Personalizar PowerShell modificando su perfil

Usted ya debe conocer el concepto de perfil porque se lleva utilizando desde hace mucho tiempo en Windows con, entre otros, el famoso «perfil Windows» (que puede ser local o móvil), así como el perfil Outlook. Un perfil es simplemente un archivo (o un conjunto de archivos) que contiene las preferencias del usuario y que le permite personalizar su entorno.

De ahora en adelante habrá que contar con perfiles adicionales, los de PowerShell. Y pueden ser numerosos ya que existen cuatro perfiles diferentes.

Es necesario en primer lugar distinguir dos tipos de perfiles:

- Los perfiles de usuario (siendo estos dos) que se aplican al usuario actual.
- Los perfiles de máquina (siendo estos dos también) que se aplican a los usuarios de una máquina determinada.

Otro concepto que es preciso conocer con PowerShell es el concepto de «Shell» o «entorno» en castellano. La consola instalada de origen con PowerShell constituye un entorno. Debe saber que Microsoft Exchange 2007 (plataforma de mensajería de empresa Microsoft) así como System Center Operation Manager 2007 (antigua MOM (*Microsoft Operation Manager*) es la solución de supervisión de sistemas) y todos los otros productos de la gama Microsoft System Center aparecidos después de 2009, tienen ya o tendrán su propia consola PowerShell; se trata también aquí de nuevos entornos. Microsoft ofrece, pues, con todo software, la posibilidad de crear un perfil PowerShell propio en cada entorno.

1. Perfiles de usuarios

Si en su empresa son varios administradores los que utilizan PowerShell, querrá que cada uno pueda personalizar su entorno de trabajo, y sin modificar el de su vecino. En este caso, este tipo de perfil esta hecho para usted.

Existen dos perfiles de usuario, cada uno con un nombre distinto:

- `%UserProfile%\Mis documentos\WindowsPowerShell\`**`profile.ps1`**
- `%UserProfile%\Mis documentos\WindowsPowerShell\`**`Microsoft.PowerShell_profile.ps1`**

El primer perfil es un perfil común a todos los entornos mientras que el segundo es propio del entorno PowerShell instalado por defecto. En otras palabras, si usted crea un archivo **`profile.ps1`**, todas las modificaciones hechas en éste serán válidas tanto en la consola Exchange, como en la consola SCOM, así como en la consola por defecto.

Debido a que el identificador del entorno PowerShell instalado por defecto se llama «Microsoft.PowerShell» el nombre del perfil comienza así. Para verificarlo, escriba el comando siguiente: `Get-Item variable:Shellid`

Ciertas consolas y, en particular la de PowerShell ISE, pueden tener en cuenta su propio perfil de usuario. Ejemplo del perfil PowerShell ISE: `%UserProfile%\Mis documentos\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1`

---

2. Perfiles de máquina

Todos los cambios que usted puede aportar a estos perfiles serán efectivos sólo en un ordenador pero se aplicarán a todos los usuarios.

Existen dos perfiles de máquinas, cada uno con nombre diferente:

- `%windir%\system32\WindowsPowerShell\v1.0\profile.ps1`
- `%windir%\system32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1`

Para la plataforma Windows 64 bits, el emplazamiento de estos archivos es diferente:

- `%windir%\syswow64\WindowsPowerShell\v1.0\profile.ps1`
- `%windir%\syswow64\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1`

El principio es el mismo que para los perfiles de usuarios, es decir, el archivo **`profile.ps1`** se aplicará a todos los entornos instalados en la máquina y a todos los usuarios, mientras que el segundo será específico del entorno **`Microsoft.PowerShell`**.

Es preferible manipular en primer lugar los perfiles de usuarios antes que los perfiles de máquinas ya que los primeros pueden seguir si utiliza los perfiles Windows móviles o si ha establecido una política de grupo que redirija su directorio Mis documentos hacia un partición de red. Si usted se encuentra en este último caso, y utiliza PowerShell en un servidor, no olvide desactivar la configuración de Modo protegido de Internet Explorer. Sin lo cual, en función de su política de ejecución del script en curso, PowerShell puede impedirle ejecutar su perfil.

