

Tabla de contenido

FORMATO DE LOS COMANDOS POWERSHELL	2
OBTENER INFORMACIÓN	3
TUBERÍAS Y REDIRECCIÓN	5
MOSTRANDO ARCHIVOS	5
FORMATEANDO LA SALIDA	6
FILTRANDO RESULTADOS	7
ADMINISTRAR LA UBICACIÓN ACTUAL	10
Obtener la ubicación actual (Get-Location)	10
Establecer la ubicación actual (Set-Location)	10
Almacenar y recuperar ubicaciones recientes (Push-Location y Pop-Location)	11
USANDO ARCHIVOS DE TEXTO	11
TRABAJAR CON ARCHIVOS Y CARPETAS	12
Enumerar archivos y carpetas (Get-ChildItem)	12
Manipular elementos directamente	15
Crear nuevos elementos (New-Item)	15
Ver el contenido (Get-Content)	16
Cambiar nombres de elementos existentes (Rename-Item)	16
Eliminar elementos (Remove-Item)	16
Mover elementos (Move-Item)	17
Copiar elementos (Copy-Item)	17
Ejecutar elementos (Invoke-Item)	17
Imprimir elementos (Out-Printer)	18
ALIAS DE COMANDOS	19
PROCESOS Y SERVICIOS	21
LENGUAJE DE SCRIPT EN POWERSHELL	21
SEPARACIÓN DE COMANDOS	21
COMENTARIOS	22
OBTENER AYUDA	22
VARIABLES	22
VARIABLES PREDEFINIDAS	23
CONSTANTES	24

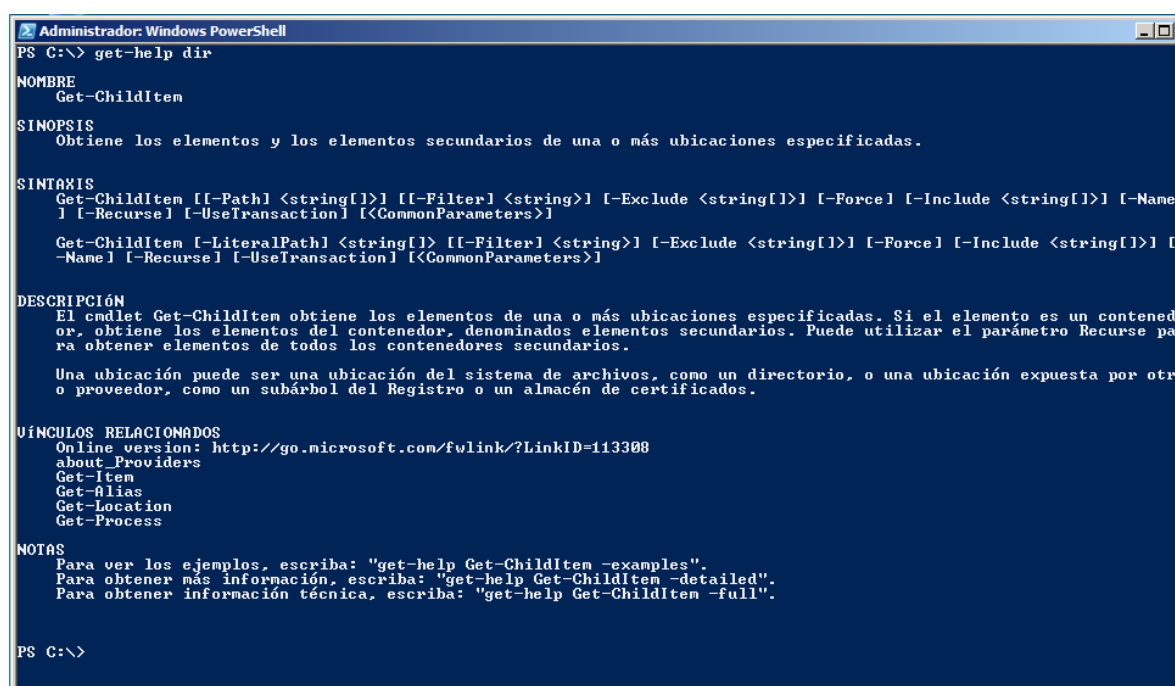
FORMATO DE LOS COMANDOS POWERSHELL

Los comandos de Powershell se denominan **cmdlets** (commandlets).

Estos comandos tienen una sintaxis particular que se compone de un verbo, un sustantivo y opciones y argumentos:

verbo-sust -parametro1 argumento1 -parametro2 argumento2 ...

Como se puede apreciar, el verbo y el sustantivo están separados por un guion (-) y los parámetros y argumentos por espacios.



```
Administrador: Windows PowerShell
PS C:\> get-help dir

NOMBRE
    Get-ChildItem

SINOPSIS
    Obtiene los elementos y los elementos secundarios de una o más ubicaciones especificadas.

SINTAXIS
    Get-ChildItem [[-Path] <string[]>] [[-Filter] <string>] [-Exclude <string[]>] [-Force] [-Include <string[]>] [-Name]
    [-Recurse] [-UseTransaction] [[<CommonParameters>]]

    Get-ChildItem [-LiteralPath] <string[]> [[-Filter] <string>] [-Exclude <string[]>] [-Force] [-Include <string[]>] [-Name]
    [-Recurse] [-UseTransaction] [[<CommonParameters>]]

DESCRIPCIÓN
    El cmdlet Get-ChildItem obtiene los elementos de una o más ubicaciones especificadas. Si el elemento es un contenedor, obtiene los elementos del contenedor, denominados elementos secundarios. Puede utilizar el parámetro Recurse para obtener elementos de todos los contenedores secundarios.

    Una ubicación puede ser una ubicación del sistema de archivos, como un directorio, o una ubicación expuesta por otro proveedor, como un subárbol del Registro o un almacén de certificados.

VÍNCULOS RELACIONADOS
    Online version: http://go.microsoft.com/fwlink/?LinkID=113308
    about_Providers
    Get-Item
    Get-Alias
    Get-Location
    Get-Process

NOTAS
    Para ver los ejemplos, escriba: "get-help Get-ChildItem -examples".
    Para obtener más información, escriba: "get-help Get-ChildItem -detailed".
    Para obtener información técnica, escriba: "get-help Get-ChildItem -full".

PS C:\>
```

Ejemplos:

get-history

Muestra una lista de los últimos comandos (por defecto 32)

get-history -count 10

Muestra los últimos 10 comandos ejecutados

get-history 7

Muestra solamente el comando número 7 de la lista

get-history -id 7

Muestra solamente el comando número 7 de la lista

Si necesitáramos una ayuda más amplia (con los parámetros, por ejemplo) utilizaremos:

PowerShell en Windows Server

```
PS>get-help dir -detailed
```

Y si queremos obtener toda la ayuda sobre “dir” escribimos:

```
PS>get-help dir -full
```

Si queremos obtener ejemplos:

```
PS>get-help dir -Examples
```

Hay otras maneras, conocidas en el entorno Unix, para obtener ayuda en PowerShell como son: “man” y “help”.

