Scripts con PowerShell

Introducción

Un script es un archivo en texto plano que contiene órdenes para automatizar tareas.

Para que sea tratado como script, el archivo de texto necesita la extensión .ps1

Ejemplos:

Comprobar si un servicio está detenido y en ese caso, lanzarlo de nuevo.

Hacer copias de seguridad de una serie de archivos todos los viernes a una hora de la tarde

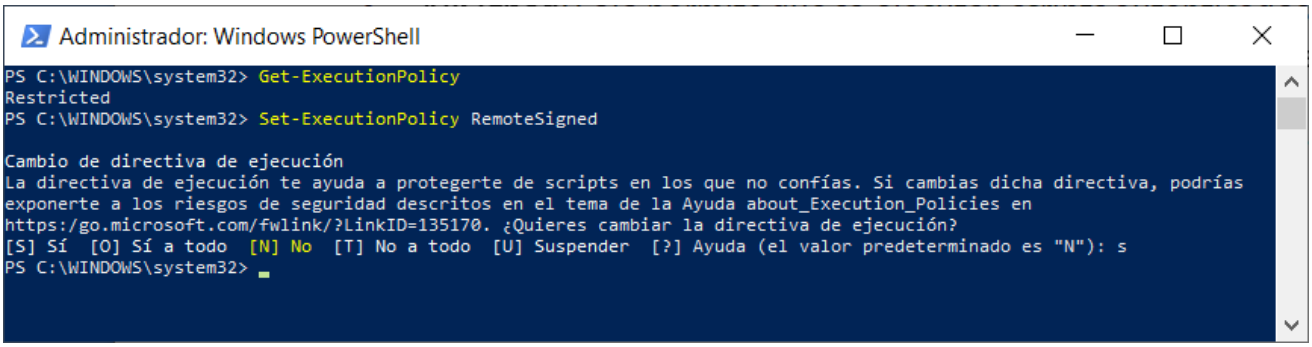
Crear usuarios de forma masiva

Seguridad

Hay 4 niveles de seguridad

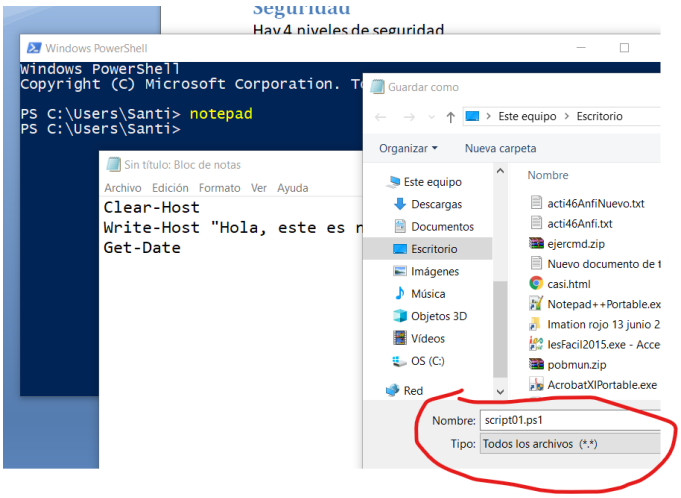
- **Rectricted**: No se permite la ejecución de scripts (por defecto)
- **AllSigned**: Sólo permite que se ejecuten scripts autenticados
- **RemoteSigned**: Sólo deben estar autenticados los scripts que proceden de una ubicación remota. Nuestros scripts sí pueden ejecutarse.
- **Unrestricted**: se puede ejecutar cualquier script sin importar su origen. La opción menos segura.

Ver y cambiar la seguridad:
En una consola de PS en modo administrador:



Primer script:

Desde powershell ejecutamos el comando notepad, escribimos el script, y lo guardamos con la extensión ps1



Ojo!: Hay que cambiar el tipo de archivo de .txt a todos (*.*) para poder poner la extensión ps1

Ojo!: En el caso de la imagen e arriba vemos que el script queda guardado en el escritorio., y para ejecutarlo deberíamos buscarlo en esa ubicación.

