

¿Qué es PowerShell?

PowerShell es un framework de Microsoft que se compone de dos partes:

1. Un intérprete de comandos.
2. Un lenguaje para la escritura de scripts.

Como principal diferencia con otros lenguajes de scripting destacaría que al estar basado en .NET, permite la orientación a objetos. Por tanto, al trabajar con objetos, los cuales vienen caracterizados por propiedades y métodos, el trabajo con PowerShell es razonablemente cómodo de escribir siendo al mismo tiempo muy potente al acceder de una manera estructurada a los elementos que componen el objeto.

Compatibilidad con CMD

Los comandos en PowerShell en realidad se llaman **cmdlets** (leído 'commandlets'). El término cmdlet proviene de *Command Applet*.

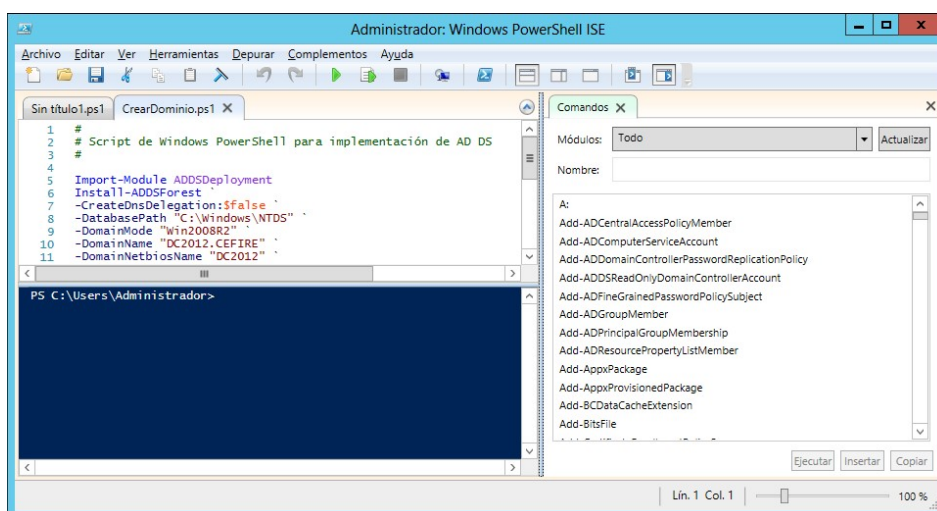
La versión 3.0 de PowerShell -implementada tanto en Windows 8 como en Windows Server 2012- recoge una cantidad vastísima de estos cmdlets. Sin embargo, en PowerShell siguen funcionando los comandos que utilizábamos con el clásico CMD, aunque en realidad en algunos casos son **alias** (es decir etiquetas que apuntan a cmdlets) y en otros son **funciones**, pero que en ambos casos ofrecen las mismas salidas que los comandos a los que estamos habituados. Por ejemplo, para obtener un listado de los elementos que tenemos en un directorio utilizábamos dir en CMD. Este comando también funciona en PowerShell, aunque en realidad apunta al cmdlet Get-ChildItem. Además, algunos comandos Unix como ls también están recogidos como alias en PowerShell. Si queremos saber a qué cmdlet apunta un alias escribiremos: Get-alias *comando_CMD*.

Versiones de PowerShell

En la actualidad existen tres versiones de PowerShell:

- 1.0: Apareció en 2006 incluyéndose posteriormente como una característica opcional en Windows Server 2008.
- 2.0: Integrada en Windows Server 2008 R2 y Windows 7 mejoró aspectos como la ejecución remota de los scripts, la aparición del entorno de desarrollo ISE (figura 4.2-1).

