

WINDOWS POWERSHELL

OBJETOS Y VARIABLES

Contenido

1. Objetos en Powershell
2. Propiedades y métodos
3. Variables

Contenido

1. Objetos en Powershell
2. Propiedades y métodos
3. Variables

Objetos

1. Objetos en PowerShell

- ❑ Los *cmdlets* de PowerShell aceptan, procesan y devuelven **objetos** (.NET)
- ❑ Un **objeto** es colección de datos estructurada que representa un **ítem** individual
 - ❑ Almacén de datos
 - ❑ Estado del ordenador
- ❑ Cada objeto pertenece a un **tipo** y contiene distintos **miembros** (propiedades y métodos)

Tipo de un objeto

1. Objetos en PowerShell

- El **tipo** de un objeto:
 - ▣ establece la **clase** a la que pertenece el objeto
 - ▣ indica el **ítem** que representa (un fichero, un proceso,...)
 - ▣ define sus **miembros**

- Ejemplos:

El <i>cmdlet</i>	Devuelve objeto(s) de tipo
Get-Date	DateTime
Get-Location	PathInfo
Get-Process	Process[]

PS C:\> get-date

miércoles, 21 de enero de 2015 19:05:35

PS C:\> get-location

Path

C:\

PS C:\> get-process -name notepad

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
85	7	1228	7028	89	0,02	1280	notepad
85	7	1232	6996	89	0,02	1324	notepad
85	7	1240	7008	93	0,00	4828	notepad

PS C:\>

Contenido

1. Objetos en Powershell

2. Propiedades y métodos (Get-Member)

3. Variables

```
PS C:\> get-help get-member
```

NOMBRE

```
Get-Member
```

SINOPSIS

```
Gets the properties and methods of objects.
```

SINTAXIS

```
Get-Member [[-Name] <String[]>] [-Force] [-InputObject <PSObject>] [-MemberType <PSMemberTypes>] [-Static] [-View  
<PSMemberViewTypes>] [<CommonParameters>]
```

DESCRIPCIÓN

```
The Get-Member cmdlet gets the "me
```

```
To specify the object, use the Inp  
static members (members of the cla  
members, such as NoteProperties, u
```



PSMemberTypes = { Property | Method |... }

VÍNCULOS RELACIONADOS

```
Online Version: http://go.microsoft.com/fwlink/p/?linkid=293971
```

```
Add-Member
```

```
Get-Command
```

```
Get-Help
```

```
Get-PSDrive
```

```
about_Automatic_Variables
```

```
about_Properties
```

```
about_Methods
```

```
about_Objects
```

NOTAS

```
Para ver los ejemplos, escriba: "get-help Get-Member -examples".
```

```
Para obtener más información, escriba: "get-help Get-Member -detailed".
```

```
Para obtener información técnica, escriba: "get-help Get-Member -full".
```

```
Para obtener ayuda disponible en línea, escriba: "get-help Get-Member -online"
```



```
PS C:\> get-member -inputobject (get-date) -membertype property
```

```
TypeName: System.DateTime
```

Name	MemberType	Definition
Date	Property	datetime Date {get;}
Day	Property	int Day {get;}
DayOfWeek	Property	System.DayOfWeek DayOfWeek {get;}
DayOfYear	Property	int DayOfYear {get;}
Hour	Property	int Hour {get;}
Kind	Property	System.DateTimeKind Kind {get;}
Millisecond	Property	int Millisecond {get;}
Minute	Property	int Minute {get;}
Month	Property	int Month {get;}
Second	Property	int Second {get;}
Ticks	Property	long Ticks {get;}
TimeOfDay	Property	timespan TimeOfDay {get;}
Year	Property	int Year {get;}

```
PS C:\> (get-date).date
```

```
miércoles, 21 de enero de 2015 0:00:00
```

```
PS C:\> (get-date).day
```

```
21
```

```
PS C:\> (get-date).month
```

```
1
```

```
PS C:\> (get-date).year
```

```
2015
```

```
PS C:\> (get-date).hour
```

```
19
```

```
PS C:\> (get-date).minute
```

```
48
```

```
PS C:\> █
```

Las **propiedades** de un objeto son los **datos** asociados con (almacenados en) el objeto

Cada propiedad se define mediante un **nombre** y un **tipo** de datos asociado

Para acceder al **valor** de una propiedad, se utiliza la **notación** **objeto.propiedad**

```
PS C:\> get-member -inputobject (get-date) -membertype method
```

