

La gestión de archivos

La gestión de archivos nunca ha sido tan sencilla... Si alguno de ustedes ha tenido la oportunidad de pelearse con VBScript estará muy contento de aprender esto. En efecto, con PowerShell, ya no es necesario instanciar los objetos de tipo filesystem, de abrirlos especificando el modo de acceso (lectura o escritura), y después cerrarlos. PowerShell aporta un juego de commandlets dedicado a la gestión de archivos y veremos que esto representa una enorme mejora de la productividad en la escritura de scripts.

En el capítulo Descubra PowerShell, estábamos interesados en el recipiente (el archivo en sí mismo), y habíamos visto cómo crearlos, moverlos, renombrarlos, etc. En estos momentos, nos interesaremos por el contenido, y veremos entre otras cosas, cómo generarlo y cómo releerlo.

Es importante señalar que PowerShell trata generalmente los archivos de texto en Unicode de forma nativa (con algunas excepciones), contrariamente a CMD.exe que únicamente manipula ASCII y las páginas de código de caracteres. Sin embargo, por razones de compatibilidad, es posible forzar a los commandlets a utilizar otras codificaciones tales como ASCII, UTF8, UTF32, etc.

1. Envío de datos a un archivo

Existen dos maneras básicas de proceder para escribir datos en un fichero. Podemos utilizar `Set-Content`, o bien `Out-File`.

Aunque estos dos comandos sirven para hacer lo mismo: crear ficheros y datos, hay una diferencia significativa que es importante conocer pero que no es fácilmente detectable cuando uno está empezando.

Cuando se utiliza `out-File`, va a tratar de formatear, al igual que los otros comandos `out-*`, el flujo antes de escribirlo en el archivo.

`set-Content`, por su parte, no pretende formatear el flujo pero le aplica el método `ToString` para estar seguro de escribir los caracteres. Ésta es la principal diferencia. Sin embargo, aunque a primera vista pueda parecer insignificante, se llevará una sorpresa si intenta escribir un objeto en un archivo con `set-Content` sin haberlo formateado previamente.

Por ejemplo, el resultado de este comando escribirá en un archivo el tipo de objeto en lugar de su contenido:

```
PS > Get-Process powershell | Set-Content MiArchivo.txt
PS > Get-Content MiArchivo.txt

System.Diagnostics.Process (powershell)
```

Mientras que el comando siguiente nos devuelve el resultado esperado:

```
PS > Get-Process powershell | Out-File MiArchivo.txt
PS > Get-Content MiArchivo.txt

Handles  NPM(K)  PM(K)  WS(K)  VM(M)  CPU(s)  Id  ProcessName
-----  -
533      13      64608  65376   219    38,39   2080 powershell
```

Para obtener el mismo resultado con `set-Content`, habría sido necesario efectuar un «transtypage» previo en el objeto antes de escribirlo, como esto:

```
PS > Get-Process powershell | Out-String -Stream | Set-Content
MiArchivo.txt
```

`out-string` nos permite convertir los objetos emitidos representándolos en forma de cadena. El parámetro `-stream` permite enviar a la tubería tantas cadenas como objetos recibidos, en lugar de enviar una cadena única que contenga la representación de todos los objetos.

Si deseamos personalizar el resultado, podríamos escribir lo siguiente:

```
Get-Process powershell | Format-Table id, processname | Out-String |  
Set-Content MiArchivo.txt
```

Otra diferencia interesante es que `set-content` permite escribir directamente los bytes en un archivo gracias al parámetro `-Encoding Byte`. El valor «Byte» de este parámetro es propio de `set-content`, y no existe en `out-file`. Esto va a permitir manipular archivos distintos a los archivos de texto escribiendo directamente los bytes.

En resumen, es preferible usar `out-file` para crear archivos de texto, y `set-content` para los archivos binarios.

a. Los archivos de texto con Out-File

Este commandlet tan potente nos permitirá crear archivos y sus contenidos asociados. Tiene una funcionalidad parecida a los operadores de redirección (que veremos en la siguiente sección), salvo que se puede especificar en `out-file` un cierto número de parámetros suplementarios.

Veamos ahora la lista de parámetros:

Parámetros	Descripción
FilePath <String>	Archivo destino.
Encoding <String>	Tipo de codificación (defecto: unicode).
Append <Switch>	Añadir el contenido a un archivo existente.
Width <Int>	Número de caracteres máximo por línea.
InputObject <PSObject>	Objeto a escribir en el archivo.
NoClobber <Switch>	Indica no reemplazar el archivo existente.


Los valores posibles para el parámetro de codificación son los siguientes:

Nombre	Descripción
Ascii	Fuerza la codificación a ASCII de base (juego de caracteres 0 a 127, 7 bits).
UTF7	Fuerza la codificación a Unicode UTF7 (<i>Unicode Transformation Format</i>).
UTF8	Fuerza la codificación a Unicode UTF8.
Unicode	Fuerza la codificación a Unicode UTF16 LittleEndian.
BigEndianUnicode	Fuerza la codificación a Unicode UTF16 BigEndian.
UTF32	Fuerza la codificación a Unicode UTF32.
Default	Utiliza la codificación de la página de códigos ANSI actuales del sistema.
Oem	Utiliza el identificador de la página de códigos del fabricante de ordenadores OEM (<i>Original Equipment Manufacturer</i>) actual para el sistema operativo.



Microsoft Windows trabaja internamente en Unicode UTF16 LittleEndian. LittleEndian significa que en una palabra (2 bytes), el byte menos significativo se colocará en primer lugar. Al contrario de la notación BigEndian

donde el byte más significativo está en primer lugar. Por ejemplo, si se deseaba codificar el cifra 10 (base decimal) en hexadecimal de 16 bits, daría: 00 0A en LittleEndian, 0A 00 en BigEndian.

 Generalmente es más eficaz utilizar el orden de byte nativo para almacenar caracteres Unicode. Así es preferible utilizar el orden de byte LittleEndian en las plataformas Little-endian de tipo Intel y el orden de byte BigEndian en las plataformas Motorola.

Ejemplo:

Creación de un archivo ASCII que contenga la información sobre un proceso del sistema.

```
PS > Get-Process powershell |  
    Out-File c:\temp\test\MiArchivo.txt -Encoding ascii
```


Este comando crea el archivo ASCII MiArchivo.txt en el directorio c:\temp\test. Este archivo contendrá el resultado de la ejecución del comando anterior pasado a través de la tubería.


Ejemplo 2:

Adición de datos a un archivo existente.

```
PS > Get-Date | Out-File c:\temp\test\MiArchivo.txt -Append -Encoding ascii
```

En este ejemplo, se añaden los datos al archivo que hemos creado en el ejemplo anterior. Tenga cuidado de especificar *siempre* el *mismo* formato de codificación cuando añada datos a un archivo. PowerShell no se lo impedirá pero, si los formatos de sus datos difieren, su archivo se volverá ilegible.

 Cuando añada datos a un archivo de texto, tenga siempre en cuenta su codificación, sabiendo que puede hacer que su archivo quede ilegible. Un método sencillo cuando usted no conoce el origen de un archivo y tiene datos por añadir, es abrirlo con el bloc de notas y hacer como si quisiera guardarlo con «Guardar como». Así en el pie de la ventana, podrá ver una lista desplegable llamada «codificación» donde podrá seleccionar la codificación deseada, suponiendo que la codificación propuesta por defecto es la del archivo que ha abierto.

 Existe otro editor de texto muy bueno y freeware llamado «ConTEXT» que le recomendamos. Con ConTEXT, cuando se abre un archivo, su tipo de codificación se muestra en la barra de estado situada en el pie de la ventana, lo que es muy práctico. Y, naturalmente tendrá derecho, como todo editor de textos digno de este nombre, a la coloración de la sintaxis, así como otras funciones.

b. Redirección del flujo estándar

Creación de archivos

Hemos visto en el capítulo Fundamentos que existe un operador de redirección, el operador *mayor que* «>». Este operador representa la forma más sencilla de crear un archivo. Funciona de manera idéntica que con CMD.exe (con la excepción de que el tipo de codificación por defecto es Unicode). Sepa que cuando se utiliza, el flujo de salida estándar se envía a un archivo de texto.

Ejemplo:

```
PS > Get-childItem C:\temp > dir.txt
```

Esta línea de comandos lista los archivos y carpetas contenidos en el directorio c:\temp en el archivo dir.txt.

Para los que estén acostumbrados a CMD.exe, no hay cambios en el funcionamiento de este operador.

Adición de datos a un archivo

No hay cambios tampoco en la adición de datos, que se sigue realizando con el operador de redirección «>>». Así, gracias a este operador, podemos añadir el contenido al final de un archivo existente.

Ejemplo:

```
PS > Get-Date >> dir.txt
```

Esta línea de comandos tiene como resultado añadir la fecha actual al final del archivo `dir.txt`, conservando el contenido del interior del archivo.

