salidas.

9. **Gestión de Versiones y Almacenamiento:**
   - Se abordará la gestión de versiones y el almacenamiento seguro de scripts. Los estudiantes aprenderán a utilizar sistemas de control de versiones y a proteger sus scripts mediante el uso de medidas de seguridad adecuadas.