TypeName: System.DateTime

Name	MemberType	Definition
-----	-----	-----
Add	Method	datetime Add(timespan value)
AddDays	Method	datetime AddDays(double value)
AddHours	Method	datetime AddHours(double value)
AddMilliseconds	Method	datetime AddMilliseconds(double value)
AddMinutes	Method	datetime AddMinutes(double value)
AddMonths	Method	datetime AddMonths(int months)
AddSeconds	Method	datetime AddSeconds(double value)
AddTicks	Method	datetime AddTicks(long value)
AddYears	Method	datetime AddYears(int value)
CompareTo	Method	int CompareTo(System.Object value), int
Equals	Method	bool Equals(System.Object value), bool
GetDateTimeFormats	Method	string[] GetDateTimeFormats(), string[]
GetHashCode	Method	int GetHashCode()
GetObjectData	Method	void ISerializable.GetObjectData(System.
GetType	Method	type GetType()
GetTypeCode	Method	System.TypeCode GetTypeCode(), System.
IsDaylightSavingTime	Method	bool IsDaylightSavingTime()
Subtract	Method	timespan Subtract(datetime value), datet
ToBinary	Method	long ToBinary()
ToBoolean	Method	bool IConvertible.ToBoolean(System.IForma
ToByte	Method	byte IConvertible.ToByte(System.IForma
ToChar	Method	char IConvertible.ToChar(System.IForma
ToDateTime	Method	datetime IConvertible.ToDateTime(System.
ToDecimal	Method	decimal IConvertible.ToDecimal(System.
ToDouble	Method	double IConvertible.ToDouble(System.IFor
ToFileTime	Method	long ToFileTime()
ToFileTimeUtc	Method	long ToFileTimeUtc()
ToInt16	Method	int16 IConvertible.ToInt16(System.IForma
ToInt32	Method	int IConvertible.ToInt32(System.IForma
ToInt64	Method	long IConvertible.ToInt64(System.IForma
ToLocalTime	Method	datetime ToLocalTime()
ToLongDateString	Method	string ToLongDateString()
ToLongTimeString	Method	string ToLongTimeString()

Los **métodos** de un objeto son las **operaciones** definidas para manipular ese objeto

Cada método se define mediante su **nombre**, sus **parámetros** y su **valor de retorno**

Para **invocar** un método, se utiliza la **notación** `objeto.metodo()`

```
PS C:\> get-date
```

```
lunes, 16 de febrero de 2015 23:14:55
```

```
PS C:\> (get-date).AddDays(1)
```

```
martes, 17 de febrero de 2015 23:15:00
```

```
PS C:\>
```

```
PS C:\>
```

```
PS C:\> (get-date).GetType()
```

IsPublic	IsSerial	Name	BaseType
-----	-----	----	-----
True	True	DateTime	System.ValueType

```
PS C:\> █
```

El método `GetType()` devuelve el **tipo** del objeto al que se aplica

Contenido



1. Objetos en Powershell
2. Propiedades y métodos
- 3. Variables**

Variables

3. Variables

- Una **variable** es una unidad de memoria que:
 - ▣ Se referencia por **nombre**, y
 - ▣ almacena uno o varios **valores**
- Se **suelen utilizar** en expresiones y mandatos
- Un **nombre** de variable:
 - ▣ Es una **cadena que comienza por el carácter \$**
 - ▣ Se **recomienda no utilizar espacios o caracteres especiales**
 - ▣ No **distingue entre mayúsculas y minúsculas**

Tipos de variables

3. Variables

□ Automáticas

- ▣ Almacenan el estado de PowerShell, y el usuario no puede crearlas, borrarlas o cambiar su valor
- ▣ Ejemplos: `$home`, `$null`, `$true`, `$false`

□ De preferencia



- ▣ Almacenan ciertas preferencias del usuario sobre PowerShell, y por ello el usuario sólo puede cambiar su valor
- ▣ Ejemplos: `$MaximumHistoryCount`

□ Creadas por el usuario

- ▣ En la consola o ejecución de un script

```
PS C:\> $num = 5
PS C:\>
PS C:\>
PS C:\> $Cadena = "hola mundo"
PS C:\>
PS C:\>
PS C:\> $hoy = Get-Date
PS C:\>
PS C:\>
PS C:\> $varios = 3, "hola", (Get-Location)
PS C:\>
PS C:\>
PS C:\> $num2 = 10 + $num
PS C:\>
PS C:\>
PS C:\> $cadena = $cadena + "!!!"
PS C:\>
PS C:\>
PS C:\> $varios = $varios + 3.1415926
PS C:\>
PS C:\>
PS C:\> $num3 = $num2 + $num4
PS C:\>
```

Las **variables** se **crean** cuando se utilizan, no es necesario definirlas previamente

Lo habitual es comenzar por **asignarles** uno o varios valores

Luego pueden ser **utilizadas** en expresiones o *cmdlets*

Si se utiliza una variable **sin valor** previo, éste es **\$null**

```
PS C:\> $num
5
PS C:\>
PS C:\> $num2
15
PS C:\>
PS C:\> $num3
15
PS C:\>
PS C:\> $num4
PS C:\>
PS C:\> $cadena
hola mundo!!!
PS C:\>
PS C:\> $varios
3
hola
Path
----
C:\
3,1415926

PS C:\> $hoy
jueves, 26 de febrero de 2015 20:09:30

PS C:\> $hoy.year
2015
PS C:\>
PS C:\> $hoy.GetType()

IsPublic IsSerial Name                                     BaseType
-----
True     True     DateTime                                     System.ValueType

PS C:\>
```

Una sentencia que sólo contiene el nombre de una variable, la **visualiza** por pantalla

Cuando una variable contiene un **objeto**, podemos acceder a sus **propiedades y métodos** con “.”