También podríamos haber hecho:

PS> notepad script01.ps1

Con lo que nos facilita todo, el script se guarda con extensión ps1 y en la ubicación desde donde hemos puesto el comando.

Para ejecutar el script solo hay que escribir su nombre:

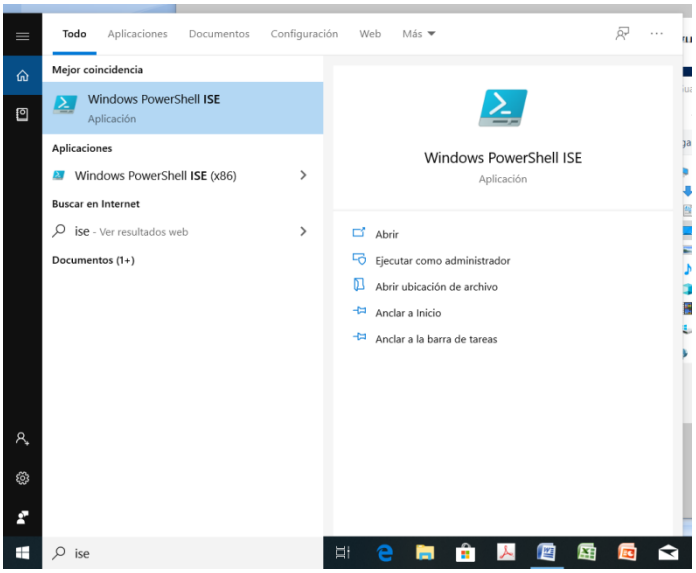
PS > .\scrip01.ps1

Entorno de script integrado (ISE)

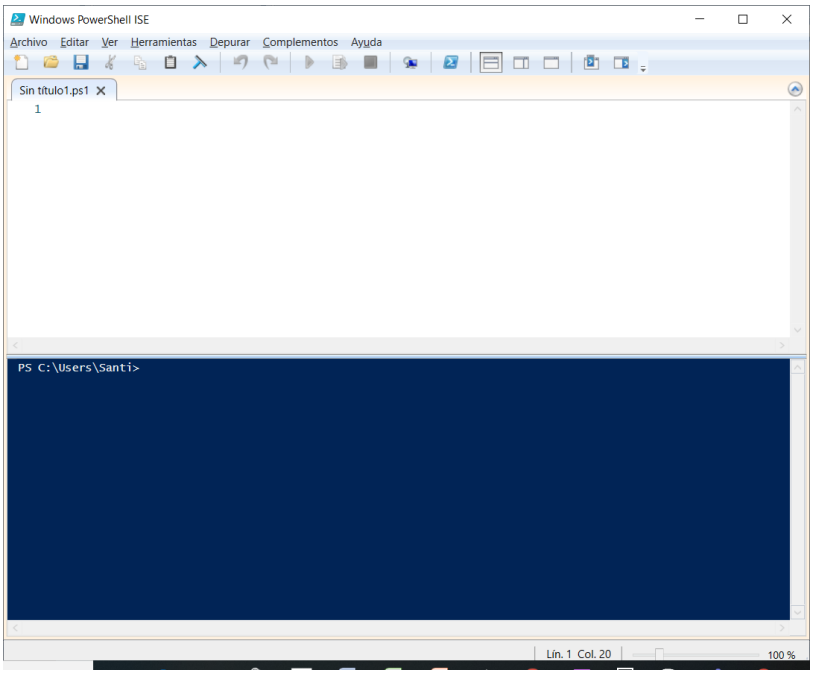
ISE: Integrated Scripting Environment.

Este entorno integra las tareas relativas a la escritura, depuración y ejecución de scripts.

