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

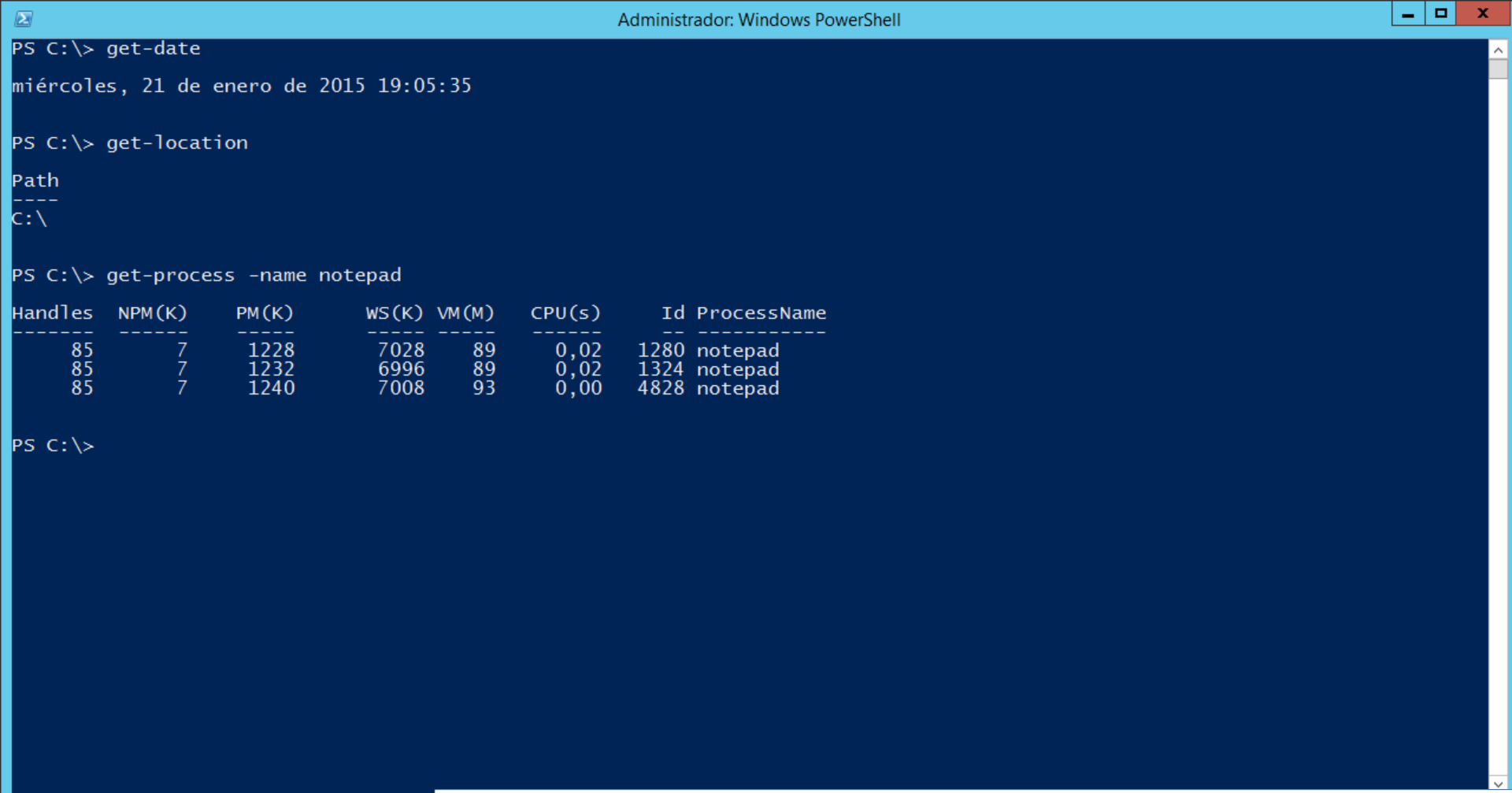
- Los *cmdlets* de PowerShell aceptan, procesan y devuelven **objetos** (.NET)
- Un **objeto** es colección de datos estructurada que representa un **ítem** individual
  - Almacén de datos
  - Estado del ordenador
- Cada objeto pertenece a un **tipo** y contiene distintos **miembros** (propiedades y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

Tipo de un objeto

- 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:

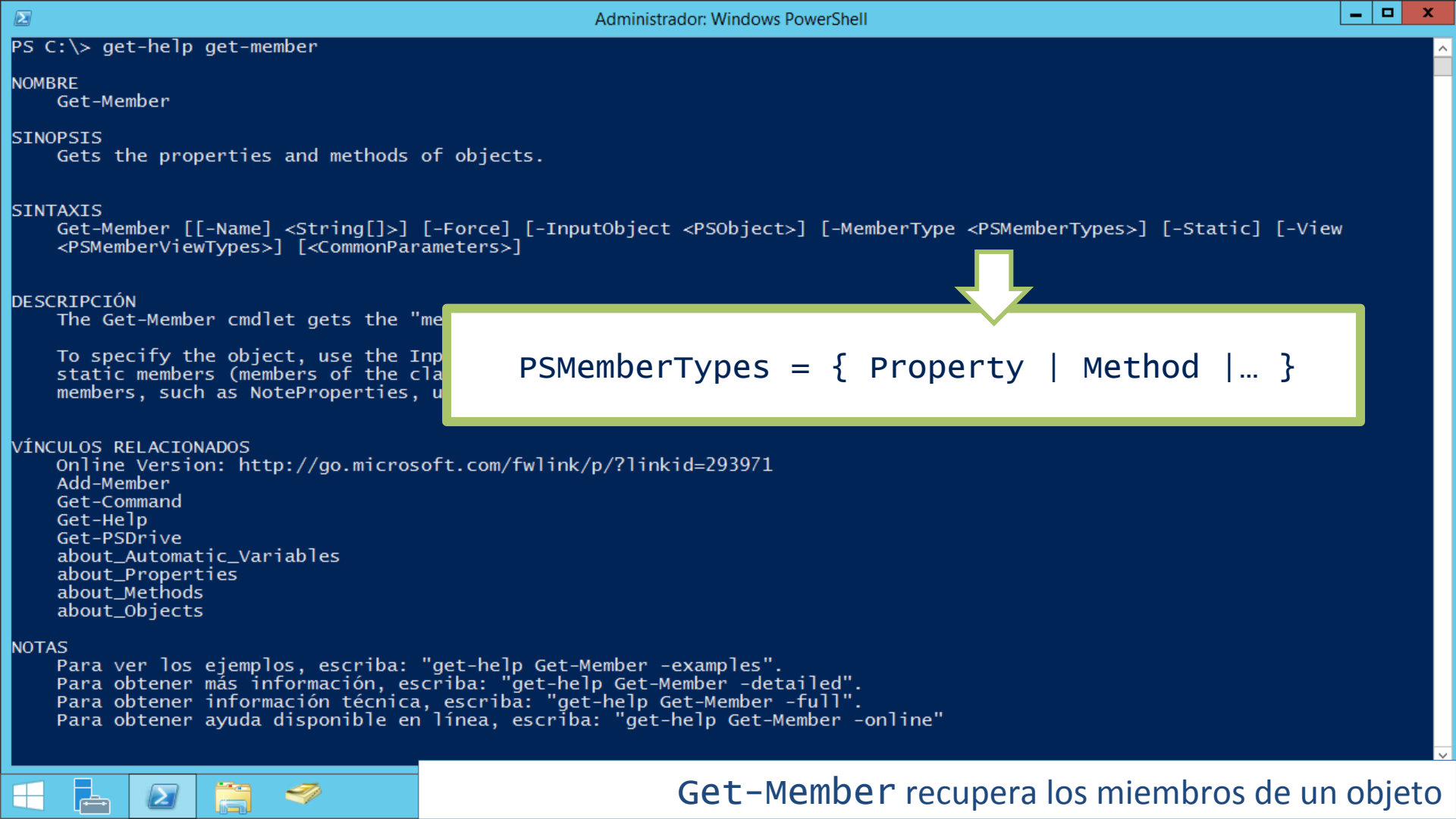
El *cmdlet*	Devuelve objeto(s) de tipo
`Get-Date`	`DateTime`
`Get-Location`	`PathInfo`
`Get-Process`	`Process[]`