➤ Los operadores de redirección de flujo «>>» y «>>>» en realidad llaman al commandlet `out-file`. Para salir de dudas, llamaremos al rescate `Trace-Command` (que detallaremos en el próximo capítulo) para tratar de descubrir lo que hay en el interior de la bestia... Probemos esto:

```
PS > Trace-command -Name CommandDiscovery -Expression `{get-date > test.txt} -PSHost

DEPURACIÓN : CommandDiscovery Information: 0 : Looking up command: get-date
DEPURACIÓN : CommandDiscovery Information: 0 :
              Attempting to resolve function or filter: get-date
DEPURACIÓN : CommandDiscovery Information: 0 :
              Cmdlet found:
Get-Date   Microsoft.PowerShell.Commands.GetDateCommand,
              Microsoft.PowerShell.Commands.Utility,
              Version=1.0.0.0, Culture=neutral,
              PublicKeyToken=31bf3856ad364e35
DEPURACIÓN : CommandDiscovery Information: 0 : Looking up command: out-file
DEPURACIÓN : CommandDiscovery Information: 0 :
              Attempting to resolve function or filter: out-file
DEPURACIÓN : CommandDiscovery Information: 0 :
              Cmdlet found: Out-File Microsoft.PowerShell.
              Commands.OutFileCommand,
              Microsoft.PowerShell.Commands.Utility,
              Version=1.0.0.0, Culture=neutral,
              PublicKeyToken=31bf3856ad364e35
```

➤ Vemos aparecer en la última línea «Cmdlet found: Out-File», ¡CQD! Aún podemos mejorarlo mirando cuáles son los valores que PowerShell asigna a los diferentes parámetros de **Out-File**. Probemos esta línea de comandos:

```
PS > Trace-command -Name ParameterBinding -Expression `{get-date > test.txt} -PSHost
```

➤ Por motivos de encubrimiento debidos a la verborrea de `Trace-Command`, no visualizaremos la totalidad del resultado sino únicamente las líneas más significativas. Así verá lo siguiente:

```
BIND arg [test.txt] to param [FilePath]
BIND arg [unicode] to parameter [Encoding]
BIND arg [16/08/2009 19:50:19] to parameter [InputObject]
```

➤ Esto confirma lo que veníamos diciendo, PowerShell codifica por defecto sus archivos en Unicode. Observaremos también que el nombre de nuestro archivo se pasa con el parámetro `-FilePath`. Este mecanismo de asociación

de parámetros se aplica también a todos los commandlets. Por lo tanto, cuando nos limitamos a pasar un valor a un parámetro opcional (tal como `Get-Childitem miArchivo` en lugar de `Get-Childitem -FilePath miArchivo`), la asociación valor/parámetro se hace automáticamente de forma interna.

c. Creación de archivos binarios con Set-Content

Contrariamente a `out-File`, este commandlet escribe los datos tal como los recibe. La gran fuerza de `set-Content` es poder escribir directamente los bytes en un archivo, sea cual sea el tipo de archivo (texto o binario). Pero atención, `set-content` borra el contenido del archivo de destino ya que no posee el switch `-append` como `out-File`.

No hay que olvidar que `set-content` forma parte de la familia de commandlets `*-Content`:

- **Add-Content**: añade datos a un archivo existente,
- **Clear-Content**: borra los datos presentes en un archivo, pero no el archivo,
- **Get-Content**: lee el contenido de un archivo. Estudiaremos este commandlet en detalle un poco más adelante.

Veamos los parámetros de `set-Content`:

Parámetros	Descripción
Path <String[]>	Archivo destino receptor de los datos.
Value <Object[]>	Datos a escribir (reemplazaran el contenido existente).
Include <String[]>	Modifica únicamente los elementos especificados.
Exclude <String[]>	Omite los elementos especificados.
Filter <String>	Especifica un filtro en el formato o el lenguaje del proveedor.
PassThru <Switch>	Pasa el objeto creado por el commandlet a través de la tubería.
Force <Switch>	Fuerza el comando a ejecutarse sin comprometer la seguridad, por ejemplo, creando el directorio de destino si no existe.
Credential <PSCredential>	Utiliza la información de identificación para validar el acceso al archivo.
Encoding <String>	Tipo de codificación (valor por defecto: « default », en ANSI).

Los valores posibles para el parámetro de codificación son los siguientes:

Nombre	Descripción
ASCII	Fuerza la codificación a ASCII de base (juego de caracteres 0 a 127, 7 bits).
UTF7	Fuerza la codificación a Unicode UTF7.
UTF8	Fuerza la codificación a Unicode UTF8.
Unicode	Fuerza la codificación a Unicode UTF16 LittleEndian.
BigEndianUnicode	Fuerza la codificación a Unicode UTF16 BigEndian.

Byte	Fuerza la codificación a byte.
String	Utiliza la codificación de la página de códigos ANSI actuales del sistema.
Unknown	Ídem Unicode.

Tenga especial cuidado ya que no son los mismos valores que para el commandlet `Out-File`.

Aunque es posible que se escriban datos textuales con `Set-Content` (tomando las precauciones enunciadas en la introducción), lo más interesante es la posibilidad de escribir directamente los bytes en un archivo.



Si enviamos los datos de tipo `string` a un archivo sin especificar explícitamente la codificación deseada, el archivo resultante será un archivo ANSI. Es decir, un archivo ASCII ampliado con su página de código actual para tener en cuenta los caracteres acentuados.

Ejemplo:

Envío de datos textuales a un archivo.

```
PS > 'AAéBB' | set-content test.txt
```

Esta línea de comandos crea el archivo `test.txt` en formato ANSI. Veamos ahora cual es el tamaño de este fichero:

```
PS > Get-ChildItem test.txt

Directorio: C:\temp

Mode                LastWriteTime         Length Name
----                -
-a---          28/08/2009   23:53             7 test.txt
```

¿Por qué diablos tenemos un archivo de 7 bytes si únicamente hemos enviado cinco caracteres a su interior?

Gracias a una pequeña función personalizada de nuestra cosecha, vamos a poder pasar detalladamente por todos los bytes que componen nuestro archivo.

Nuestra función `Get-Dump`, como su nombre inglés indica, «descarga» el contenido de un archivo decimal, hexadecimal y ASCII:

```
function Get-Dump
{
    param ([string]$path=$(throw 'Ruta no encontrada'),
          [int]$tamaño=(gci $path).Length)

    $fic = Get-Content -Path $path -Encoding byte -TotalCount $tamaño
    [string]$strDest = ''
    [string]$strAsciiDest = ''
    [string]$strHexDest = ''
    for ($i=0; $i -lt $tamaño; $i++)
    {
        $StrDest += $fic[$i]
        $StrDest += ' '
        $strAsciiDest += [char]$fic[$i]
        $strHexDest += ('{0:x}' -f $fic[$i]).PadLeft(2,'0')
        $strHexDest += ' '
    }
}
```

```

    }

    Write-host "DEC:    $StrDest"
    Write-host "HEX:    $strHexDest"
    Write-host "ASCII: $strAsciiDest"
}

```

Esta función debería ayudarnos a comprender de dónde procede esta diferencia de tamaño.

```

PS > Get-Dump test.txt

DEC:    65 65 233 66 66 13 10
HEX:    41 41 e9 42 42 0d 0a
ASCII:  AAéBB

```

65, 66, y 233 son respectivamente los códigos ASCII de los caracteres «A», «B», y «é»; hasta aquí todo normal. Pero podemos observar que tenemos dos bytes adicionales al final del fichero que se han añadido automáticamente. Estos bytes 13 y 10 en decimal o 0D, 0A en hexadecimal corresponden a los caracteres **CR** (*Carriage Return*) y **LF** (*Line Feed*). Dicho de otra forma, un retorno de carro y un retorno de línea.

Esto es completamente normal ya que en la plataforma Windows (ya pasaba en DOS), cada línea de un archivo de texto se concluye por **CR** y **LF**. Mientras que en Unix (y otros derivados) una línea termina únicamente por **LF**. Esto es lo que explica porque se dan algunos problemas de formato en el intercambio de archivos de texto entre estas plataformas...



La adición de los códigos de control **CR** y **LF** se producen igualmente con el commandlet `Out-File`.

Ejemplo:

*Escritura de un flujo de bytes en un archivo sin **CR LF**.*

En este ejemplo intentaremos escribir una cadena de caracteres en un archivo, pero esta vez vamos a procurar que **CR** y **LF** no se añadan al final de línea. Para lograrlo, vamos a enviar los bytes correspondientes a los códigos ASCII de la cadena a escribir; después especificaremos el tipo de codificación `byte` para `Set-Content`.

```
PS > [byte[]][char[]]'AAéBB' | Set-Content test.txt -Encoding byte
```

Haciendo esto, convertimos la cadena «AAéBB» en una tabla de caracteres, que a continuación convertimos en una tabla de bytes, y que después pasamos todo a `Set-Content` donde no olvidamos de añadir el parámetro `-encoding byte`.