En “icono de Windows—buscar” escribimos “ise” y encontramos el Windows PowerShell ISE

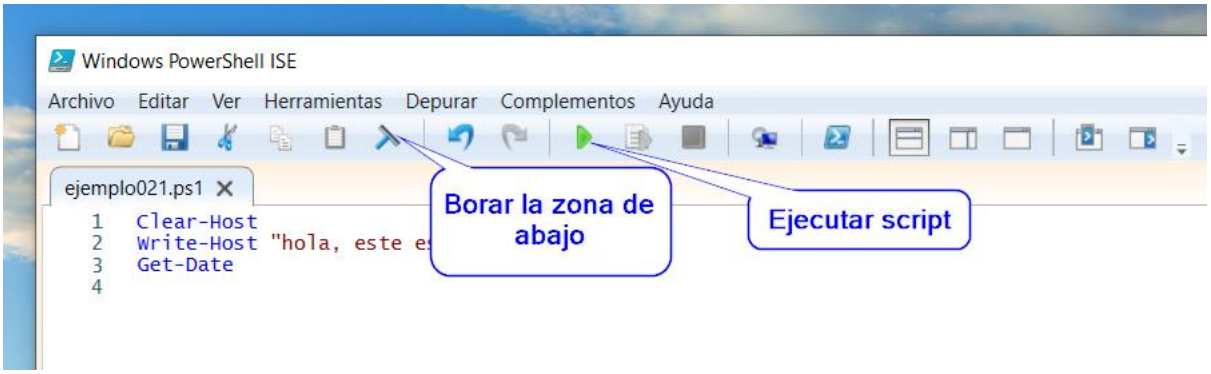


Una vez en ejecución vemos 2 partes o zonas.



En la zona de arriba (la blanca) es donde escribiremos el script.

En la de abajo es donde veremos la salida que se muestra al ejecutarse los scripts, y también podemos escribir comandos.



Primero se guarda y luego se ejecuta.

Si selecciona una línea, puedo ejecutar esa línea, pulsando el botón “Ejecutar selección” que está a la derecha del de “ejecutar script”.

Puntos de interrupción o de ruptura:

Podemos establecer uno o más puntos de interrupción.

F9→ pone o quita pto interrupción

F11 ejecuta paso a paso

Comentarios

Son muy importantes.

Comienzan por # → para comentar una línea

<#

Comentar un bloque

#>

Variables

Una variable es un espacio de memoria que contiene un valor que puede cambiar.

No es obligatorio declararla ni inicializarla.

Basta con asignar un valor con el signo igual (=) a una variable y PowerShell la crea y le asigna un tipo.

El primer carácter debe ser siempre el “símbolo dólar” ($)

```
# ejemplo de variables
$edad=23
$nombre="Federico"
$edad
$nombre
```

```
PS C:\Users\Santi\Desktop\scripts> C:\Users\Santi\Desktop\scripts\variable.ps1
23
Federico

PS C:\Users\Santi\Desktop\scripts> $nombre.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     String                                   System.Object


PS C:\Users\Santi\Desktop\scripts> $edad.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Int32                                    System.ValueType


PS C:\Users\Santi\Desktop\scripts>
```

Constantes

Son espacios de memoria que no cambian.

```
# ejemplo de constantes
New-Variable -Name c -Value 300000 -Option Constant
$c=45
$c
```

```
PS C:\Users\Santi\Desktop\scripts> C:\Users\Santi\Desktop\scripts\variable.ps1
New-Variable : Ya existe una variable con el nombre 'c'.
En C:\Users\Santi\Desktop\scripts\variable.ps1: 2 Carácter: 1
+ New-Variable -Name c -Value 300000 -Option Constant
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceExists: (c:String) [New-Variable], SessionStateException
    + FullyQualifiedErrorId :
VariableAlreadyExists,Microsoft.PowerShell.Commands.NewVariableComm
   and

No se puede sobrescribir la variable c porque es de solo lectura o constante.
En C:\Users\Santi\Desktop\scripts\variable.ps1: 3 Carácter: 1
+ $c=45
+ ~~~~~
    + CategoryInfo          : WriteError: (c:String) [],
SessionStateUnauthorizedAccessException
    + FullyQualifiedErrorId : VariableNotWritable

300000

PS C:\Users\Santi\Desktop\scripts>
```

Si pretendemos durante un script que una constante cambie su valor, dará error.

Tipos de datos

Los tipos básicos son

Tipo de datos	Descripción
Int	Entero con signo
Double	Múmero decimal
Char	Un solo carácter
String	Cadena de texto
Boolean	Valor lógico (True o False)

Hay más tipos de datos, pero estos son lo básicos.

Operaciones básicas con variables

Read-Host → Guarda en una variable lo que escriba el usuario, pero como texto ¡¡Ojo!!

Write-Host→ Muestra en pantalla un texto o el contenido de una variable