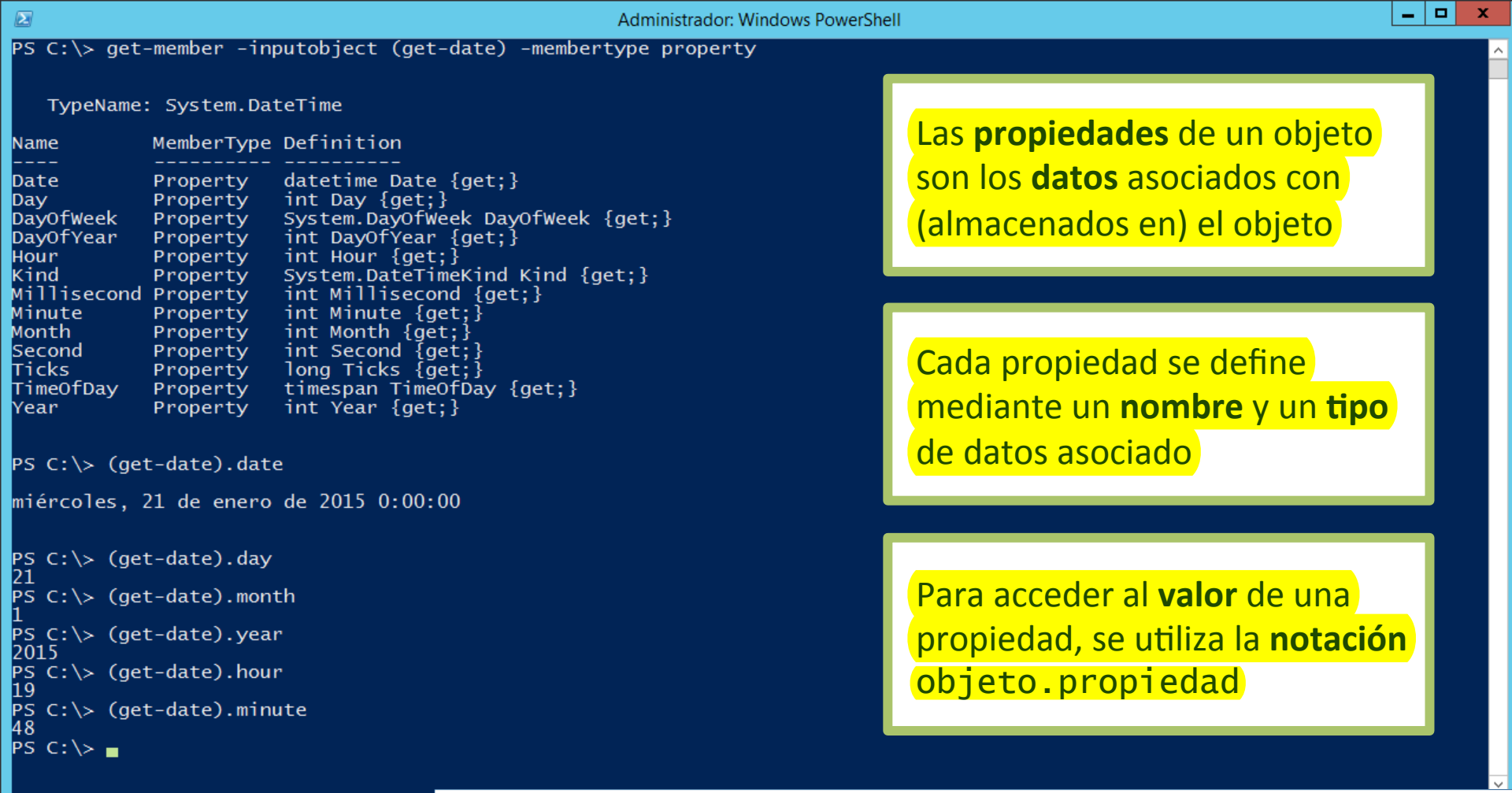
Ejemplos de cmdlets que devuelven objetos

Contenido

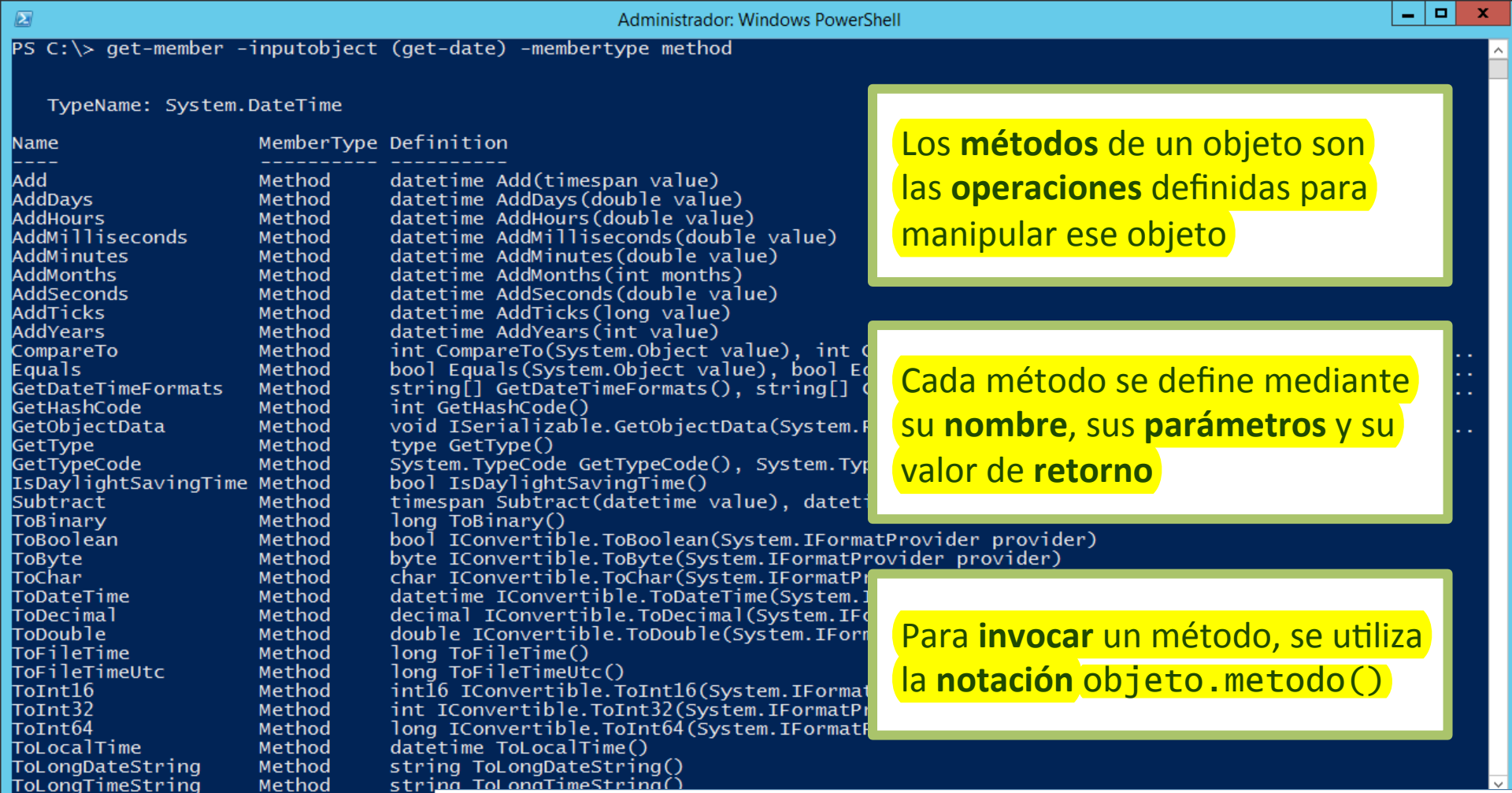
1. Objetos en Powershell
2. Propiedades y métodos (`Get-Member`)
3. Variables