Ejemplo:

*Convertir un archivo de texto Unix a **DOS**.*

```

# convert-Unix2Dos.ps1

param ($path=$(throw 'archivo no encontrado'), $dest=$path)

$tab = get-content $path -encoding byte
for ($i=0;$i -lt $tab.length; $i++)
{
    if ($tab[$i] -eq 10)
    {
        $tab=$tab[0..$(($i-1)]+[byte]13+$tab[$i..$tab.length]
        $i++
    }
}

```

```
}
}
$tab | Set-Content $dest -encoding Byte
```

Este pequeño script convierte un archivo de tipo Unix en un archivo compatible DOS/Windows insertando el carácter de control **CR** (13 Dec.) delante de cada carácter **LF** (10 Dec.).

El grupo de bytes siguiente: 68 74 57 98 102 **10** 65 66 48 **10** 125 139 78

se transformará en: 68 74 57 98 102 **13 10** 65 66 48 **13 10** 125 139 78

Gracias a la instrucción `param` y a la inicialización automática de los parámetros, se producirá una excepción si no especifica el archivo fuente. Además, si omite especificar un archivo de destino, el archivo fuente se utilizará como archivo de destino y su contenido existente se borrará.

Se guarda el contenido del archivo fuente en forma de un conjunto de bytes en la tabla `$tab`. Después la labor es algo más ardua: se recorre la totalidad de la tabla `$tab` buscando el carácter **LF**. Cuando se encuentra uno, se concatena al inicio de nuestra tabla con CR y al final de nuestra tabla, después se reinyecta el nuevo contenido en nuestra tabla `$tab`. Resumiendo, borramos en cada iteración el contenido de `$tab` por un nuevo contenido modificado. Hacemos esto porque no existe un método para insertar un elemento en una tabla en un punto determinado. Por último, incrementamos nuestra variable de índice en una posición ya que hemos agregado un elemento en `$tab`; en caso contrario la verificación sería siempre verdadera y entraríamos en un bucle infinito. Por último nuestra tabla de bytes se pasa mediante la tubería a `set-content` sin olvidar de especificar el tipo de codificación `byte`.

2. Lectura de datos con Get-Content

Como usted esperaba y como su nombre indica `Get-Content` va a permitirnos leer el contenido de un archivo. Este puede ser de tipo texto o de tipo binario, poco importa, `Get-Content` se adapta a partir del momento en que se le precisa. Por defecto este commandlet espera leer archivos de texto.

Veamos los parámetros de `Get-Content`:

Parámetros	Descripción
Path <String[]>	Archivo fuente con los datos a leer.
TotalCount <Int64>	Número de líneas por leer. Por defecto, todas (valor -1).
ReadCount <Int64>	Número de líneas de contenido enviadas simultáneamente a la tubería. Por defecto se envían una por una (valor 1). Un valor 0 indica que se quiere enviar todas las líneas de golpe.
Include <String[]>	Recuperará únicamente los elementos especificados.
Exclude <String[]>	Omite los elementos especificados.
Filter <String>	Especifica un filtro en el formato o el lenguaje del proveedor.
Force <Switch>	Fuerza el comando a ejecutarse sin comprometer la seguridad.
Credential <PSCredential>	Utiliza la información de autenticación para validar el acceso al archivo.
Encoding <String>	Especifica el tipo de codificación de los caracteres utilizados para mostrar el contenido.

Los valores posibles para el parámetro de codificación son los siguientes:

Nombre	Descripción
ASCII	Fuerza la codificación a ASCII de base (juego de caracteres 0 a 127, 7 bits).
UTF7	Fuerza la codificación a Unicode UTF7.
UTF8	Fuerza la codificación a Unicode UTF8.
Unicode	Fuerza la codificación a Unicode UTF16 LittleEndian.
BigEndianUnicode	Fuerza la codificación a Unicode UTF16 BigEndian.
Byte	Fuerza la codificación a byte.
String	Utiliza la codificación de la página de códigos ANSI actual del sistema.
Unknown	Ídem Unicode.

Ejemplo:

Funcionalidades básicas.

```
PS > Get-Date > misProcesos.txt
PS > Get-Process >> misProcesos.txt

PS > Get-Content misProcesos.txt -Totalcount 10

domingo 20 septiembre 2009 11:22:22

Handles  NPM(K)  PM(K)  WS(K) VM(M) CPU(s)  Id  ProcessName
-----  -
91         5   3280   1496   62    0,22  3408 ashDisp
129        140  4340   2072   67           2380 ashMaiSv
351        10  28156  15888   140          1676 ashServ
140        40  16312  32156   112          2416 ashWebSv
30         2    836    396    23          1664 aswUpdSv
```

En este ejemplo, hemos creamos un archivo de texto con el operador de redirección «mayor que» (unicode) que contiene la fecha y hora, así como la lista de los procesos en ejecución. Luego, hacemos una llamada a `Get-Content` para leer y mostrar por pantalla las diez primeras líneas del archivo.

Ejemplo:

Manipular un archivo como una tabla.

```
PS > $fic = Get-Content FábulaLaFontana.txt
PS > $fic[14]
La hormiga no presta;
```

Utilizando una variable para recibir el resultado del comando `Get-Content`, estamos creamos en realidad una tabla de líneas. Y visualizaremos acto seguido la línea situada en el índice 14 de la tabla (en realidad la décimo quinta línea del archivo, porque no olvide que los índices de la tabla comienzan por cero).

Además, como una cadena es también una tabla de caracteres, se puede leer cualquier carácter utilizando la sintaxis de tablas de dos dimensiones. Por ejemplo, la «i» de la palabra hormiqa que se encuentra en el índice 8:

```
PS > $fic[14][8]
i
```

Para finalizar con este ejemplo, si aplicamos el método `Length` a nuestra tabla `$fic`, obtendremos el número de elementos que la componen, o sea el número de líneas de nuestro archivo de texto.

```
PS > $fic.Length
22
```

Veintidós es el número de líneas de nuestro archivo.

Ejemplo:

Lectura de un archivo en modo «bruto».

Como comentamos al introducir este comando, `Get-Content` sabe leer los bytes. Esta funcionalidad es particularmente interesante para revelar el contenido real de los archivos, es en cierta forma un modo de acceso de bajo nivel al contenido.


En efecto, ¿qué diferencia a un archivo de texto de un archivo binario? La respuesta es fácil: el contenido o la interpretación del mismo. En los dos casos, un archivo posee atributos que caracterizan su nombre, su extensión, su tamaño, su fecha de creación, etc.

Un archivo de texto contiene, así como su homólogo el archivo binario, un conjunto de bytes con una cierta estructura.

Probemos de abrir en modo bruto un archivo de texto Unicode, pero antes vamos a crear un archivo nuevo:

```
PS > 'PowerShell' > test.txt
PS > Get-Content test.txt -Encoding byte

255 254 80 0 111 0 119 0 101 0 114 0 83 0 104 0 101 0 108 0 108 0 13 0 10 0
```

 Los bytes se muestran en realidad verticalmente, pero para facilitar la lectura y la comprensión del ejemplo los hemos transcrito horizontalmente.

Un ojo experto, con los archivos de texto ASCII observaría las siguientes dos cosas:

- El archivo empieza por dos bytes extraños: 255 y 254.
- Todos los caracteres están codificados en dos bytes siendo uno de los dos cero.

También observará la presencia de los bytes 13 y 10 al final de línea que corresponden a **CR** y **LF** (ver un poco más arriba en este capítulo).

La presencia de los bytes 255 y 254 se explica por el hecho de que cualquier archivo Unicode comienza por una cabecera cuya longitud oscila entre 2 y 4 bytes. Esto difiere según la codificación Unicode escogida (UTF8, UTF16, UTF32).

En nuestro caso, 255 254 (FF FE en notación hexadecimal) significa que hemos asignado un archivo **UTF16 Little Endian**.

La presencia de ceros se explica ya que en un archivo UTF16 todos los caracteres están codificados con dos bytes.

Ejemplo:

Determinar el tipo de codificación de un archivo.

La pregunta que nos formulábamos desde hace ya algunas páginas, es decir: «¿Cómo reconocer el tipo de codificación de un archivo de texto?» ha logrado encontrar una respuesta en el ejemplo anterior. Los primeros bytes de un archivo de texto nos dan su codificación.