```
# ejemplo de tipo de datos
Clear-Host
$nombre= Read-Host "Cómo de llamas?"
$edad=Read-Host "¿Qué edad tienes?"
$edad=$edad+2
Write-Host "Hola, $nombre, tienes $edad años"
```

```
Cómo de llamas?: Federico
¿Qué edad tienes?: 23
Hola, Federico, tienes 232 años

PS C:\Users\Santi\Desktop\scripts>
```

Observa que no sumó 2 años a la edad, añadió el texto 2 a la edad.

Vamos a forzar a que la variable edad sea del tipo int. Poniendo `[int]$edad`

```
# ejemplo de tipo de datos
Clear-Host
$nombre= Read-Host "Cómo te llamas?"
[int]$edad=Read-Host "¿Qué edad tienes?"
$edad=$edad+2
Write-Host "Hola, $nombre, tienes $edad años"
```

```
Cómo te llamas?: Peláez
¿Qué edad tienes?: 23
Hola, Peláez, tienes 25 años

PS C:\Users\Santi\Desktop\scripts>
```

Ahora vemos como sí suma 2 a la edad.

También podríamos haber usado la expresión PS> $edad.GetType()

Operadores

Existen numerosos operadores: de cadenas, aritméticos, lógicos…

Veremos algunos

Operadores de cadena

Concatenación: permite unir dos o más variables de texto. (string)

```
# ejemplo de concatenación
Clear-Host
$nombre="Federico"
$apellido="Peláez"
$completo=$nombre+" "+$apellido
Write-Host "Te llamas $completo"
```

Operadores numéricos o aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto de la división

```
# ejemplo de operaciones aritméticas
Clear-Host
$x=9
$y=4
$suma=$x+$y
$resto=$x%$y
Write-Host "La suma es $suma"
Write-Host "El resto es $resto"
```

```
La suma es 13
El resto es 1
PS C:\Users\Santi\Desktop\scripts>
```

Operaciones booleanas

Los operadores lógicos se utilizan mucho en las estructuras condicionales

Operador	Descripción
-eq	Igual a
-ne	Distinto a (not equal)
-lt	Menor que
-le	Menor o igual que
-gt	Mayor que
-ge	Mayor o igual que

```
# ejemplo de operaciones lógicas
Clear-Host
$x=9
$y=4
$iguales = $x -eq $y
Write-Host "Los números $x, $y son iguales? $iguales"
```

```
Los números 9, 4 son iguales? False
PS C:\Users\Santi\Desktop\scripts>
```

```
# ejemplo de operaciones lógicas
Clear-Host
$x=7
$y=7
$iguales = $x -eq $y
Write-Host "Los números $x, $y son iguales? $iguales"
```

```
Los números 7, 7 son iguales? True
PS C:\Users\Santi\Desktop\scripts>
```

```
# ejemplo de operaciones lógicas
# ejemplo con números decimales
Clear-Host

[double]$y=0
[double]$x=Read-Host "Valor de x:"
$y=Read-Host "Valor de y:"
$iguales = $x -eq $y
Write-Host "Los números $x, $y son iguales? $iguales"
$mayor= $x -gt $y
iWrite-Host "$x es mayor que $y ? $mayor"
```

```
Valor de x:: 2.4
Valor de y:: 2.1
Los números 2.4, 2.1 son iguales? False
2.4 es mayor que 2.1 ? True

PS C:\Users\Santi>
```

Estructuras de control (condicionales y repetitivas)

Como todos los lenguajes de script, PowerShell también tiene **estructuras de control** que permiten modificar el flujo de ejecución de un programa.

Snipptes: Son estructuras de código listas para ser utilizadas. Se activan con Ctrl+J

Estructuras condicionales: ejecutan un grupo de sentencias en función del valor de una condición.

- if
- if-else
- switch