Al igual que para el perfil de usuario, algunas consolas como PowerShell ISE pueden utilizar su propio perfil de máquina. Ejemplo del perfil máquina PowerShell ISE: %windir%\system32\WindowsPowerShell\v1.0\Microsoft.PowerShellISE_profile.ps1 y %windir%\syswow64\WindowsPowerShell\v1.0\Microsoft.PowerShellISE_profile.ps1

---

3. Orden de aplicación de los perfiles

El orden de aplicación de los perfiles es importante, PowerShell los aplica en este orden:

- `%windir%\system32\WindowsPowerShell\v1.0\profile.ps1`
- `%windir%\system32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1`
- `%UserProfile%\Mis documentos\WindowsPowerShell\profile.ps1`
- `%UserProfile%\Mis documentos\WindowsPowerShell\Microsoft.PowerShell_profile.ps1`

Como de costumbre los parámetros más próximos al usuario son los más prioritarios y, por tanto se aplican en último lugar. Por ejemplo, si usted define varias veces la misma variable en sus perfiles, la última definición que se aplique será la que prevalezca.

4. Creación del perfil

Por defecto, no se crea ningún perfil. El método más sencillo para crear su perfil consiste en apoyarse en la variable predefinida **`$profile`**. Esta variable contiene la ruta completa hacia su perfil de usuario de entorno por defecto **`Microsoft.PowerShell`**, y esto es así, aunque no lo haya creado todavía.

Veamos que contiene **`$profile`**:

```
PS > $profile
C:\Users\Eduardo\Documents\WindowsPowerShell\
Microsoft.PowerShell_profile.ps1
```

Para crear su perfil, teclee el comando:

```
PS > New-Item -Path $profile -ItemType file -Force
```

Felicitaciones, su perfil se ha creado pero sólo tiene cero bytes ya que está vacío. Para modificarlo con el bloque de notas, escriba el comando siguiente:

```
PS > notepad $profile
```

Ahora está usted preparado para personalizar su entorno preferido. Puede por ejemplo cambiar el color de fondo de la ventana, su tamaño, el color de los caracteres, añadir nuevos alias, o nuevas funciones, etc.

Veamos por ejemplo el contenido de nuestro perfil en este momento:

```
# profile.ps1 version 0.7
# Definición del alias Out-Clipboard para enviar el flujo
# al portapapeles.
#
Set-Alias -name Out-Clipboard -value 'c:\windows\system32\clip.exe'
Set-alias -name grep -value select-string

# Definición de las funciones
Function cd.. {cd ..}

# Modificación de las variables de preferencia
$VerbosePreference = 'continue' # por defecto "silentlycontinue"
$DebugPreference   = 'continue'
$WarningPreference = 'continue'

# Mensaje de bienvenida personalizado
#
$UserType = 'Usuario'
$CurrentUser = [System.Security.Principal.WindowsIdentity]::GetCurrent()
$principal =
   new-object System.Security.principal.windowsprincipal($CurrentUser)
if ($principal.IsInRole('Administradores'))
{
    $UserType = 'Administrador'
    $host.ui.RawUI.BackGroundColor = 'DarkMagenta'
    Clear-Host
}
else
{
    $host.ui.RawUI.BackGroundColor = 'DarkMagenta'
    Clear-Host
}

Write-Host '+------------------------------------------------+'
Write-Host "+- Hola $(($CurrentUser.Name).split('\')[1])"
Write-Host "+- Usted está conectado como: $UserType"
Write-Host '+------------------------------------------------+'

# Modificación del color del prompt a amarillo, reemplazar
# el Prompt por PS > y mostrar la ruta actual en la barra de título
# en la ventana de la consola
function prompt
{
    Write-Host ('PS ' + '>') -nonewline -fore yellow
    $host.ui.RawUI.Set_windowtitle("$(get-location) ($UserType)")
    return ' '
```

```
}
```

Vamos a ver en la parte siguiente, un amplio abanico de lo que es posible hacer para personalizar la ventana PowerShell.