`Get-Member` recupera los miembros de un objeto



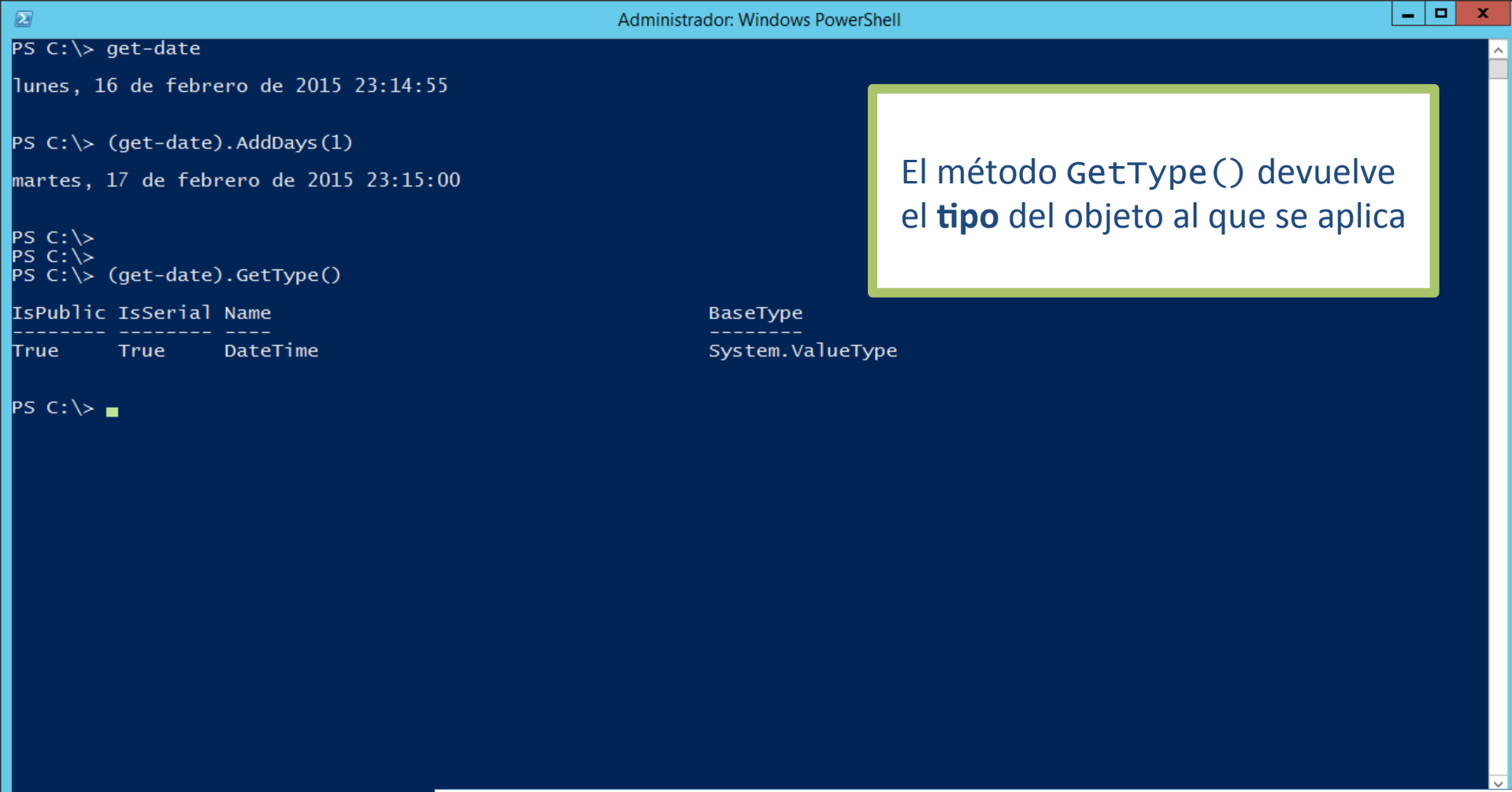
Propiedades de los objetos



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()`


Administrador: Windows PowerShell
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

