Sistemas Operativos en Red

# UD 04. Introducción a PowerShell - Parte 3




Autor: Sergi García, Gloria Muñoz

Actualizado Abril 2024

## Licencia



## Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

# ÍNDICE

# UNIDAD 04. INTRODUCCIÓN A POWERSHELL - PARTE 3

## 1. MANIPULACIÓN DE ARCHIVOS CON POWERSHELL

Vamos a ver ejemplos que cubren una amplia gama de operaciones comunes relacionadas con archivos en PowerShell, desde la creación y manipulación de contenido hasta la gestión de archivos en el sistema de archivos

### 1.1 Ejemplos de manipulación de Archivos

**Leer el Contenido de un Archivo**

```
$contenido = Get-Content "C:\temp\archivo.txt"
Write-Host "Contenido del archivo: $contenido"
```

**Explicación:**

Este comando utiliza Get-Content para leer todo el contenido de un archivo especificado y lo almacena en la variable $contenido. Luego, imprime ese contenido en la consola.

**Escribir Contenido en un Archivo**

```
Set-Content -Path "C:\temp\nuevoArchivo.txt" -Value "Hola, mundo"
```

**Explicación:**

Set-Content escribe el texto "Hola, mundo" en el archivo especificado. Si el archivo no existe, lo crea; si existe, sobrescribe su contenido con el nuevo texto.

**Agregar contenido al final de un archivo**

```
Add-Content -Path "C:\temp\nuevoArchivo.txt" -Value "Adiós, mundo"
```

**Explicación:**

Add-Content añade la línea "Adiós, mundo" al final del archivo especificado, preservando el contenido existente.

**Copiar un archivo**

```
Copy-Item "C:\temp\archivo.txt" -Destination "C:\temp\archivoCopia.txt"
```

**Explicación:**

Este comando copia un archivo de una ubicación a otra. Si el archivo de destino ya existe, será sobrescrito sin advertencia.

**Mover un archivo**

```
Move-Item "C:\temp\archivoCopia.txt" -Destination
"C:\temp\nuevaUbicacion\archivoCopia.txt"
```

**Explicación**:

Move-Item mueve un archivo desde una ubicación a otra. Es útil para reorganizar archivos o para limpieza de datos.

**Eliminar un Archivo**

```
Remove-Item "C:\temp\nuevaUbicacion\archivoCopia.txt"
```

**Explicación:**

Elimina el archivo especificado de la ubicación dada. Si el archivo no existe, PowerShell puede mostrar un error.

**Crear un archivo temporal**

```
$tempFile = New-TemporaryFile
Write-Host "Archivo temporal creado: $($tempFile.FullName)"
```

**Explicación**:

Crea un archivo temporal en la ubicación temporal predeterminada del sistema y muestra su ruta completa.

**Listar Todos los Archivos en un Directorio**

```
$archivos = Get-ChildItem "C:\temp" -File
foreach ($archivo in $archivos) {
    Write-Host "Archivo encontrado: $($archivo.Name)"
}
```

**Explicación:**

Get-ChildItem se utiliza para obtener todos los archivos de un directorio especificado y los enumera uno por uno.

**Buscar archivos por extensión**

```
$archivosTxt = Get-ChildItem "C:\temp" -Filter "*.txt"
foreach ($archivo in $archivosTxt) {
    Write-Host "Archivo de texto: $($archivo.FullName)"
}
```

**Explicación:**

Encuentra y lista todos los archivos con extensión .txt en un directorio específico, mostrando su ruta completa.

**Leer las primeras líneas de un archivo**

```
$lineas = Get-Content "C:\temp\archivo.txt" -TotalCount 5
foreach ($linea in $lineas) {
    Write-Host "Línea: $linea"
}
```

**Explicación:**

Lee y muestra las primeras cinco líneas de un archivo, lo cual es útil para previsualizar el contenido de archivos grandes.

**Escribir en un Archivo Línea por Línea**

```
$lineas = "Primera línea", "Segunda línea", "Tercera línea"
$lineas | Set-Content "C:\temp\lineas.txt"
```

**Explicación:**

Escribe varias líneas en un archivo, una por una, sobrescribiendo el contenido existente.