Los cmdlets según el verbo utilizado tienen una tarea específica:

get -> obtener datos
set -> establecer o modificar datos
format -> aplicar formato a los datos
out -> enviar la salida a un destino

EJERCICIO: Ejecuta los comandos siguientes en la consola de Powershell e indica qué realizan:

```
PS>Get-help
PS>Get-command
PS>Get-Process
PS>Get-Service
PS>Get-Date
PS>Get-Command *.exe
```

OBTENER INFORMACIÓN

Es muy importante manejar el sistema de ayuda de Powershell. Para acceder a la ayuda disponemos de varios comandos:

get-help

Cmdlet para acceder a la ayuda. Se puede solicitar ayuda sobre cmdlets concretos o sobre temas de ayuda (por ejemplo: `about_arithmetic_operators`). Si no se especifica otro cmdlet muestra ayuda sobre el propio `get-help`.

Opciones:

-name cmdlet

Muestra la ayuda de un cmdlet determinado (se puede omitir)

-detailed

Muestra más detalle

-full

Muestra la ayuda completa

-examples

Muestra ejemplos de uso del cmdlet

Ejemplos:

get-help

Muestra un resumen de cmdlet get-help

get-help -detailed

Muestra más detalles del cmdlet get-help

get-help -full get-command

Muestra la ayuda completa del comando get-command

get-help -examples get-command

Muestra ejemplos de uso del cmdlet get-command

get-help about_arithmetic_operators

Muestra ayuda sobre los operadores aritméticos

get-help -examples get-command

Muestra ejemplos de uso del cmdlet get-command

get-help about_arithmetic_operators

Muestra ayuda sobre los operadores aritméticos

get-command

Muestra información básica sobre los cmdlets. Se suele utilizar para obtener listas de cmdlets.

Opciones:**-name cmdlet**

Muestra la ayuda de un cmdlet determinado (se puede usar *)

-verb verbo

Muestra la ayuda de los comandos con el verbo indicado

-noun sustantivo

Muestra la ayuda de los comandos con el sustantivo indicado

Ejemplos:

get-command

Muestra una lista de los cmdlets

get-command -name get-a*

Muestra los cmdlets que comiencen con get-a

get-command -verb set

Muestra los cmdlets con el verbo set

get-command -noun alias

Muestra los cmdlets con el sustantivo alias

get-member

Todo lo que devuelve un cmdlet es un objeto. Un objeto es una forma de almacenar información, en la que se asocia a cada elemento una serie de datos denominados propiedades y una serie de acciones denominadas métodos. Con get-member podemos obtener la lista de métodos y propiedades de los objetos proporcionados por otros cmdlets usando el mecanismo de tubería (|).

PowerShell en Windows Server

Opciones:

-name nombre

Especifica los miembros de los que obtener información

-membertype tipo

Indica el tipo de objetos sobre los que obtener información

Usaremos fundamentalmente los tipos: Method y Property

Ejemplos:

```
ls | get-member
```

Muestra los miembros resultantes de un cmdlet ls

```
ls | get-member -membertype property
```

Muestra solo las propiedades del resultado de ls

TUBERÍAS Y REDIRECCIÓN

El mecanismo de tubería conocido en los sistemas Linux funciona de igual forma en Powershell, pero su uso es aun más intensivo, ya que la información que muestran los comandos, el formato de salida, etc, se controla usando este mecanismo.

Además las redirecciones se hacen más necesarias al disponer solo del comando more para hacer visible las salidas de comandos que ocupan más de una pantalla.

Ejemplos:

```
ls c:\Windows | more
```

```
ls c:\Windows > archivos.txt
```

```
ls c:\ >> archivos.txt
```

```
get-command -noun alias | get-help -examples
```

Obtiene ejemplos de los comandos con el sustantivo alias

MOSTRANDO ARCHIVOS

Para mostrar la lista de archivos los comandos tradicionales son ls y dir. El cmdlet al que hacemos referencia al ejecutar ls o dir en Powershell se denomina get-childitem, por lo que está claro que es más cómodo usar sus alias.

get-childitem

Muestra los archivos de una ubicación especificada.

Opciones:

-path ubicacion

Muestra la lista de archivos de ubicación (se puede omitir)

-name nombre

Muestra los archivos especificados en nombre

-include patrón

Solo incluye los archivos que cumplen el patrón

-exclude patrón

Excluye los archivos que cumplen el patrón

-recurse

Muestra también las subcarpetas

Ejemplos:

get-childitem

Muestra la lista de archivos de la carpeta actual

get-childitem -recurse

Muestra además la lista de archivos de las subcarpetas

get-childitem -include *.txt

Solo incluye los archivos .txt

FORMATEANDO LA SALIDA

Powershell proporciona una serie de cmdlets que permiten personalizar la salida por pantalla de otros cmdlets. Se utilizan principalmente recibiendo a través de una tubería el resultado de un cmdlet. Para aprovechar al máximo sus posibilidades es necesario conocer los elementos del resultado del cmdlet a personalizar mediante get-member.

format-list

Formato de lista. Cada elemento de cada objeto del resultado se muestra en una línea.

Opciones:

-property elementos

Muestra solo los elementos indicados.

format-wide

Formato en columnas. Se muestra un solo elemento de cada objeto.

Opciones:

-property elementos

Muestra solo el primero de los elementos indicados.