- 3.0: Integrada en Windows Server 2012 y Windows 8, sus mejoras más destacables respecto a las anteriores son la facilidad en la escritura de scripts mediante la inclusión de la funcionalidad *Intellisense* la cual muestra información sobre los cmdlets que estamos escribiendo de una manera automatizada, la persistencia de las conexiones con máquinas remotas aún en caso de fallos en la red, y el Script Explorer que nos permite acceder a un gran número de scripts de ejemplo.
- 4.0: Integrada en Windows Server 2012 R2 y Windows 8.1. Incluye alguna característica nueva respecto a su antecesora.
- 5.0: Integrada en Windows 10 y Windows Server 2016. Las principales mejoras se encuentran en las áreas de Configuración de estado deseado, seguridad, rendimiento, comunicación remota y mejoras del lenguaje. Existe una versión 5.1 que aparece con la actualización del aniversario en Windows 10.



Entorno de desarrollo ISE.

Si queremos conocer la versión de PowerShell que tenemos instalada en nuestro sistema, abriremos PowerShell .



A continuación escribiremos la siguiente instrucción:

```
> Get-Host | Select-Object Version
```

Y el sistema nos devolverá la versión instalada. El cmdlet Get-Host obtiene (*get*) información del entorno de ejecución. Como solo nos interesa la versión, la cual es una

de las propiedades del objeto devuelto, bastará con utilizar a través de una tubería el cmdlet *Select-Object* para especificar la propiedad del objeto con la que queremos quedarnos. Puede parecer una manera algo farragosa de trabajar, pero en los siguientes apartados iremos viendo con mayor detalle cómo operar con PowerShell explotando la potencia que nos va a proporcionar la orientación a objetos.

Estructura de los cmdlets

Como norma general, los cmdlets están formados por un verbo (describe la acción a realizar) y un nombre (indica el objeto sobre el que se aplica la acción) unidos mediante un guión, como por ejemplo los que hemos visto anteriormente: *Get-Host* o *Select-Object*.

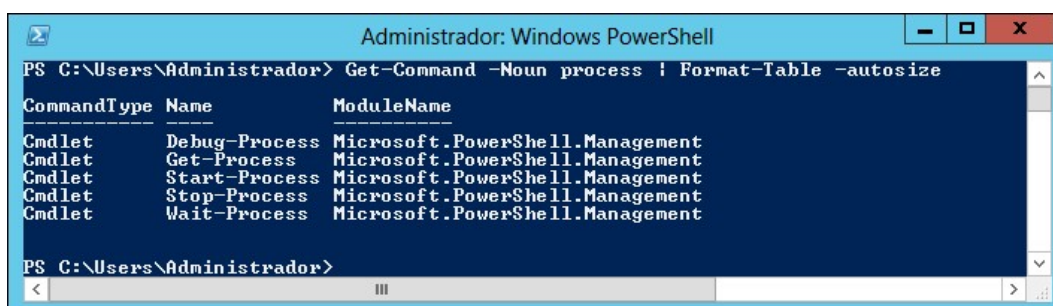
La lista de cmdlets es extensísima, por lo que es imposible revisar ni tan siquiera una parte importante de los mismos en un curso de estas características. Sin embargo, sí que veremos los más habituales y sobre todo, aquellos que nos pueden permitir encontrar el cmdlet que necesitamos, ver qué propiedades tiene dicho cmdlet y averiguar cómo se utiliza.

Get-Command

El cmdlet *Get-Command* sirve para obtener un listado de los cmdlets que existen en PowerShell. Si lo introducimos veremos una larga secuencia de cmdlets lo cual no es muy útil. En cambio, sí puede resultarnos útil cuando utilizamos la opción *-Noun* para averiguar los cmdlets relacionados con un determinado elemento del sistema:

```
> Get-Command -Noun process
```

El cmdlet anterior nos muestra un listado de los cmdlets y funciones disponibles en el sistema para gestionar procesos.