**Verificar si un archivo existe**

```
$existe = Test-Path "C:\temp\archivo.txt"
if ($existe) {
    Write-Host "El archivo existe."
} else {
    Write-Host "El archivo no existe."
}
```

**Explicación:**

Utiliza Test-Path para verificar la existencia de un archivo y muestra un mensaje apropiado.

**Cambiar el nombre de un archivo**

```
Rename-Item "C:\temp\lineas.txt" -NewName "nuevoNombre.txt"
```

**Explicación:**

Cambia el nombre de un archivo existente a un nuevo nombre especificado.

**Leer y mostrar un archivo de configuración (CSV)**

```
$datos = Import-Csv "C:\temp\config.csv"
foreach ($fila in $datos) {
    Write-Host "Nombre: $($fila.Nombre), Valor: $($fila.Valor)"
}
```

**Explicación**:

Lee un archivo CSV y muestra cada fila, proporcionando una forma de ver datos estructurados de manera clara y organizada.

Excelente, vamos a explorar cómo trabajar con el pipeline en PowerShell, una característica fundamental que facilita la transmisión de datos de salida de un cmdlet a otro cmdlet como entrada. El pipeline permite manipular y transformar datos de manera eficiente y efectiva.

## 2. Pipeline en PowerShell

En PowerShell, el pipeline es representado por el operador |. Permite pasar la salida de un comando directamente como entrada al siguiente comando sin necesidad de almacenarla temporalmente. Es muy útil para filtrar, ordenar y manipular datos de manera fluida.

## 2.1 Ejemplos de uso del pipeline

**Listar y filtrar Procesos**

```
Get-Process | Where-Object {$_.WS -gt 100MB}
```

**Explicación**:

Este comando lista todos los procesos cuyo espacio de trabajo (Working Set - memoria física utilizada) excede los 100MB. Utiliza Get-Process para obtener los procesos y Where-Object para filtrarlos basándose en su consumo de memoria.

**Obtener y seleccionar propiedades específicas de archivos**

```
Get-ChildItem | Select-Object Name, Length
```

**Explicación:**

Este comando lista todos los archivos y directorios en el directorio actual, pero solo selecciona y muestra los nombres y tamaños de los archivos utilizando Select-Object.

**Ordenar Procesos por Uso de Memoria**

```
Get-Process | Sort-Object -Property WS -Descending
```

**Explicación:**

Ordena todos los procesos en función de su uso de memoria (Working Set), de mayor a menor, usando Sort-Object.

**Convertir datos a formato JSON**

```
Get-Process | Where-Object {$_.Id -lt 1000} | ConvertTo-Json
```

**Explicación:**

Filtra los procesos cuyos ID son menores que 1000 y convierte la información de estos procesos en formato JSON, útil para integraciones o almacenamiento de datos.

**Contar archivos en un directorio**

```
Get-ChildItem | Measure-Object
```

**Explicación:**

Utiliza Get-ChildItem para obtener todos los archivos y directorios del directorio actual y Measure-Object para contar cuántos elementos hay.

**Filtrar y contar procesos por su nombre**

```
Get-Process | Where-Object {$_.ProcessName -eq "notepad"} |
Measure-Object
```

**Explicación:**

Cuenta cuántas instancias del proceso "notepad" están corriendo actualmente en el sistema.

**Capturar y mostrar información detallada de eventos**

```
Get-EventLog -LogName System | Where-Object {$_.EntryType -eq "Error"}
| Format-List
```

**Explicación:**

Obtiene registros de eventos del sistema que son de tipo "Error" y los muestra en formato de lista para una revisión detallada.

**Buscar archivos grandes y ordenarlos**

```
Get-ChildItem -Recurse | Where-Object {$_.Length -gt 1MB} | Sort-Object
Length -Descending
```

**Explicación:**

Busca todos los archivos mayores de 1MB en todos los subdirectorios del directorio actual y los ordena de mayor a menor tamaño.

**Exportar la información de configuración a un archivo CSV**

```
Get-ItemProperty -Path
'HKLM:\Software\Microsoft\Windows\CurrentVersion' | Export-Csv -Path
"C:\temp\config.csv"
```

**Explicación:**

Exporta propiedades de una clave específica del registro de Windows a un archivo CSV, útil para análisis o documentación.