-groupby elemento

Agrupar por valor similar del elemento

-column n

Numero de columnas en pantalla

format-table

Formato de tabla. Se muestran los elementos de cada objeto en columnas.

Opciones:

-property elementos

Muestra solo el primero de los elementos indicados.

-groupby elemento

Agrupar por valor similar del elemento

Ejemplos:

ls c:\Windows | format-list

ls c:\Windows | format-list -property name, length

ls c:\Windows | format-wide -column 4

PowerShell en Windows Server

```
ls c:\Windows | format-wide -groupby extension
ls c:\Windows | format-table -property name, length, extension
ls c:\Windows | format-table -groupby extensión
```

sort-object

Ordena la salida.

Opciones:

-property elemento

Ordena por el elemento indicado

-descending

Usa orden descendente

Además de los anteriores, el cmdlet “out” permite redirigir la salida de datos. Cuando utilicemos algunas de sus opciones, necesitaremos aplicar el “operador de canalización”, con el símbolo “|”, para que la acción tenga su efecto en los datos de entrada.

Tenemos cuatro modalidades con el cmdlet “out”, que son:

out-host -> dirige los datos de salida a la pantalla

out-printer -> dirige los datos de salida a la impresora

out-null -> para descartar datos innecesarios producto de una ejecución

out-file -> para almacenar en un fichero los datos de salida

Veamos un ejemplo con “out-host”. Supongamos, que necesitamos dirigir los datos obtenidos sobre los comandos de una manera paginada, entonces podemos aplicar la siguiente sintaxis:

```
PS>Get-Command | out-host -paging
```

Con esto logramos que por pantalla veamos la lista de todos los comandos y de una manera paginada.

FILTRANDO RESULTADOS

Cuando se muestra una lista de resultados, siempre es interesante la posibilidad de filtrarlos usando sus propiedades.

Para filtrar resultados en Powershell se utiliza el cmdlet where-object a través del mecanismo de tubería.

Un filtro consiste en una expresión condicional donde se hace referencia a propiedades del objeto. Para expresar los filtros se utilizan operadores de comparación y lógicos.

Operadores de comparación

Con los operadores de comparación se expresan preguntas como .es mayor?, .es igual?, etc. Su resultado es un valor lógico que puede ser verdadero (true) o falso (false).

Estos son los operadores de comparación mas utilizados:

-eq	Es igual
-ne	Es diferente

PowerShell en Windows Server

-gt	Es mayor
-ge	Es mayor o igual
-lt	Es menor
-le	Es menor o igual
-like	Coincide con una expresión regular (con caracteres comodín)
-notlike	No coincide con una expresión regular
-match	Con propiedades que almacenan varios valores, si coincide uno
-notmatch	Con varios valores almacenados, si no coincide ninguno

Más información:

get-help about_comparison

Operadores lógicos

Los operadores lógicos permiten crear expresiones donde se evalúan varias comparaciones. Casos como .se cumple esto y aquello? , .se cumple esto o eso?, etc. Estos son los operadores de comparación más utilizados:

-and	y
-or	o
-not	no

Más información:

get-help about_logic

where-object

Cmdlet para filtrar resultados. Se usa fundamentalmente recibiendo a través de la tubería el resultado de otro cmdlet.

Sintaxis:

cmdlet | where-object { expresión_lógica }

Para hacer referencia a las propiedades de los objetos recibidos a través de la tubería de usa la siguiente sintaxis:

\$_propiedad

Ejemplos:

ls | where-object { \$_.length -gt 1000000 }
Filtrar los archivos con tamaño superior a 1MB

ls | where-object { \$_.lastwritetime -gt (get-date "2014-05-01") }
Filtrar los archivos modificados desde el 1 de mayo de 2014

ls | where { (\$_.length -ge 1000) -and (\$_.length -le 5000) }
Filtrar los archivos con tamaño entre 1KB y 5KB

PowerShell en Windows Server

Se puede usar where en vez de where-object (es un alias)

```
ls | ? { $_.fullname -like "*.pdf" }
```

Filtra los archivos que terminen en .pdf

También se puede usar ? (es otro alias)

```
ls | ? { $_.fullname -notlike "*.pdf" }
```

Filtra los archivos que no terminen en .pdf

```
ls | ? { -not ($_.length -gt 100000) }
```

Filtra los archivos que no tengan el tamaño superior a 100KB

```
ls | where { ($_.length -lt 1000) -or ($_.length -ge 5000) }
```

Filtra los archivos con tamaño menor que 1KB o mayor que 5KB

```
ls | ? { $_.attributes -match "readonly" }
```

Filtra los archivos que tienen entre sus atributos el de solo lectura.

EJERCICIOS:

1. Obtener la lista de los archivos .pdf de tu disco C: que no empiecen por a, b o c.
2. Obtener la lista de los archivos de tu HOME y sus subcarpetas modificados desde el 15 de abril.
3. Obtener la lista de los directorios de tu carpeta HOME (C:\Users\usuario).
4. Obtener la lista de directorios de tu carpeta HOME que contengan archivos ocultos.
5. Obtener la lista de archivos creados durante el curso que ocupen mas de 10MB. Debe mostrarlos ordenados de forma descendente por tamaño.
6. Obtener la lista de los archivos de tu carpeta HOME menos los directorios. Mostrar solo nombre de archivo y atributos.
7. Obtener la lista de todos los archivos menos los .pdf de tu carpeta HOME y sus subcarpetas. Mostrar solo el nombre con cuatro columnas.
8. Obtener las lista de los archivos de tu carpeta HOME y sus subcarpetas que contengan la palabra som. Mostrar los resultados en formato lista con las propiedades: nombre, tamaño y atributos.
9. Obtener las lista de los archivos de tu carpeta HOME y sus subcarpetas que no contengan la palabra som. Mostrar los resultados agrupados por extensión.
10. Obtener la lista de los archivos de tu carpeta HOME y sus subcarpetas que empiecen por r o tengan extensión .pdf. Mostrarlos ordenados por tamaño ascendente.

ADMINISTRAR LA UBICACIÓN ACTUAL

Windows PowerShell utiliza el sustantivo Location para hacer referencia al directorio de trabajo e implementa un conjunto de cmdlets para examinar y manipular la ubicación.

