

## 5. Conceptos básicos de PowerShell

Monday, September 7, 2020 7:36 PM

### ? PowerShell es Orientado a Objetos

La primera cosa que debemos comprender sobre PowerShell es que TODO es un objeto.

Los conceptos básicos que debemos conocer para comenzar a trabajar con PowerShell son:

- **VARIABLES** - Lugares para almacenar valores
  - Todas variables comienzan con el símbolo: \$
  - Ejemplo: PS> \$MaximumHistoryCount
  - Esta variable integrada determina el número máximo de comandos que PowerShell guarda en su historial de comandos.
  - Las variables pueden ser variables definidas por el usuario o variables automáticas. Las primeras las creo el usuario y las segundas, ya existen en PowerShell.
  - Si intentas acceder a una variable que no existe recibirás un error de operación inválida.
  - Para crear una variable es necesario declararla y asignarle un valor (inicializarla).
  - Ejemplo: PS> \$color = 'blue'
  - Una vez creada, la variable ya puede ser referenciada sin problemas.
  - Para fijar el valor de una variable también podemos usar el comando:  
PS> Set-Variable -Name color -Value blue
  - Para obtener (o accesar) a su valor, podemos usar el comando:  
PS> Get-Variable -Name color
  - Las variables automáticas también pueden ser modificadas, sin embargo, el realizarlo puede traer consecuencias inesperadas. Se recomienda tratar todas las variables automáticas como *read-only*.
  - Un tipo especial de variable es \$null, ya que representa nada. Si asignamos este valor a una variable podemos crearla sin asignarle un valor real.
  - Otra variable automática importante es \$LASTEXITCODE, la cual recibe el valor de salida que se recibió de una ejecución de una aplicación externa. Ejemplo: al hacer ping a una IP, el valor de salida, si todo fue exitoso, debe ser 0, sino es que fallo.

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> ping.exe -n 1 google.com

Pinging google.com [2607:f8b0:4012:80a::200e] with 32 bytes of data:
Reply from 2607:f8b0:4012:80a::200e: time=31ms

Ping statistics for 2607:f8b0:4012:80a::200e:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 31ms, Maximum = 31ms, Average = 31ms
PS C:\WINDOWS\system32> $LASTEXITCODE
0
PS C:\WINDOWS\system32>
```

- **Tipos de datos** - PowerShell tiene múltiples tipos de datos como bools, strings y enteros.
  - Es posible cambiar el tipo de datos de las variables sin errores, ya que PowerShell deduce el tipo de dato basado en los valores que se le proveen.

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> $foo = 1
PS C:\WINDOWS\system32> $foo = 'one'
PS C:\WINDOWS\system32> $foo = $true
PS C:\WINDOWS\system32>
```

- Booleanos: \$true y \$false
- Enteros y punto flotante: los datos numéricos en PowerShell corresponden a estos tipos.
  - **Int** - Los datos enteros generalmente se manejan como int32, es decir, datos de 32 bits.
  - **Float y Double** - Para números decimales contamos con estas 2 opciones que varían en su precisión decimal. El valor por default es Double.

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> $num = 0.1234567890
PS C:\WINDOWS\system32> $num.GetType().name
Double
```

```
PS C:\WINDOWS\system32> $num + $num
0.246913578
PS C:\WINDOWS\system32> [Float]$num + [Float]$num
0.246913582086563
PS C:\WINDOWS\system32>
```

- String: Corresponden con cadenas de caracteres

- Considera que siempre debe ir su valor encerrado entre comillas simples, aunque también pueden usar comillas dobles.

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> $language = 'PowerShell'
PS C:\WINDOWS\system32> $color = 'blue'
PS C:\WINDOWS\system32> $sentence = "Today, you learned that $language loves the color $color"
PS C:\WINDOWS\system32> $sentence
Today, you learned that PowerShell loves the color blue
PS C:\WINDOWS\system32>
```

- **Objetos** - Un objeto es la instancia de una clase, la cual tiene métodos que definen su comportamiento y atributos, cuyo valor es particular de cada objeto, que lo describen. A lo largo de esta UA manearemos los objetos de la forma más práctica posible, evitando entrar en la medida de lo posible, en las complejidades de la POO.
- **Estructuras de datos** - Permiten estructurar múltiples datos bajo un único identificador.
  - En PowerShell pueden ser arreglos, ArrayLists y hashtables (tablas hash).
  - Arreglos:

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> $colorPicker = @('blue','white','yellow','black')
PS C:\WINDOWS\system32> $colorPicker[0]
blue
PS C:\WINDOWS\system32> $colorPicker[1]
white
PS C:\WINDOWS\system32> $colorPicker[1..3]
white
yellow
black
PS C:\WINDOWS\system32> $colorPicker[1] = 'gray'
PS C:\WINDOWS\system32> $colorPicker[1]
gray
PS C:\WINDOWS\system32> $colorPicker = $colorPicker + 'white'
PS C:\WINDOWS\system32> $colorPicker
blue
gray
yellow
black
white
PS C:\WINDOWS\system32> $colorPicker += 'orange'
PS C:\WINDOWS\system32> $colorPicker
blue
gray
yellow
black
white
orange
PS C:\WINDOWS\system32>
```

- ArrayList: Los arreglos en PowerShell son de tamaño fijo por lo que, cada que se agrega o se elimina un elemento, la realidad es que PowerShell crea un nuevo arreglo con los elementos finales. Esta característica puede ocasionar problemas de rendimiento en arreglos grandes. Para solucionar esto, PowerShell cuenta con los ArrayList que se comportan casi igual que un arreglo, pero sin tamaño fijo.

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> $colorPicker = [System.Collections.ArrayList]@('blue','white','yellow','black')
PS C:\WINDOWS\system32> $colorPicker
blue
white
yellow
black
PS C:\WINDOWS\system32> $colorPicker.Add('gray')
4
PS C:\WINDOWS\system32> $null = $colorPicker.Add('orange')
PS C:\WINDOWS\system32> $colorPicker
blue
```

```

blue
white
yellow
black
gray
orange
PS C:\WINDOWS\system32> $colorPicker.Remove('gray')
PS C:\WINDOWS\system32>

```

- Tablas Hash (hashtables): Es el equivalente a un diccionario en Python.

```

Administrator: Windows PowerShell
PS C:\WINDOWS\system32> $users = @{user1 = 'Pedro Picapiedra'; user2 = 'Pablo Marmol'; user3 = 'Dino'}
PS C:\WINDOWS\system32> $users

Name          Value
----          -----
user1        Pedro Picapiedra
user3        Dino
user2        Pablo Marmol

PS C:\WINDOWS\system32> $users['user1']
Pedro Picapiedra
PS C:\WINDOWS\system32> $users.Keys
user1
user3
user2
PS C:\WINDOWS\system32> $users.Values
Pedro Picapiedra
Dino
Pablo Marmol
PS C:\WINDOWS\system32> $users.Add('root', 'sudo')
PS C:\WINDOWS\system32> $users

Name          Value
----          -----
root          sudo
user1        Pedro Picapiedra
user3        Dino
user2        Pablo Marmol

```

```

PS C:\WINDOWS\system32> $users.ContainsKey('noEsta')
False
PS C:\WINDOWS\system32> $users['root'] = 'La Profe de PC'
PS C:\WINDOWS\system32> $users

Name          Value
----          -----
root          La Profe de PC
user1        Pedro Picapiedra
user3        Dino
user2        Pablo Marmol

PS C:\WINDOWS\system32> $users.Remove('root')
PS C:\WINDOWS\system32> $users

Name          Value
----          -----
user1        Pedro Picapiedra
user3        Dino
user2        Pablo Marmol

```