5. Personalización del entorno

Todo lo que vamos a ver ahora está hecho para incluirlo en su perfil. Puede elegir cuál será el fichero de perfil más adecuado a sus necesidades.

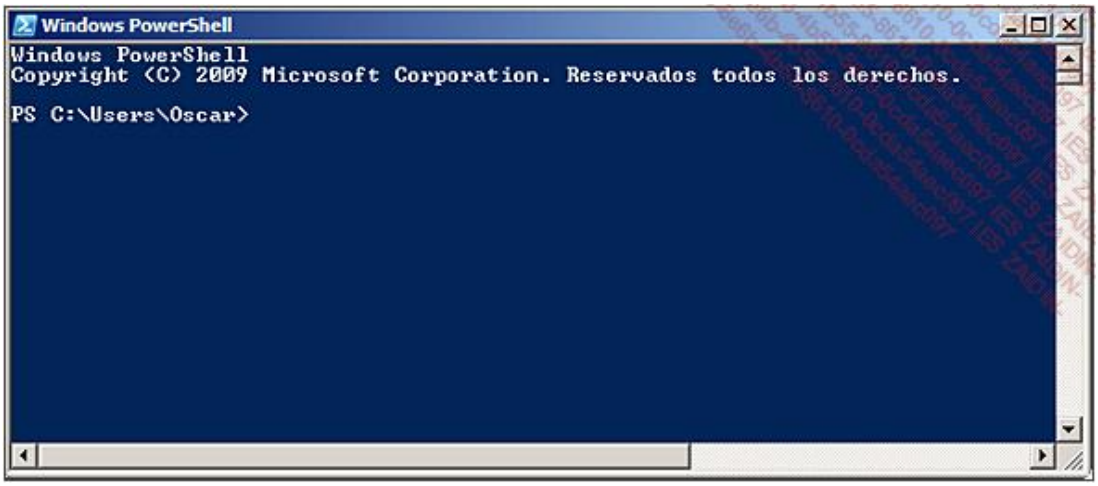
a. Modificación del prompt

El prompt es el conjunto de caracteres que indica que el ordenador está preparado para recibir una entrada mediante el teclado.

Por defecto presenta la forma siguiente: *PS RUTA_EN_CURSO>*

Se encontrará, al iniciar PowerShell, en el directorio raíz de su perfil Windows (en Windows 7 y Vista: **C:\Users\NombreDelPerfil**, en Windows XP: **C:\Documents and Settings\NombreDelPerfil)**.

Veamos como se muestra en Windows 7 el prompt por defecto:



Vista del promt por defecto

Para cambiarlo, bastará modificar la función **Prompt** intrínseca de PowerShell. Veamos qué contiene en su configuración original. Teclee el comando `Get-Content function:prompt`. He aquí el resultado obtenido con PowerShell v1:

```
PS > Get-Content function:prompt
'PS ' + $(Get-Location) + $(if ($nestedpromptlevel -ge 1) { '>' }) + '> '
```

Y aquí con PowerShell v2:

```
PS > Get-Content function:prompt
$(if (test-path variable:/PSDebugContext) { '[DBG]: ' } else { '' }) `
+ 'PS ' + $(Get-Location) + $(if ($nestedpromptlevel -ge 1) { '>>' }) + '> '
```

La función **Prompt** por defecto puede parecer un poco burda a primera vista pero mirándola detenidamente se podrán observar los siguientes elementos:

- Concatena cuatro cadenas de caracteres separadas por el operador de adición « + ».
- **$(Get-Location)** devuelve la ruta en curso.