CommandType	Name	ModuleName
Cmdlet	Debug-Process	Microsoft.PowerShell.Management
Cmdlet	Get-Process	Microsoft.PowerShell.Management
Cmdlet	Start-Process	Microsoft.PowerShell.Management
Cmdlet	Stop-Process	Microsoft.PowerShell.Management
Cmdlet	Wait-Process	Microsoft.PowerShell.Management

Sin embargo, si no solo quisiéramos hallar cmdlets, sino por ejemplo también ficheros .exe, utilizaríamos la opción *-Name*:

```
> Get-Command -Name *file*
```

```

Administrador: Windows PowerShell
PS C:\Users\Administrador> Get-Command -Name *file* | Format-Table -autosize
CommandType Name                               ModuleName
-----
Function Close-SmbOpenFile                SmbShare
Function Disable-NetIPHttpsProfile      NetworkTransition
Function Enable-NetIPHttpsProfile      NetworkTransition
Function Get-FileIntegrity              Storage
Function Get-NetConnectionProfile      NetConnection
Function Get-NetFirewallProfile        NetSecurity
Function Get-NfsOpenFile                NFS
Function Get-RDFileTypeAssociation      RemoteDesktop
Function Get-SmbOpenFile                SmbShare
Function Get-SupportedFileSystems      Storage
Function Publish-BCFileContent          BranchCache
Function Repair-FileIntegrity           Storage
Function Revoke-NfsOpenFile             NFS
Function Set-FileIntegrity              Storage
Function Set-NetConnectionProfile      NetConnection
Function Set-NetFirewallProfile        NetSecurity
Function Set-RDFileTypeAssociation      RemoteDesktop
Cmdlet Add-BitsFile                      BitsTransfer
Cmdlet Get-AppLockerFileInformation     AppLocker
Cmdlet New-ADDCCloneConfigFile          ActiveDirectory
Cmdlet New-PSSessionConfigurationFile  Microsoft.PowerShell.Core
Cmdlet Out-File                         Microsoft.PowerShell.Utility
Cmdlet Test-PSSessionConfigurationFile  Microsoft.PowerShell.Core
Cmdlet Unblock-File                    Microsoft.PowerShell.Utility
Application forfiles.exe
Application openfiles.exe

PS C:\Users\Administrador>

```

En el ejemplo anterior buscamos todos los elementos (funciones, cmdlets, aplicaciones, etc.) que contienen file en su nombre. Como puede observarse en los ejemplos anteriores, pueden utilizarse comodines del tipo *. Comprobad como difiere el resultado en la ejecución de:

```
> Get-Command -Noun file
```

