SCRIPTS EN POWERSHELL

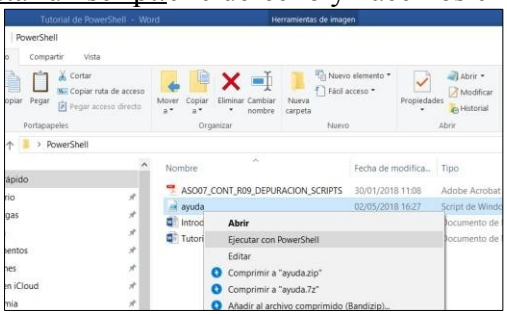
1. INTRODUCCIÓN

Un script contiene un conjunto de comandos (**cmdlets**) y elementos de programación: (bucles, condicionales, comparaciones, etc.).

Los scripts de PowerShell se almacenan en archivos **.ps1**

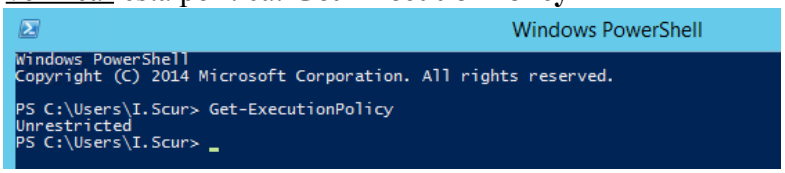
Para evitar daños accidentales al sistema no se puede ejecutar una secuencia de comandos simplemente haciendo doble clic en un archivo.

Para ejecutar un script:clic derecho y hacemos clic en "Ejecutar con PowerShell":



Hay una POLÍTICA que restringe la ejecución de scripts.

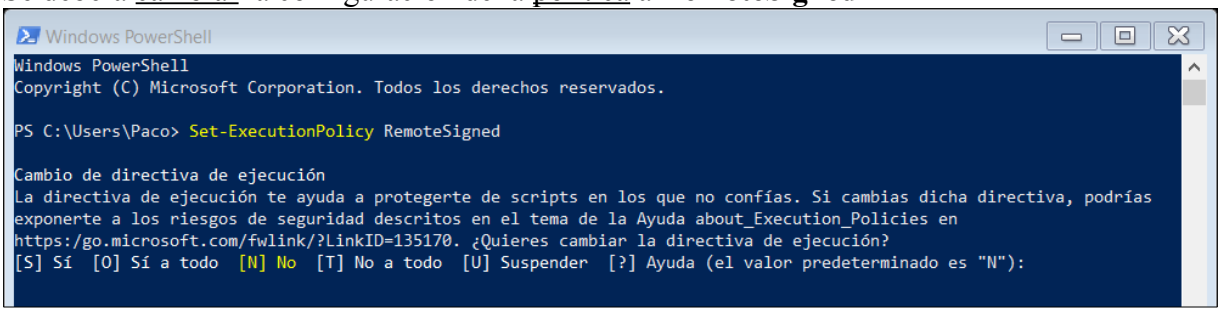
Para verificar esta política: **Get-ExecutionPolicy**



Obtendrá uno de los siguientes valores:

- **Restricted**: no se permiten scripts. Esta es la configuración predeterminada.
- **Unrestricted**: si se permiten scripts.
- **AllSigned**: se permiten scripts firmados por un desarrollador.
- **RemoteSigned**: se permiten scripts propios y scripts firmados por un desarrollador.

Se deberá cambiar la configuración de la política a **RemoteSigned**



2. VARIABLES Y PARÁMETROS

Todos los nombres de variables **empiezan** con el signo **$**

VARIABLES PROPIAS de PowerShell:

Variable	Contiene
$?	**Estado** del último comando ejecutado. **True** si fue exitoso, **False** en caso de error
$Errors	Array con los **últimos errores,** siendo $Error[0] el más reciente
$False	Constante **False**
$True	Constante **True**
$Null	Constante **Null**
$PSVersionTable	**Versión** de PowerShell
$Pwd	**Path** actual

