Actividad 6 – Powershell

1. Crear una variable llamada prueba y asignarle un texto. Comprobar las acciones que podemos realizar y usar al menos tres de ellas.

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba = "hola"
```

Para comprobar sus acciones usamos get-member

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba | get-member

   TypeName: System.String

Name                MemberType          Definition
----                ----------          ----------
Clone               Method              System.Object Clone(), System.Object ICloneable.Clone()
CompareTo           Method              int CompareTo(System.Object value), int CompareTo(string s…
Contains            Method              bool Contains(string value), bool Contains(string value, S…
CopyTo              Method              void CopyTo(int sourceIndex, char[] destination, int desti…
EndsWith            Method              bool EndsWith(string value), bool EndsWith(string value, S…
EnumerateRunes      Method              System.Text.StringRuneEnumerator EnumerateRunes()
Equals              Method              bool Equals(System.Object obj), bool Equals(string value),…
GetEnumerator       Method              System.CharEnumerator GetEnumerator(), System.Collections.…
GetHashCode         Method              int GetHashCode(), int GetHashCode(System.StringComparison…
GetPinnableReference Method             System.Char&, System.Private.CoreLib, Version=8.0.0.0, Cul…
GetType             Method              type GetType()
GetTypeCode         Method              System.TypeCode GetTypeCode(), System.TypeCode IConvertibl…
IndexOf             Method              int IndexOf(char value), int IndexOf(char value, int start…
```

Usamos 3 de ellas:

Método replace():

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba.replace("h", "b")
bola
```

Método equals():

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba.equals("adios")
>> ;
False
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba.equals("hola")
>> ;
True
```

Método toUpper():

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba.toUpper()
HOLA
```

2. Crear una variable llamada prueba2 y asignarle un número. Comprobar las acciones que podemos realizar y usar al menos tres de ellas.

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba2 = 10
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba2 | get-member

   TypeName: System.Int32

Name                 MemberType Definition
----                 ---------- ----------
CompareTo            Method     int CompareTo(System.Object value), int CompareTo(int value), int ICo…
Equals               Method     bool Equals(System.Object obj), bool Equals(int obj), bool IEquatable…
GetByteCount         Method     int IBinaryInteger[int].GetByteCount()
GetHashCode          Method     int GetHashCode()
GetShortestBitLength Method     int IBinaryInteger[int].GetShortestBitLength()
GetType              Method     type GetType()
GetTypeCode          Method     System.TypeCode GetTypeCode(), System.TypeCode IConvertible.GetTypeCo…
ToBoolean            Method     bool IConvertible.ToBoolean(System.IFormatProvider provider)
ToByte               Method     byte IConvertible.ToByte(System.IFormatProvider provider)
ToChar               Method     char IConvertible.ToChar(System.IFormatProvider provider)
ToDateTime           Method     datetime IConvertible.ToDateTime(System.IFormatProvider provider)
ToDecimal            Method     decimal IConvertible.ToDecimal(System.IFormatProvider provider)
```

Métodos:

GetType().FullName && toString()

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> Write-Host "El tipo de dato de prueba2 es: $($prueba2.Get
Type().FullName)"
El tipo de dato de prueba2 es: System.Int32
PS /home/noe/Documentos/DAW1/ScriptsSistemas> write-host "El valor de prueba2 es" $($prueba2.toString()
)
El valor de prueba2 es 10
```

equals()

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba2.equals(2)
False
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba2.equals(10)
True
```

3. Comprobar el uso de las comillas dobles y simples junto con el cmdlet Write-Host

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> # Definir una variable
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $nombre = "Juan"
PS /home/noe/Documentos/DAW1/ScriptsSistemas>
PS /home/noe/Documentos/DAW1/ScriptsSistemas> # Usar comillas dobles con Write-Host
PS /home/noe/Documentos/DAW1/ScriptsSistemas> Write-Host "Hola, $nombre!"
Hola, Juan!
PS /home/noe/Documentos/DAW1/ScriptsSistemas>
PS /home/noe/Documentos/DAW1/ScriptsSistemas> # Usar comillas simples con Write-Host
PS /home/noe/Documentos/DAW1/ScriptsSistemas> Write-Host 'Hola, $nombre!'
Hola, $nombre!
```

EL uso de dobles comillas permite la expansión de variables además la interpretación de caracteres de escape, mientras que las comillas simples no realizan ninguna expansión ni interpretación y lo imprime literalmente, es decir, al usarlo con comillas dobles permite la salida por pantalla del dato de la variable almacenada, en cambio si usamos comillas simple solo devolverá el nombre de la variable sin el dato almacenada que hay en ella.

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> write-host '$prueba2'
$prueba2
PS /home/noe/Documentos/DAW1/ScriptsSistemas> write-host "$prueba2"
10
```

Visualmente si escribimos en la terminal el nombre de nuestra variable sin comillas dobles devolverá el mismo resultado

```
PS /home/noe/Documentos/DAW1/ScriptsSistemas> $prueba2
10
```