- **`$nestedpromptlevel`** indica si nos encontramos en un entorno anidado o no (véase el capítulo Gestión de los errores y depuración, sección La depuración - Los puntos de interrupción (break points)). Si esta variable contiene un número superior a cero, nos encontramos en un entorno anidado, entonces en este caso se añade al prompt un carácter «>» adicional.

- Por último se añade al prompt el carácter final «>» seguido de un espacio para que la entrada no quede pegada al prompt.

- Destacar que en PowerShell v2, una verificación sobre las condiciones de depuración se efectúa al inicio de la función (véase capítulo Gestión de los errores y depuración para conocer el significado de la verificación).

Es muy fácil redefinir esta función, así podríamos por ejemplo suprimir del promt la ruta en curso, ya que con frecuencia a causa de esto, el promt es infinitamente largo cuando se exploran estructuras donde muchos directorios están interrelacionados. Sin embargo, para no privarse de esta interesante información, vamos a mostrarla en el título de la ventana, en lugar del título habitual «Windows PowerShell».

```
function prompt
{
    'PS > '
    $host.ui.RawUI.set_windowtitle($(get-location))
}
```

Observará que el título de la ventana se actualiza cada vez que cambiamos de directorio. La realidad es un poco diferente ya que la función **`Prompt`** de hecho se recalcula cada vez que PowerShell nos devuelve el prompt para introducir una nueva línea de comandos.

`$host` es el objeto que corresponde a nuestro entorno. Posee un gran número de propiedades y métodos que pueden servir para personalizar nuestra ventana PowerShell.

Un prompt coloreado

Para dar un pequeño toque simpático a nuestro prompt, podemos añadirle un poco de color, del siguiente modo:

```
function prompt
{
    Write-Host ('PS ' + $(get-location) +'>') `
-NoNewLine -ForegroundColor yellow
    ' '
}
```

Actuando de este modo, mostraremos una cadena de caracteres en color con la commandlet **`Write-Host`**, a la que pedimos no cambiar de línea con el conmutador **`-NoNewLine`**. Luego redefinimos nuestro prompt a su mínima expresión: un espacio. Es obligatorio que la función prompt devuelva una cadena de caracteres, porque sino el prompt por defecto « PS>" aparece. En lugar de escribir «' '» en la función, lo que puede parecer un poco extraño, habríamos podido escribir **`return ' '`**. Para más información referente al retorno de las funciones, consulte el capítulo Fundamentos - Las funciones.

Un prompt siempre en hora

¿Tal vez prefiere mostrar la fecha y hora en lugar de la ruta actual?

Nada más sencillo, pruebe lo siguiente:

```
function prompt
{
    Write-Host ('PS ' + $(get-date) +'>') -NoNewLine -Foreg yellow
    return ' '
}
```

```
PS 09/18/2009 23:45:46>
```

Puede hacer cualquier tipo de cosas en la función **Prompt**, pero recuerde esto: su función deberá devolver siempre un valor de tipo String, sin lo cual PowerShell mostrará el prompt por defecto "PS>"; para mayor claridad pruebe a limitar su prompt a una sola línea, lo más breve posible; a cada retorno al prompt, es decir al final de cada comando, PowerShell volverá a evaluar la función **Prompt**. Intente pues no hacer demasiadas cosas complicadas en su función, lo que podría tener como consecuencia un cierto ralentizamiento del sistema.

b. Modificar el tamaño de la ventana

Puede actuar sobre la ventana de la consola para modificar su tamaño, su color, su título, su posición, etc.