TIPOS: NO ES NECESARIO DECLARAR EL TIPO DE DATO

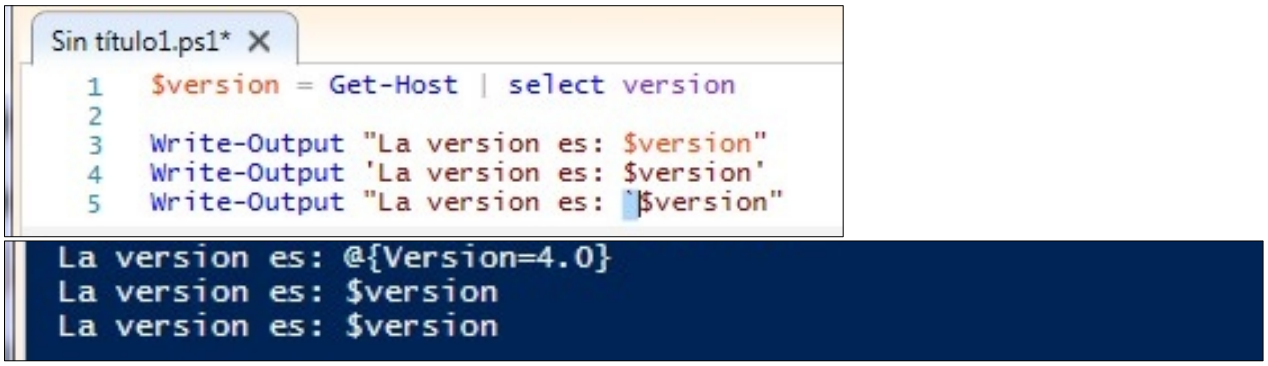
Pero se puede hacer si se quiere, indicando el tipo antes y entre []

[Int]$num=5

Para ciertas operaciones es necesario especificar los tipos de los números.

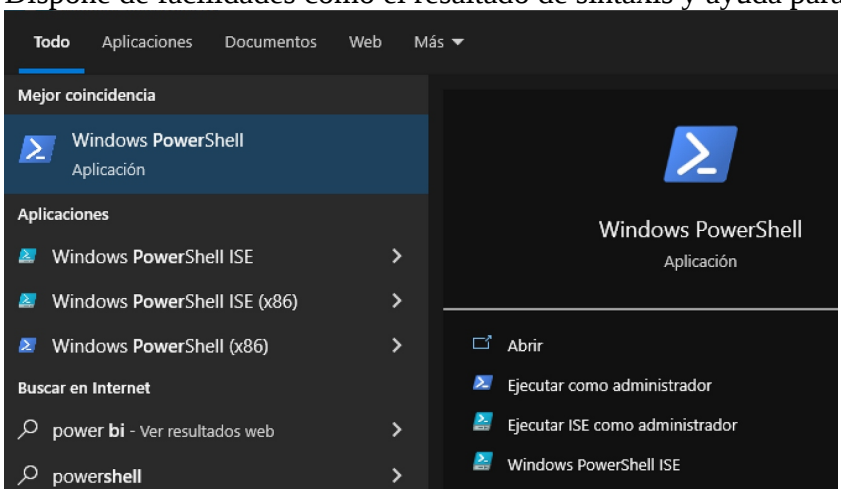
3. USO DE COMILLAS

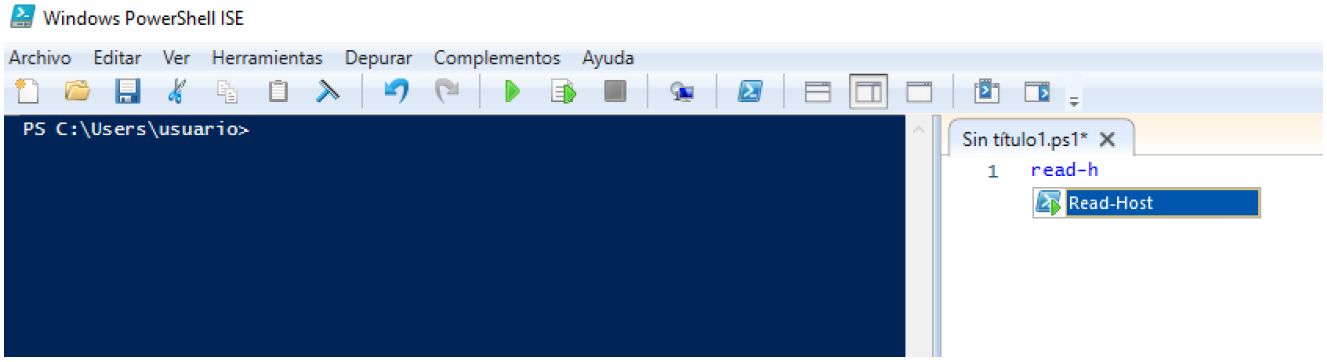
- Las comillas **dobles (“)**, también denominadas comillas débiles, permiten utilizar texto e interpretar variables al mismo tiempo.
- Las comillas **simples (‘)**, también denominadas comillas fuertes, generan el texto, lo que evita que se interpreten las variables.
- El acento **grave (`)**, es el carácter de escape para evitar la interpretación de los caracteres especiales, como por ejemplo el símbolo $.