**Filtrar, agrupar y resumir información de archivos por extensión**

```
Get-ChildItem -Recurse | Group-Object Extension | Sort-Object Count
-Descending | Select-Object Name, Count
```

**Explicación:**

Agrupa archivos por su extensión y cuenta cuántos archivos hay para cada tipo, mostrando los resultados en orden descendente.

**Copiar archivos modificados recientemente a otra carpeta**

```
Get-ChildItem -Filter "*.txt" | Where-Object {$_.LastWriteTime -gt
(Get-Date).AddDays(-10)} | Copy-Item -Destination "C:\temp\RecentFiles"
```

**Explicación:**

Encuentra archivos .txt que han sido modificados en los últimos 10 días y los copia a otra carpeta.

**Obtener Información detallada del sistema y vonvertirla a HTML**

```
Get-SystemInfo | ConvertTo-Html | Out-File "C:\temp\systemInfo.html"
```

**Explicación:**

Genera un informe detallado del sistema en formato HTML y lo guarda en un archivo. Este

comando podría requerir adaptaciones dependiendo del módulo específico para Get-SystemInfo.

**Filtrar logs de Eventos y contar los de tipo error**

```
Get-EventLog -LogName Application | Where-Object {$_.EntryType -eq
"Error"} | Measure-Object
```

**Explicación:**

Cuenta la cantidad de eventos de tipo "Error" en los logs de aplicaciones, ayudando en diagnósticos y auditorías.

**Generar una Lista de procesos y exportarlos a XML**

```
Get-Process | Export-Clixml -Path "C:\temp\processes.xml"
```

**Explicación:**

Guarda la lista actual de procesos en un archivo XML, lo cual es útil para análisis posteriores o para transferir datos entre sesiones de PowerShell.

**Leer un archivo CSV y filtrar ciertas filas**

```
Import-Csv "C:\temp\data.csv" | Where-Object {$_.Age -gt 30} |
Export-Csv "C:\temp\filteredData.csv"
```

**Explicación:**

Lee un archivo CSV, filtra las filas donde la edad es mayor que 30, y guarda los resultados filtrados en un nuevo archivo CSV.

## 3. Uso de Select-Object en PowerShell

El cmdlet Select-Object permite extraer ciertas propiedades de objetos o un número determinado de objetos de una secuencia. También puede ser utilizado para crear nuevos objetos calculados a partir de las propiedades existentes.

### 3.1 Ejemplos de Uso de Select-Object

**Seleccionar propiedades específicas de procesos**

```
Get-Process | Select-Object Name, ID, CPU
```

**Explicación**:

Este comando extrae solo los nombres, identificadores (ID), y el uso de CPU de los procesos en ejecución. Es útil para obtener rápidamente información crítica sobre los recursos que están consumiendo los procesos.

**Seleccionar los primeros 5 procesos**

```
Get-Process | Select-Object -First 5
```

**Explicación:**

Muestra solo los primeros cinco procesos de la lista de procesos activos. Es una forma efectiva de limitar la salida a un número manejable de elementos para análisis rápido.

**Seleccionar los últimos 3 procesos**

```
Get-Process | Select-Object -Last 3
```

**Explicación**:

Similar al anterior, pero se enfoca en los últimos tres procesos de la lista, lo que puede ser útil para ver los procesos más recientemente lanzados.

**Seleccionar propiedades únicas de un objeto**

```
Get-Process | Select-Object -Unique
```

**Explicación**:

Filtra la lista para eliminar objetos duplicados, basándose en todas las propiedades de los procesos. Es útil para identificar procesos únicos cuando se investigan incidencias de ejecuciones múltiples de una aplicación.

**Seleccionar y calcular una nueva propiedad**

```
Get-Process | Select-Object Name,
@{Name='Threads';Expression={$_.Threads.Count}}
```

**Explicación**:

Añade una propiedad calculada que cuenta los hilos de cada proceso. Este enfoque es valioso para análisis detallados de la utilización de recursos por proceso.