Realicemos un pequeño script que nos dirá de que tipo de codificación tiene un archivo a partir de estos primeros bytes.

```
# Get-FileTypeEncoding.ps1

param ([string]$path=$(throw 'Ruta no encontrada'))

# definición de las variables y constantes
$ANSI=0
Set-Variable -Name UTF8      -Value 'EFBBBF'   -Option constant
Set-Variable -Name UTF16LE -Value 'FFFE'       -Option constant
Set-Variable -Name UTF16BE -Value 'FEFF'       -Option constant
Set-Variable -Name UTF32LE -Value 'FFFE0000' -Option constant
Set-Variable -Name UTF32BE -Value '0000FEFF' -Option constant

$fic = Get-Content -Path $path -Encoding byte -TotalCount 4
# Organizar los bytes leídos en 2 caracteres y conversión hexadecimal
# ex: 0 -> 00, ou 10 -> 0A en lugar de A
# y concatenación de los bytes en una cadena para efectuar la comparación
[string]$strLeido = [string]('{0:x}' -f $fic[0]).PadLeft(2, '0') +
                   [string]('{0:x}' -f $fic[1]).PadLeft(2, '0') +
                   [string]('{0:x}' -f $fic[2]).PadLeft(2, '0') +
                   [string]('{0:x}' -f $fic[3]).PadLeft(2, '0')
Switch -regex ($strLeido){
    "^$UTF32LE" {write-host 'Unicode UTF32LE'; break}
    "^$UTF32BE" {write-host 'Unicode UTF32BE'; break}
    "^$UTF8"    {write-host 'Unicode UTF8    '; break}
    "^$UTF16LE" {write-host 'Unicode UTF16LE'; break}
    "^$UTF16BE" {write-host 'Unicode UTF16BE'; break}
    default
    {
        # Búsqueda de un byte con valor > 127
        $fic = Get-Content -Path $path -Encoding byte
        for ($i=0; $i -lt (gci $path).Length; $i++){
            if ([char]$fic[$i] -gt 127){
                $ANSI=1
                break
            }
            else {
                $ANSI=0
            }
        } #fin for
        if ($ANSI -eq 1){
            Write-Host 'Archivo ANSI'
        }
        else{
            Write-Host 'Archivo ASCII'
        }
    } #fin default
} #fin switch
```

Este script lee los cuatro primeros bytes del archivo, los formatea y los compara a la firma Unicode para determinar el tipo de codificación. Si no se ha encontrado ninguna firma, indica que el archivo es de tipo ASCII puro (caracteres US de 0 a 127), o bien de tipo ANSI (ASCII extendido, o bien ASCII + página de códigos para generar los caracteres acentuados).



Información de última hora: explorando en profundidad las clases del Framework .NET (que descubrirá en el capítulo .NET) hemos descubierto que existe una clase que permite determinar el tipo de codificación de un archivo.

Ejemplo:

```
PS > $sr = new-object system.io.streamreader c:\temp\MiArchivo.txt
PS > $sr.CurrentEncoding

BodyName           : utf-8
EncodingName       : Unicode (UTF-8)
HeaderName         : utf-8
WebName            : utf-8
WindowsCodePage    : 1200
IsBrowserDisplay   : True
IsBrowserSave      : True
IsMailNewsDisplay  : True
IsMailNewsSave     : True
IsSingleByte       : False
EncoderFallback    : System.Text.EncoderReplacementFallback
DecoderFallback    : System.Text.DecoderReplacementFallback
IsReadOnly         : True
CodePage           : 65001
```



Esto nos simplifica enormemente la tarea. ¡He aquí la prueba de que hojeando un poco en Framework .NET se puede ganar mucho tiempo! El ejemplo sigue siendo interesante, pues conocerá finalmente un poco más sobre la codificación Unicode.


3. Búsqueda del contenido con **Select-String**

Gracias a **select-string** vamos a poder revisar el contenido de una variable de tipo cadena, de un archivo, o de un gran número de archivos en la búsqueda de una cadena de caracteres en forma de expresión regular. Los usuarios Unix que ya conocen el comando **Grep** no se encontrarán muy desorientados.

Veamos los parámetros de **select-string** (los parámetros señalados con un asterisco sólo están disponibles con PowerShell v2):

Parámetros	Descripción
Pattern <String[]>	Cadena o expresión regular a buscar.
Path <String[]>	Objetivo de la búsqueda: cadena(s) o archivo(s).
InputObject <PSObject>	Acepta un objeto como entrada.
Include <String[]>	Recupera únicamente los elementos especificados.
Exclude <String[]>	Omite los elementos especificados.
SimpleMatch <Switch>	Especifica que debe utilizarse una correspondencia simple, en vez de una correspondencia de expresión regular.

CaseSensitive <Switch>	Hace las correspondencias sensibles a mayúsculas y minúsculas.
Quiet <Switch>	Reemplaza el resultado del comando por un valor booleano.
List <Switch>	Especifica que debe devolverse una sola correspondencia para cada archivo de entrada.
AllMatches (*) <Switch>	Busca varias coincidencias en cada línea de texto. Sin este parámetro, <code>select-string</code> busca únicamente la primera correspondencia en cada línea de texto.
Context (*) <Int32>	Permite seleccionar un número específico de líneas antes y después de la línea que contiene la correspondencia (permite ver de este modo el contenido buscado en su contexto).
Encoding (*) <String>	Indica la codificación del flujo de texto al que debe aplicarse <code>select-string</code> . Los valores pueden ser: UTF7, UTF8, UTF32, Ascii, Unicode, BigIndian, Default o OEM.
NotMatch (*) <Switch>	Indica qué modelo no retornará la búsqueda. Este parámetro es muy útil para realizar una búsqueda inversa (no seleccionando las líneas basadas en el modelo). Equivalente a Grep -v.

 Los caracteres acentuados no se tienen correctamente en cuenta en las búsquedas en el interior de los archivos ANSI. En cambio, todo funciona correctamente con los archivos Unicode.

Ejemplo:

Búsqueda simple.

```
PS > select-string -Path c:\temp\*.txt -Pattern 'hormiga'

C:\temp\CigarraHormiga.txt:8:Casa de la hormiga su vecina,
C:\temp\CigarraHormiga.txt:15:La hormiga no presta;
C:\temp\hormigasÚtiles.txt:1:Las hormigas son muy útiles.
```

En este ejemplo, buscamos la cadena «hormiga» en todos los archivos de texto del directorio `c:\temp`.

Obtendremos retornado el nombre de los archivos (o del archivo si sólo hubiese uno) que contengan la cadena buscada. Los valores 8, 15 y 1 corresponden al número de la línea en el archivo donde se ha encontrado la ocurrencia.

A veces cuando los resultados son numerosos, es interesante utilizar el conmutador `-List` para especificar al commandlet que retorne únicamente el primer resultado encontrado por archivo.

Veamos cual será el resultado con `-List`:

```
PS > select-string -Path c:\temp\*.txt -Pattern 'hormiga' -List

C:\temp\CigarraHormiga.txt:8:Casa de la hormiga su vecina,
C:\temp\hormigasÚtiles.txt:1:Las hormigas son muy útiles.
```

Los resultados obtenidos son de tipo `Microsoft.PowerShell.Commands.MatchInfo`. De este modo, es posible obtener y manipular un cierto número de información complementaria pasando por una variable intermedia, como en el caso siguiente:

```
PS > $var = Select-String -Path c:\temp\*.txt -Pattern 'hormiga'
```

```
PS > $var | Get-Member -MemberType property
```

```
TypeName: Microsoft.PowerShell.Commands.MatchInfo
```

Name	MemberType	Definition
Context	Property	Microsoft.PowerShell.Commands.MatchInfoContext Context {get;set;}
Filename	Property	System.String Filename {get;}
IgnoreCase	Property	System.Boolean IgnoreCase {get;set;}
Line	Property	System.String Line {get;set;}
LineNumber	Property	System.Int32 LineNumber {get;set;}
Matches	Property	System.Text.RegularExpressions.Match[] Matches {get;set;}
Path	Property	System.String Path {get;set;}
Pattern	Property	System.String Pattern {get;set;}

Ahora, probamos de forzar una vista en forma de lista:

```
PS > $var | Format-List
```

```
IgnoreCase : True
LineNumber : 8
Line       : Casa de la hormiga su vecina,
Filename   : CigarraHormiga.txt
Path       : C:\temp\CigarraHormiga.txt
Pattern    : hormiga
Context    :
Matches    : {Hormiga}
```

```
IgnoreCase : True
LineNumber : 15
Line       : La hormiga no presta;
Filename   : CigarraHormiga.txt
Path       : C:\temp\CigarraHormiga.txt
Pattern    : hormiga
Context    :
Matches    : {Hormiga}
```

```
IgnoreCase : True
LineNumber : 1
Line       : Las hormigas son muy utiles.
Filename   : hormigasUtiles.txt
Path       : C:\temp\hormigasUtiles.txt
Pattern    : hormiga
Context    :
Matches    : {Hormiga}
```

De este modo podremos pedir el número de la línea del primer caso:

```
PS > $var[0].LineNumber
8
```


Ejemplo:

Otra búsqueda sencilla.

De la misma manera también podemos utilizar `select-string` pasándole los datos específicos mediante la tubería:

```
PS > Get-Item c:\temp\*.txt | Select-String -Pattern 'hormiga'
```

Los resultados obtenidos serán los mismos que en el ejemplo anterior.