PowerShell le permite actuar sobre la consola a través del objeto **host.ui.RawUI**. Listemos sus propiedades para ver sobre cuáles podríamos actuar:

```
PS > $host.UI.RawUI

ForegroundColor       : DarkYellow
BackgroundColor       : DarkMagenta
CursorPosition        : 0,2999
WindowPosition        : 0,2948
CursorSize            : 25
BufferSize            : 140,3000
WindowSize            : 140,52
MaxWindowSize         : 140,81
MaxPhysicalWindowSize : 182,81
KeyAvailable          : False
WindowTitle           : www.PowerShell-Scripting.com : Usuario
```

Si queremos ajustar horizontalmente nuestra ventana no sólo tendremos que modificar el tamaño de ésta sino también será necesario modificar la memoria tampón (el buffer) asociado. Esto se hace del siguiente modo:

```
PS > $buff = $host.ui.RawUI.BufferSize    # inic. de la variable $buff
PS > $buff.width = 150                    # def. del num. de car. por línea
PS > $buff.Height = 3000                  # def. del num. de las líneas v...
PS > $host.ui.RawUI.BufferSize = $buff
PS > $tamaño = $host.ui.RawUI.WindowSize  # se inicializa la variable
PS > $tamaño.Width = $buff.width          # num. de caracteres en la hori...
PS > $tamaño.Height = 60                  # numero de líneas verticales
PS > $host.ui.RawUI.WindowSize = $tamaño
```

El tamaño de la memoria tampón y el de la ventana deben ser exactamente idénticos si no quiere tener acceso con la barra de desplazamiento horizontal.

c. Modificación de los colores

Tiene la opción de elegir los colores de su entorno preferido, tanto para los caracteres como para el color de fondo de la ventana.

He aquí la lista de los colores posibles:

Black	Blue	Cyan	DarkBlue
DarkCyan	DarkGray	DarkGreen	DarkMagenta

DarkRed	DarkYellow	Gray	Green
Magenta	Red	White	Yellow

Para asignarlos, puede basarse en estos ejemplos:

```
PS > $host.ui.RawUI.ForeGroundColor = 'White' # Color del texto
PS > $host.ui.RawUI.BackGroundColor = 'Black' # Color del fondo
```

Cuando cambiamos el color de fondo de la ventana con **`$host.ui.RawUI.BackGroundColor`**, es necesario que ejecutemos a continuación un **`Clear-Host`** o **`cls`**. Si no lo hace, el color de fondo sólo se aplicará a los nuevos caracteres; lo que no es normalmente un efecto muy bonito. También puede asignar colores diferentes a los definidos por defecto para los mensajes de error y depuración. Para consultarlo, teclee `$host.privatedata`.

Veamos la lista de las propiedades y de los colores por defecto:

```
PS > $host.privatedata
ErrorForegroundColor    : Red
ErrorBackgroundColor    : Black
WarningForegroundColor  : Yellow
WarningBackgroundColor  : Black
DebugForegroundColor    : Yellow
DebugBackgroundColor    : Black
VerboseForegroundColor  : Yellow
VerboseBackgroundColor  : Black
ProgressForegroundColor : Yellow
ProgressBackgroundColor : DarkCyan
```

d. Modificación del título de la ventana

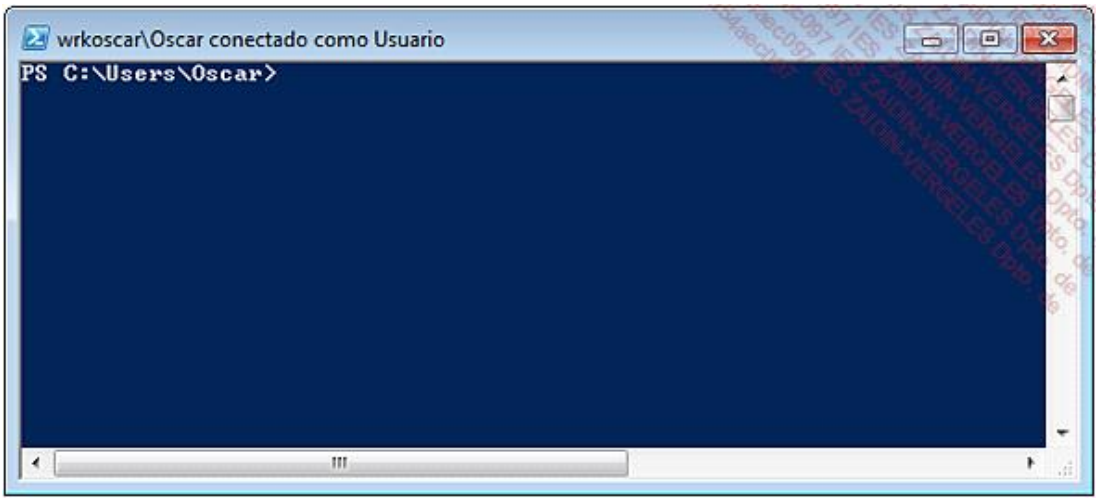
El título de la ventana se modifica gracias a la propiedad **`WindowTitle`**, como se muestra a continuación:

```
PS > $host.ui.RawUI.WindowTitle = 'www.PowerShell-Scripting.com'
```