**Seleccionar ciertos archivos y sus propiedades**

```
Get-ChildItem | Select-Object Name, Length, LastAccessTime
```

**Explicación:**

Lista los archivos en el directorio actual, mostrando su nombre, tamaño (Length) y la última vez que fueron accedidos (LastAccessTime). Ideal para auditorías de archivos o limpieza de datos.

**Seleccionar propiedades y renombrarlas**

```
Get-Service | Select-Object
@{Label='ServiceName';Expression={$_.Name}}, Status
```

**Explicación:**

Personaliza la salida al renombrar la propiedad Name a ServiceName y muestra el estado actual del servicio. Esta técnica mejora la claridad de la presentación de datos.

Seleccionar propiedades de un objeto y limitar la cantidad

```
Get-EventLog -LogName Application | Select-Object -First 10 -Property
EntryType, TimeWritten, Message
```

**Explicación:**

Recupera y muestra los primeros 10 registros del log de aplicaciones, enfocándose en el tipo de entrada, el tiempo registrado y el mensaje. Esencial para el diagnóstico de aplicaciones.

**Seleccionar mediante índices específicos**

```
Get-Process | Select-Object -Index 0,5,10
```

**Explicación:**

Selecciona procesos específicos basándose en sus índices en la lista de procesos. Útil para comparaciones directas o para revisar puntos de datos específicos.

**Seleccionar y excluir ciertas propiedades**

```
Get-Process | Select-Object -Property * -ExcludeProperty Handles, VM
```

**Explicación:**

Muestra todas las propiedades de los procesos, excepto los contadores de manejadores (Handles) y el tamaño de la memoria virtual (VM), permitiendo centrarse en otros detalles más relevantes.

**Crear una tabla a partir de propiedades seleccionadas**

```
Get-Process | Select-Object Name, CPU | Format-Table
```

**Explicación**:

Organiza la información del nombre y uso de CPU de los procesos en una tabla, facilitando la comparación y el análisis visual.

**Seleccionar una lista de valores de una sola propiedad**

```
Get-Process | Select-Object -ExpandProperty ProcessName
```

**Explicación:**

Extrae y muestra únicamente los nombres de los procesos, proporcionando una lista limpia y enfocada que es fácil de revisar o documentar.

**Combinar selecciones con condiciones**

```
Get-Process | Where-Object {$_.WorkingSet -gt 100MB} | Select-Object
Name, WorkingSet
```

**Explicación:**

Filtra los procesos que utilizan más de 100MB de memoria de conjunto de trabajo y muestra sus nombres junto con el valor de memoria utilizado. Perfecto para monitorear el uso de recursos.

**Seleccionar propiedades y formatear como Lista**

```
Get-Service | Select-Object Name, Status | Format-List
```

**Explicación:**

Presenta una vista detallada de los servicios y su estado en formato de lista, proporcionando una descripción clara y legible.

**Utilizar Select-Object para depuración**

```
Get-Process | Select-Object -First 1 | Format-List *
```

**Explicación**:

Muestra todas las propiedades del primer proceso de la lista en formato detallado. Es una herramienta excelente para la depuración, permitiendo ver toda la información disponible sobre un proceso.

## 4. USO DE FORMAT-TABLE EN POWERSHELL

Format-Table permite visualizar datos en formato de columnas y filas, haciéndolo ideal para presentar múltiples propiedades de objetos de forma simultánea. Es altamente personalizable y se puede ajustar para mostrar solo ciertas columnas, cambiar el ancho de las columnas, y mucho más.

### 4.1 Ejemplos de uso de Format-Table

**Mostrar procesos en una tabla**

```
Get-Process | Format-Table Name, Id, CPU
```

**Explicación**:

Este comando lista los procesos activos y los presenta en una tabla con columnas para el nombre del proceso, ID y uso de CPU. Es útil para una revisión rápida del consumo de recursos por los procesos.

**Formatear la salida de servicios**

```
Get-Service | Format-Table Name, Status -AutoSize
```

**Explicación**:

Muestra los servicios del sistema en una tabla que incluye el nombre y el estado actual de cada servicio, ajustando automáticamente el tamaño de las columnas para una visualización óptima.

