

PowerShell

Anexo

Import-Csv

Supongamos el siguiente archivo de productos:

```

1 Código;Tipo;Nombre;Precio;
2 1;Yerba Mate;Aguantadora;120;
3 2;Harina;Blancaflor;70;
4 3;Yerba Mate;Cruz de Malta;180;
5 4;Café;La Morenita;140;
6 5;Té;La Virginia;60;
7 6;Yerba Mate;Secadero;130;
8 7;Leche en polvo;Cremigal;199;
9 8;Dulce de leche;Estancia "El Placer";90;

```

Vemos que respeta formato de archivo separado por comas (CSV), en este caso utiliza el ; como separador de campos. A continuación leeremos su contenido a través del comando Import-Csv y lo enviaremos a pantalla:

```

1 Param(
2     [ Parameter( Mandatory = $True, HelpMessage = 'Archivo de productos con extensión .csv') ]
3     [ ValidateScript( {
4         if( Test-Path $_ ){ $True } else{ Throw "La ruta al archivo no es válida" }
5         if( (Get-Item $_) -is [System.IO.FileInfo] ){ $True } else{ Throw $_+" no es un archivo" }
6     } ) ]
7 )
8 [ ValidateNotNullOrEmpty() ]
9 [ String ] $archivo
10 )
11
12 Import-Csv $archivo -Delimiter ';' -Encoding UTF7

```

Agregamos algunas validaciones de parámetros sobre el archivo de entrada y especificamos en el comando Import-Csv, el delimitador y la codificación del archivo por si tuviéramos problemas con acentos y la letra ñ.

```

Administrator: Windows PowerShell
PS C:\PowerShell> .\ImportCsv.ps1 .\Productos.csv

```

Código	Tipo	Nombre	Precio
1	Yerba Mate	Aguantadora	120
2	Harina	Blancaflor	70
3	Yerba Mate	Cruz de Malta	180
4	Café	La Morenita	140
5	Té	La Virginia	60
6	Yerba Mate	Secadero	130
7	Leche en polvo	Cremigal	199
8	Dulce de leche	Estancia "El Placer"	90

Ahora, supongamos que queremos actualizar el precio de todos los productos de un tipo dado, incrementando su precio actual en cierto porcentaje:

```

1 Param(
2     [ Parameter( Mandatory = $True, HelpMessage = "Archivo de productos con extensión .csv") ]
3     [ ValidateScript( {
4         if( Test-Path $_ ){ $True } else{ Throw "La ruta al archivo no es válida" }
5         if( (Get-Item $_) -is [System.IO.FileInfo] ){ $True } else{ Throw $_ " no es un archivo" }
6     } )
7 ]
8     [ ValidateNotNullOrEmpty() ][ String ] $archivo,
9     [ Parameter( Mandatory = $True ) ][ ValidateRange(0,100) ][ Int ] $porcentaje ,
10    [ Parameter( Mandatory = $False ) ][ String ] $tipo = "Todos"
11 )
12
13 $hash = [ordered]@{}
14 Import-Csv $archivo -Delimiter ';' -Encoding UTF7 |
15     ForEach-Object {
16         $precio_nuevo = [Int]$_ .Precio
17         if( $tipo -eq $_.Tipo -or $tipo -eq "Todos" ){
18             $precio_nuevo+= $precio_nuevo * ($porcentaje/100)
19         }
20         $hash.add( $_.Código, @( $_.Tipo, $_.Nombre, $precio_nuevo ) )
21     }
22
23 $hash

```

Agregamos algunas validaciones mas, como el rango del porcentaje y un valor por defecto para el tipo de producto, dado que también queremos poder actualizar todos los productos de alguna forma, en caso de no pasar parámetro a la variable tipo, ésta tomará como valor el valor “Todos”.

Declaramos un array asociativo ordenado por clave de manera ascendente (variable \$hash), ahora la salida del Import-Csv la enviamos a un Foreach, en el cual actualizaremos el precio si corresponde, y almacenaremos en el array asociativo como clave el código, y como valor un array normal con el tipo, nombre y precio actualizado como elementos.

Al ejecutar por ejemplo para la Yerba Mate a un 50 % de aumento:

```

Administrator: Windows PowerShell
PS C:\PowerShell> .\ImportCsv.ps1 .\Productos.csv 50 "Yerba Mate"

Name                               Value
----                               -
1 {Yerba Mate, Aguantadora, 180}
2 {Harina, Blancaflor, 70}
3 {Yerba Mate, Cruz de Malta, 270}
4 {Café, La Morenita, 140}
5 {Té, La Virginia, 60}
6 {Yerba Mate, Secadero, 195}
7 {Leche en polvo, Cremigal, 199}
8 {Dulce de leche, Estancia "El Placer", 90}

```

Al ejecutar para todos un 50%

```

Administrator: Windows PowerShell
PS C:\PowerShell> .\ImportCsv.ps1 .\Productos.csv 50

Name                               Value
----                               -
1 {Yerba Mate, Aguantadora, 180}
2 {Harina, Blancaflor, 105}
3 {Yerba Mate, Cruz de Malta, 270}
4 {Café, La Morenita, 210}
5 {Té, La Virginia, 90}
6 {Yerba Mate, Secadero, 195}
7 {Leche en polvo, Cremigal, 298,5}
8 {Dulce de leche, Estancia "El Placer", 135}

```

Ahora supongamos que queremos mejorar la salida por pantalla, para que se parezca a la visualizada con el comando Import-Csv, podemos hacer algo con el comando Format-Table:

```

1 Param(
2     [ Parameter( Mandatory = $True, HelpMessage = "Archivo de productos con extensión .csv") ]
3     [ ValidateScript( {
4         if( Test-Path $_ ){ $True } else{ Throw "La ruta al archivo no es válida" }
5         if( (Get-Item $_) -is [System.IO.FileInfo] ){ $True } else{ Throw $_+" no es un archivo" }
6     } )
7 ]
8     [ ValidateNotNullOrEmpty() ][ String ] $archivo,
9     [Parameter( Mandatory = $True )][ ValidateRange(0,100) ][ Int ] $porcentaje ,
10    [Parameter( Mandatory = $False )][ String ] $tipo = "Todos"
11 )
12
13 $hash = [ordered]@{}
14 Import-Csv $archivo -Delimiter ';' -Encoding UTF7 |
15     ForEach-Object {
16         $precio_nuevo = [Int]$_ .Precio
17         if( $tipo -eq $_.Tipo -or $tipo -eq "Todos"){
18             $precio_nuevo+=$precio_nuevo*($porcentaje/100)
19         }
20         $hash.add( $_.Código, @( $_.Tipo, $_.Nombre, $precio_nuevo ) )
21     }
22
23 $hash | Format-Table @{Label='Código';Expression={$_.Name}},
24     @{Label='Tipo';Expression={$_.Value[0]}},
25     @{Label='Nombre';Expression={$_.Value[1]}},
26     @{Label='Precio';Expression=[Int32]$_ .Value[2]} -AutoSize
27

```

Ejecutamos:

```

Administrator: Windows PowerShell
PS C:\PowerShell> .\ImportCsv.ps1 .\Productos.csv 50

Código Tipo      Nombre          Precio
-----
1      Yerba Mate     Aguantadora     180
2      Harina         Blancaflor      105
3      Yerba Mate     Cruz de Malta   270
4      Café          La Morenita     210
5      Té            La Virginia     90
6      Yerba Mate     Secadero        195
7      Leche en polvo Cremigal    298
8      Dulce de leche Estancia "El Placer" 135

```

Export-Csv

Ahora supongamos que queremos actualizar el archivo original con los nuevos precios, utilizaremos el comando Export-Csv. La última línea de código hace la exportación, pero previamente se realiza un Foreach-Object donde para cada entrada de \$hash se agrega en el array \$salida un objeto con los datos correspondientes a la entrada analizada. Notemos que se utiliza el comando New-Object para crear el nuevo objeto.

```

1 Param(
2     [ Parameter( Mandatory = $True, HelpMessage = "Archivo de productos con extensión .csv") ]
3     [ ValidateScript( {
4         if( Test-Path $_ ) { $True } else{ Throw "La ruta al archivo no es válida" }
5         if( (Get-Item $_) -is [System.IO.FileInfo] ) { $True } else{ Throw $_ + " no es un archivo" }
6     } ) ]
7 )
8 [ ValidateNotNullOrEmpty() ][ String ] $archivo,
9 [Parameter( Mandatory = $True )][ ValidateRange(0,100) ][ Int ] $porcentaje ,
10 [Parameter( Mandatory = $False )][ String ] $tipo = "Todos"
11 )
12
13 $hash = [ordered]@{}
14 Import-Csv $archivo -Delimiter ';' -Encoding UTF7 |
15    ForEach-Object {
16         $precio_nuevo = [Int]$_ .Precio
17         if( $tipo -eq $_.Tipo -or $tipo -eq "Todos" ){
18             $precio_nuevo += $precio_nuevo * ($porcentaje / 100)
19         }
20         $hash.add( $_.Codigo, @( $_.Tipo, $_.Nombre, $precio_nuevo ) )
21     }
22
23 $salida = @()
24 foreach ( $i in $hash.Keys ){
25     $salida += New-Object PsObject -property ([ordered]@{
26         'Codigo' = $i
27         'Tipo' = $hash[$i][0]
28         'Nombre' = $hash[$i][1]
29         'Precio' = $hash[$i][2]
30     })
31 }
32 $salida | Export-Csv $archivo -delimiter ';' -NoTypeInfo -Encoding UTF8

```

ConvertTo-Json

Si quisiéramos por ejemplo exportar los datos actualizados a un archivo con formato JSON, podemos hacerlo con el comando `ConvertTo-Json`, solo habría que agregar la línea al final del archivo:

```
$salida | ConvertTo-Json | Out-File salida.json
```

Si vemos el archivo de salida:

```

1 [
2     {
3         "Codigo": "1",
4         "Tipo": "Yerba Mate",
5         "Nombre": "Aguantadora",
6         "Precio": 180
7     },
8     {
9         "Codigo": "2",
10        "Tipo": "Harina",
11        "Nombre": "Blancaflor",
12        "Precio": 105
13    },
14    {
15        "Codigo": "3",
16        "Tipo": "Yerba Mate",
17        "Nombre": "Cruz de Malta",
18        "Precio": 270
19    },
20    {
21        "Codigo": "4",
22        "Tipo": "Café",
23        "Nombre": "La Morenita",
24        "Precio": 210
25    },
26    {
27        "Codigo": "5",
28        "Tipo": "Té",
29        "Nombre": "La Virginia",
30        "Precio": 90
31    },
32    {
33        "Codigo": "6",
34        "Tipo": "Yerba Mate",
35        "Nombre": "Secadero",
36        "Precio": 195
37    },
38    {
39        "Codigo": "7",
40        "Tipo": "Leche en polvo",
41        "Nombre": "Cremigal",
42        "Precio": 298.5
43    },
44    {
45        "Codigo": "8",
46        "Tipo": "Dulce de leche",
47        "Nombre": "Estancia \"El Placer\"",
48        "Precio": 135
49    }
50 ]

```

FileSystemWatcher

Si quisiéramos monitorear eventos en el sistema de archivos como por ejemplo la creación de archivos en un directorio dado, podemos utilizar el objeto FileSystemWatcher. A continuación se presenta un script que ante la creación de un archivo o subdirectorio en el directorio "pepe" contenido en la misma ubicación donde se ejecuta el script, mostrará por pantalla el nombre y la ruta absoluta del elemento creado:

```
1 #El Objeto FileSystemWatcher monitorea por cambios en el contenido de un directorio específico
2 $monitor=New-Object System.IO.FileSystemWatcher #Creamos un objeto del tipo FSWatcher
3 $monitor.path = "$PWD\pepe" #Especificamos el directorio a monitorear
4 $monitor.filter = "*.*)" #Especificamos que nos interesa cualquier tipo de archivo
5
6 #Especificamos la acción que queremos que se ejecute cuando se produzca la creación de un archivo
7 #en el directorio
8 $accion={
9     $nombre = $Event.SourceEventArgs.Name #Nombre del archivo creado
10    $path_absoluto = $EventArgs.FullPath #Ruta completa del archivo creado
11    write-host $nombre $path_absoluto
12 }
13
14 #Por ultimo debemos asociar el FSWatcher a un evento que es quien dispara la acción
15 Register-ObjectEvent $monitor Created -SourceIdentifier CreatedFile -Action $accion
```

Básicamente tenemos tres elementos involucrados:

1. El objeto FileSystemWatcher, que monitorea el directorio y ante la creación de un archivo, emite una señal.
2. La acción que se realizará al detectarse la creación del archivo.
3. Un objeto evento que conectará los dos anteriores, es decir al recibir la señal del FileSystemWatcher disparará la acción.

Al ejecutar el script, el Register-ObjectEvent muestra los datos del Evento creado:

```
PS C:\PowerShell> mkdir pepe

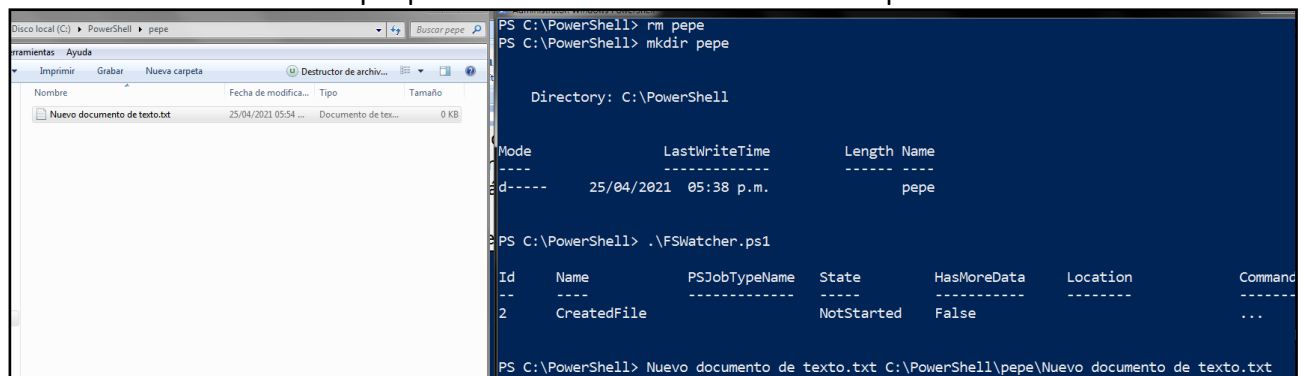
Directory: C:\PowerShell

Mode                LastWriteTime         Length Name
----                -
d-----          25/04/2021   05:38 p.m.         pepe

PS C:\PowerShell> .\FSWatcher.ps1

Id      Name      PSJobTypeName    State      HasMoreData   Location      Command
--      -
2      CreatedFile                NotStarted   False                ...
```

Al crear un archivo vemos que por consola se muestran los datos especificados en la acción.



```
PS C:\PowerShell> rm pepe
PS C:\PowerShell> mkdir pepe

Directory: C:\PowerShell

Mode                LastWriteTime         Length Name
----                -
d-----          25/04/2021   05:38 p.m.         pepe

PS C:\PowerShell> .\FSWatcher.ps1

Id      Name      PSJobTypeName    State      HasMoreData   Location      Command
--      -
2      CreatedFile                NotStarted   False                ...

PS C:\PowerShell> Nuevo documento de texto.txt C:\PowerShell\pepe\Nuevo documento de texto.txt
```

Para eliminar el evento debemos hacerlo con el comando Unregister-Event y para ver los Eventos activos lo vemos con el comando Get-EventSubscriber.

```
Administrator: Windows PowerShell
PS C:\PowerShell> Unregister-Event 2
PS C:\PowerShell> Get-EventSubscriber
PS C:\PowerShell> .\FSWatcher.ps1

Id      Name      PSJobTypeName  State      HasMoreData  Location  Command
--      -
3      CreatedFile  -----      NotStarted  False        -----
...

PS C:\PowerShell> Get-EventSubscriber

SubscriptionId : 3
SourceObject   : System.IO.FileSystemWatcher
EventName      : Created
SourceIdentifier : CreatedFile
Action         : System.Management.Automation.PSEventJob
HandlerDelegate :
SupportEvent   : False
ForwardEvent    : False
```

Timer

El objeto **timer** debe asociarse a un **evento**, el evento tendrá una **acción** asociada, cuando se dispara el timer, se realiza la acción a través del evento.

TIMER

- **Creación**

```
PS A:\powershell> $timer=New-Object System.Timers.Timer
PS A:\powershell> $timer

AutoReset      : True
Enabled        : False
Interval       : 100
Site           :
SynchronizingObject :
Container      :
```

- **Configuración:**

Interval: Cada cuanto se va a disparar el timer (ms), por defecto está en 100 ms.

AutoReset: indica si una vez que se disparó el timer se debe empezar de nuevo con otra cuenta regresiva, por defecto está en True.

Enabled: se puede controlar con \$timer.start() y \$timer.stop() para habilitar o deshabilitar el timer, también se puede asignar directamente así:

```
PS A:\powershell> $timer.Interval=10000
PS A:\powershell> $timer.Enabled=$true
PS A:\powershell>
```

ACCIÓN

- **Creación**

A continuación crearemos un pequeño script que será la acción que ejecute el evento cuando se dispare el timer.

```
PS A:\powershell> $accion={write-host "HOLA MUNDO"}
PS A:\powershell>
```

EVENTO

- **Registro**

A continuación debemos asociar timer y acción a un evento

```
PS A:\powershell> Register-ObjectEvent $timer Elapsed -SourceIdentifier Reloj -Action $accion
```

Id	Name	PSJobTypeName	State	HasMoreData	Location	Command
2	Reloj		NotStarted	False		write-host "HOLA MUNDO"

En el comando, **Elapsed** es el evento del timer que voy a esperar que ocurra (Transcurrido) es decir cuando se agote el tiempo establecido en **Interval**, en pocas palabras cuando se dispare.

SourceIdentifier es un identificador que le pondremos al evento para poder manejarlo rápidamente, podría no ponerse nada, en este caso yo lo llamé Reloj.

Finalmente, si el timer fue puesto en enabled empezará solo, sino para poner todo en marcha hay que iniciar el timer:

```
PS A:\powershell> $timer.Start()
PS A:\powershell> HOLA MUNDO
HOLA MUNDO
HOLA MUNDO
HOLA MUNDO
HOLA MUNDO
HOLA MUNDO
$timer.stop()
PS A:\powershell>
```