```
# ejemplo para comprobar la conexión con el servidor
# Clear-Host
Write-Host "Conectividad"
$ip=Read-Host "Introduce la Ip del servidor"
if(Test-Connection $ip -Count 1 -Quiet){
    Write-Host "$ip conexión establecida"
}
else{
    Write-Host "Error de conexión con $ip"
}
# conunt 1 es para que haga un solo ping
# -quite es para que no nos ofrezca demasiada conexión
```

```
Conectividad
Introduce la Ip del servidor: 2.2.2.2
Error de c onexión con 2.2.2.2

PS C:\Users\Santi\Desktop\scripts> C:\Users\Santi\Desktop\scripts\variable.ps1
Conectividad
Introduce la Ip del servidor: 127.0.0.1
127.0.0.1 conexión establecida

PS C:\Users\Santi\Desktop\scripts>
```

Estructuras repetitivas: nos permiten repetir un bloque de instrucciones.

- while
- do-while
- for
- foreach

```
# ejemplo para comprobar la conexión con el servidor
#Clear-Host
Write-Host "Conectividad"
# servidores.txt es un archivo que contiene una lista de ip's de servidores
$datos=Get-Content .\servidores.txt #indicando la ruta del archivo

foreach ($item in $datos)
{
    $respuesta=Test-Connection $item -Count 1 -Quiet
    if ($respuesta -eq "true"){
        Write-Host "$item Conexión establecida"
    }
    else
    {
        Write-Host "Error de conexión con $item"
    }
}
# count 1 es para que haga un solo ping
# -quite es para que no nos ofrezca demasiada conexión
```

```
PS C:\Users\Santi\Desktop\scripts> Get-Content .\servidores.txt
192.168.1.1
192.168.1.2
192.168.1.34
8.8.8.8
2.12.22.22

PS C:\Users\Santi\Desktop\scripts> C:\Users\Santi\Desktop\scripts\variable.ps1
Conectividad
192.168.1.1 Conexión establecida
Error de conexión con 192.168.1.2
192.168.1.34 Conexión establecida
8.8.8.8 Conexión establecida
Error de conexión con 2.12.22.22

PS C:\Users\Santi\Desktop\scripts>
```

Funciones

Definición:

Una función es un conjunto de instrucciones que tiene un nombre.

Una función siempre devuelve un dato y solo uno.

Una función puede ser invocada con datos de entrada.

Por tanto, una función es un conjunto de instrucciones que devuelven un único valor para un número indeterminado de datos de entrada.

Una función tiene que ser declarada antes de ser usada.

```
# ejemplo para comprobar la conexión con el servidor

#Definición de funciones
Function conecta ($fichero){
    Write-Host "Comprobando conexiones"
    foreach ($item in $fichero){
      $respuesta=Test-Connection $item -Count 1 -Quiet
      if ($respuesta -eq "true"){
         Write-Host "$item Conexión establecida"
      }
      else {
```

```
            Write-Host "Error de conexión con $item"
        }
    }
}

#inicio del script
#Clear-Host

$datos1=Get-Content .\servidores.txt
Conecta ($datos1)  #así llamamos a la función

$datos2=Get-Content .\clientes.txt
Conecta ($datos2)  #así llamamos a la función
```

```
PS C:\Users\Santi\Desktop\scripts> Get-Content .\servidores.txt
192.168.1.1
192.168.1.2
192.168.1.34
8.8.8.8
2.12.22.22

PS C:\Users\Santi\Desktop\scripts> Get-Content .\clientes.txt
127.0.0.1
192.168.1.2
192.168.1.34

PS C:\Users\Santi\Desktop\scripts> C:\Users\Santi\Desktop\scripts\variable.ps1
Comprobando conexiones
192.168.1.1 Conexión establecida
Error de conexión con 192.168.1.2
192.168.1.34 Conexión establecida
8.8.8.8 Conexión establecida
Error de conexión con 2.12.22.22
Comprobando conexiones
127.0.0.1 Conexión establecida
Error de conexión con 192.168.1.2
192.168.1.34 Conexión establecida

PS C:\Users\Santi\Desktop\scripts>
```