Obtener la ubicación actual (Get-Location)

Para determinar la ruta de acceso a la ubicación de directorio actual, escriba el comando **Get-Location**:

```
PS> Get-Location
Path
----
C:\Documents and Settings\PowerUser
```

Nota:

El cmdlet Get-Location es similar al comando pwd del shell BASH. El cmdlet Set-Location es similar al comando cd de Cmd.exe.

Establecer la ubicación actual (Set-Location)

El comando **Get-Location** se utiliza con el comando **Set-Location**. El comando **Set-Location** permite especificar la ubicación de directorio actual.

```
PS> Set-Location -Path C:\Windows
```

Después de escribir el comando, observará que no recibe ninguna información directa sobre el efecto del comando. La mayoría de los comandos de Windows PowerShell que realizan una acción generan pocos resultados o ninguno, ya que éstos no siempre son útiles. Para comprobar que se ha producido el cambio correcto de directorio al escribir el comando **Set-Location**, incluya el parámetro **-PassThru** al escribir el citado comando:

```
PS> Set-Location -Path C:\Windows -PassThru
Path
----
C:\WINDOWS
```

El parámetro **-PassThru** se puede utilizar con muchos comandos Set de Windows PowerShell para obtener información sobre el resultado en los casos en que no se muestran resultados de forma predeterminada.

Se puede especificar rutas de acceso relativas a la ubicación actual de la misma manera que lo haría en la mayoría de los shells de comandos de UNIX y Windows, de modo que la carpeta actual se representa mediante un punto (.) y el directorio principal de la ubicación actual se representa mediante dos puntos (..).

Por ejemplo, si se encuentra en la carpeta **C:\Windows**, un punto (.) representa **C:\Windows** y dos puntos (..) representan **C:**. Para cambiar de la ubicación actual a la raíz de la unidad C:, se escribe:

PowerShell en Windows Server

```
PS> Set-Location -Path .. -PassThru
Path
----
C:\
```

Puede escribir Set-Location o usar cualquiera de los alias integrados de Windows PowerShell para Set-Location (cd, chdir, sl). Por ejemplo:

```
cd -Path C:\Windows
chdir -Path .. -PassThru
sl -Path HKLM:\SOFTWARE -PassThru
```

Almacenar y recuperar ubicaciones recientes (Push-Location y Pop-Location)

Al cambiar de ubicación, resulta útil registrar la ubicación anterior y poder volver a ella. El cmdlet **Push-Location** de Windows PowerShell crea un historial ordenado (una "pila") de rutas de acceso a directorios en las que ha estado y el cmdlet **Pop-Location** complementario permite retroceder por este historial.

Por ejemplo, Windows PowerShell se inicia normalmente en el directorio principal del usuario.

```
PS> Get-Location

Path
----
C:\Documents and Settings\PowerUser
```

USANDO ARCHIVOS DE TEXTO

En determinadas ocasiones es bastante útil disponer en un archivo de texto elementos sobre los que después realizaremos ciertas operaciones. Para conseguirlo es necesario usar el cmdlet get-content combinado con foreach-object.

get-content

Muestra el contenido de un archivo de texto.

Ejemplos:

```
get-content lista.txt
Muestra el contenido de lista.txt
get-content lista.txt | % { rm $_.tostring() }
Borra los archivos de la lista contenida en lista.txt
```

Para usar el contenido se usa el método tostring() para convertirlo en un formato adecuado para los cmdlets

TRABAJAR CON ARCHIVOS Y CARPETAS

Windows PowerShell utiliza el sustantivo **Item** para hacer referencia a elementos que se encuentran en una unidad de Windows PowerShell. Cuando utilice el proveedor FileSystem de Windows PowerShell, un **Item** puede ser un archivo, una carpeta o la unidad de Windows PowerShell. La enumeración y el uso de estos elementos constituyen una importante tarea básica en la mayoría de las configuraciones administrativas y, por este motivo, queremos tratar detenidamente estas tareas.

Enumerar archivos y carpetas (Get-ChildItem)

Dado que obtener una colección de elementos de una ubicación concreta es una tarea muy habitual, el cmdlet **Get-ChildItem** se ha diseñado específicamente para devolver todos los elementos incluidos en un contenedor, como por ejemplo una carpeta.

Si desea devolver todos los archivos y carpetas que contiene directamente la carpeta C:\Windows, se escribe:

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

La lista es similar a la que se mostraría al escribir el comando **dir** en **Cmd.exe** o el comando **ls** en un shell de comandos de UNIX. Se puede usar parámetros del cmdlet **Get-ChildItem** para crear listas muy complejas. Analizaremos varios escenarios a continuación. Para ver la sintaxis del cmdlet **Get-ChildItem**, se escribe:

```
PS> Get-Command -Name Get-ChildItem -Syntax
```

Estos parámetros se pueden combinar y comparar para obtener resultados muy personalizados.

Crear una lista de todos los elementos contenidos (-Recurse)

Para ver tanto los elementos incluidos en una carpeta de Windows como los elementos contenidos en las subcarpetas, utilice el parámetro **Recurse** de **Get-ChildItem**. La lista mostrará todo el contenido de la carpeta de Windows, así como los elementos incluidos en las subcarpetas. Por ejemplo:

```
PS> Get-ChildItem -Path C:\WINDOWS -Recurse
```

```
Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\WINDOWS
Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\WINDOWS\AppData
Mode                LastWriteTime         Length Name
----                -
-a---             2004-08-04   8:00 AM       1852416 AcGenral.dll
...
```

Filtrar elementos por nombre (-Name)

Para mostrar únicamente los nombres de los elementos, utilice el parámetro **Name** de **Get-Childitem**:

```
PS> Get-ChildItem -Path C:\WINDOWS -Name
addins
```

PowerShell en Windows Server

```
AppPatch
assembly
...
```

Forzar la presentación de los elementos ocultos (-Force)

Los elementos que no se muestran normalmente en el Explorador de Windows o en Cmd.exe tampoco se muestran en el resultado del comando `Get-ChildItem`. Para mostrar los elementos ocultos, utilice el parámetro `Force` de `Get-ChildItem`. Por ejemplo:

```
Get-ChildItem -Path C:\Windows -Force
```

Este parámetro recibe el nombre de `Force` porque invalida forzosamente el comportamiento normal del comando **Get-ChildItem**. El parámetro `Force` se usa a menudo y fuerza una acción que un cmdlet no realizaría normalmente, aunque no ejecutará ninguna acción que pueda poner en peligro la seguridad del sistema.