**Mostrar archivos con tamaño formateado**

```
Get-ChildItem | Format-Table Name, @{Name="Size";Expression={$_.Length
/ 1Kb};FormatString="N2"}, LastAccessTime
```

**Explicación**:

Lista los archivos en el directorio actual, mostrando el nombre, tamaño en KB con dos decimales, y la última fecha de acceso en una tabla formateada.

**Usar condicionales en la tabla**

```
Get-Process | Format-Table Name, @{Name='HighCPU';Expression={if
($_.CPU -gt 100) {"High"} else {"Low"}}}
```

**Explicación:**

Muestra los procesos junto con una columna calculada que indica si el uso de CPU es alto ("High") o bajo ("Low"), basado en una condición.

**Visualizar propiedades seleccionadas de eventos**

```
Get-EventLog -LogName Application -Newest 50 | Format-Table
TimeGenerated, EntryType, Message -Wrap
```

**Explicación**:

Muestra los últimos 50 eventos del log de aplicaciones, incluyendo la hora en que se generaron, el tipo de entrada y el mensaje, con ajuste de texto para asegurar que el contenido no se recorte.

**Mostrar configuración de red en una tabla**

```
Get-NetIPAddress | Format-Table IPAddress, InterfaceAlias,
AddressFamily -AutoSize
```

**Explicación**:

Lista las direcciones IP del sistema, mostrando la dirección IP, el alias de la interfaz y la familia de direcciones, ajustando el tamaño automáticamente.

**Tabla con datos de sistema de archivos**

```
Get-Volume | Format-Table DriveLetter, FileSystem,
@{Name="Capacidad";Expression={$_.Size / 1Gb -as [int]}},
@{Name="Libre";Expression={$_.SizeRemaining / 1Gb -as [int]}} -AutoSize
```

**Explicación:**

Muestra información sobre los volúmenes del sistema, incluyendo la letra del disco, el sistema de archivos y el espacio total y libre expresado en GB.

**Listar módulos de PowerShell en una tabla**

```
Get-Module | Format-Table Name, Version
```

**Explicación:**

Presenta una lista de los módulos de PowerShell cargados en la sesión actual, incluyendo sus nombres y versiones.

**Mostrar información de usuarios del sistema**

```
Get-LocalUser | Format-Table Name, Enabled, LastLogon
```

**Explicación:**

Lista los usuarios locales del sistema, mostrando sus nombres, si están habilitados y la última fecha de inicio de sesión.

**Tabla de datos personalizados con colores**

```
Get-Process | Format-Table Name, @{Name="Memory";Expression={$_.WS /
1Mb -as [int]};Label="Memory (MB)"}, CPU -GroupBy Company | Out-String
-Stream | Foreach-Object {$_ -replace "Microsoft Corporation",
"`e[32mMicrosoft Corporation`e[0m"}
```

**Explicación:**

Agrupa los procesos por compañía y muestra los nombres, uso de memoria en MB y CPU. Además, personaliza el color de los textos para "Microsoft Corporation" usando secuencias de escape ANSI.

**Comparar objetos con formato de tabla**

```
Compare-Object (Get-Process -Before) (Get-Process -After) |
Format-Table -Property Name, SideIndicator
```

**Explicación:**

Compara dos conjuntos de procesos (antes y después de un evento) y muestra las diferencias en una tabla indicando qué procesos fueron añadidos o eliminados.

**Visualización de propiedades con múltiples formatos de columna**

```
Get-Process | Format-Table -Property
@{Label="ProcessName";Expression={$_.Name};Width=25},
@{Label="MemoryUsage";Expression={$_.WS / 1Mb};FormatString="N0"}
```

**Explicación:**

Este comando muestra los procesos con columnas personalizadas para el nombre del proceso y el uso de memoria. Las columnas están configuradas específicamente con etiquetas personalizadas, anchos de columna fijos y formatos numéricos que mejoran la legibilidad. La columna "ProcessName" se establece con un ancho fijo para alinear uniformemente los nombres, mientras que "MemoryUsage" muestra la memoria en MB sin decimales. Esta forma de presentación facilita la rápida evaluación visual del uso de memoria entre múltiples procesos, útil en el diagnóstico y monitoreo de recursos del sistema.

**Mostrar espacio en disco de manera estructurada**

```
Get-PSDrive | Where-Object {$_.Provider -like "FileSystem"} |
Format-Table Name, Used, Free -AutoSize
```

**Explicación**:

Este comando filtra y muestra las unidades de disco en el sistema, mostrando el nombre de la unidad, el espacio usado y el espacio libre en un formato tabular que ajusta automáticamente el tamaño de las columnas. Es ideal para obtener una visión rápida del estado del almacenamiento del sistema, ayudando a identificar posibles problemas de espacio.

## 5. EJERCICIOS CON ENUNCIADOS Y SOLUCIONES

**Ejercicio 1: Bucle While con Archivos**

Escribe un script en PowerShell que utilice un bucle while para leer nombres de archivos de un directorio hasta que encuentre un archivo llamado "stop.txt", que no deberá ser procesado.

**Solución:**

```
$stop = $false
while (-not $stop) {
      $files = Get-ChildItem -Path "C:\temp\"
      foreach ($file in $files) {
      if ($file.Name -eq "stop.txt") {
            $stop = $true
            break
      } else {
            Write-Host "Procesando archivo: $($file.Name)"
      }
      }
}
```

**Explicación:**

Este script busca archivos en el directorio especificado y procesa cada uno hasta encontrar un archivo llamado "stop.txt". Utiliza un bucle while para repetir la lectura y un bucle foreach para iterar sobre cada archivo. El script se detiene cuando encuentra el archivo "stop.txt".

**Ejercicio 2: Función con pipeline**

Crea una función en PowerShell que acepte un pipeline de nombres de archivos, lea el contenido de cada archivo y lo imprima en la consola.

**Solución:**

```
function Read-FileContent {
      param(
      [Parameter(ValueFromPipeline=$true)]
      [string]$Path
      )
      process {
      Get-Content $Path | Write-Host
      }
}

# Uso de la función
"file1.txt", "file2.txt" | Read-FileContent
```

**Explicación**:

La función Read-FileContent está diseñada para aceptar rutas de archivo desde el pipeline. Para cada archivo recibido, lee y muestra su contenido. Esto demuestra cómo las funciones pueden ser diseñadas para integrarse en flujos de trabajo de pipeline en PowerShell.

**Ejercicio 3: Bucle for para procesos**

Utiliza un bucle for para listar los nombres y el uso de CPU de los 10 primeros procesos ordenados por uso de memoria, utilizando Format-Table.

**Solución:**

```
$processes = Get-Process | Sort-Object -Property WS -Descending |
Select-Object -First 10
$processes | ForEach-Object {
     Write-Host "$($_.Name) - CPU: $($_.CPU)"
} | Format-Table
```

**Explicación:**

Este script primero ordena los procesos por uso de memoria en orden descendente y selecciona los primeros 10. Luego utiliza un bucle foreach para mostrar el nombre y el uso de CPU de cada proceso, aunque la intención de usar Format-Table directamente en el pipeline parece incorrecta aquí y podría requerir ajustes para funcionar correctamente.

**Ejercicio 4: Seleccionar y formatear datos de red**

Escribe un script que obtenga la dirección IP, máscara de subred y estado del adaptador de red, y lo presente en una tabla formateada.

**Solución:**

```
Get-NetIPAddress | Select-Object IPAddress, PrefixLength,
InterfaceAlias, InterfaceIndex | Format-Table -AutoSize
```

**Explicación:**

Este comando utiliza Get-NetIPAddress para obtener información detallada sobre las direcciones IP del sistema y luego selecciona y formatea específicamente las propiedades relevantes en una tabla, optimizando el tamaño de la columna automáticamente para una mejor legibilidad.

**Ejercicio 5: Función para calcular espacio en disco**

Desarrolla una función que muestre el espacio libre y total de cada unidad de disco en el sistema en GB, utilizando Format-Table.

**Solución:**

```
function Get-DiskSpace {
     Get-Volume | Format-Table DriveLetter,
@{Name="TotalSpace(GB)";Expression={[math]::Round($_.Size / 1GB, 2)}},
@{Name="FreeSpace(GB)";Expression={[math]::Round($_.SizeRemaining /
1GB, 2)}}
}

Get-DiskSpace
```

**Explicación:**

La función Get-DiskSpace recupera información sobre todas las unidades de disco y muestra el espacio total y libre en formato tabular. Utiliza expresiones calculadas dentro de Format-Table para convertir los valores de espacio de bytes a gigabytes y redondearlos para facilitar la visualización.

**Ejercicio 6: Archivos modificados recientemente**

Usa un bucle for para listar todos los archivos en "C:\temp" modificados en los últimos 7 días y muestra sus nombres y fechas de última modificación.

**Solución:**

```
$files = Get-ChildItem -Path "C:\temp\" | Where-Object
{$_.LastWriteTime -gt (Get-Date).AddDays(-7)}
foreach ($file in $files) {
      Write-Host "Archivo: $($file.Name), Modificado:
$($file.LastWriteTime)"
}
```

**Explicación:**

Este script recopila todos los archivos en el directorio especificado que han sido modificados en la última semana. Utiliza un bucle foreach para imprimir el nombre y la fecha de última modificación de cada archivo, proporcionando una manera eficaz de monitorear cambios recientes en un directorio.

**Ejercicio 7: Uso de pipeline para filtrar servicios**

Filtra todos los servicios que están corriendo y los presenta en una tabla con el nombre y el estado del servicio.

**Solución:**

```
Get-Service | Where-Object {$_.Status -eq 'Running'} | Format-Table
Name, Status
```

**Explicación:**

Este comando filtra los servicios activos (en estado 'Running') y los presenta en una tabla con las columnas para el nombre y el estado del servicio. Es útil para administradores de sistemas que necesitan una visión rápida de los servicios actualmente activos.

**Ejercicio 8: Función con parámetros de entrada**

Crea una función que acepte una lista de procesos y muestre su nombre y memoria de trabajo en MB.

**Solución:**

```
function Show-ProcessMemory {
      param([Parameter(ValueFromPipeline=$true)][string[]]$Name)
      process {
      Get-Process -Name $Name | Select-Object Name,
@{Name="MemoryMB";Expression={$_.WS / 1MB -as [int]}}
      }
}
```

```
# Ejemplo de uso
"chrome", "explorer" | Show-ProcessMemory
```

**Explicación:**

La función Show-ProcessMemory toma una lista de nombres de procesos y muestra el nombre y el uso de memoria en MB de cada proceso especificado. Aprovecha la capacidad de aceptar entrada a través de un pipeline, permitiendo flexibilidad en cómo se proporcionan los nombres de los procesos.

**Ejercicio 9: Monitor de recursos en tiempo real**

Desarrolla un script en PowerShell que funcione como un monitor de recursos simplificado. Este script debe ejecutarse en un bucle infinito, actualizando cada 10 segundos. Debe mostrar una tabla con los cinco procesos que más CPU consumen en ese momento, incluyendo detalles como el nombre del proceso, ID, uso de CPU y memoria (en MB). El script debe permitir al usuario salir del bucle presionando la tecla 'q'.

**Solución:**

```
function Monitor-Resource {
      do {
      Clear-Host
      $processes = Get-Process | Sort-Object CPU -Descending |
Select-Object -First 5 Name, Id, CPU,
@{Name="MemoryMB";Expression={$_.WS / 1Mb -as [int]}}
      $processes | Format-Table -AutoSize
      Write-Host "Presione 'q' para salir o espere 10 segundos para la
próxima actualización..."
      $host.UI.RawUI.ReadKey("NoEcho,IncludeKeyUp") | Where-Object
{$_.Character -eq 'q'} | ForEach-Object { return }
      Start-Sleep -Seconds 10
      } while ($true)
}

Monitor-Resource
```

**Explicación:**

Este script utiliza un bucle do-while que se ejecuta indefinidamente hasta que el usuario decide salir presionando 'q'. Dentro del bucle, limpia la consola con Clear-Host, recupera los cinco procesos que más CPU consumen, y los muestra en una tabla con información relevante como el nombre, ID, CPU y memoria en MB. La tabla se actualiza cada 10 segundos, ofreciendo un monitor en tiempo real del consumo de recursos.

**Ejercicio 10: Análisis y reporte de uso de espacio en disco**

Escribe un script en PowerShell que analice el uso del espacio en todas las unidades de disco disponibles en un sistema. Debe generar un reporte en un archivo CSV que incluya la letra de la unidad, el sistema de archivos, el espacio total, el espacio usado y el espacio libre, ambos en GB. Además, debe identificar aquellas unidades con menos del 20% de espacio libre y listarlas en la consola al final de la ejecución.

**Solución:**

```
function Report-DiskUsage {
      $volumes = Get-Volume | Where-Object {$_.DriveType -eq 'Fixed'}
      $report = foreach ($volume in $volumes) {
      [pscustomobject]@{
            DriveLetter = $volume.DriveLetter
            FileSystem = $volume.FileSystem
            TotalSpaceGB = [math]::Round($volume.Size / 1Gb)
            UsedSpaceGB = [math]::Round(($volume.Size -
$volume.SizeRemaining) / 1Gb)
            FreeSpaceGB = [math]::Round($volume.SizeRemaining / 1Gb)
      }
      }
      $report | Export-Csv -Path "C:\temp\disk_usage_report.csv"
-NoTypeInformation
      $lowSpace = $report | Where-Object {$_.FreeSpaceGB /
$_.TotalSpaceGB -lt 0.2}
      Write-Host "Unidades con menos del 20% de espacio libre:"
      $lowSpace | Format-Table -AutoSize
}

Report-DiskUsage
```

**Explicación:**

Este script genera un informe detallado del uso del espacio en disco de todas las unidades fijas del sistema. Para cada unidad, calcula el espacio total, usado y libre en GB y exporta estos datos a un archivo CSV. Adicionalmente, identifica y muestra las unidades con menos del 20% de espacio libre, ayudando en la gestión proactiva del espacio en disco.

**Ejercicio 11: Gestión de logs de eventos**

Desarrolla un script en PowerShell para gestionar logs de eventos de Windows. Este script debe permitir al usuario elegir entre exportar logs de errores de los últimos 7 días a un archivo HTML o visualizar en consola los logs de advertencia del mismo período. El usuario debe poder hacer su elección mediante una entrada en la consola, y el script debe manejar adecuadamente la entrada para evitar errores.

**Solución:**

```
function Manage-EventLogs {
      Write-Host "Seleccione una opción:"
      Write-Host "1. Exportar logs de errores a HTML"
      Write-Host "2. Mostrar logs de advertencia en consola"
      $option = Read-Host "Ingrese su opción (1 o 2)"
      switch ($option) {
      '1' {
```

```
            $errorLogs = Get-EventLog -LogName Application -EntryType
Error -After (Get-Date).AddDays(-7)
            $errorLogs | ConvertTo-Html | Out-File
"C:\temp\errorLogs.html"
            Write-Host "Logs de errores exportados a
'C:\temp\errorLogs.html'."
      }
      '2' {
            $warningLogs = Get-EventLog -LogName Application -EntryType
Warning -After (Get-Date).AddDays(-7)
            $warningLogs | Format-Table TimeGenerated, Message -AutoSize
      }
      default {
            Write-Host "Opción no válida, intente nuevamente."
      }
      }
}

Manage-EventLogs
```

**Explicación:**

Este script ofrece al usuario la opción de gestionar logs de eventos de dos maneras: exportando los logs de error a un archivo HTML o mostrando los logs de advertencia directamente en la consola. Utiliza un switch para manejar la elección del usuario y ejecuta la acción correspondiente, demostrando una integración efectiva de la interfaz de usuario en scripts PowerShell y proporcionando herramientas útiles de monitoreo y reporte.

## 6. BIBLIOGRAFÍA

https://somebooks.es/scripts-powershell-guia-principiantes/
https://learnxinyminutes.com/docs/es-es/powershell-es/
https://github.com/fleschutz/PowerShell
https://learn.microsoft.com/es-es/powershell/scripting/overview?view=powershell-7.4