y

```
> Get-Command -Name *file
```

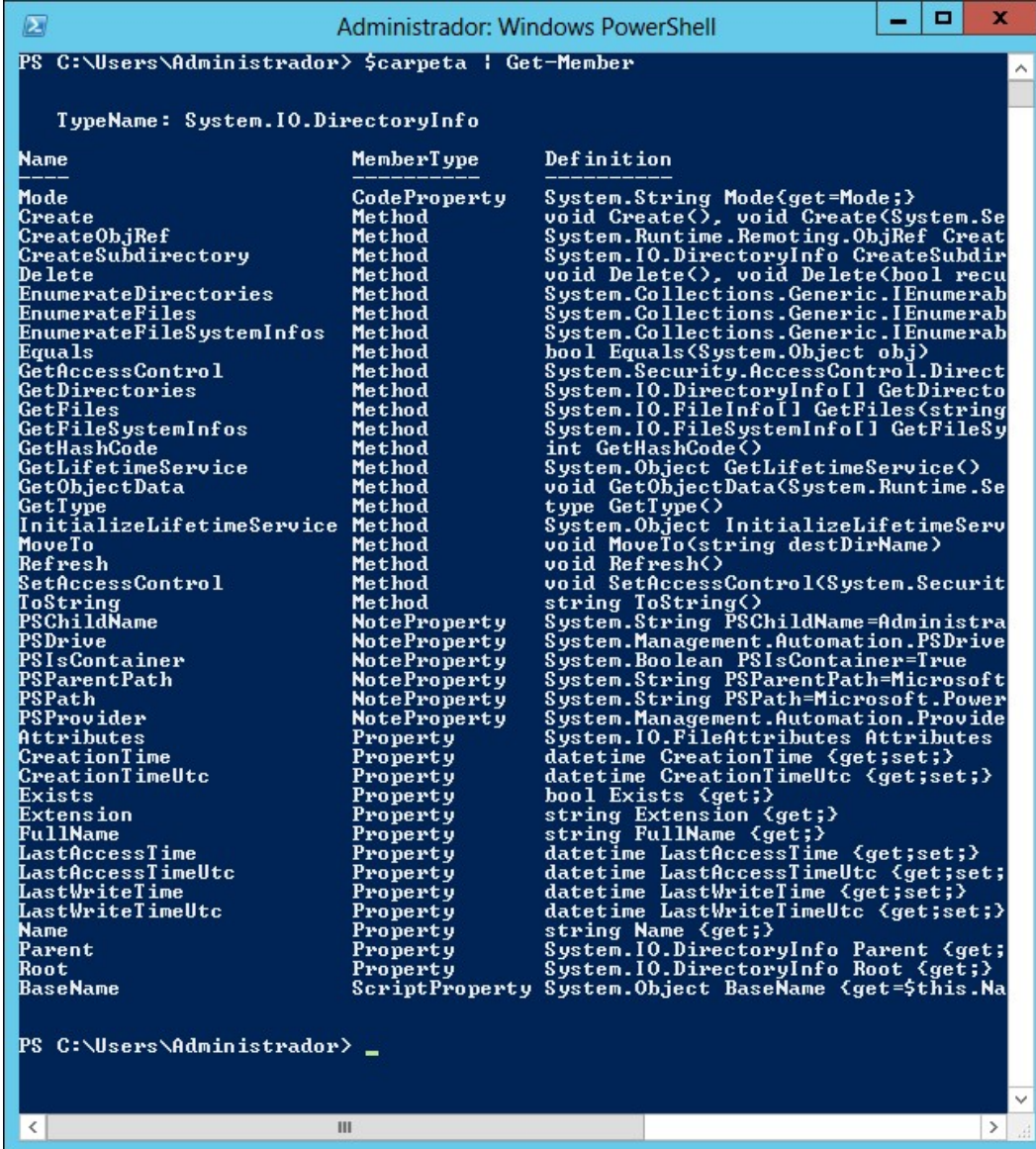
Get-Member

El cmdlet *Get-Member* indica las propiedades y métodos soportados por un objeto, veamos cómo puede sernos útil. Supongamos que queramos almacenar en una variable la referencia a un directorio concreto. Como PowerShell trata **todos los elementos como objetos**, utilizaremos el cmdlet *Get-Item* para asignar el objeto directorio a la variable que queremos crear:

```
> $carpeta = Get-Item C:\Users\Administrador
```