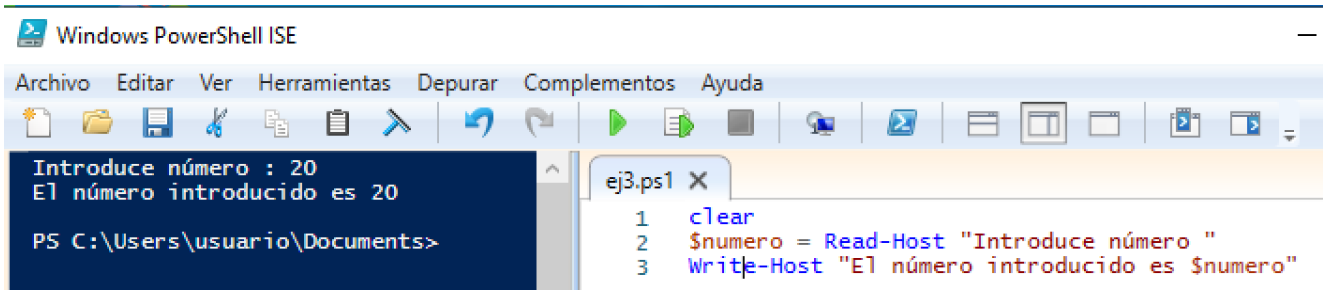


4. POWERSHEL ISE

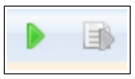
Dispone de facilidades como el resaltado de sintaxis y ayuda para la introducción de órdenes:







4.1. EJECUTAR SCRIPT

Con Botón 

En línea de comandos **.\nombre_fichero**

5. ENTRADA Y SALIDA DE DATOS
ENTRADA DE DATOS

Read-Host

```
$num= Read-Host 'Ingrese un número' #con mensaje y almacenamiento en variable
```

las “” y ‘’ se pueden omitir

SALIDA DE DATOS Y MENSAJES

Write-Output
Write-Host
Write se puede omitir

```
PS C:\Users\usuario\Documents\scripts>
Ingrese un número: 20
Con write-host: 20
Con write-output 20
Con write 20
Write
sin
comillas
20
Sin write 20
```

ej6.ps1

```
$num= Read-Host Ingrese un número #co
write-host Con write-host: $num
Write-Output "Con write-output $num"
Write "Con write $num"
write Write sin comillas $num
"Sin write $num"
```

6. OPERADORES

- **Aritméticos**: +, -, *, /, %
- **Asignación**; =, +=, -=, *=, /=, %=
- **Comparación**: -eq, -ne, -gt, -lt, -le, -ge
- **Lógicos**: -and, -or, -xor, -not, !
- **De patrones:** -like, - nolike
- **De tipo:** -is, -isnot, -as
- **De redirección**: >, >>
- **De coincidencia:**-match, -nomatch
- **De sustitución** :-replace
- **De división y combinación**: -split, -join

Operador	Significado	Ejemplo (devuelve el valor True)
-eq	Es igual a	1 -eq 1
-ne	Es distinto de	1 -ne 2
-lt	Es menor que	1 -lt 2
-le	Es menor o igual que	1 -le 2
-gt	Es mayor que	2 -gt 1
-ge	Es mayor o igual que	2 -ge 1
-like	Es como (para cadenas de caract)	"file.doc" -like "f*.do?"
-notlike	No es como	"file.doc" -notlike "p*.doc"
-contains	Contiene	1,2,3 -contains 1
-notcontains	No contiene	1,2,3 -notcontains 4

