El ámbito de las variables

El ámbito de una variable determinará la visibilidad de una variable en PowerShell, este concepto de ámbito es muy importante, ya que garantiza la independencia de las variables, y evita así las probables interferencias de valores.

Supongamos por un instante que ejecutamos un script y que una vez el script ha terminado deseamos comprobar un valor escribiendo su nombre en la consola, ¡vaya! No es posible, la variable no está disponible, esto es debido sencillamente al ámbito de las variables.

PowerShell utiliza el concepto de ámbito Padre y ámbito Hijo. Un ámbito Hijo es un ámbito creado en el interior de un ámbito Padre. Es decir, cada vez que ejecutamos una función, un script, o un bloque de instrucciones, se crea un nuevo ámbito. Y, salvo que especifiquemos lo contrario, PowerShell define que una variable puede leerse en su ámbito, así como en los ámbitos hijos. Pero únicamente puede modificarse en el ámbito donde se creó. Además, los ámbitos padres no pueden ni leer, ni modificar las variables definidas en sus ámbitos hijos.

El esquema siguiente muestra la accesibilidad de una variable a través del ámbito hijo.

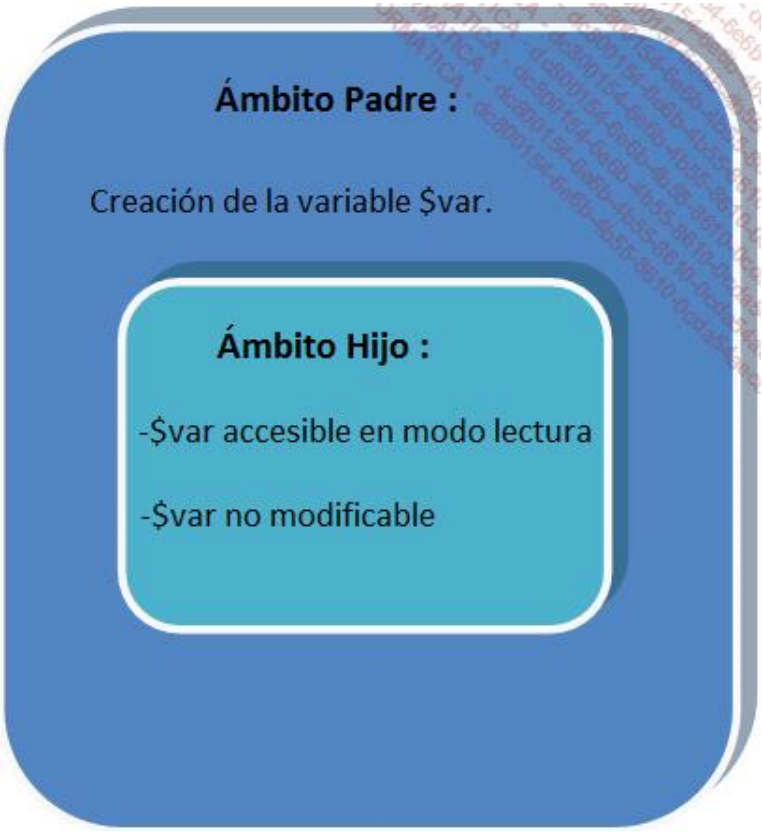

Ilustración de la accesibilidad de una variable

Admitamos que una variable `$valor` se ha inicializado a 0 en la consola. A continuación, creamos la función «leer» siguiente:

```
function Leer
{
   $valor
}
```

Hasta aquí todo va bien, podemos leer la variable `$valor` cada vez que llamamos a la función «leer». Nos encontramos en el caso de una variable creada en un ámbito padre, con acceso de lectura en un ámbito hijo.

Ahora, si creamos la función «añadir», que permite incrementar en 1 la variable `$valor` cada vez que se llama a la función:

```
function Añadir
```

```
{
   $valor++
}
```

Si ha seguido hasta aquí, sabrá que una vez la función termina, la variable no se incrementa y la variable `$valor` será siempre igual a 0. Simplemente porque no hemos especificado nada en los ámbitos de las variables, y que por consiguiente, no estamos autorizados a modificar este valor en un ámbito hijo.

```
PS > $valor = 0
PS > Añadir
PS > $valor
0
```

Para solucionar esto, será necesario adaptar el ámbito de las variables según tres tipos:

- el ámbito global,
- el ámbito local,
- el ámbito script.

El ámbito global (global):

El ámbito global se aplica al inicio de PowerShell, es decir que si al inicio inicializamos una variable, por defecto su alcance será global.

Las variables de ámbito global, serán accesibles en modo lectura en el ámbito hijo sin especificarlas.

Sin embargo, para poder modificar una variable de ámbito global desde un ámbito hijo, es obligatorio especificar una etiqueta « global:» antes del nombre de la variable.

Por ejemplo:

```
function Añadir
{
   $global:valor++
}
```

Así, al regresar al ámbito Padre, la variable `$valor` se ha modificado correctamente.

```
PS > $valor = 0
PS > Añadir
PS > $valor
1
```

El ámbito local (local):

El ámbito local es el ámbito en el que nos encontramos en un momento T (ámbito actual).

Un nuevo ámbito local se crea cada vez que ejecutamos una función, un script o un bloque de instrucciones.

El ámbito local responde a las normas de base, es decir, una variable creada en un ámbito padre no puede ser modificada en un ámbito hijo.

El ámbito script:

El ámbito script es, como su nombre indica, un ámbito creado durante el tiempo de ejecución del script, y deja de existir una vez termina el script. A semejanza de la consola, un script puede verse obligado a crear varios ámbitos hijos, ellos mismos susceptibles de crear variables. El ámbito script permite acceder a estas variables, pero en el exterior de la función. Como ejemplo, tomemos el script siguiente:

```
#script1.ps1
#Script acerca del tipo de ámbito "script"

Function Nuevo_valor
{
   $valor = 10
}

$valor = 0          #Inicialización de la variable
Write-Host "El valor de origen es: $valor"
Nuevo_valor         #Llamada de la función
Write-Host "El nuevo valor es: $valor"
```

En este script, inicializamos una variable a 0, luego llamamos una función que asignará el valor 10 a nuestra variable, y finalmente, mostramos la variable.

Si no se especifica ningún ámbito, al ejecutar el script, se puede ver que la variable no se ha modificado:

```
PS > ./Script1.ps1

Valor de origen: 0
Nuevo valor: 0
```

Si ahora, integramos la etiqueta «script:» delante del nombre de la variable en la función, la variable se identifica como parte del ámbito script:

```
#script1.ps1
#Script sobre el tipo de ámbito "script"

Function Nuevo_valor
{
   $script:valor = 10
}

$valor = 0                  #Inicialización de la variable
Write-Host "Valor de origen: $valor"
Nuevo_valor                 #Llamada de la función
Write-Host "Nuevo valor: $valor"
```

El resultado no será el mismo. Esta vez, es la variable creada en el ámbito script la que va a ser modificada.

```
PS > ./Script1.ps1

Valor de origen: 0
Nuevo valor: 10
```

También conviene destacar que podemos del mismo modo utilizar la etiqueta «private:» en la asignación de la variable en una función. Esto tendrá por efecto hacer que nuestra variable tome un valor solamente durante el período de vida de la función.