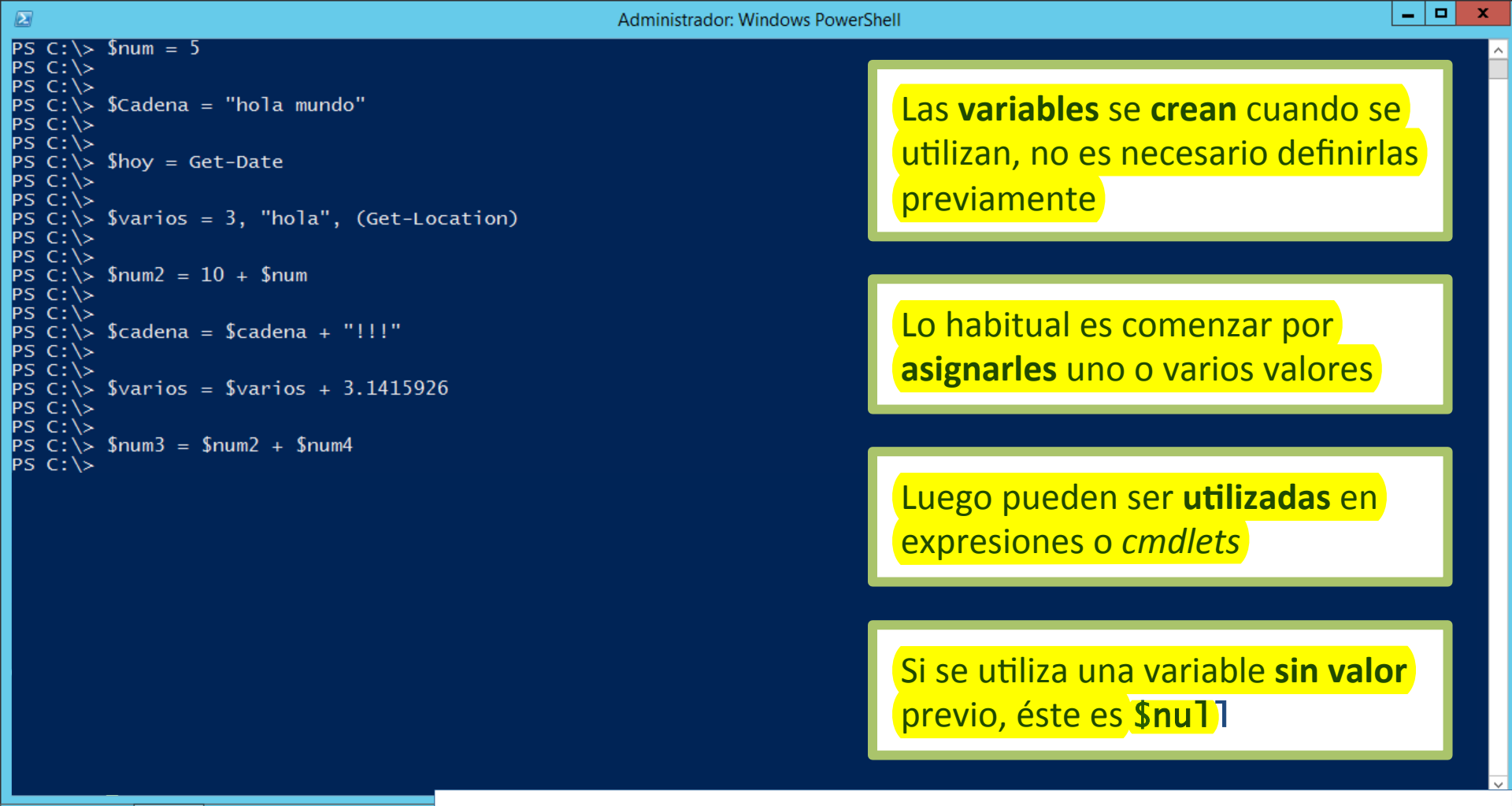
1. Objetos en Powershell
2. Propiedades y métodos
3. Variables

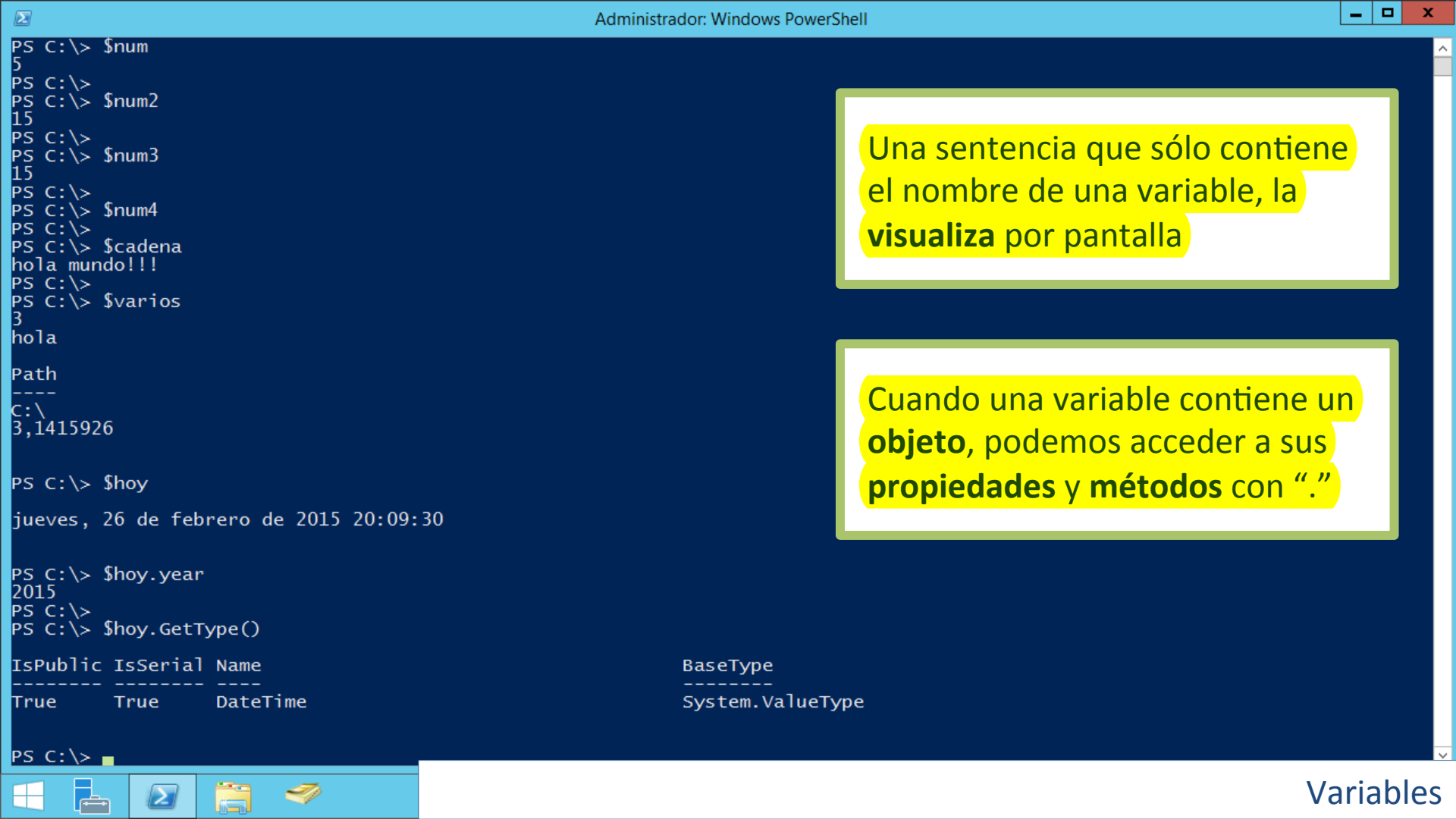
Variables

- 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

- **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


Administrador: Windows PowerShell
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