 ¡No se equivoque! Utilice `Get-Item` o `Get-Childitem` y no `Get-Content` ya que todo y que también podría funcionar, pueden darse efectos no deseados. En efecto, pasaría al «pipeline» el contenido de los archivos y no los propios archivos, y el contenido en cierto modo se concatenaría. Lo que tendría como consecuencia falsear el valor de la propiedad `LineNumber`.

Ejemplo:

```
PS > $var = Get-Content c:\temp\*.txt | Select-String -Pattern 'hormiga'
PS > $var | Format-List

IgnoreCase : True
LineNumber : 8
Line       : Casa de la hormiga su vecina,
Filename   : InputStream
Path       : InputStream
Pattern    : hormiga
Context    :
Matches    : {Hormiga}

IgnoreCase : True
LineNumber : 15
Line       : La hormiga no presta;
Filename   : InputStream
Path       : InputStream
Pattern    : hormiga
Context    :
Matches    : {Hormiga}

IgnoreCase : True
LineNumber : 23
Line       : Las hormigas son muy utiles.
Filename   : InputStream
Path       : InputStream
Pattern    : hormiga
Context    :
Matches    : {Hormiga}
```

En este ejemplo, observará que:

- El nombre del archivo ha desaparecido de los resultados al ser reemplazado por un «InputStream» que indica la procedencia de los datos.
- En cualquier vínculo cuyo fichero de origen haya desaparecido, el número de línea es relativo a la totalidad del flujo, lo que da un resultado potencialmente erróneo si se espera tener la posición en el archivo (observe el tercer y último caso anterior).


Ejemplo 3:

Búsqueda en base a expresiones regulares.

```
PS > Get-item $pshome/es-ES/*.txt | Select-String -Pattern 'item$'

C:\...\es-ES\about_Alias.help.txt:159:          get-childitem
C:\...\es-ES\about_Core_Commands.help.txt:23:    Get-ChildItem
C:\...\es-ES\about_Core_Commands.help.txt:36:    APPLETS DE COMANDO ITEM
C:\...\es-ES\about_Core_Commands.help.txt:45:    Set-Item
C:\...\es-ES\about_Environment_Variable.help.txt:35: get-childitem
C:\...\es-ES\about_Environment_Variable.help.txt:100: get-childitem
C:\...\es-ES\about_Parameter.help.txt:35:        help Get-ChildItem
C:\...\es-ES\about_Provider.help.txt:137:        get-childitem
C:\...\es-ES\about_Special_Characters.help.txt:46: $a = Get-ChildItem
C:\...\es-ES\about_Wildcard.help.txt:82:         help Get-ChildItem
```

Esta línea de comandos va a explorar todos los ficheros cuya extensión sea «.txt» buscando una cadena que termine por «item».

 El ejemplo anterior también se basaba en una expresión regular. Simplemente, su sintaxis no lo distingue de una expresión literal. Es preciso utilizar el parámetro `-simpleMatch` para que `select-string` haga una búsqueda sobre una expresión literal en lugar de sobre una expresión regular.

Ejemplo:

Búsqueda donde el resultado sea un booleano.

```
PS > Select-String C:\temp\CigarraHormiga.txt -Pattern 'hormiga' -Quiet
True

PS > Select-String C:\temp\CigarraHormiga.txt -Pattern 'elefante' -Quiet
False
```

Ejemplo:

Búsqueda de una cadena que muestre su contexto (2 líneas antes y 2 líneas después).

```
PS > Select-String Cigarrahormiga.txt -Pattern 'Agosto' -Context 2

Cigarrahormiga.txt:11:Hasta la nueva estación
Cigarrahormiga.txt:12:"Le pagaré, le dijo ella,
Cigarrahormiga.txt:13:Antes de agosto, fe de animal,
Cigarrahormiga.txt:14:Interés y principal."
Cigarrahormiga.txt:15:La hormiga no presta;
```

Siendo rigurosos, tendríamos que añadir el parámetro `-simpleMatch` con el fin de precisar que nuestra búsqueda se realiza sobre una cadena y no sobre una expresión regular. Funcionará correctamente ya que no hay caracteres especiales en nuestra cadena de búsqueda.

4. Gestión de archivos CSV: Export-CSV / Import-CSV

Los archivos CSV (*Comma Separated Values*) son archivos de texto cuyos valores están separados por comas. Generalmente, la primera línea de estos archivos es la cabecera. Ésta incluye el nombre de cada columna de datos, y los valores que la componen también están separados por comas.

Veamos un ejemplo de archivo CSV:


```
Sexo,Apellido,Año_de_nacimiento
M,Eduardo,1982
M,Jose,1974
F,Eleonor,2004
```

PowerShell comprende un juego de dos commandlets para gestionar estos archivos: **Export-csv** para crear un archivo, **Import-csv** para leerlo.

Veamos los diferentes parámetros de **Export-csv** (los parámetros señalados con un asterisco sólo están disponibles con PowerShell v2):

Parámetros	Descripción
Path <String>	Ruta del archivo de destino.
InputObject <PSObject>	Acepta un objeto como entrada.
Force <Switch>	Reemplaza el archivo especificado si el destino ya existe.
Encoding <String>	Tipo de codificación del archivo a crear (véase Out-File para la lista de valores posibles). ASCII es el tipo por defecto.
NoTypeInfoInformation <Switch>	Por defecto, una cabecera está escrita con el tipo de datos. Si este conmutador está especificado, esta cabecera no se escribirá.
NoClobber <Switch>	No sobrescribir el fichero si ya existe.
Delimiter (*) <Char>	Muy útil, este parámetro permite especificar un carácter delimitador para separar los valores de propiedad. El valor por defecto es una coma (,).
UseCulture (*)<Switch>	En lugar del parámetro Delimitador, también puede usar UseCulture. Especificando una cultura específica, PowerShell adaptará el delimitador (el delimitador para la cultura es-ES es el punto y coma). Para comprobarlo, pruebe: (Get-Culture).TextInfo.ListSeparator

Y ahora los de **Import-csv**:

Parámetros	Descripción
Path <String[]>	Ruta del archivo fuente.
Delimiter (*) <Char>	Muy útil, este parámetro permite especificar un carácter delimitador para separar los valores de propiedad. El valor por defecto es una coma (,).
UseCulture (*)<Switch>	En lugar del parámetro Delimitador, también puede usar UseCulture. Especificando una cultura específica, PowerShell adaptará el delimitador (el delimitador para la cultura es-ES es el punto y coma). Para comprobarlo, pruebe: (Get-Culture).TextInfo.ListSeparator
Header (*) <String[]>	Permite especificar otra línea de encabezado de columna para el archivo importado.

Ejemplo: Export-CSV

```
PS > Get-Eventlog system -Newest 5 |
    Select-Object TimeGenerated,EntryType,Source,EventID |
```

```
Export-Csv c:\temp\EventLog.csv -Encoding Unicode
PS > Get-Content c:\temp\EventLog.csv
```

```
#TYPE System.Management.Automation.PSCustomObject
TimeGenerated,EntryType,Source,EventID
"20/09/2009 12:31:29","Information","Service Control Manager","7036"
"20/09/2009 12:21:29","Information","Service Control Manager","7036"
"20/09/2009 12:00:01","Information","EventLog","6013"
"20/09/2009 11:50:38","Information","VPCNetS2","12"
"20/09/2009 11:50:38","Information","VPCNetS2","5"
```

Acabamos de crear un archivo Unicode llamado **EventLog.csv**. Contiene las propiedades **timeGenerated,Entrytype,source,EventID** de un objeto de tipo diario de eventos. Observe que la primera línea del archivo comienza por **#TYPE** seguido del tipo del objeto contenido en nuestro archivo; dicho de otra forma se trata del tipo generado por **Get-EventLog**.



Tenga cuidado, ya que por defecto este comando genera archivos de tipo ASCII.