Para comprobar que ahora en la variable `$carpeta` no solo tenemos la ruta del directorio, sino todas las propiedades del objeto, escribiremos lo siguiente:

```
> $carpeta | Get-Member
```



```

Administrador: Windows PowerShell
PS C:\Users\Administrador> $carpeta | Get-Member

TypeName: System.IO.DirectoryInfo

Name                MemberType          Definition
-----
Mode                CodeProperty        System.String Mode{get=Mode;}
Create              Method               void Create(), void Create(System.Se
CreateObjRef        Method               System.Runtime.Remoting.ObjRef Creat
CreateSubdirectory  Method               System.IO.DirectoryInfo CreateSubdir
Delete              Method               void Delete(), void Delete(bool recu
EnumerateDirectories Method               System.Collections.Generic.IEnumerab
EnumerateFiles       Method               System.Collections.Generic.IEnumerab
EnumerateFileSystemInfos Method               System.Collections.Generic.IEnumerab
Equals              Method               bool Equals(System.Object obj)
GetAccessControl     Method               System.Security.AccessControl.Direct
GetDirectories       Method               System.IO.DirectoryInfo[] GetDirecto
GetFiles             Method               System.IO.FileInfo[] GetFiles(string
GetFileSystemInfos   Method               System.IO.FileSystemInfo[] GetFileSy
GetHashCode          Method               int GetHashCode()
GetLifetimeService  Method               System.Object GetLifetimeService()
GetObjectData        Method               void GetObjectData(System.Runtime.Se
GetType             Method               type GetType()
InitializeLifetimeService Method               System.Object InitializeLifetimeServ
MoveIo               Method               void MoveIo(string destDirName)
Refresh             Method               void Refresh()
SetAccessControl     Method               void SetAccessControl(System.Securit
ToString            Method               string ToString()
PSChildName         NoteProperty         System.String PSChildName=Administra
PSDrive             NoteProperty         System.Management.Automation.PSDrive
PSIsContainer        NoteProperty         System.Boolean PSIsContainer=True
PSParentPath        NoteProperty         System.String PSParentPath=Microsoft
PSPath              NoteProperty         System.String PSPath=Microsoft.Power
PSProvider          NoteProperty         System.Management.Automation.Provide
Attributes           Property             System.IO.FileAttributes Attributes
CreationTime         Property             datetime CreationTime {get;set;}
CreationTimeUtc      Property             datetime CreationTimeUtc {get;set;}
Exists               Property             bool Exists {get;}
Extension            Property             string Extension {get;}
FullName             Property             string FullName {get;}
LastAccessTime       Property             datetime LastAccessTime {get;set;}
LastAccessTimeUtc    Property             datetime LastAccessTimeUtc {get;set;}
LastWriteTime        Property             datetime LastWriteTime {get;set;}
LastWriteTimeUtc     Property             datetime LastWriteTimeUtc {get;set;}
Name                 Property             string Name {get;}
Parent               Property             System.IO.DirectoryInfo Parent {get;
Root                 Property             System.IO.DirectoryInfo Root {get;}
BaseName             ScriptProperty       System.Object BaseName {get=$this.Na
  
```

Veamos cómo podemos aprovechar alguna de las propiedades del objeto anterior. Si nos fijamos en el listado que hemos obtenido, veremos que existe una propiedad denominada `parent`. Si escribimos:

```
> $carpeta.parent
```

La salida será como sigue:

```

Administrador: Windows PowerShell
PS C:\Users\Administrador> $carpeta.parent

Mode                LastWriteTime         Length Name
----                -
d-r--           01/05/2013         7:51         Users

PS C:\Users\Administrador>

```

Podemos comprobar que se nos muestra el nombre del directorio del que cuelga *\$carpeta*. Con PowerShell también podemos obtener de la misma manera propiedades del directorio 'padre':

```
> $carpeta.parent.parent
```

```

Administrador: Windows PowerShell
PS C:\Users\Administrador> $carpeta.parent.parent

Mode                LastWriteTime         Length Name
----                -
d--hs           26/04/2013         7:34         C:\

PS C:\Users\Administrador>

```

Si solo quisiéramos obtener el nombre del directorio de nivel superior, utilizaríamos la propiedad *name*:

```
> $carpeta.parent.name
```

Por otra parte, también podemos obtener información adicional, como por ejemplo cuándo se realizó el último acceso al directorio aprovechando la propiedad *LastAccessTime*:

```
> $carpeta.LastAccessTime
```

```

Administrador: Windows PowerShell
PS C:\Users\Administrador> $carpeta.LastAccessTime

miércoles, 1 de mayo de 2013 15:13:15

PS C:\Users\Administrador>

```

También podemos realizar consultas sobre el tipo de objeto. En el siguiente ejemplo obtenemos el valor de la propiedad *PsIsContainer*, que nos indica si el objeto es una carpeta, devolviéndo True o False:

```
> $carpeta.PsIsContainer
```

Para finalizar, podemos utilizar el método `delete()` para borrar no solo la referencia al objeto, sino el objeto en si (**no lo ejecutaremos sobre *C:\Users\Administrador*, ya que es el directorio personal del administrador**):

```
> $carpeta.delete()
```