Y para terminar...

get-help

Get-Member

about_objects

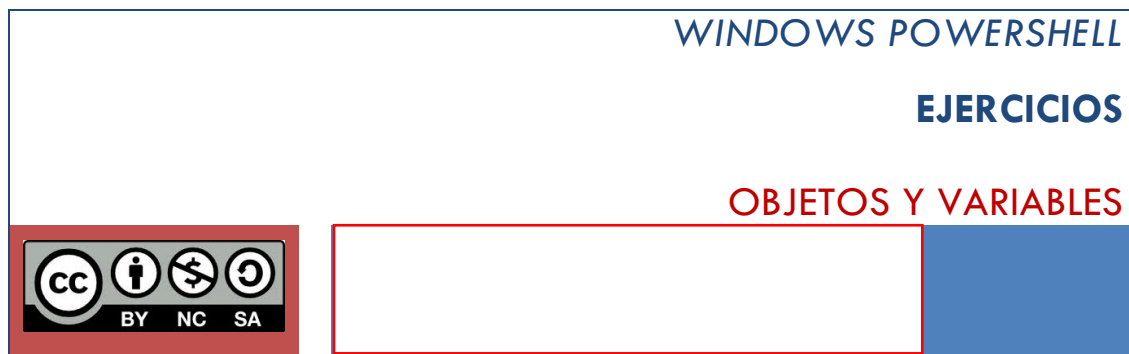
about_properties

about_methods

about_variables

about_automatic_variables

about_preference_variables



1 Ejecuta cada uno de los siguientes cmdlets en la consola:

- a) `Get-Date`
- b) `Get-Location`
- c) `Get-Alias cd`
- d) `Get-ChildItem c:\Windows\explorer.exe`
- e) `Get-Process system`
- f) `Get-Service w32time`

A continuación, utiliza el cmdlet `Get-Member` para averiguar el tipo de objeto que devuelve cada una de esas ejecuciones, y compáralo con la salida obtenida previamente. ¿A qué tipo de elemento real del sistema crees que corresponde cada uno?

2 En primer lugar, ejecuta el siguiente cmdlet:

```
Set-Location c:\Windows
```

Comprueba que efectivamente la ruta actual es la indicada, ejecutando `Get-Location`. A continuación, diseña cmdlets (o expresiones) para obtener la información siguiente, y compruébalas en la consola:

- a) Los nombres de las propiedades del objeto devuelto por `Get-Location`
- b) El valor actual de cada una de dichas propiedades

3 En primer lugar, lanza el bloc de notas de Windows, ejecutando en la consola

```
notepad.exe
```

Vuelve a la consola (sin cerrar el bloc de notas), y utiliza `Get-Process` para comprobar que efectivamente existe un proceso que ejecuta este programa. A continuación, ejecuta en la consola cmdlets (o expresiones) para conseguir lo siguiente:

- a) Obtener los nombres de los métodos del objeto devuelto por `Get-Process notepad` (NOTA: Asegúrate que sólo hay un proceso bloc de notas en ejecución; si hay más de uno, la salida de este cmdlet no será la adecuada).

- b) De entre los métodos anteriores, invocar aquel que termina (“mata”) el proceso sobre el objeto anterior. En ese momento, el bloc de notas debe cerrarse automáticamente.

4 En la consola, asigna a las variables siguientes los valores iniciales especificados en la tabla que se muestra a continuación, en el orden en que aparecen:

Variable	Valor Inicial
\$num	1000
\$cadena1	“En un lugar de la Mancha”
\$cadena2	“de cuyo nombre no quiero acordarme”
\$fecha	Get-Date
\$file	get-childitem c:\Windows\System32\Notepad.exe
\$proceso	Start-Process \$file -Passthru

A continuación, resuelve cada uno de los ejercicios siguientes mediante un cmdlet o expresión que resulte adecuado, y ejecútalo en la consola para comprobarlo:

- Suma 2000 a \$num, y guarda el resultado en \$num2.
- Visualiza en pantalla el valor de \$num2
- Inicializa la variable \$cadena3 como suma de \$cadena1, una cadena con un espacio en blanco y \$cadena2.
- Obtén el tipo de objeto de \$cadena3, así como sus propiedades y métodos.
- Localiza de entre las propiedades del ejercicio anterior una que devuelve la longitud de una cadena, y útilízala para averiguar la longitud de \$cadena3.
- Localiza de entre los métodos del ejercicio c) uno que subdivide la cadena en subcadenas. Utiliza este método con la variable \$cadena3 en una expresión que muestre en pantalla su subdivisión en palabras (NOTA: Este método, invocado sin parámetros, divide la cadena en subcadenas al encontrar el carácter “espacio en blanco”).
- Visualiza en pantalla el contenido de \$fecha.
- Localiza entre los miembros de \$fecha la propiedad que representa el día (del mes), y escribe una expresión que sume 40 al valor de esta propiedad de \$fecha.
- Localiza entre los miembros de \$fecha el método que permite añadir un número de días a una fecha. Invoca este método sobre \$fecha, con 40 como argumento. ¿En qué se diferencia este resultado del obtenido en el apartado anterior?
- Visualiza por pantalla el valor de \$proceso. A continuación, escribe una expresión que visualice el identificador interno del proceso representado por la variable \$proceso, utilizando la propiedad que expresa este identificador.
- Utiliza el método adecuado sobre la variable \$proceso para finalizar la ejecución del proceso representado por esta variable.

5 En el ejercicio anterior, la variable \$proceso se ha inicializado mediante la salida del cmdlet Start-Process. Averigua el papel que juega el parámetro -PassThru en esta invocación. ¿Qué habría ocurrido si no se hubiera utilizado?

WINDOWS POWERSHELL

CANALIZACIÓN (*PIPELINE*)

Contenido

1. Concepto de canalización (*pipeline*)
2. Cómo funcionan las canalizaciones
3. *Cmdlets* útiles para canalizaciones

Contenido

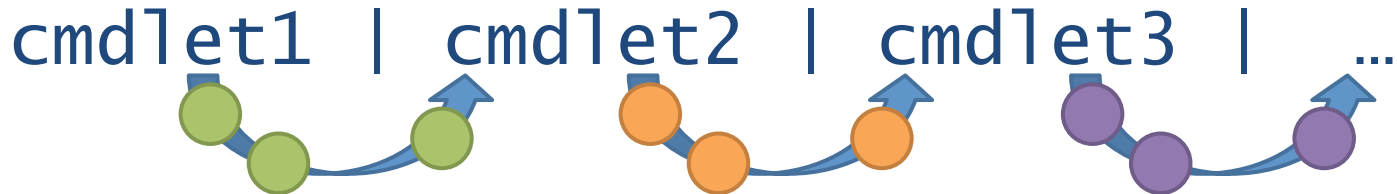
1. Concepto de canalización (*pipeline*)
2. Cómo funcionan las canalizaciones
3. *Cmdlets* útiles para canalizaciones

Canalización

1. Concepto de canalización (*pipeline*)

□ Canalización (*pipeline*):

- ▣ es una **serie** o **cadena** de mandatos (*cmdlets*),
- ▣ **conectados** entre sí por el operador de canalización “|”
- ▣ que **envía** los datos (**objetos**) de salida de cada uno como datos de entrada del siguiente:



- ▣ los objetos se envían **uno a uno**, mejorando la eficiencia

Objetos vs. texto

1. Concepto de canalización (*pipeline*)

- En una canalización:
 - ▣ los *cmdlets* participantes se envían **objetos** (.NET)
 - ▣ el último *cmdlet* convierte sus objetos de salida **en texto**, y los muestra por pantalla
- El **intercambio** basado en **objetos**:
 - ▣ Clarifica y simplifica la **compatibilidad** entre *cmdlets*
 - ▣ **Evita** tener que interpretar o formatear la salida de un *cmdlet* para que lo entienda el siguiente