7. ESTRUCTURAS DE CONTROL

 # Condiciones

```
if ($variable -eq 'valor')
{ ... }
elseif ($variable -lt 'valor')
{ ... }
else
{ ... }
```

 # Ciclo for

```
for ($i = 0; $i -lt 10; $i++)
{ ... }
```

 # Ciclo while

```
while ($variable -lt 5)
{ ... }
```

Switch

```
switch (variable){
  valor1{ comandos  }
  valor2{ comandos  }
}
```

 # Ciclo foreach

```
foreach ($item in $array)
{ ... }
```

 # Ciclo do while

```
do
{ ... }
while ($variable -lt 10)
```

8. PARÁMETROS

Se puede indicar nombre a los parámetros.

Hay que indicarlos al principio del scripts con param ($par1,$par2)

EJECUTAR SCRIPT CON PARÁMETROS

.\nombre_fichero lista de parámetros separados por espacio

```
PS C:\Users\usuario\Documents\scripts> .\ej5.ps1 10 20 30
primer parámetro     10
segundo parámetro    20
tercer parámetro     30

PS C:\Users\usuario\Documents\scripts>
```

ej5.ps1

```
param ($1,$2, $3) #debe ir antes de cualquier código
write-host "primer parámetro`t" $1
write-host "segundo parámetro`t" $2
write-host "tercer parámetro`t" $3
```

9. COMENTARIOS

```
<#
comentario varias líneas
sigue el comentario
continúa el comentario
#>
param ($1,$2, $3) #debe ir antes de cualquier código
write-host "primer parámetro`t" $1 # `t es una tabulación
write-host "segundo parámetro`t" $2
write-host "tercer parámetro`t" $3
```

10. VECTORES

```
Primer elemento: 200
Todos los elementos: 200 250 300 350 400
Obtención del doble:
200 * 2 = 400
250 * 2 = 500
300 * 2 = 600
350 * 2 = 700
400 * 2 = 800

PS C:\Users\usuario\Documents\scripts>
```

ej7.ps1

```
clear
#declaración de vector
$lista = 200,250,300,350,400
#acceso a un elemento
Write-Host 'Primer elemento:'$lista[0]
#visualización de todos los elementos
write-host "Todos los elementos:"$lista
#recorrido de elementos
write-host "Obtención del doble:"
foreach ($i in $lista){
    write-host "$i * 2 ="($i*2)
}
```

11. FUNCIONES

Se declaran y luego se llaman.

```
Introduzca su nombre: María
Tu nombre es: María

PS C:\Users\usuario\Documents>
```

ej3.ps1

```
function Get-Nombre{
    clear
    #solicitud de nombre
    $Name = $(
        Write-Host 'Introduzca su nombre: '-ForegroundColor Yellow -NoNewLine
        Read-Host
    )
    #visualización de nombre
        Write-host "Tu nombre es: $Name"
}
#Llamada a la función
Get-Nombre
```

FUNCIÓN CON PARÁMETROS:

Función altura:

```
Introduce valor: 20
la altura de 20 = 1962

PS C:\Users\usuario>
```

ej9.ps1

```
$g=9.81
#declaración de función
function altura{
    param($t)
    if($t -eq 0){
        return 0
    }
    else{
        return ($g*[System.Math]::Pow($t,2))/2
    }
}
clear
$valor=read-host "Introduce valor"
write-host -nonewline "la altura de $valor = "
#llamada a función
altura($valor)
```

Función recursiva: factorial de un número

```
Ingrese un número: 5
5! = 120

PS C:\Users\usuario>
```

ej8.ps1

```
function Get-Factorial(){
    param ([int]$num)

    if($num -lt 0)    {
        $result = 0
    }
    elseif ($num -le 1)    {
        $result = 1
    }
    else    {
        $result = $num * (Get-Factorial ($num-1))
    }
    return $result
}
clear
$value = Read-Host "Ingrese un número"
$result = Get-Factorial $value
Write-Output "$value! = $result"
```

El uso de funciones, nos permite usarlas en la línea de comandos:

Indicar el nombre de la función y los parámetros que recibe:

```
PS C:\Users\usuario> Get-Factorial -num 5
120
```