Get-Help

El cmdlet *Get-Help* nos va a permitir obtener información acerca de cómo utilizar un determinado cmdlet. Antes de empezar a utilizarlo, actualizaremos el módulo de ayuda escribiendo el siguiente cmdlet:

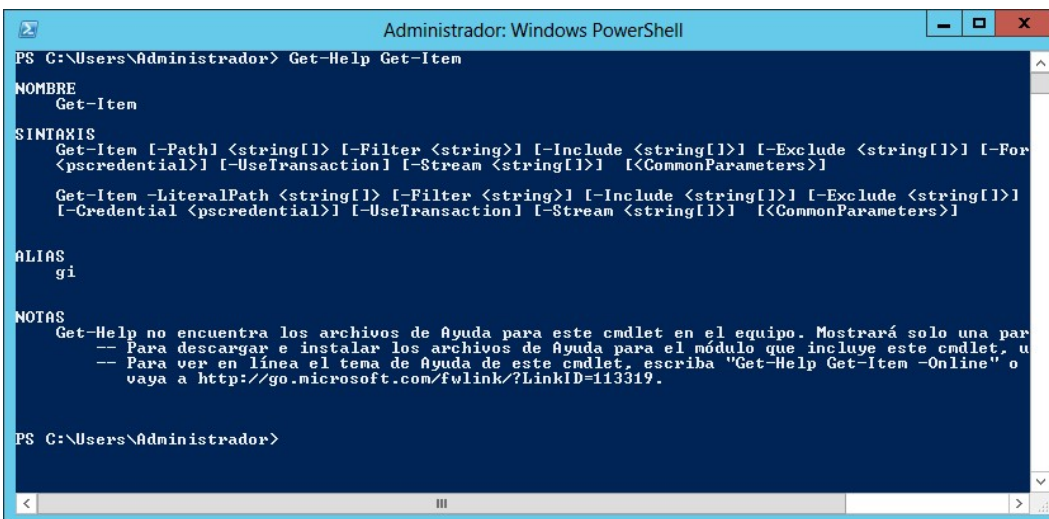
```
> Update-Help
```

Para poder actualizar correctamente el módulo de ayuda, necesitaremos que nuestro equipo tenga conexión a Internet.

La utilización habitual de *Get-Help* sería *Get-Help CMDLET_A_CONSULTAR*, como por ejemplo:

```
> Get-Help Get-Item
```

Cuya salida sería:



```

Administrador: Windows PowerShell
PS C:\Users\Administrador> Get-Help Get-Item

NOMBRE
    Get-Item

SINTAXIS
    Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-For
    <pscredential>] [-UseTransaction] [-Stream <string[]>] [<CommonParameters>]

    Get-Item -LiteralPath <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>]
    [-Credential <pscredential>] [-UseTransaction] [-Stream <string[]>] [<CommonParameters>]

ALIAS
    gi

NOTAS
    Get-Help no encuentra los archivos de Ayuda para este cmdlet en el equipo. Mostrará solo una par
    -- Para descargar e instalar los archivos de Ayuda para el módulo que incluye este cmdlet, u
    -- Para ver en línea el tema de Ayuda de este cmdlet, escriba "Get-Help Get-Item -Online" o
    -- vaya a http://go.microsoft.com/fwlink/?LinkID=113319.

PS C:\Users\Administrador>
  
```

Como se puede ver, se nos ofrece un pequeño resumen de la funcionalidad del cmdlet, la sintaxis con sus opciones, una descripción detallada, otros cmdlets relacionados y finalmente algunas observaciones adicionales.