Usar caracteres comodín para buscar nombres de elementos

El comando **Get-ChildItem** acepta caracteres comodín en la ruta de acceso de los elementos que se van a enumerar. Dado que el motor de Windows PowerShell controla la búsqueda con caracteres comodín, todos los cmdlets que aceptan caracteres comodín utilizan la misma notación y tienen el mismo comportamiento en la búsqueda de coincidencias. La notación de caracteres comodín de Windows PowerShell es la siguiente:

- El asterisco (*) busca cero o más instancias de cualquier carácter.
- El signo de interrogación (?) busca exactamente un carácter.
- Los caracteres de corchete izquierdo ([) y corchete derecho (]) rodean un conjunto de caracteres de los que se van a buscar coincidencias.

A continuación se muestran algunos ejemplos.

Para buscar todos los archivos del directorio Windows que tengan la extensión **.log** y exactamente cinco caracteres en el nombre base, escriba el comando siguiente:

```
PS> Get-ChildItem -Path C:\Windows\?????.log
Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\Windows
Mode                LastWriteTime         Length Name
----                -
...
-a---             2006-05-11   6:31 PM         204276 ocgen.log
-a---             2006-05-11   6:31 PM          22365 ocmsn.log
...
-a---             2005-11-11   4:55 AM           64 setup.log
-a---             2005-12-15   2:24 PM         17719 VxSDM.log
...
```

Para buscar todos los archivos del directorio Windows que comiencen por la letra **x**, escriba:

PowerShell en Windows Server

```
Get-ChildItem -Path C:\Windows\x*
```

Para buscar todos los archivos cuyo nombre comience por la letra **x** o **z**, escriba:

```
Get-ChildItem -Path C:\Windows\[xz]*
```

Excluir elementos (-Exclude)

Puede excluir elementos específicos usando el parámetro **Exclude** de **Get-ChildItem**. Este parámetro permite aplicar filtros complejos en una sola instrucción.

Por ejemplo, suponga que desea buscar el archivo DLL del servicio de hora de Windows en la carpeta System32 y todo lo que recuerda del nombre del archivo DLL es que comienza por "W" y que contiene "32".

Una expresión como **w*32*.dll** encontrará todos los archivos DLL que cumplan las condiciones indicadas, pero también puede devolver los archivos DLL de compatibilidad con las versiones Windows 95 y Windows de 16 bits que tienen "95" o "16" en sus nombres. Para omitir los archivos cuyo nombre contenga estos números, use el parámetro **Exclude** con el patrón ***[9516]***:

```
PS> Get-ChildItem -Path C:\WINDOWS\System32\w*32*.dll -Exclude *[9516]*
Directory: Microsoft.PowerShell.Core\FileSystem::C:\WINDOWS\System32
Mode LastWriteTime Length Name
----
-a--- 2004-08-04 8:00 AM 174592 w32time.dll
-a--- 2004-08-04 8:00 AM 22016 w32topl.dll
-a--- 2004-08-04 8:00 AM 101888 win32spl.dll
-a--- 2004-08-04 8:00 AM 172032 wldap32.dll
-a--- 2004-08-04 8:00 AM 264192 wow32.dll
-a--- 2004-08-04 8:00 AM 82944 ws2_32.dll
-a--- 2004-08-04 8:00 AM 42496 wsnmp32.dll
-a--- 2004-08-04 8:00 AM 22528 wsock32.dll
-a--- 2004-08-04 8:00 AM 18432 wtsapi32.dll
```

Combinar parámetros de Get-ChildItem

Puede utilizar varios de los parámetros del cmdlet **Get-ChildItem** en un mismo comando. Antes de combinar parámetros, asegúrese de que entiende cómo se pueden realizar búsquedas de coincidencias con caracteres comodín. Por ejemplo, el siguiente comando no devuelve ningún resultado:

```
PS> Get-ChildItem -Path C:\Windows\*.dll -Recurse -Exclude [a-y]*.dll
```

No se obtienen resultados, aunque hay dos archivos DLL que comienzan por la letra "z" en la carpeta Windows.

El motivo por el que no se devuelven resultados es que hemos especificado el carácter comodín como parte de la ruta de acceso. Aunque el comando era recursivo, el cmdlet **Get-ChildItem** ha limitado los elementos a los incluidos en la carpeta Windows cuyo nombre termina en ".dll".

PowerShell en Windows Server

Para especificar una búsqueda recursiva de archivos cuyo nombre coincide con un patrón especial, utilice el parámetro **-Include**:

```
PS> Get-ChildItem -Path C:\Windows -Include *.dll -Recurse -Exclude [a-y]*.dll
```

```
Directory: Microsoft.Windows
PowerShell.Core\FileSystem::C:\Windows\System32\Setup
```

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a---	2004-08-04 8:00 AM	8261	zoneoc.dll

```
Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\Windows\System32
```

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a---	2004-08-04 8:00 AM	337920	zipfldr.dll

Manipular elementos directamente

Los elementos que se muestran en las unidades de Windows PowerShell, como los archivos y carpetas de las unidades del sistema de archivos, así como las claves del Registro incluidas en las unidades del Registro de Windows PowerShell, reciben el nombre de *elementos* en Windows PowerShell. Los cmdlets que permiten trabajar con estos elementos tienen el sustantivo **Item** en sus nombres.

El resultado del comando **Get-Command -Noun Item** muestra que hay nueve cmdlets **Item** en Windows PowerShell:

```
PS> Get-Command -Noun Item
```

CommandType	Name	Definition
-----	----	-----
Cmdlet	Clear-Item	Clear-Item [-Path] <String[]>...
Cmdlet	Copy-Item	Copy-Item [-Path] <String[]>...
Cmdlet	Get-Item	Get-Item [-Path] <String[]> ...
Cmdlet	Invoke-Item	Invoke-Item [-Path] <String[]>...
Cmdlet	Move-Item	Move-Item [-Path] <String[]>...
Cmdlet	New-Item	New-Item [-Path] <String[]> ...
Cmdlet	Remove-Item	Remove-Item [-Path] <String[]>...
Cmdlet	Rename-Item	Rename-Item [-Path] <String>...
Cmdlet	Set-Item	Set-Item [-Path] <String[]> ...

Crear nuevos elementos (New-Item)

Para crear un nuevo elemento en el sistema de archivos, utilice el cmdlet **New-Item**. Asimismo, incluya el parámetro **Path** con la ruta de acceso al elemento y el parámetro **ItemType** con el valor "file" o "directory".

Por ejemplo, para crear un directorio llamado "directorio1 " en el directorio C:\Temp, escriba:

```
PS> New-Item -Path c:\temp\directorio1 -ItemType Directory
```

PowerShell en Windows Server

Para crear un archivo, cambie el valor del parámetro **ItemType** a "file". Por ejemplo, para crear un archivo llamado "file1.txt" en el directorio "directorio1", escriba:

```
PS> New-Item -Path C:\Directorio1\file1.txt -ItemType file
```

Para crear un archivo no vacío usamos el parámetro **Value** "texto". Por ejemplo, para crear un archivo llamado "file2.txt" en el directorio "directorio1", con el texto "datos" escriba:

```
PS> New-Item -Path C:\Directorio1\file2.txt -ItemType file -Value "datos"
```

Ver el contenido (Get-Content).

Para ver el contenido de un archivo usamos el cmdlet **Get-Content**. Por ejemplo, para ver el contenido del archivo llamado "file2.txt" en el directorio "directorio1", escriba:

```
PS> Get-Content C:\Directorio1\file2.txt
```

Cambiar nombres de elementos existentes (Rename-Item)

El cmdlet **Rename-Item** permite cambiar el nombre de un archivo o una carpeta. El comando siguiente cambia el nombre del archivo **file1.txt** a **fileOne.txt**.

```
PS> Rename-Item -Path C:\Directorio1\file1.txt fileOne.txt
```

El cmdlet **Rename-Item** puede cambiar el nombre de un archivo o una carpeta, pero no puede mover un elemento.

Eliminar elementos (Remove-Item)

El cmdlet **Remove-Item** permite eliminar archivos y carpetas. Los cmdlets de Windows PowerShell que pueden realizar cambios irreversibles importantes, como es el caso de **Remove-Item**, suelen solicitar confirmación cuando se escriben sus comandos. Por ejemplo, si intenta quitar la carpeta **Directorio1**, se le pedirá que confirme el comando, ya que esta carpeta contiene archivos:

```
PS> Remove-Item C:\Directorio1
```

Dado que la respuesta predeterminada es **Yes**, para eliminar la carpeta y los archivos que contiene, deberá presionar la tecla **Entrar**. Para quitar la carpeta sin que se solicite confirmación, utilice el parámetro **-Recurse**.

```
PS> Remove-Item C:\temp\New.Directory -Recurse
```

Para eliminar los archivos txt de la carpeta actual por ejemplo, escribimos.

```
PS> Remove-Item *.txt
```


PowerShell en Windows Server

Mover elementos (Move-Item)

El cmdlet **Move-Item** permite mover un archivo o una carpeta.

Por ejemplo, el siguiente comando mueve el directorio Directorio1 desde el directorio C:\ a C:\Temp:. Para comprobar que se ha movido el elemento, incluya el parámetro **PassThru** del cmdlet **Move-Item**. Sin **PassThru**, el cmdlet **Move-Item** no muestra ningún resultado.

```
PS> Move-Item -Path C:\Directorio1 -Destination C:\Temp -PassThru
```

Copiar elementos (Copy-Item)

A diferencia de las operaciones de copia de otros shells, puede que el comportamiento del cmdlet **Copy-Item** de Windows PowerShell resulte inusual. **Cuando se copia un elemento de una ubicación a otra, Copy-Item no copia su contenido de forma predeterminada.**

Por ejemplo, si copia el directorio **Directorio1** de la unidad C: al directorio C:\Directorio2, el comando se ejecutará correctamente, pero los archivos del directorio Directorio1 no se copiarán.

```
PS> Copy-Item -Path C:\Directorio1 -Destination C:\Directorio2
```

Si muestra el contenido del directorio **C:\Directorio2**, se observará que no contiene ningún archivo:

```
PS> Get-ChildItem -Path C:\Directorio2
```

¿Por qué el cmdlet **Copy-Item** no copia el contenido en la nueva ubicación?

El cmdlet **Copy-Item** se ha diseñado para uso genérico, no sólo para copiar archivos y carpetas. Así, incluso al copiar archivos y carpetas, es posible que desee copiar únicamente el contenedor y no los elementos que contiene.

Para copiar todo el contenido de una carpeta, incluya el parámetro **Recurse** del cmdlet **Copy-Item** en el comando. Si ya ha copiado el directorio sin su contenido, agregue el parámetro **Force**, que permite sobrescribir la carpeta vacía. El parámetro **-PassThru** devuelve el resultado de la copia de cada elemento.

```
PS> Copy-Item -Path C:\Directorio1 -Destination C:\temp -Recurse -Force -  
Passthru
```

Ejecutar elementos (Invoke-Item)

Windows PowerShell utiliza el cmdlet **Invoke-Item** para realizar una acción predeterminada relativa a un archivo o una carpeta. Esta acción predeterminada está determinada por el controlador de aplicaciones predeterminado en el Registro; el efecto es el mismo que si se hace doble clic en el elemento en el Explorador de Windows.

PowerShell en Windows Server

Por ejemplo, ejecuta el siguiente comando:

```
PS> Invoke-Item C:\WINDOWS
```

Aparecerá una ventana del Explorador que se encuentra en C:\Windows, como si hubiera hecho doble clic en la carpeta C:\Windows.

Si invoca el archivo **Boot.ini** en un sistema anterior a Windows Vista:

```
PS> Invoke-Item C:\boot.ini
```

Si el tipo de archivo .ini está asociado al Bloc de notas, el archivo boot.ini se abrirá en esta aplicación

Imprimir elementos (Out-Printer)

Podemos imprimir un archivo desde el PowerShell usamos el cmdlet Out-Printer con los parámetros **-InputObject** nombre-archivo y **-Name** Nombre-Dispositivo.