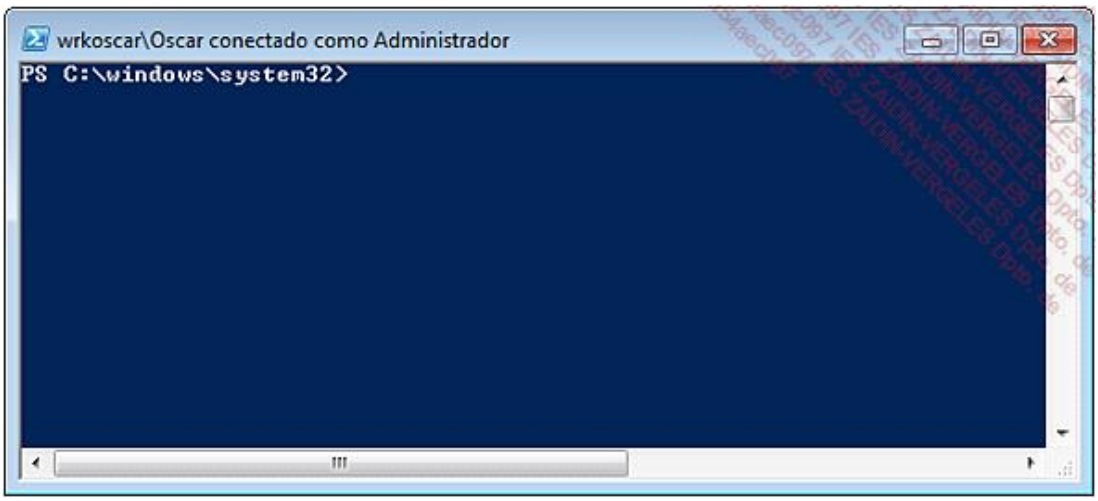
Recuerde que esta propiedad no es dinámica. No podrá mostrar la hora del sistema y ver cómo se actualiza en tiempo real. Por el contrario, puede utilizar la función prompt que se encargará de actualizar el título de la ventana regularmente.

Veamos un ejemplo donde mostraremos, en función del usuario conectado, su perfil (usuario o administrador) en el título de la ventana. Este ejemplo, no tiene ningún interés en Windows XP o Windows Server, pero tiene sentido con Windows 7 y Vista en la medida en que incluso conectado como Administrador, decida ejecutar por defecto PowerShell como simple usuario.

```
$UserType = 'Usuario'
$CurrentUser = [System.Security.Principal.WindowsIdentity]::GetCurrent()
$principal =
new-object System.Security.principal.windowsprincipal($CurrentUser)
if ($principal.IsInRole('Administrators'))
{
    $UserType = 'Administrador'
}
$host.ui.RawUI.WindowTitle = "$($CurrentUser.Name) conectado como $UserType"
```



Modificar el título de la consola: Modo Usuario



Modificar el título de la consola: Modo Administrador

En Windows 7 y Vista, para iniciar PowerShell en modo administrador, deberá hacer clic en el botón secundario del ratón en el icono PowerShell y seleccionar «Ejecutar como administrador».