Administrador: Windows PowerShell
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
```

WINDOWS POWERSHELL

EJERCICIOS

OBJETOS Y 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:

a) Suma `2000` a `$num`, y guarda el resultado en `$num2`.

b) Visualiza en pantalla el valor de `$num2`

c) Inicializa la variable `$cadena3` como suma de `$cadena1`, una cadena con un espacio en blanco y `$cadena2`.

d) Obtén el tipo de objeto de `$cadena3`, así como sus propiedades y métodos.

e) Localiza de entre las propiedades del ejercicio anterior una que devuelve la longitud de una cadena, y utilízala para averiguar la longitud de `$cadena3`.

f) 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”).

g) Visualiza en pantalla el contenido de `$fecha`.

h) 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`.

i) 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?

j) 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.

k) 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

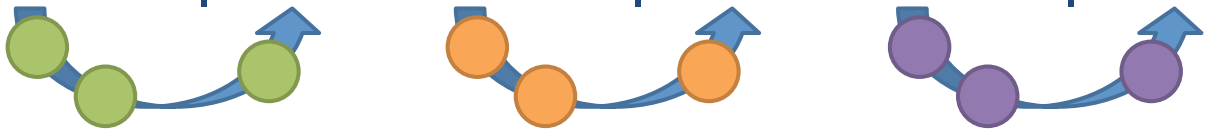
Contenido