Por ejemplo, suponga que ejecuta el siguiente comando:

```
PS> Out_Printer -InputObject file1 -Name Impresora1
```

EJERCICIOS:

1. Crear los siguientes directorios en el directorio raíz o principal de la unidad de disco:

AGORA\ASO
AGORA\ASO\WIN
AGORA\ASO\LINUX
AGORA\FOL
2. Crear un fichero no vacío llamado apuntes.doc en el directorio WIN de ASO.
3. Crear un fichero no vacío llamado practicas.doc en el directorio WIN de ASO
4. Crear un fichero sin contenido llamado apuntes_linux.txt en el directorio LINUX de ASO.
5. Visualizar el contenido del fichero practicas.doc.
6. Copiar los ficheros de WIN A FOL.
7. Copiar el fichero apuntes-linux.txt de LINUX a FOL con el nombre apuntes-fol.txt.

8. Cambiar el nombre del fichero practicas.doc de FOL por el nombre practicas-fol.doc.
9. Eliminar el fichero apuntes.doc del directorio WIN.
10. Visualizar los atributos de los archivos de WIN.
11. Copiar el directorio ASO con todo su contenido a AGORA con el nombre SO
12. Eliminar el directorio ASO con todo su contenido.
13. Mostrar los ficheros txt del árbol AGORA.
14. Mostrar los ficheros doc del árbol AGORA.
15. Mostrar los ficheros doc que no empiecen por la letra "p" del árbol AGORA.
16. Visualiza la fecha y hora actuales.
17. Estudia como cambiarlas.

ALIAS DE COMANDOS

Para hacer mas comodo el uso de Powershell disponemos del mecanismo de alias de cmdlets. Los alias de un cmdlet son diferentes formas de escribirlo. Por ejemplo, cuando ejecutamos un ls realmente estamos usando el cmdlet get-childitem.

get-alias

Obtiene la lista de alias definidos.

Opciones:

-name alias

Muestra el cmdlet asociado a un alias (se puede omitir)

-definition cmdlet

Muestra los alias de cmdlet

Ejemplos:

get-alias

Muestra la lista de todos los alias definidos

get-alias ls

Muestra el cmdlet asociado a ls

get-alias -definition get-childitem

Muestra los alias del cmdlet get-childitem

get-alias -definition *childitem

Muestra los alias de los cmdlets que terminen con childitem

get-alias w*

Muestra los cmdlet de los alias que empiecen por w

set-alias

Cambia un alias existente o lo crea si no existe. Los alias que viene predefinidos en el sistema son la mayoría de solo lectura, por lo que no se pueden modificar.

PowerShell en Windows Server

Opciones:

-name alias

Nombre del alias (se puede omitir)

-value cmdlet

Cmdlet asociado al alias

Ejemplos:

```
set-alias lista -value get-childitem
```

Define el alias lista para el cmdlet get-childitem

```
set-alias lista -value get-service
```

Modifica el alias lista para que se refiera a otro cmdlet, en este caso get-service

export-alias

Guarda las definiciones de alias actuales en un archivo de texto.

Ejemplo:

```
export-alias alias.txt
```

Guarda las definiciones de alias en alias.txt

import-alias

Recupera las definiciones de alias de un archivo de texto.

Ejemplo:

```
import-alias alias.txt
```

Recupera las definiciones de alias del archivo alias.txt

Cuando se crea un nuevo alias, este se pierde al cerrar la ventana de Powershell. Para hacerlos para un usuario se debe conocer el nombre del archivo que se ejecuta cada vez que abrimos un Powershell. Esto se hace comprobando el valor de una variable denominada profile.

\$profile

Muestra el nombre del archivo que se ejecuta cada vez que abrimos una Powershell.

Este archivo se puede editar con notepad (es probable que no exista si no lo hemos creado con anterioridad) e incluir los alias que necesitemos o un import-alias de un archivo de texto que debemos haber creado previamente con export-alias.

Además, para que este archivo se ejecute debemos cambiar la política de ejecución de scripts de Powershell, esto se realiza ejecutando el siguiente comando como Administrador.

set-executionpolicy

Modifica los permisos de ejecución de scripts.

Ejemplo:

```
set-executionpolicy Unrestricted
```

Permite la ejecución de scripts libremente

PROCESOS Y SERVICIOS

Disponemos de cmdlets para conocer los procesos en ejecución, terminarlos e iniciar nuevos procesos.

get-process Lista de procesos en ejecución

stop-process Termina un proceso. Se puede realizar a través del ID o el nombre

start-process Inicia un nuevo proceso, se suele usar su alias **start** para ejecutar programas

Desde la powershell podemos gestionar los servicios del sistema. Estos son los cmdlets asociados a los servicios más interesantes:

get-service Lista los servicios definidos en el sistema

start-service Inicia un servicio determinado

restart-service Reinicia un servicio determinado

stop-service Para un servicio determinado

LENGUAJE DE SCRIPT EN POWERSHELL

El lenguaje de Script que Powershell (**PSL**) pone a nuestra disposición incluye variables, bucles, condicionales, funciones y manejo de errores. Este lenguaje de script no es algo creado desde cero, sino que esta “inspirado” en los lenguajes shell de Unix como PERL, PHP, PYTHON...

SEPARACIÓN DE COMANDOS

Cada línea en un script de Windows Powershell es un comando. Un comando puede contener varios cmdlets separados por el símbolo “|”.

También podemos escribir varios comandos en una misma línea separándolos con “*punto y coma*” (;).

Cuando una línea de un comando es muy larga, podemos escribir el carácter “*comilla invertida*” (``) al final de la línea para indicar que en la siguiente línea continua el comando.

Ejemplo.

```
Get-Command | where-object `
{$_ .Name -like "get-*"}
```

Crea un script llamado “**separacion.ps1**” que contenga las dos líneas anteriores escribiendo el texto en un editor de textos.

Luego ejecuta el **script** de la forma siguiente:

PowerShell en Windows Server

PS> .\separacion.ps1

COMENTARIOS

Los comentarios en WPS se escriben utilizando el símbolo de *almohadilla* (#). Todo lo que se encuentre después de este carácter, no será interpretado por **Powershell**.

Ejemplo

#Esto es un comentario.

OBTENER AYUDA

También podemos obtener ayuda sobre algunos de las características del lenguaje de **script**. Por ejemplo, si queremos obtener ayuda sobre el bucle “for”, podemos escribir:

PS> get-help about_for

Si escribimos:

PS> get-help about

Nos mostrará la lista de todos los documentos “**about**”.

VARIABLES

Las variables comienzan con el caracter “\$”. Las variables en **WPS** pueden contener letras, números e incluso el carácter de subrayado (_). Las variables en **WPS**, como en todo lenguaje de programación, pueden ser de varios tipos.

Ejemplo 1

Si queremos crear una variable que contenga el número **23**, haremos lo siguiente:

\$numero = 23

o si queremos guardar una frase:

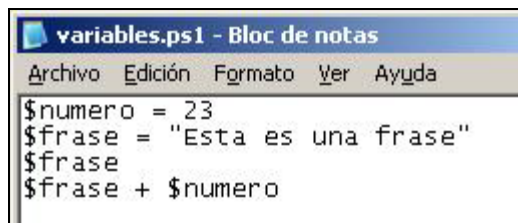
\$frase = “Esta es una frase”

y para mostrar el contenido de la variable basta con especificarla junto con el caracter “\$”.

\$frase

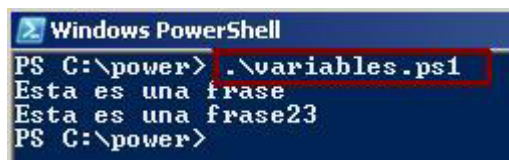
Crea un fichero llamado “*variables.ps1*” cuyo contenido podemos ver en la siguiente captura:

PowerShell en Windows Server



```
variables.ps1 - Bloc de notas
Archivo Edición Formato Ver Ayuda
$numero = 23
$frase = "Esta es una frase"
$frase
$frase + $numero
```

En la última línea se concatenan dos variables. El resultado de este **script** se muestra a continuación:



```
Windows PowerShell
PS C:\power> .\variables.ps1
Esta es una frase
Esta es una frase23
PS C:\power>
```

Las variables serán validas en el ámbito en el que han sido creadas o declaradas. Esto quiere decir que si hemos declarado una variable dentro de un bucle “**for**”, esta variable existirá o estará accesible siempre que estemos dentro del bucle “**for**”, una vez que haya finalizado el bucle o antes de comenzarlo, esta variable no existirá.

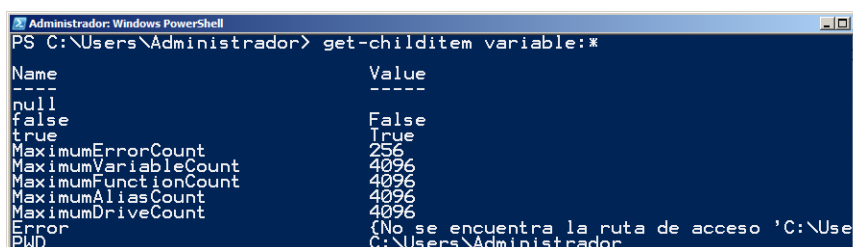
VARIABLES PREDEFINIDAS

Windows Powershell dispone de muchas variables predefinidas tambien llamadas “*variables integradas*” o “*variables internas*”.

\$true	Valor true.
\$false	Valor false.
\$home	El directorio home del usuario actual.
\$psHOME	Directorio de instalación de windows powershell.
\$host	Información de instalación del host.
\$error	Lista de los errores que han ocurrido desde que se ha iniciado WPS.

Para obtener una lista de todas las variables disponibles en la sesión actual de **Windows Powershell**, tanto las internas como las que hemos definido nosotros mismos:

>get-childitem variable:*



```
Administrador: Windows PowerShell
PS C:\Users\Administrador> get-childitem variable:*

Name                Value
----                -
null                False
false               True
true                256
MaximumErrorCount   4096
MaximumVariableCount 4096
MaximumFunctionCount 4096
MaximumAliasCount    4096
MaximumDriveCount    4096
Error                (No se encuentra la ruta de acceso 'C:\Use
PWD                 C:\Users\Administrador
```

Y si lo que queremos es obtener todas las variables cuyo nombre comience por “p”:

>get-childitem variable:p*

PowerShell en Windows Server

Pero para obtener información sobre las variables, disponemos de otro cmdlet “**get-variable**”.

CONSTANTES

Las constantes se definen para valores que no cambian. El comando a utilizar para crear una constante seria:

Set-Variable -name nombreconstante -option readonly -value contenido

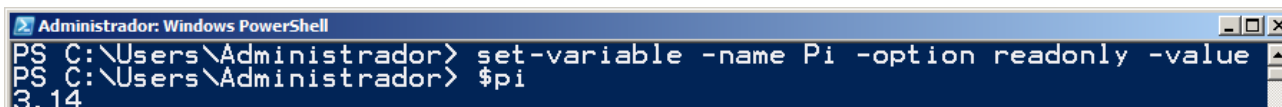
Ejemplo

>set-variable -name Pi -option readonly -value 3.14

En este caso no hemos utilizado el caracter \$. Con Set-Variable no lo utilizaremos. Sin embargo, si queremos trabajar con la constante, si que tendremos que utilizarlo.

Para ver el contenido de la constante Pi, haremos igual que si fuera una variable:

>\$Pi



```
Administrador: Windows PowerShell
PS C:\Users\Administrador> set-variable -name Pi -option readonly -value 3.14
PS C:\Users\Administrador> $pi
3.14
```

EJERCICIOS:

1. Definir un alias del comando para reiniciar el ordenador denominado reinicia. Hacer este alias persistente. Reinicia el ordenador de un compañero usando el alias.
2. Listar los archivos de c:\users\usuario y sus subcarpetas modificados hoy en una lista de ordenadores del aula.
3. Copiar solo los archivos (no las carpetas) modificados hoy en c:\users\usuario y sus subcarpetas dentro de una carpeta c:\users\usuario\seguridad (es necesario excluir esta carpeta de la copia).
4. Ejecuta el navegador Mozilla firefox, observa el id del proceso y páralo.
5. Escribe get-help about_while y crea un script powershell que muestre los números del 1 al 100.