PS C:\> get-date | get-member

TypeName: System.DateTime

Name	MemberType	Definition
-----	-----	-----
Add	Method	datetime Add(timespan value)
AddDays	Method	datetime AddDays(double value)
AddHours	Method	datetime AddHours(double value)
AddMilliseconds	Method	datetime AddMilliseconds(double value)
AddMinutes	Method	datetime AddMinutes(double value)
AddMonths	Method	datetime AddMonths(int months)
AddSeconds	Method	datetime AddSeconds(double value)
AddTicks	Method	datetime AddTicks(long value)
AddYears	Method	datetime AddYears(int value)
CompareTo	Method	int CompareTo(System.Object value), int CompareTo(datetime value), int IComparable.Com...
Equals	Method	bool Equals(System.Object value), bool Equals(datetime value), bool IEquatable[datetim...
GetDateTimeFormats	Method	string[] GetDateTimeFormats(), string[] GetDateTimeFormats(System.IFormatProvider prov...
GetHashCode	Method	int GetHashCode()
GetObjectData	Method	void ISerializable.GetObjectData(System.Runtime.Serialization.SerializationInfo info, ...
GetType	Method	type GetType()
GetTypeCode	Method	System.TypeCode GetTypeCode(), System.TypeCode IConvertible.GetTypeCode()
IsDaylightSavingTime	Method	bool IsDaylightSavingTime()
Subtract	Method	timespan Subtract(datetime value), datetime Subtract(timespan value)
ToBinary	Method	long ToBinary()
ToBoolean	Method	bool IConvertible.ToBoolean(System.IFormatProvider provider)
ToByte	Method	byte IConvertible.ToByte(System.IFormatProvider provider)
ToChar	Method	char IConvertible.ToChar(System.IFormatProvider provider)
ToDateTime	Method	datetime IConvertible.ToDateTime(System.IFormatProvider provider)
ToDecimal	Method	decimal IConvertible.ToDecimal(System.IFormatProvider provider)
ToDouble	Method	double IConvertible.ToDouble(System.IFormatProvider provider)
ToFileTime	Method	long ToFileTime()
ToFileTimeUtc	Method	long ToFileTimeUtc()
ToInt16	Method	int16 IConvertible.ToInt16(System.IFormatProvider provider)
ToInt32	Method	int IConvertible.ToInt32(System.IFormatProvider provider)
ToInt64	Method	long IConvertible.ToInt64(System.IFormatProvider provider)
ToLocalTime	Method	datetime ToLocalTime()
ToLongDateString	Method	string ToLongDateString()
ToLongTimeString	Method	string ToLongTimeString()

```
PS C:\> get-service | get-member -membertype property
```

TypeName: System.ServiceProcess.ServiceController

Name	MemberType	Definition
CanPauseAndContinue	Property	bool CanPauseAndContinue {get;}
CanShutdown	Property	bool CanShutdown {get;}
CanStop	Property	bool CanStop {get;}
Container	Property	System.ComponentModel.IContainer Container {get;}
DependentServices	Property	System.ServiceProcess.ServiceController[] DependentServices {get;}
DisplayName	Property	string DisplayName {get;set;}
MachineName	Property	string MachineName {get;set;}
ServiceHandle	Property	System.Runtime.InteropServices.SafeHandle ServiceHandle {get;}
ServiceName	Property	string ServiceName {get;set;}
ServicesDependedOn	Property	System.ServiceProcess.ServiceController[] ServicesDependedOn {get;}
ServiceType	Property	System.ServiceProcess.ServiceType ServiceType {get;}
Site	Property	System.ComponentModel.ISite Site {get;set;}
Status	Property	System.ServiceProcess.ServiceControllerStatus Status {get;}

```
PS C:\> get-member -inputobject (get-service) -membertype property
```

TypeName: System.Object[]

Name	MemberType	Definition
IsFixedSize	Property	bool IsFixedSize {get;}
IsReadOnly	Property	bool IsReadOnly {get;}
IsSynchronized	Property	bool IsSynchronized {get;}
Length	Property	int Length {get;}
LongLength	Property	long LongLength {get;}
Rank	Property	int Rank {get;}
SyncRoot	Property	System.Object SyncRoot {get;}

```
PS C:\>
```

```
PS C:\> get-childitem c:\windows |  
>> where-object -property psiscontainer -eq $false |  
>> sort-object -property length |  
>> select-object Name,Length |  
>> format-table -autosize  
>>
```

Name	Length
-----	-----
setuperr.log	0
win.ini	92
system.ini	219
PFR0.log	808
DtcInstall.log	3608
vmgcoinstall.log	5446
winhlp32.exe	10752
write.exe	10752
setupact.log	16072
hh.exe	17408
ServerWeb.xml	27640
ServerStandard.xml	27657
mib.bin	43131
twain_32.dll	51712
bfsvc.exe	56832
bootstat.dat	67584
splwow64.exe	125952
regedit.exe	151552
notepad.exe	217600
WMSysPr9.prx	316640
WindowsUpdate.log	361565
HelpPane.exe	973312
explorer.exe	2373784

```
PS C:\>
```

Get-Childitem (obtener hijos)

Where-Object (filtrar objetos)

Sort-Object (ordenar objetos)

Select-Object (sel. propiedades)

Format-Table (formatear salida)

Contenido



1. Concepto de canalización (*pipeline*)
- 2. Cómo funcionan las canalizaciones**
3. *Cmdlets* útiles para canalizaciones

Cmdlets anterior y posterior

2. Cómo funcionan las canalizaciones

`cmdlet1 | cmdlet2 -param1 -param2...`

- El *cmdlet* **anterior** genera objetos y los envía a la canalización (sin restricciones)
- El *cmdlet* **posterior** debe estar diseñado para poder recibir datos de entrada de una canalización
 - ▣ Uno o varios **parámetros** de entrada