Una opción sumamente interesante del cmdlet *Get-Help* consiste en la utilización del argumento *-examples* para mostrarnos cómo puede utilizarse un determinado cmdlet:

```
> Get-Help Get-Item -examples
```

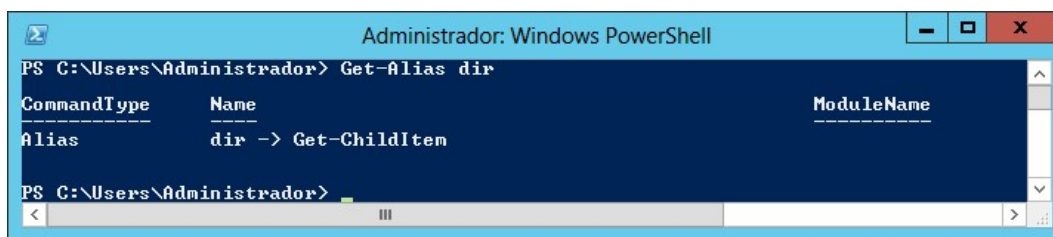
Los tres cmdlets revisados hasta ahora (*Get-Command*, *Get-Member* y *Get-Help*) nos van a permitir manejarnos bastante bien con PowerShell sin tener un conocimiento exacto a priori ni del cmdlet a utilizar ni de su sintaxis u opciones. Sin embargo, iremos introduciendo en esta sección y en las posteriores, los cmdlets que nos van a resultar más útiles en las tareas de administración de nuestro sistema.

Get-ChildItem

Habitualmente, en los sistemas Windows, para obtener un listado de los directorios y ficheros, utilizábamos el comando *dir*. Este comando también funciona en PowerShell, pero en realidad es un alias del cmdlet *Get-ChildItem*. De hecho, si escribimos en PowerShell el siguiente cmdlet:

```
> Get-Alias dir
```

Obtendremos algo similar a lo siguiente:



Cmdlet al que apunta *dir*.

Con *Get-Alias* lo que hemos hecho es obtener el cmdlet correspondiente al alias *dir*. Si queremos obtener un listado de los elementos de un directorio escribiremos:

```
> Get-ChildItem
```



```

Administrador: Windows PowerShell
PS C:\Users\Administrador> Get-ChildItem

Directorio: C:\Users\Administrador

Mode                LastWriteTime         Length Name
----                -
d-----         01/05/2013          7:35         a
d-r-----        17/04/2013          7:30       Contacts
d-r-----        26/04/2013          7:32       Desktop
d-r-----        17/04/2013          7:30     Documents
d-r-----        17/04/2013          7:30     Downloads
d-r-----        17/04/2013          7:30     Favorites
d-r-----        17/04/2013          7:30       Links
d-r-----        17/04/2013          7:30       Music
d-r-----        17/04/2013          7:30     Pictures
d-r-----        17/04/2013          7:30   Saved Games
d-r-----        17/04/2013          7:30     Searches
d-r-----        17/04/2013          7:30     Videos
-a-----         01/05/2013        11:55         397 crearCompartidos.ps1
-a-----         01/05/2013        7:54         395 ListProcessesSortResults.ps1
-a-----         01/05/2013          9:29         419 ping-rango.ps1
-a-----         01/05/2013          9:50         535 ping-rango2.ps1
-a-----         01/05/2013       10:51      121530 Registro
  