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

Canalización

- 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:

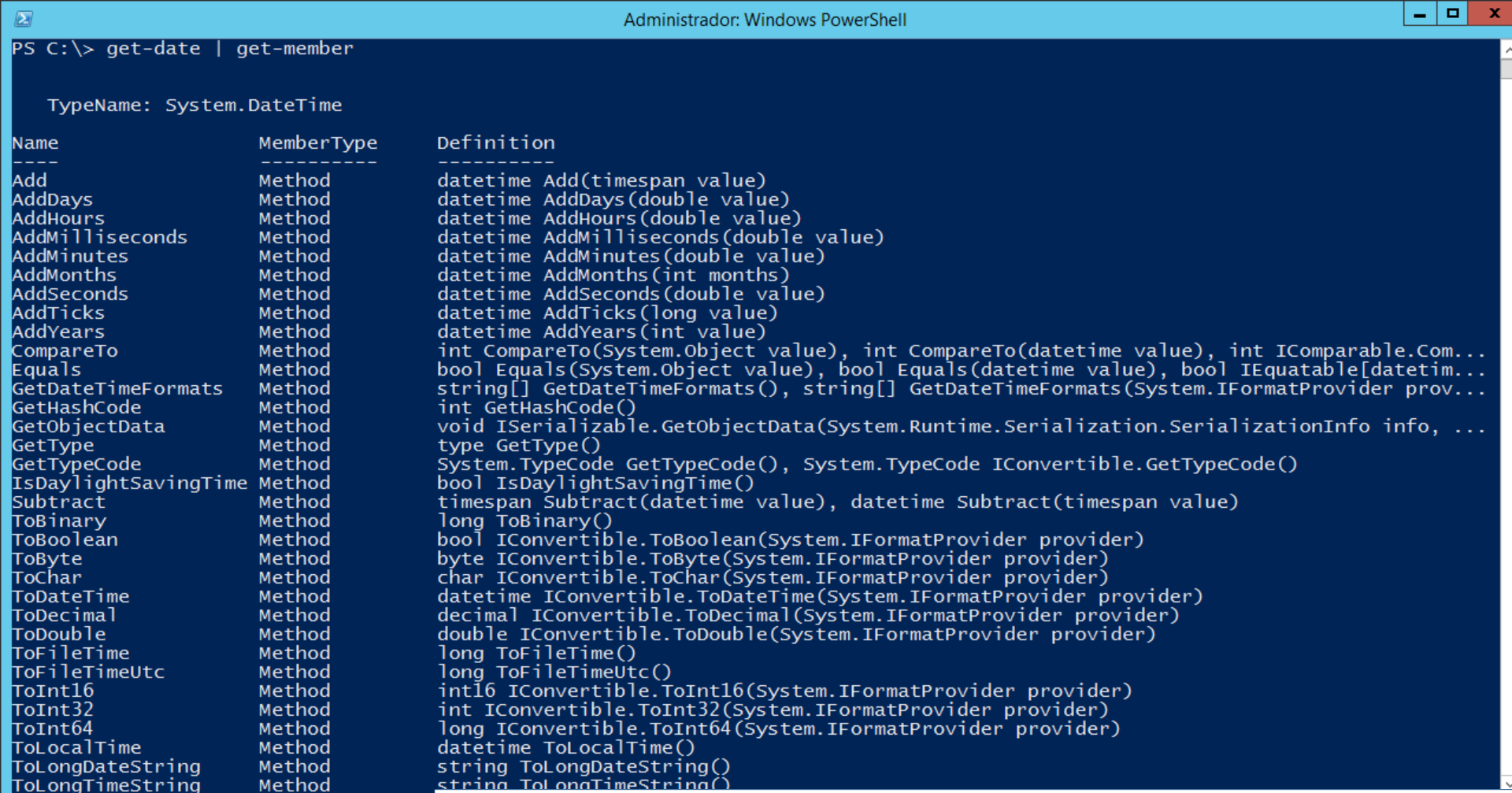
```
cmdlet1 | cmdlet2 | cmdlet3 | …
```



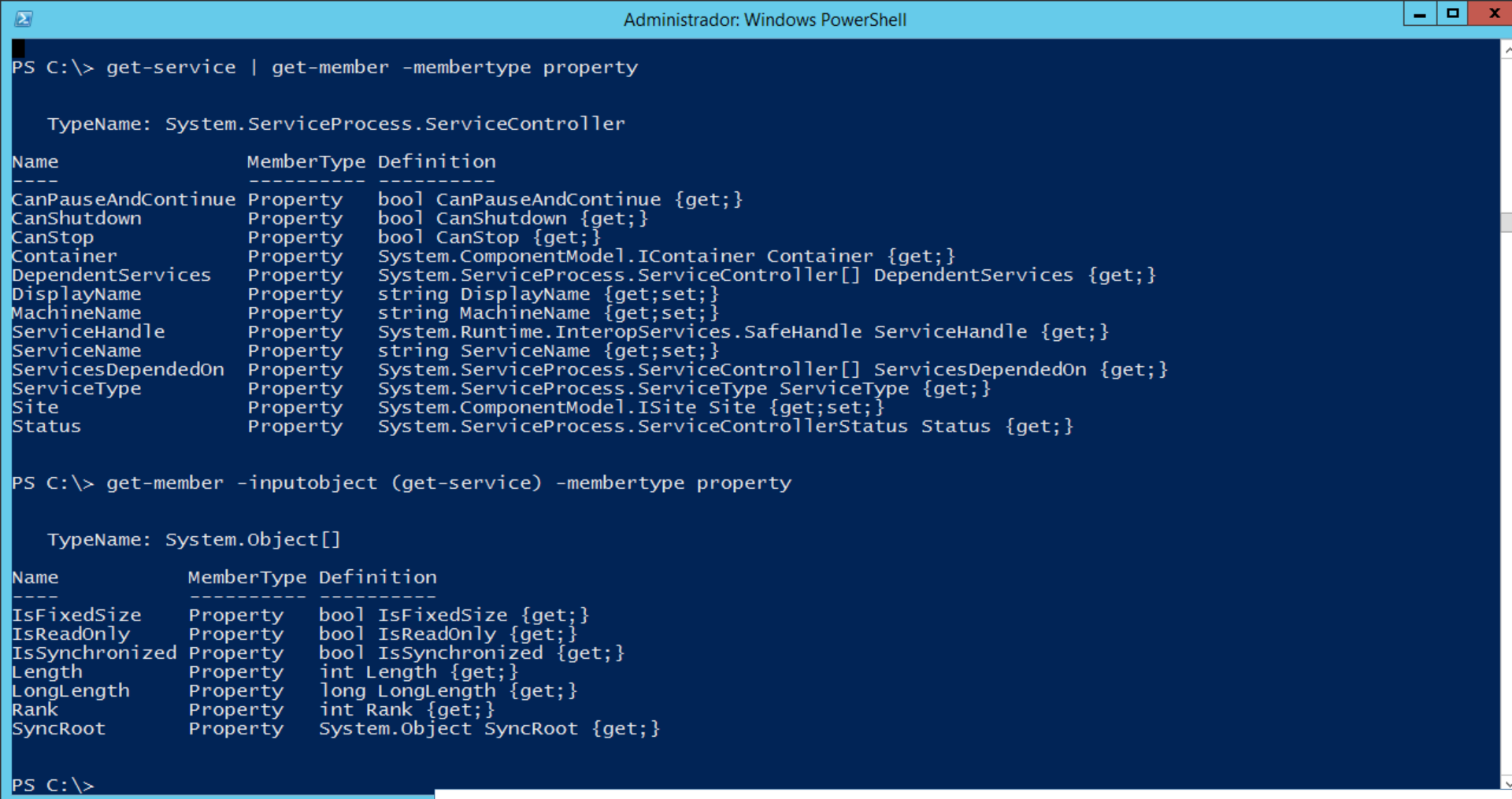
  - los objetos se envían **uno a uno**, mejorando la eficiencia

Objetos vs. texto

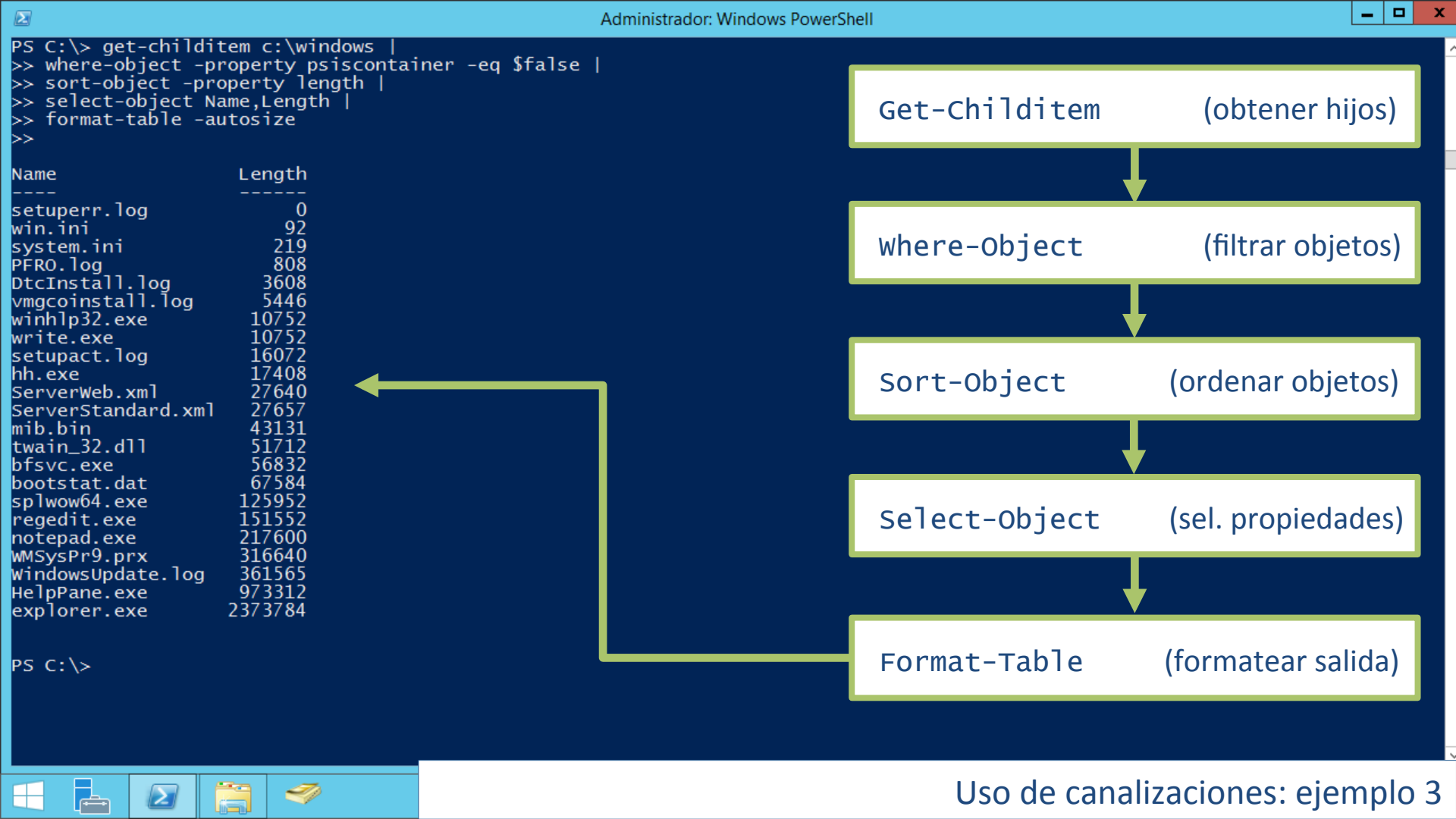
- En una canalización:
  - los *cmdlets* participantes se envían **objetos** (.NET)
  - el último *cmdlet* convierte sus objetos de salida **en texto**, y los muestra por pantalla
- El **intercambio** basado en **objetos**:
  - Clarifica y simplifica la **compatibilidad** entre *cmdlets*
  - **Evita**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



Uso de canalizaciones: ejemplo 1



Uso de canalizaciones: ejemplo 2



Uso de canalizaciones: ejemplo 3

Contenido

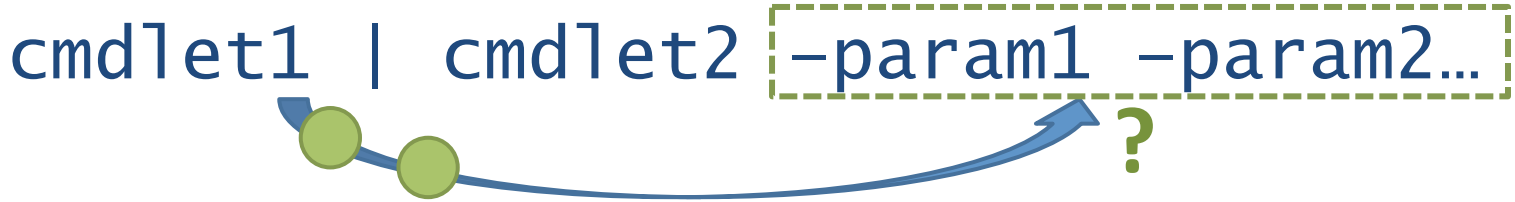
1. Concepto de canalización (*pipeline*)
2. Cómo funcionan las canalizaciones
3. *Cmdlets* útiles para canalizaciones

Cmdlets anterior y posterior


```
cmdlet1 | cmdlet2 -param1 -param2…
```

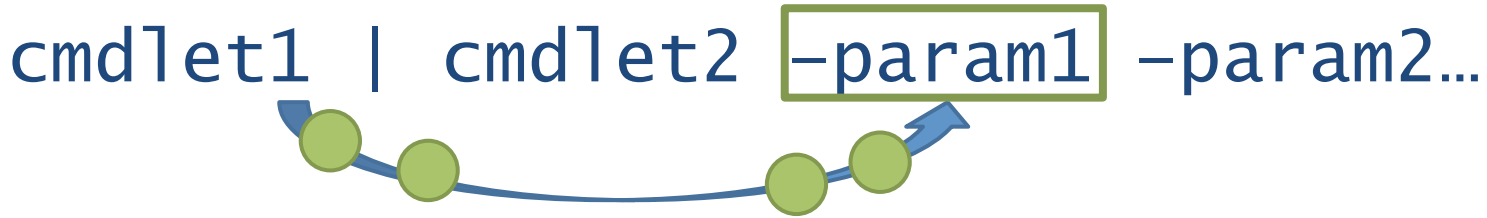
- El *cmdlet* **anterior** genera objetos y los envía a la canalización (sin restricciones)
- El *cmdlet* **posterior** debe estar diseñado para poder recibir datos de entrada de una canalización
  - Uno o varios **parámetros** de entrada

Asociación (*binding*)



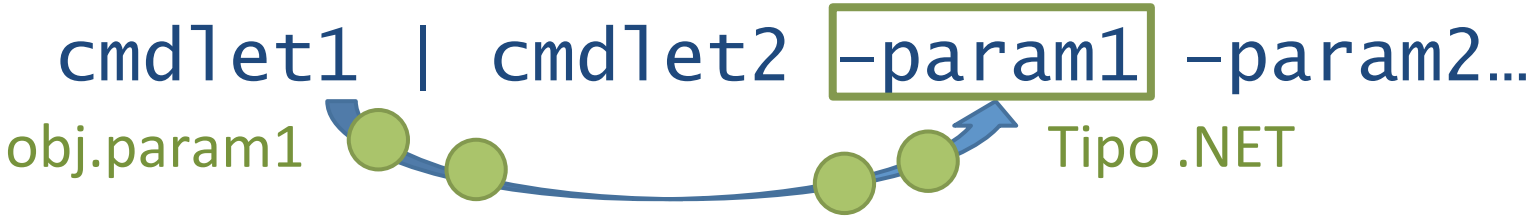
- PowerShell intenta **localizar** un **parámetro** que:
  - **Admita** recibir objetos mediante canalización
  - Tenga un **tipo compatible** con los objetos generados
  - **No** esté ya en uso por el *cmdlet*

Asociación (*binding*)

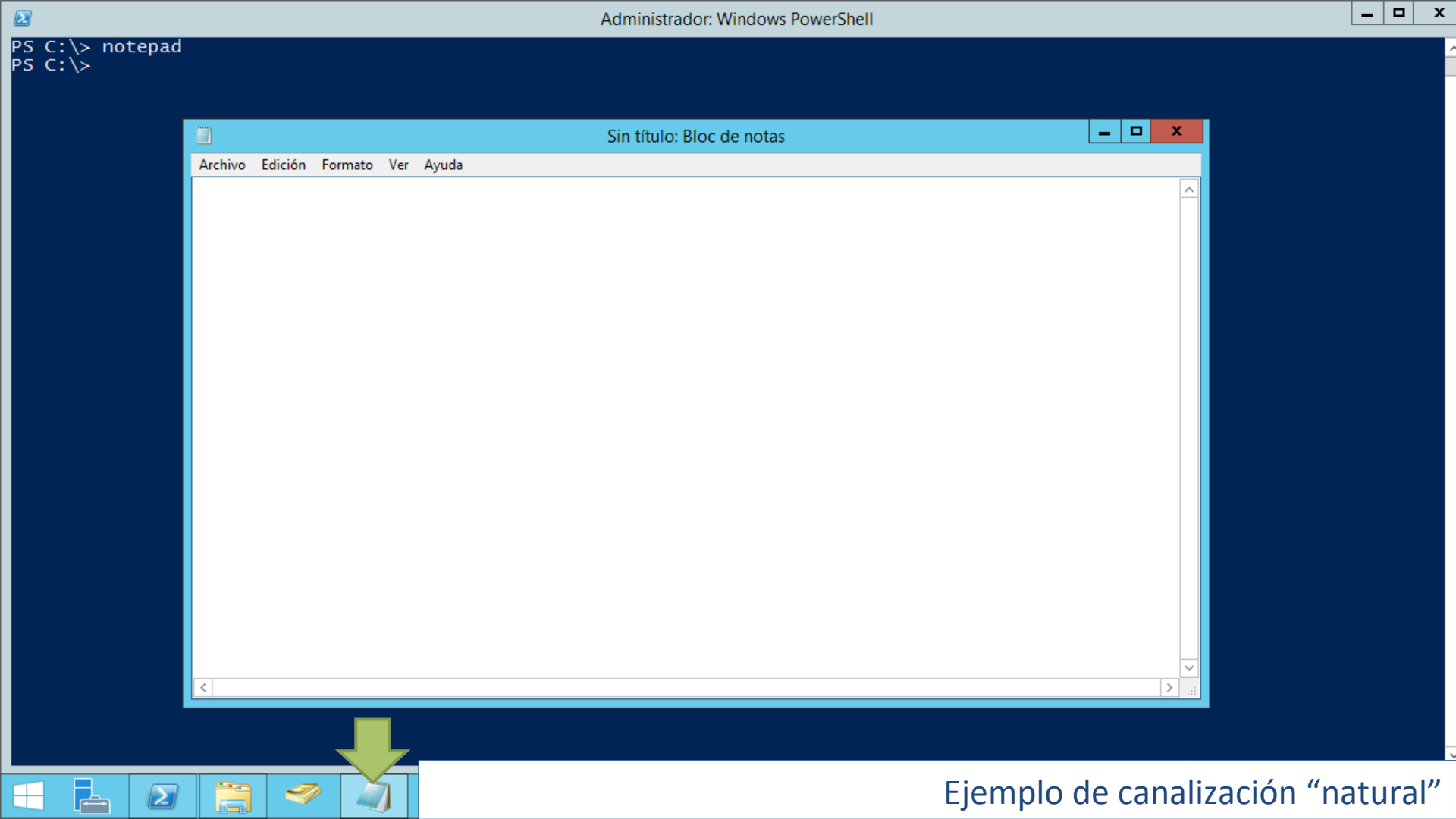


- El **resultado**:
  - Si 1 parámetro cumple, se utiliza
  - Si 1+ parámetros cumplen, se utiliza el **más eficiente**
  - Si ningún parámetro cumple, se genera un **error**

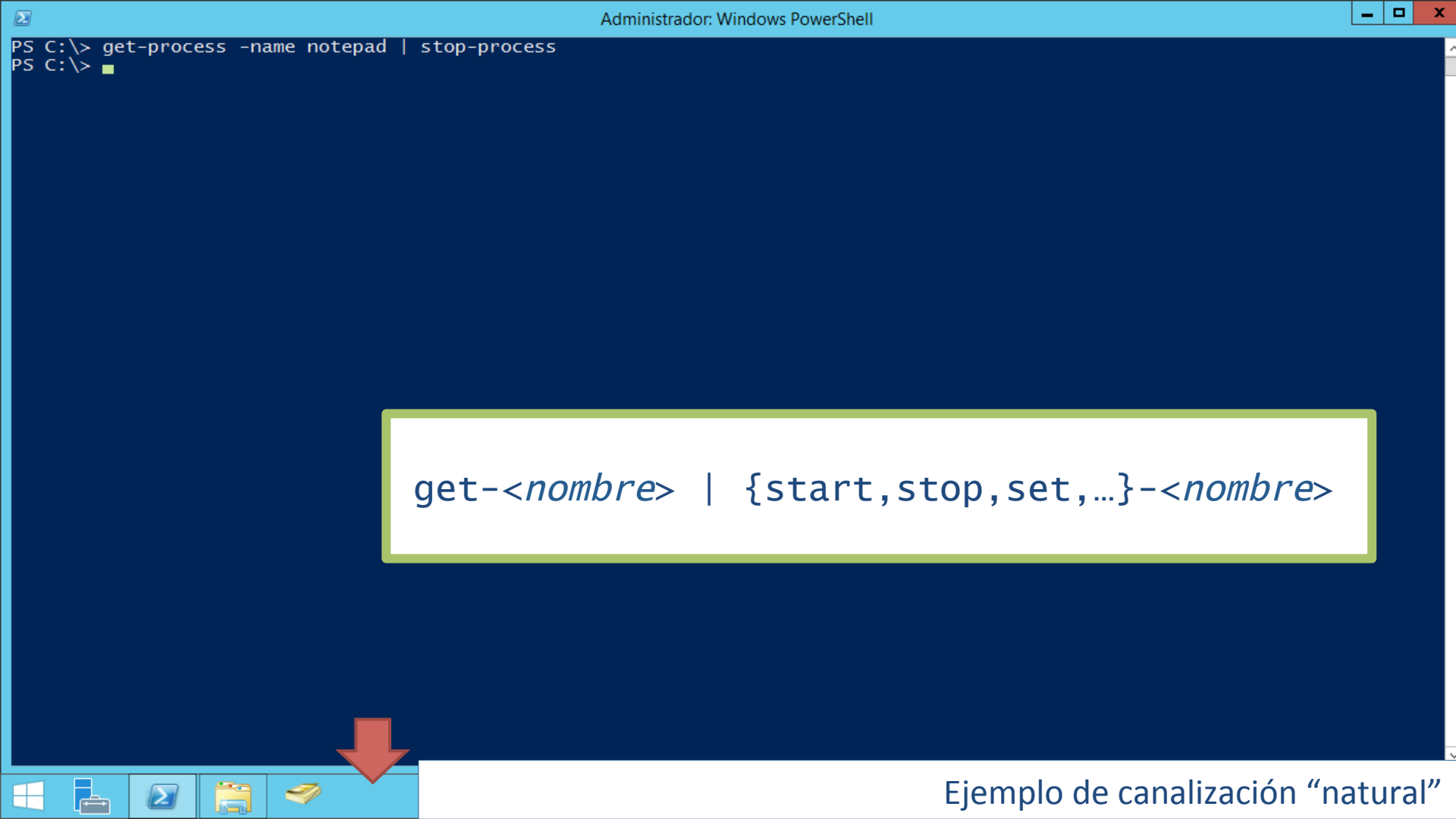
Métodos de aceptación



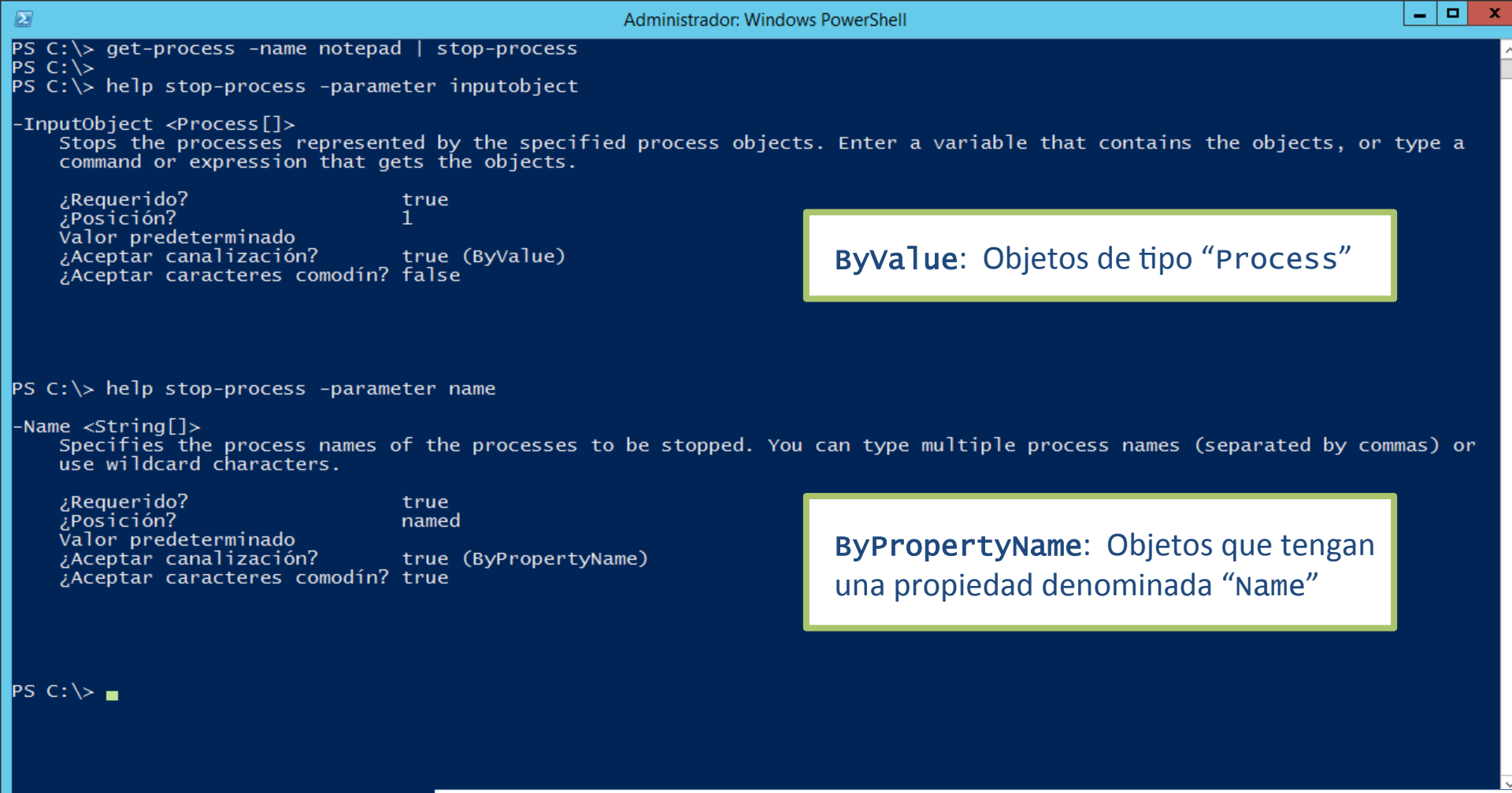
- Un **parámetro** puede aceptar objetos de 2 formas:
  - **`ByValue`**
    - Acepta si su **tipo** .NET es **compatible** con el tipo del objeto
  - **`ByPropertyName`**
    - Acepta si su **nombre coincide** con alguna propiedad del objeto



Ejemplo de canalización “natural”



Ejemplo de canalización “natural”



Averiguando qué parámetros aceptan canalizaciones

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 `$_` referencia el objeto actual)

Formatear/visualizar objetos

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

Cmdlet	Uso
`Import-CSV`	**Crea objetos** definidos por el usuario a partir del contenido de un fichero CSV (*comma-separated values*). 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

WINDOWS POWERSHELL

EJERCICIOS

CANALIZACIONES

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:

a) Obtener las propiedades de los objetos devueltos por `Get-Service`.

b) Obtener la lista de los servicios del sistema, ordenados por su estado ("`Status`").

c) Obtener la lista de los servicios del sistema cuyo estado sea en ejecución.

d) Obtener la lista de los servicios en ejecución, en orden alfabético inverso (NOTA: Ver propiedad "`-Descending`" de `Sort-Object`).

e) Obtener la cantidad de servicios en ejecución (NOTA: Ver `Measure-Object`).

f) 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:

a) Obtener las propiedades de los objetos devueltos por `Get-Process`.

b) Obtener la lista de los procesos en ejecución, ordenada por identificador del proceso.

c) Obtener la lista de los procesos en ejecución, ordenada inversamente por la cantidad de CPU utilizada por cada proceso.

d) 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 acho 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).