Asociación (*binding*)

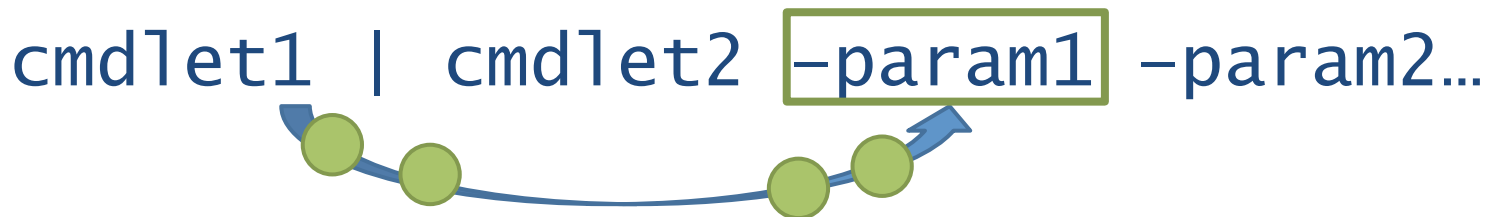
2. Cómo funcionan las canalizaciones



- ❑ PowerShell intenta **localizar** un **parámetro** que:
 - ❑ **Admita** recibir objetos mediante canalización
 - ❑ Tenga un **tipo compatible** con los objetos generados
 - ❑ **No** esté ya en uso por el *cmdlet*

Asociación (*binding*)

2. Cómo funcionan las canalizaciones

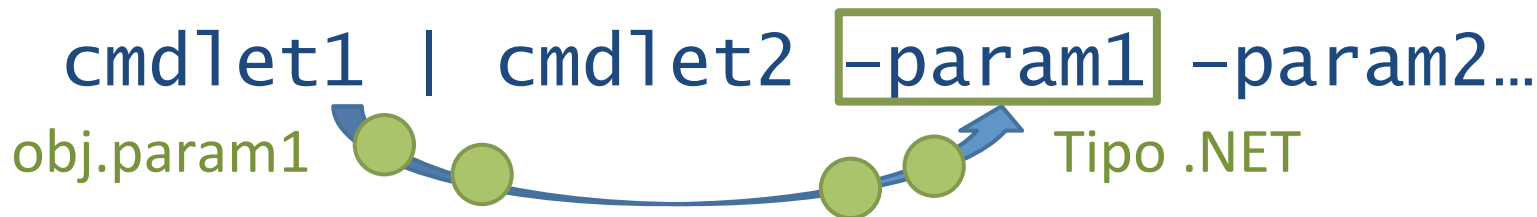


□ El **resultado**:

- ▣ Si 1 parámetro cumple, se utiliza
- ▣ Si 1+ parámetros cumplen, se utiliza el **más eficiente**
- ▣ Si ningún parámetro cumple, se genera un **error**

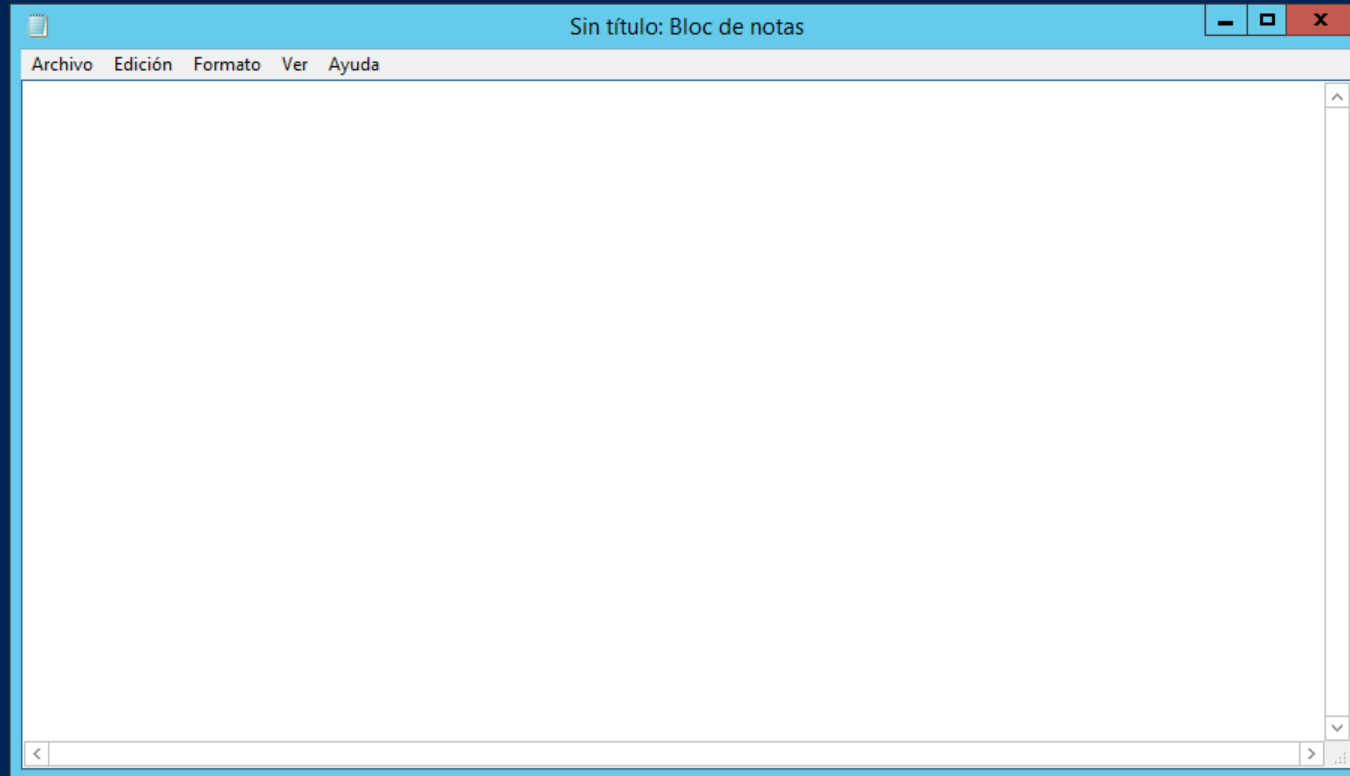
Métodos de aceptación

2. Cómo funcionan las canalizaciones



- Un **parámetro** puede aceptar objetos de 2 formas:
 - ▣ **ByValue**
 - Acepta si su **tipo .NET** es **compatible** con el tipo del objeto
 - ▣ **ByPropertyName**
 - Acepta si su **nombre coincide** con alguna propiedad del objeto


```
PS C:\> notepad  
PS C:\>
```



Ejemplo de canalización “natural”

```
PS C:\> get-process -name notepad | stop-process  
PS C:\> █
```

`get-<nombre> | {start,stop,set,...}-<nombre>`



```
PS C:\> get-process -name notepad | stop-process
PS C:\>
PS C:\> help stop-process -parameter inputobject
```

-InputObject <Process[]>

Stops the processes represented by the specified process objects. Enter a variable that contains the objects, or type a command or expression that gets the objects.

¿Requerido?	true
¿Posición?	1
Valor predeterminado	
¿Aceptar canalización?	true (ByValue)
¿Aceptar caracteres comodín?	false

ByValue: Objetos de tipo "Process"

```
PS C:\> help stop-process -parameter name
```

-Name <String[]>

Specifies the process names of the processes to be stopped. You can type multiple process names (separated by commas) or use wildcard characters.

¿Requerido?	true
¿Posición?	named
Valor predeterminado	
¿Aceptar canalización?	true (ByPropertyName)
¿Aceptar caracteres comodín?	true

ByPropertyName: Objetos que tengan una propiedad denominada "Name"

```
PS C:\> █
```

Contenido



1. Concepto de canalización (*pipeline*)
2. Cómo funcionan las canalizaciones
- 3. *Cmdlets* útiles para canalizaciones**

Gestión de objetos

3. Cmdlets útiles para canalizaciones

Cmdlet	Uso
Sort-Object	Ordena objetos por el valor de sus propiedades
Select-Object	Selecciona propiedades concretas de objetos y objetos de una colección en función de su posición
Where-Object	Filtra objetos de una colección en función del valor de sus propiedades
Measure-Object	Calcula propiedades numéricas sobre una colección de objetos (máximo, mínimo, media, número,...)
ForEach-Object	Ejecuta un código PowerShell sobre cada ítem de una colección (la variable <code>\$_</code> referencia el objeto actual)

Formatear/visualizar objetos

3. Cmdlets útiles para canalizaciones

Cmdlet	Uso
Format-Table, Format-List, Format-Wide, ...	Formatea objetos para una visualización más adecuada: en forma de tabla, en forma de lista, en formato ancho, etc. Estos cmdlets suelen ubicarse al final de la canalización
Out-File, Out-Printer, Out-GridView, ...	Redirecciona la salida a una ubicación concreta: a un fichero, a una impresora, a una nueva ventana, etc. Estos cmdlets suelen ubicarse al final de la canalización

Importar/exportar objetos

3. Cmdlets útiles para canalizaciones

Cmdlet	Uso
Import-CSV	Crea objetos definidos por el usuario a partir del contenido de un fichero CSV (<i>comma-separated values</i>). Este cmdlet se suele ubicar al principio de una canalización
Export-CSV	Convierte objetos en una serie de cadenas de texto en formato CSV, y las almacena en un archivo CSV
ConvertTo-Html	Convierte objetos en código HTML que puede ser visualizado en un navegador web

Y para terminar...



`get-help`

`about_pipeline`

... y sobre el resto de *cmdlets* de la presentación

**NOTA INICIAL**

De cara a resolver los siguientes ejercicios en la consola de manera más cómoda y compacta, recuerda que muchos de los cmdlets que se suelen utilizar en canalizaciones tienen alias más cortos y fáciles de teclear. Por ejemplo:

Cmdlet	Alias
Sort-Object	sort
Select-Object	select
Where-Object	where
Measure-Object	measure
Get-Service	gsv
Get-Process	ps
Get-ChildItem	gci, dir, ls

1 Escribe canalizaciones que resuelvan cada uno de los siguientes casos:

- Obtener las propiedades de los objetos devueltos por `Get-Service`.
- Obtener la lista de los servicios del sistema, ordenados por su estado ("Status").
- Obtener la lista de los servicios del sistema cuyo estado sea en ejecución.
- Obtener la lista de los servicios en ejecución, en orden alfabético inverso (NOTA: Ver propiedad "-Descending" de `Sort-Object`).
- Obtener la cantidad de servicios en ejecución (NOTA: Ver `Measure-Object`).
- Reiniciar los servicios que comiencen con el prefijo "net" y que actualmente se encuentren en ejecución.

2 Escribe canalizaciones que resuelvan cada uno de los siguientes casos:

- Obtener las propiedades de los objetos devueltos por `Get-Process`.
- Obtener la lista de los procesos en ejecución, ordenada por identificador del proceso.
- Obtener la lista de los procesos en ejecución, ordenada inversamente por la cantidad de CPU utilizada por cada proceso.
- Obtener, de la lista anterior, el proceso que más CPU ha consumido (NOTA: Ver `Select-Object`, que permite seleccionar elementos de una lista por posición, quedarse con los primeros o los últimos N objetos, etc.).

- e) Completar la canalización anterior para que el objeto resultante sólo tenga las propiedades "Name" y "Cpu".
- f) Completar la canalización anterior para visualizar el resultado en forma de tabla (NOTA: Ver `Format-Table`), pero que no ocupe todo el ancho disponible de la consola, sino que ajuste el ancho de cada columna al espacio necesario.
- g) Obtener la cantidad de procesos en ejecución en el sistema, así como el máximo, mínimo y valor medio de su consumo de CPU.

3 Escribe canalizaciones que resuelvan cada uno de los siguientes casos:

- a) Obtener las propiedades de los objetos devueltos por `Get-Childitem c:\Windows` (nótese que hay dos tipos de objetos, con algunas propiedades distintas).
- b) Obtener la lista de archivos con extensión ".EXE" de `c:\Windows\System32`.
- c) Repetir la canalización anterior, pero de manera recursiva, es decir, incluyendo las subcarpetas de `c:\Windows\System32`. (NOTA: Revisar los parámetros del cmdlet `Get-Childitem`).
- d) Obtener la lista de archivos con extensión ".EXE" de `c:\Windows\System32`, incluyendo sus subcarpetas, y ordenar la salida por tamaño de archivo, en orden inverso.

Notarás un retardo inicial en visualizar el resultado esta canalización. ¿A qué crees que es debido?
- e) Obtener la lista de los 10 archivos con extensión ".EXE" de mayor tamaño de la carpeta `c:\Windows\System32` y sus subcarpetas.
- f) Obtener la suma del tamaño de los 10 archivos con extensión ".EXE" de mayor tamaño de la carpeta `c:\Windows\System32` y sus subcarpetas.

4 Te han encargado que obtengas información acerca de los 10 procesos que más CPU han consumido en un momento dado en un sistema Windows. En primer lugar, escribe una canalización que resuelva esta tarea y muestre su resultado en pantalla (NOTA: Puedes basarte en el ejercicio 2 de este boletín).

A continuación, completa esa canalización de las formas siguientes, donde cada una establece un formato o soporte de salida distintos, y que puede resultarte útil en ocasiones diferentes:

- a) Visualizar solamente el nombre de los procesos implicados, o bien los consumos de CPU (ver `Format-Wide`).
- b) Visualizar el resultado en una ventana distinta a la consola (ver `Out-GridView`).
- c) Guardar el resultado en un fichero de texto denominado `MasCPU.txt`. (Posteriormente, puedes abrir este fichero con el Bloc de notas o cualquier procesador de textos).
- d) Guardar el resultado en un fichero HTML denominado `MasCPU.html`. (Posteriormente, puedes abrir este fichero con un navegador web).
- e) Guardar el resultado en un fichero CSV denominado `MasCPU.csv`. (Este fichero puede ser abierto con una hoja de cálculo, por ejemplo, para un procesamiento posterior).