Ejemplo: Import-CSV

```
PS > $diario = Import-Csv c:\temp\EventLog.csv
PS > $diario
```

TimeGenerated	EntryType	Source	EventID
-----	-----	-----	-----
20/09/2009 12:31:29	Information	Service Control Manager	7036
20/09/2009 12:21:29	Information	Service Control Manager	7036
20/09/2009 12:00:01	Information	EventLog	6013
20/09/2009 11:50:38	Information	VPCNetS2	12
20/09/2009 11:50:38	Information	VPCNetS2	5

Ahora, observemos las propiedades y métodos de **\$diario**:

```
PS > $diario | Get-Member
```

```
TypeName: CSV:System.Management.Automation.PSCustomObject
```

Name	MemberType	Definition
----	-----	-----
Equals	Method	System.Boolean Equals(Object obj)
GetHashCode	Method	System.Int32 GetHashCode()
GetType	Method	System.Type GetType()
ToString	Method	System.String ToString()
EntryType	NoteProperty	System.String EntryType=Information
EventID	NoteProperty	System.String EventID=1103
Source	NoteProperty	System.String Source=Dhcp
TimeGenerated	NoteProperty	System.String TimeGenerated=20/09/2009 12:31:29

Podemos ver que tenemos propiedades que corresponden al nombre de nuestra línea de cabecera: ¡lo que va a ser muy útil para recuperar los valores! Por ejemplo:

```
PS > $diario[0].EventID
7036
```

Ejemplo 2: Export-CSV e Import-CSV

Supongamos ahora que tiene que realizar la búsqueda de resultados en un archivo .csv y luego de sus modificaciones. Tomemos como ejemplo el fichero .csv siguiente:

```
Apellido,Nombre,Dominio,Ultima_Conexion
Lemesle,Robin,powershell-scripting.com,20/09/2009
Petitjean,Arnaud,powershell-scripting.com,21/09/2009
Teixeira,Jessica,,20/09/2009
```

En este archivo, tres personas son identificadas entre ellas, una no se identifica como perteneciente al dominio powershell-scripting.com.

En un primer paso, importamos el archivo.

```
PS > $usuarios = Import-Csv ./usuarios.csv
PS > $usuarios

Apellido      Nombre      Dominio      Ultima_Conexion
---      -
Lemesle      Robin      powershell-scripting.com  20/09/2009
Petitjean    Arnaud     powershell-scripting.com  21/09/2009
Teixeira     Jessica    powershell-scripting.com  20/09/2009

PS > $usuarios[0]

Apellido      Nombre      Dominio      Ultima_Conexion
---      -
Lemesle      Robin      powershell-scripting.com  20/09/2009
```

Como se puede ver, la variable \$usuarios se comporta como una tabla en la que cada línea corresponde a un número de índice. Además cada objeto definido en la tabla dispone de las propiedades Nombre,Apellido,Dominio,Ultima_Conexion, y los nombres de las columnas de nuestro archivo CSV.

```
PS > $usuarios[0].Apellido
Ledesma

PS > $usuarios[0].Dominio
powershell-scripting.com
```

Naturalmente, la tabla puede recorrerse con un bucle y cada uno de los valores es modificable, como se ve en el caso inferior. Para cada usuario, si no está especificado un dominio, se le añade el valor PowerShell-scripting.com.

```
PS > Foreach($usuario in $usuarios){
if(($usuario.Dominio) -eq ''){
    $usuario.Dominio = 'PowerShell-scripting.com'
    Write-Host "el usuario $($usuario.Apellido) se ha añadido al dominio"
}
}
el usuario Teixeira se ha añadido al dominio

PS > $usuarios

Apellido      Nombre      Dominio      Ultima_Conexion
---      -
Lemesle      Robin      powershell-scripting.com  20/09/2009
Petitjean    Arnaud     powershell-scripting.com  21/09/2009
Teixeira     Jessica    powershell-scripting.com  20/09/2009
```

Lemesle	Robin	powershell-scripting.com	20/09/2009
Petitjean	Arnaud	powershell-scripting.com	21/09/2009
Teixeira	Jessica	powershell-scripting.com	20/09/2009

Por último, para tener en cuenta los cambios introducidos, el registro de la variable \$usuarios en el archivo usuarios.csv se efectuará con el comando Export-Csv.

```
PS > $usuarios | Export-Csv usuarios.csv -Encoding unicode
```



Si prefiere editar este tipo de archivos con Microsoft Excel en lugar de usar un editor de textos, le recomendamos forzar el delimitador al valor punto y coma, bien especificando el parámetro `-Delimiter ';'` , o bien añadiendo el switch `-UseCulture`. De este modo, al hacer doble clic en el archivo CSV, Excel debería convertirlo y mostrarlo automáticamente.

5. Gestión de los archivos XML: Import-Clixml/Export-Clixml

XML (*eXtensible Markup Language*) es un lenguaje basado en una jerarquización de los datos en forma de etiqueta. Para saber qué aspecto tiene del XML, lo mejor es sin duda ver un ejemplo.

```
<Libro>
  <Titulo>Windows PowerShell</Titulo>
  <SubTitulo>Guía de administración de referencia para la administración
del sistema</SubTitulo>
  <Autor>
    <Nombre>Arnaud Petitjean</Nombre>
    <AñoDeNacimiento>1974</AñoDeNacimiento>
    <Distinción>MVP PowerShell</Distinción>
  </Autor>

  <Autor>
    <Nombre>Robin Lemesle
    </Nombre>
    <AñoDeNacimiento>1985</AñoDeNacimiento>
    <Distinción>MVP PowerShell</Distinción>
  </Autor>
  <Capítulo>
    <Nombre>Introducción</Nombre>
    <NúmeroPaginas>100</NúmeroPaginas>
  </Capítulo>

  <Capítulo>
    <Nombre>Descubrir PowerShell</Nombre>
    <NúmeroPaginas>120</NúmeroPaginas>
  </Capítulo>
</Libro>
```

Para conocer todos los comandos relacionados con la utilización del archivo XML, teclee el comando siguiente:

```
PS > Get-Command -Type cmdlet *XML*
```

CommandType	Name	Definition
-----	----	-----

Cmdlet	ConvertTo-Xml	ConvertTo-Xml [-InputO...
Cmdlet	Export-Clixml	Export-Clixml [-Path]...
Cmdlet	Import-Clixml	Import-Clixml [-Path]...
Cmdlet	Select-Xml	Select-Xml [-XPath] <S...

Comando	Descripción
ConvertTo-Xml	Este commandlet permite convertir los datos a formato XML
Export-Clixml	Realiza lo mismo que ConvertTo-Xml pero que también permite almacenar el resultado en un archivo que podrá leerse por Import-Clixml.
Import-Clixml	Como su nombre indica, este comando permite importar en una variable el contenido de un archivo XML. Pero no cualquier archivo XML. En efecto, el comando Import-Clixml no permite la importación de modelos genéricos de archivos XML, sino sólo los archivos XML generados por PowerShell por el comando Export-Clixml.
Select-Xml	Permite realizar peticiones en el seno de datos XML.

Como dijimos anteriormente, el comando Import-Clixml permite únicamente la importación de archivos XML generados por PowerShell, es decir los generados por el comando Export-Clixml. Para importar nuestro archivo, tendremos que realizar un transtypage (véase el capítulo Fundamentos). El archivo XML importado no es otro que el presentado anteriormente que está contenido en el archivo Libro.xml.

```
PS > $Libro = [xml](get-content Libro.xml)
PS > $Libro

Libro
-----
Libro
```

Ahora que el archivo XML está importado, será muy sencillo de recorrer, precisando cada nodo elegido, por ejemplo:

```
PS > $Libro.Libro

Título           SubTítulo       Autor           Capítulo
-----
Windows PowerShell  Guía de admin... {Autor, Autor} {Capítulo, Cap...

PS > $Libro.Libro.Título

Windows PowerShell

PS > $Libro.Libro.Título

Windows PowerShell

PS > $Libro.Libro.Autor

Apellido          AñoDeNacimiento Distinción
-----
Arnaud Petitjean  1974             MVP PowerShell
Robin Lemesle     1985             MVP PowerShell
```

También pueden efectuarse modificaciones, para ello bastará con indicar qué nuevo valor debe adoptar el nodo en

cuestión.

```
PS > $Libro.Libro.Titulo = 'El libro de PowerShell'
PS > $Libro.Libro
```

Título	SubTítulo	Autor	Capítulo
-----	-----	-----	-----
El libro de PowerShell	Guia de admin...	{Autor, Autor}	{Capítulo,Capítulo}

Acabamos de ver cómo explorar archivos XML personales, cosa muy práctica. Pero los comandos nativos PowerShell a propósito de XML, están principalmente dedicados a un uso de almacenamiento de información procedente y propio de PowerShell. Como por ejemplo el almacenamiento de objetos PowerShell. Esto es lo que vamos a ver. Este mecanismo también se denominada «serialización» / «deserialización» de objetos.

En el momento de la recuperación de información en una sesión PowerShell, el único medio de recuperarla posteriormente a través de otra sesión PowerShell, consiste en almacenar la información en un archivo. Pero resulta que el almacenamiento de información en un archivo provoca la pérdida de toda la interactividad asociada al objeto. Tomemos el caso concreto siguiente. Supongamos que queremos guardar la información devuelta por el commandlet Get-Process para analizarla más tarde. Uno de los métodos que se nos puede ocurrir consiste en almacenar esos datos en un archivo.txt.

```
PS > Get-Process > Process.txt
```

Cuando más tarde, en el momento elegido por el usuario para recuperar sus datos, el commandlet apropiado (Get-Content) sólo podrá facilitar una cadena de caracteres como valor de retorno. Con un objeto de tipo String y no una tabla de objeto Process (como lo devuelve el comando Get-Process). La aplicación de los métodos y el acceso a las propiedades del objeto es simplemente imposible.

```
PS > $Process = Get-Content Process.txt
PS > Foreach($Item in $Process){$Item.id}
```

<Vacio>

Es entonces en ese punto donde interviene el comando Export-Clixml. Con él, el objeto transmitido se almacenará en un formato XML que PowerShell sabrá interpretar de modo que pueda «reconstruir» los datos. Retomemos nuestro ejemplo:

```
PS > Get-Process | Export-Clixml Process.xml
```

Después de almacenar el objeto vía Export-Clixml, se realiza su importación con el commandlet Import-Clixml. Y una vez importado, los métodos y propiedades del objeto son de nuevo utilizables.

```
PS > $process = Import-Clixml Process.xml
PS > Foreach($Item in $Process){$Item.id}
```

```
3900
2128
1652
2080
1612
884
2656
2864
496
```

6. Export de datos como página HTML

Si la creación de páginas HTML le motiva para presentar algunos informes estratégicos o de otro tipo, entonces el commandlet **ConvertTo-HTML** está hecho para usted. En efecto, gracias a él, la generación de páginas Web su vuelve casi un juego de niños salvo por el tema del formato.

Veamos los parámetros disponibles de **ConvertTo-HTML**:

Parámetros	Descripción
Property <Object[]>	Propiedades del objeto pasadas como parámetro al escribir en la página HTML.
InputObject <PSObject>	Acepta un objeto como entrada.
Body <String[]>	Especifica el texto a incluir en el elemento <body>.
Head <String[]>	Especifica el texto a incluir en el elemento <head>.
Title <String>	Especifica el texto a incluir en el elemento <title>.

Un poco al estilo del comando **Export-csv**, el nombre de las propiedades servirá de título para cada columna del archivo HTML.

Ejemplo:

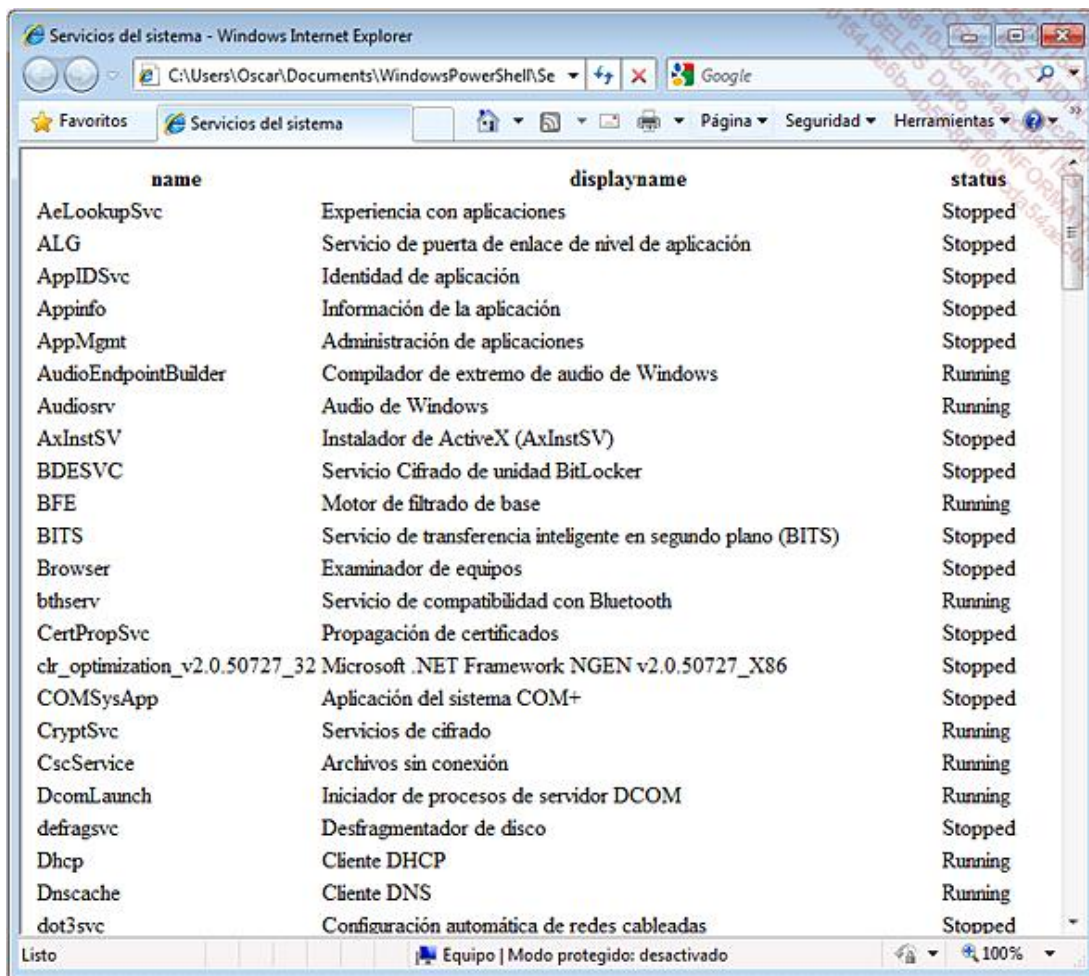
Lista de servicios del sistema.

```
PS > Get-Service |  
    ConvertTo-HTML -Property name, displayname, status -Title 'Servicios del  
sistema' | Out-File Services.htm
```

Este ejemplo nos permite crear una página HTML que contiene la lista de los servicios, sus nombres y su estado. El parámetro **-Title** se especifica con la finalidad de que la ventana tenga un título diferente al de por defecto. La totalidad se pasa a **out-File** quien creará el archivo **services.htm**.

Si no especificamos propiedades particulares, se escribirán todas las del objeto; el parámetro **-Property** tiene en cierto modo un rol de filtro.

Por otra parte, a diferencia de **Export-Csv**, **ConvertTo-HTML** necesita para funcionar plenamente que se le agregue un commandlet para escribir el flujo de texto de la generación de la página en un archivo. Por tanto, no olvide prestar atención al tipo de codificación del archivo resultante.



The screenshot shows a Windows Internet Explorer window titled 'Servicios del sistema - Windows Internet Explorer'. The address bar shows 'C:\Users\Oscar\Documents\WindowsPowerShell\Se'. The page displays a table of system services with three columns: 'name', 'displayname', and 'status'. The table lists various services such as AeLookupSvc, ALG, AppIDSvc, Appinfo, AppMgmt, AudioEndpointBuilder, Audiosrv, AxInstSV, BDESVC, BFE, BITS, Browser, bthserv, CertPropSvc, clr_optimization_v2.0.50727_32, COMSysApp, CryptSvc, CscService, DcomLaunch, defragsvc, Dhcp, Dnscache, and dot3svc, along with their display names and current status (Running or Stopped).

name	displayname	status
AeLookupSvc	Experiencia con aplicaciones	Stopped
ALG	Servicio de puerta de enlace de nivel de aplicación	Stopped
AppIDSvc	Identidad de aplicación	Stopped
Appinfo	Información de la aplicación	Stopped
AppMgmt	Administración de aplicaciones	Stopped
AudioEndpointBuilder	Compilador de extremo de audio de Windows	Running
Audiosrv	Audio de Windows	Running
AxInstSV	Instalador de ActiveX (AxInstSV)	Stopped
BDESVC	Servicio Cifrado de unidad BitLocker	Stopped
BFE	Motor de filtrado de base	Running
BITS	Servicio de transferencia inteligente en segundo plano (BITS)	Stopped
Browser	Examinador de equipos	Stopped
bthserv	Servicio de compatibilidad con Bluetooth	Running
CertPropSvc	Propagación de certificados	Stopped
clr_optimization_v2.0.50727_32	Microsoft .NET Framework NGEN v2.0.50727_X86	Stopped
COMSysApp	Aplicación del sistema COM+	Stopped
CryptSvc	Servicios de cifrado	Running
CscService	Archivos sin conexión	Running
DcomLaunch	Iniciador de procesos de servidor DCOM	Running
defragsvc	Desfragmentador de disco	Stopped
Dhcp	Cliente DHCP	Running
Dnscache	Cliente DNS	Running
dot3svc	Configuración automática de redes cableadas	Stopped

Creación de una página HTML



Para abrir este archivo directamente con Internet Explorer, será suficiente escribir el comando siguiente:
`./services.htm O Invoke-Item Services.htm.`



Como .htm es una extensión conocida por Windows, éste abrirá el archivo con la aplicación que tenga asociada (por defecto, Internet Explorer).

Gracias al parámetro `-body`, podemos especificar el contenido adicional que aparecerá en el cuerpo de la página, justo antes de los datos del objeto.