e. Adición de un mensaje de bienvenida personalizado

A demás de modificar el título de su ventana para indicar el estado del usuario conectado, podrá también mostrarlo al iniciar su consola PowerShell añadiendo el código adecuado a su perfil.

```
$UserType = 'Usuario'
$CurrentUser =[System.Security.Principal.WindowsIdentity]::GetCurrent()
$principal =
New-Object System.Security.principal.windowsprincipal($CurrentUser)
if ($principal.IsInRole('Administrators'))
{
    $UserType = 'Administrador'
}
Write-Host '+-------------------------------------------------+'
Write-Host "+- Buenos días $($CurrentUser.Name)"
Write-Host "+- Usted está conectado como: $UserType"
Write-Host '+-------------------------------------------------+'
```

Resultado:

```
------------------------------------------------+
+- Buenos días wrkoscar\Oscar
+- Usted está conectado como: Administrador
+------------------------------------------------+
```

También puede hacer que el fondo de la ventana aparezca en rojo, añadiendo el código siguiente en el bloque If:

```
$host.ui.RawUI.BackGroundColor='Red'
Clear-Host
```

f. Consideremos los scripts externos

A fuerza de utilizar PowerShell se irá constituyendo una biblioteca de scripts importante para poder reutilizarlos. Puede querer añadir sus creaciones a su perfil, pero esto tiene el peligro de sobrecargarla y hacerla difícilmente legible. Ofrecemos un pequeño fragmento de código para importarlas fácilmente en su entorno.

```
Write-Host 'Importación de scripts externos'
Get-ChildItem "$home\Scripts" | Where {$_.extension -eq '.ps1'} |
    Foreach {$_.fullname; . $_.fullname}
```

Este script busca todos los archivos con extensión « **.ps1** » en el directorio **$home\Scripts**, y después los ejecuta uno a uno en el ámbito actual.

Get-ChildItem "$home\Scripts" lista todos los archivos del directorio **Scripts** y los pasa uno a uno a la cláusula **Where** a través de la tubería. La cláusula **Where** sirve de filtro. Examina la extensión del archivo y mira si es « **.ps1** » (la verificación no es sensible a mayúsculas y minúsculas). Si la cláusula **Where** es verdadera entonces nuestro objeto archivo pasa al siguiente comando de la tubería. La instrucción **Foreach** toma entonces el relevo y para cada objeto recibido, va a devolver su nombre, después ejecutará el script en el ámbito en curso (gracias al punto y al espacio ". " colocados delante del nombre del script - es la técnica del *DotSourcing*).

Para que este script sea plenamente utilizable, necesitará escribir sus scripts en forma de función (o de filtro). Así una vez sus scripts estén cargados en memoria, no tendrá más que llamar al nombre de su función (o filtro).

g. Consideremos los archivos de definiciones de tipos personalizados

Si todavía desconoce lo que son los archivos de tipos personalizados, puede saltarse este párrafo y regresar a él posteriormente. Para los demás, continuemos...

Exactamente sobre el mismo principio que anteriormente, vamos a buscar todos los archivos cuyo nombre termine por ***.types.ps1xml**, después los importaremos gracias al comando **Update-TypeData**.

```
Write-Host 'Importación de los tipos personalizados'
gci "$home\Scripts" | ? {$_.name -match 'types.ps1xml'} |
% {$_.fullname; update-typedata $_.fullname}
```

Esta vez hemos reemplazado las instrucciones **Where** y **Foreach** por sus alias respectivos **?** y **%**.

h. Consideremos los archivos de formateo personalizados

Ya que estamos, hagamos lo mismo para importar los archivos de formateo personalizados. Esta vez vamos a buscar los archivos cuyo nombre termina por ***.format.ps1xml**.

```
Write-Host ’Importación de las vistas personalizadas’
gci "$home\Scripts" | ? {$_.name -match ’format.ps1xml’} |
% {$_.fullname; update-formatdata -prepend $_.fullname}
```