Scripts con PowerShell

Los scripts en PowerShell son una agrupación de comandos a los que le podemos añadir una estructura de programación concreta para poder automatizar ciertas tareas del sistema.

De esta manera podemos escribir comandos uno detrás de otro y a su vez aplicar ciertas estructuras de control como **condicionales, bucles o funciones**.

Seguridad

Para poder ejecutar un script, este debe ser un fichero de texto con extensión "ps1". En el sistema debemos indicar el nivel de seguridad que queremos aplicar a la ejecución de scripts:

- Restricted
- AllSigned
- RemoteSigned
- Unrestricted

Para conocer cual es la política de seguridad asignada a nuestro equipo tenemos el cmdlet:

```
Get-ExecutionPolicy
```

Y para asignar una nueva política de seguridad:

```
Set-ExecutionPolicy
```

PowerShell ISE

Podemos utilizar el PowerShell ISE para editar, depurar y ejecutar los scripts. Nos ofrece un entorno práctico en el que podemos incluso añadir puntos de ruptura para evaluar los scripts paso a paso.

Variables

A lo largo de nuestro script podemos utilizar variables para almacenar valores o resultados de los comandos utilizados. Estas variables irán precedidas del signo '$'

```
$edad=44

$nombre="Pepe Pérez"

$edad

$Nombre

$edad.GetType()     Nos dice el tipo

$Nombre.GetType()
```

Para forzar un tipo concreto de dato se puede utilizar:

```
[int]$edad=….
```

Para leer un valor por teclado:

```
$valor=Read-Host "texto"
```

Constantes

Podemos crear constantes con un nombre y valor determinado:

```
New-Variable -Name PI -Value 3.1415 -option Constant
```

Estas constantes mantendrán su valor durante todo el script y no se podrán modificar. Esto será útil para indicar valores con un identificador que se utilizarán a lo largo de todo el script.

Operadores

Se puede utilizar la concatenación de textos con el símbolo '+':

```
$texto3=$texto1+$texto2
```

También las operaciones aritméticas:

```
+ - * / %
```

Y las operaciones booleanas o relacionales:

```
-eq  igual

-ne  diferentes

-lt  menor

-le  menor o igual

-gt  mayor

-ge  mayor o igual
```

Podemos comprobar si un elemento está dentro de un conjunto de valores concreto con el operador -in:

```
$x='E'
$y=$x -in 'A','B','C'
$y
```

Si queremos utilizar el valor que nos ha devuelto un comando desde el cual hemos redireccionado el resultado con una tubería tendríamos que utilizar la variable $_ como en el siguiente ejemplo:

```
Get-Process | Where-Object {$_.ProcessName -eq 'dllhost'}
```

Donde El comando Get-Process nos devuelve una lista de procesos del sistema y nosotros filtramos cada proceso obteniendo sólo aquellos cuyo campo ProcessName es 'dllhost'

Estructuras de Control

Condicionales

usar Control+J y escribir if

```
if ($x -gt $y)
{

}


if ($x -lt $y)
{

}
else
{

}
```

Para ejecutar comandos según el valor de la misma variable podemos utilizar:

```
switch ($x)
{
	'A' {}
	'B' {}
	...
	Default {}
}
```

El siguiente ejemplo nos muestra si existe conexión con una IP determinada:

```
$ip="192.168.1.2"
if (Test-Connection $ip -count 1 -Quiet)
{
	Write-Host "Hay ping"
}
Else
{
	Write-Host "No hay ping"
}
```

Repetitivas:

Permiten realizar ejecuciones sucesivas de un determinado código:

```
while ($x -gt 0)
{

}

do
{

}while ($x -gt 0)

do
{

}until ($x -gt 0)


for ($i = 1; $i -lt 99; $i++)
{

}
```

```
foreach ($item in $collection)
{

}
```

Funciones

Las funciones son trozos de código a los que damos un identificador concreto y a los que podemos invocar en cualquier momento desde nuestro script

Hay que declarar las funciones antes de usarlas.

```
Function Nombre ($datos){



}
```

Ejemplo: Pasamos dos valores y la función muestra el resultado

```
Function Sumar ($x, $y) {
    $sumar = $x + $y
    Write-Host “La respuesta es $sumar”
}

Sumar 5 8
```

Ejemplo: Pasamos dos valores y devuelve el resultado al programa principal:

```
Function Sumar ($x, $y) {
    $sumar = $x + $y
    return $sumar
}

$v=Sumar 7 3
Write-Host $v
```