Ejemplo:

Lista de los servicios del sistema con BODY.

```
PS >Get-Service |
    ConvertTo-HTML -Property name,displayname,status -Title 'Servicios
del sistema'
    -body '<CENTER><H2>Estado de los servicios del sistema</H2></CENTER>' |
    Out-File Services.htm
```


Windows Internet Explorer - Servicios del sistema

Google

Favoritos Servicios del sistema

Estado de los servicios del sistema

name	displayname	status
AeLookupSvc	Experiencia con aplicaciones	Stopped
ALG	Servicio de puerta de enlace de nivel de aplicación	Stopped
AppIDSvc	Identidad de aplicación	Stopped
Appinfo	Información de la aplicación	Stopped
AppMgmt	Administración de aplicaciones	Stopped
AudioEndpointBuilder	Compilador de extremo de audio de Windows	Running
Audiosrv	Audio de Windows	Running
AxInstSV	Instalador de ActiveX (AxInstSV)	Stopped
BDESVC	Servicio Cifrado de unidad BitLocker	Stopped
BFE	Motor de filtrado de base	Running
BITS	Servicio de transferencia inteligente en segundo plano (BITS)	Stopped
Browser	Examinador de equipos	Stopped
bthserv	Servicio de compatibilidad con Bluetooth	Running
CertPropSvc	Propagación de certificados	Stopped
clr_optimization_v2.0.50727_32	Microsoft .NET Framework NGEN v2.0.50727_X86	Stopped
COMSysApp	Aplicación del sistema COM+	Stopped
CryptSvc	Servicios de cifrado	Running
CscService	Archivos sin conexión	Running
DcomLaunch	Iniciador de procesos de servidor DCOM	Running
defragsvc	Desfragmentador de disco	Stopped

Listo Equipo | Modo protegido: desactivado 100%

Creación de una página HTML con título

Ejemplo:

Lista de los servicios del sistema formateado con CSS.

Todavía mejor, procederemos esta vez a enmarcar nuestra tabla gracias a las hojas de estilo en cascada (*Cascading Style Sheets*). Para ello, hemos creado la hoja de estilo siguiente, que hemos nombrado *Style.css*:

```
<style type='text/css'>
  table {
    border: medium solid #000000;
    border-collapse: collapse;
  }
  td, th {
    border: thin solid #6495ed;
  }
</style>
```

Vamos a tener que incluir este archivo en el elemento **HEAD** de nuestro archivo HTML, como mostramos a continuación:

```
PS > $CSS = Get-Content Style.css
PS > Get-Service |
  ConvertTo-HTML -Property name,displayname,status -Title 'Servicios del
sistema'
  -Head $CSS -Body '<CENTER><H2>Estado de los servicios del sistema</H2>
</CENTER>' | Out-File Services.htm
```

name	displayname	status
AeLookupSvc	Experiencia con aplicaciones	Stopped
ALG	Servicio de puerta de enlace de nivel de aplicación	Stopped
AppIDSvc	Identidad de aplicación	Stopped
Appinfo	Información de la aplicación	Stopped
AppMgmt	Administración de aplicaciones	Stopped
AudioEndpointBuilder	Compilador de extremo de audio de Windows	Running
Audiosrv	Audio de Windows	Running
AxInstSV	Instalador de ActiveX (AxInstSV)	Stopped
BDESVC	Servicio Cifrado de unidad BitLocker	Stopped
BFE	Motor de filtrado de base	Running
BITS	Servicio de transferencia inteligente en segundo plano (BITS)	Stopped
Browser	Examinador de equipos	Stopped
bthserv	Servicio de compatibilidad con Bluetooth	Running
CertPropSvc	Propagación de certificados	Stopped
clr_optimization_v2.0.50727_32	Microsoft .NET Framework NGEN v2.0.50727_X86	Stopped
COMSysApp	Aplicación del sistema COM+	Stopped
CryptSvc	Servicios de cifrado	Running
CscService	Archivos sin conexión	Running
DcomLaunch	Iniciador de procesos de servidor DCOM	Running
defragsvc	Desfragmentador de disco	Stopped
Dhcp	Cliente DHCP	Running

Creación de una página HTML - Vista de una tabla

Para terminar con este comando un último ejemplo podría ser aplicar un color para cada línea de nuestra tabla. Podríamos así diferenciar los servicios en ejecución de los otros.

Ejemplo:

Lista de los servicios del sistema con análisis del contenido

```
PS > Get-Service |
  ConvertTo-Html -Property name,displayname,status -Title 'Servicios del
sistema'
-Body '<CENTER><H2>Estado de los servicios del sistema</H2></CENTER>' |
foreach {
  if($_ -match '<td>Running</td>')
  {
    $_ -replace '<tr>', '<tr bgcolor=#DDDDDD>'
  }
  elseif($_ -match '<td>Stopped</td>')
  {
    $_ -replace '<tr>', '<tr bgcolor= #6699FF>'
  }
  else
  {
    $_
  }
}
```

En este ejemplo, cuando un servicio está arrancado, se sustituye la etiqueta `tr` (indicando la línea) por la misma etiqueta `tr` con la instrucción adicional para mostrar un fondo de color (`bgcolor=#codigoColor`).



name	displayname	status
AeLookupSvc	Experiencia con aplicaciones	Stopped
ALG	Servicio de puerta de enlace de nivel de aplicación	Stopped
AppIDSvc	Identidad de aplicación	Stopped
Appinfo	Información de la aplicación	Stopped
AppMgmt	Administración de aplicaciones	Stopped
AudioEndpointBuilder	Compilador de extremo de audio de Windows	Running
Audiosrv	Audio de Windows	Running
AxInstSV	Instalador de ActiveX (AxInstSV)	Stopped
BDESVC	Servicio Cifrado de unidad BitLocker	Stopped
BFE	Motor de filtrado de base	Running
BITS	Servicio de transferencia inteligente en segundo plano (BITS)	Stopped
Browser	Examinador de equipos	Stopped
bthserv	Servicio de compatibilidad con Bluetooth	Running
CertPropSvc	Propagación de certificados	Stopped
clr_optimization_v2.0.50727_32	Microsoft .NET Framework NGEN v2.0.50727_X86	Stopped
COMSysApp	Aplicación del sistema COM+	Stopped
CryptSvc	Servicios de cifrado	Running
CscService	Archivos sin conexión	Running
DcomLaunch	Iniciador de procesos de servidor DCOM	Running
defragsvc	Desfragmentador de disco	Stopped

Creación de una página HTML - Vista de una tabla (bis)

7. Export de datos con Out-GridView

Con PowerShell v2, aparece un nuevo commandlet gráfico: se trata de `out-GridView`. Gracias a él vamos a poder visualizar los datos de forma gráfica (parecido a lo que acabamos de hacer anteriormente con una página HTML, pero sin esfuerzo) y de manera interactiva. La interactividad de la tabla se encuentra en la posibilidad de hacer clic en el título de las columnas para efectuar una clasificación de los datos por orden alfabético. Otra forma de interactividad es una función de búsqueda integrada en la tabla. Esto resulta práctico cuando los datos son abundantes y se busca uno en particular.

Ejemplo:

Lista de los servicios en ejecución.

```
PS > Get-Service | Out-GridView
```


Status	Name	DisplayName
Running	PlugPlay	Plug and Play
Running	Power	Energía
Running	ProfSvc	Servicio de perfil de usuario
Running	RpcEptMapper	Asignador de extremos de RPC
Running	RpcSs	Llamada a procedimiento remoto (RPC)
Running	SamSs	Administrador de cuentas de seguridad
Running	Schedule	Programador de tareas
Running	SENS	Servicio de notificación de eventos de sistema
Running	ShellHWDetection	Detección de hardware shell
Running	Spooler	Cola de impresión
Running	SR_Service	Check Point VPN-1 Secureremote service
Running	SR_Watchdog	Check Point VPN-1 Secureremote watchdog
Running	SSDPsrv	Detección SSDP
Running	SysMain	Superfetch
Running	Themes	Temas
Running	TrkWks	Cliente de seguimiento de vínculos distribuidos
Running	upnphost	Dispositivo host de UPnP
Running	UxSms	Administrador de sesión del Administrador de ventanas de...
Running	WdiServiceHost	Host del servicio de diagnóstico
Running	WinDefend	Windows Defender
Running	WinHttpAutoProxySvc	Servicio de detección automática de proxy web WinHTTP

Utilización de Out-GridView para mostrar la lista de los servicios

En este caso, hemos hecho clic en la columna de Status para clasificar los resultados según el estado de los servicios (Running, Stopped, etc.). Observe que encima de las columnas se encuentra la zona de búsqueda en la cual el texto por defecto es «filtro».

Ejemplo 2:

Vista gráfica del contenido de un archivo CSV

```
PS > Import-Csv usuarios.csv | Out-GridView
```

Apellido	Nombre	Dominio	UltimaConexion
Lemesle	Robin	powershell-scripting.com	20/09/2009
Petitjean	Arnaud	powershell-scripting.com	21/09/2009
Teixeira	Jessica	powershell-scripting.com	20/09/2009