```

Si queremos que **también** se muestren los ficheros y directorios del sistema añadiremos la opción **-force**:

```
> Get-ChildItem -Force
```

```

Administrador: Windows PowerShell
PS C:\Users\Administrador> Get-ChildItem -force

Directorio: C:\Users\Administrador

Mode                LastWriteTime         Length Name
----                -
d-----         01/05/2013          7:35         a
d--h-----        17/04/2013          7:30       AppData
d--hs-----        17/04/2013          7:30   Configuración local
d-r-----        17/04/2013          7:30       Contacts
d--hs-----        17/04/2013          7:30       Cookies
d--hs-----        17/04/2013          7:30   Datos de programa
d-r-----        26/04/2013          7:32       Desktop
d-r-----        17/04/2013          7:30     Documents
d-r-----        17/04/2013          7:30     Downloads
d--hs-----        17/04/2013          7:30   Entorno de red
d-r-----        17/04/2013          7:30     Favorites
d--hs-----        17/04/2013          7:30   Impresoras
d-r-----        17/04/2013          7:30       Links
d--hs-----        17/04/2013          7:30   Menú Inicio
d--hs-----        17/04/2013          7:30   Mis documentos
d-r-----        17/04/2013          7:30       Music
d-r-----        17/04/2013          7:30     Pictures
d--hs-----        17/04/2013          7:30   Plantillas
d-r-----        17/04/2013          7:30     Reciente
d-r-----        17/04/2013          7:30   Saved Games
d-r-----        17/04/2013          7:30     Searches
d--hs-----        17/04/2013          7:30     Sendto
d-r-----        17/04/2013          7:30     Videos
-a-----         01/05/2013        11:55         397 crearCompartidos.ps1
-a-----         01/05/2013        7:54         395 ListProcessesSortResults.ps1
-a-hs-----        24/04/2013          7:47      524288 NTUSER.DAT
-a-hs-----        17/04/2013          7:30      397312 ntuser.dat.LOG1
-a-hs-----        17/04/2013          7:30           0 ntuser.dat.LOG2
-a-hs-----        17/04/2013          7:37      65536 NTUSER.DAT{6f0e27bb-d6f9-11e1-93e4-d1011f353a06}.T
-a-hs-----        17/04/2013          7:37      524288 NTUSER.DAT{6f0e27bb-d6f9-11e1-93e4-d1011f353a06}.I
-a-hs-----        17/04/2013          7:37      000001.regtrans-ms
-a-hs-----        17/04/2013          7:37      524288 NTUSER.DAT{6f0e27bb-d6f9-11e1-93e4-d1011f353a06}.T
-a-hs-----        17/04/2013          7:37      000002.regtrans-ms
--hs-----        17/04/2013          7:30           20 ntuser.ini
-a-----         01/05/2013          9:29         419 ping-rango.ps1
-a-----         01/05/2013          9:50         535 ping-rango2.ps1
-a-----         01/05/2013       10:51      121530 Registro
  
```

New-Item

El cmdlet *New-Item* nos va a permitir crear una carpeta o un fichero vacío. Para ello tendremos que especificar la ruta (mediante *-path*) y el tipo de elemento (mediante *-itemtype*). Por ejemplo, con la siguiente instrucción crearemos una carpeta llamada *nueva_carpeta* en el directorio actual.:

```
> new-item -path ./nueva_carpeta -itemtype directory
```

Para crear un fichero dentro de la carpeta anterior escribiremos lo siguiente:

```
> new-item -path ./nueva_carpeta/nuevo_fichero -itemtype file
```

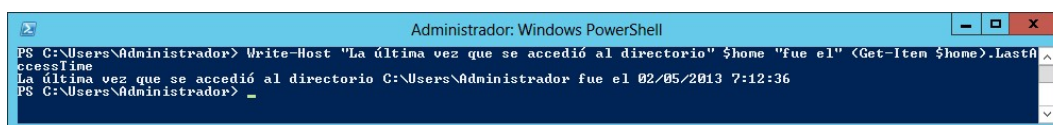
Write-Host

Otro cmdlet que utilizaremos muy a menudo será *Write-Host*, el cual nos permite mostrar por pantalla un determinado mensaje, como puede ser texto, o la salida de ejecución de una instrucción o de un script. Por ejemplo, podemos mostrar por pantalla un mensaje clásico:

```
> Write-Host "Hola Mundo"
```

También podemos mostrar la salida de otros cmdlets:

```
> Write-Host "La última vez que se accedió al directorio" $home " fue el " (Get-Item$home).LastAccessTime
```



Material elaborado exclusivamente de:

- Apuntes del curso del CEFIRE de Administración Centralizada de Redes con Windows 2012 Server del profesor José Ramón Ruiz Rodríguez bajo licencia de Creative Commons Reconocimiento-NoComercial 4.0 Internacional:

