POWERSHELL LANGUAGE

Se denomina PSL a un fichero que contiene órdenes para ser ejecutadas por el PowerShell. Mediante el lenguaje de programación que proporciona PowerShell, podemos escribir scripts que realicen tareas más complejas. El nombre del fichero puede utilizarse posteriormente para ejecutar el conjunto de órdenes almacenado, como si fuera una única orden del shell.
La ejecución del script finaliza cuando termina la ejecución de todas las órdenes del guion o se produce un error sintáctico en el lenguaje de programación.

```
$s="Hola Mundo"
write-host $s
```

EJECUCIÓN DE UN SCRIPT

Podemos ejecutar un script de varias formas:

ScriptName

ScriptName.ps1

&ScriptName.ps1

Invoke-Expression ScriptName.ps1

powershell.exe ScriptName

COMENTARIOS

Las líneas que comienzan por el carácter # son comentarios. El shell no las interpreta. Debemos utilizar los comentarios para documentar nuestros scripts Si no se coloca esta línea, el script se interpretara con el shell activo.

LA ORDEN WRITE-OUTPUT

Muestra la cadena por la salida estándar.

PS Write-Output cadena [parámetros]

VARIABLES

- Las variables empiezan con el símbolo $.

- El nombre de las variables pueden contener letras, números y el símbolo "_".
- Si se desea incluir espacios o símbolos problemáticos en el nombre de la variable, se recomienda hacer uso de llaves →{a*?,v**}.
- Las variables no son CASE-SENSITIVE.

- Podemos hacer uso de la variable especial de tubería $_ que hace referencia al objeto de canalización actual.

- Las variables son tipadas o no. Aunque se recomienda que se indiquen su tipo para evitar futuros problemas.

```
$variable=5
```

```
$variable2= [int] 8
```

- Las variables .NET hacen uso de 32 bits.
- Si la variable ya existe, modifica su valor.
- Se referencian utilizando el operador $

```
echo $variable
```

```
echo $variable/$variable2 → 5/8
```

PS dir variable: | sort name

PS dir env: | sort name

Nos mostraría las variables definidas (built-in y de entorno respectivamente).

PS Set-Variable

- Establece el valor de una variable. Crea la variable si no existe ninguna con el nombre solicitado.

PS Get-Variable

- Obtiene las variables de la consola actual.

➢ <u>Ámbito de variable</u>

- Las variables serán validas en el ámbito en el que han sido creadas o declaradas.

- SCOPE: Especifica el ámbito desde el que deben exportarse. Puede ser un ámbito con nombre, "global", "local" o "script". También puede ser el número

correspondiente al ámbito actual (entre 0 y el número de ámbitos, donde 0 es el ámbito actual y 1 su ámbito principal).

- Si hemos declarado una variable dentro de un bucle "**for**", esta variable existirá o estará accesible siempre que estemos dentro del bucle "**for**" si la hemos declarado así.

➢ Las comillas

- Comillas simples:
  - El contenido se interpreta de forma literal.

```
PS $despedida='hola $saludo'

PS echo $despedida

$despedida -> "hola $saludo"
```

- Comillas dobles:
  - Interpreta las referencias a variable.

```
$nombre="Sergio"

$saludo="Hola $nombre, que tal estas?"

$saludo → "Hola Sergio, que tal estas?"
```

➢ Sustitución de orden

- La sustitución de orden permite a la salida de una orden reemplazar al nombre de la orden.
- Bash realiza la expansión ejecutando la orden y reemplazando la orden con la salida estándar de la misma.

```
PS $dir=pwd
PS echo $dir

Path
----
C:\Documents and Settings\Administrador
```

➢ Variables a comentar…

- $^ contiene el primer comando de la última de ejecución.

- $$ contiene el último comando/parámetro de la última ejecución.
- $? estado de terminación de la última orden.
- $Error Si sucede un error, el objeto es guardado en esta variable de PowerShell.

➢ Operadores lógicos y expresiones relacionales

- Operadores lógicos:

Operador lógico	Significado	Ejemplo (devuelve el valor True)
-and	And lógico; devuelve True si ambos lados son True.	(1 -eq 1) -and (2 -eq 2)
-or	Or lógico; devuelve True si cualquiera de los lados es True.	(1 -eq 1) -or (1 -eq 2)
-not	Not lógico; invierte True y False.	-not (1 -eq 2)
!	Not lógico; invierte True y False.	!(1 -eq 2)

- Expresiones relacionales:

Operador	Significado
-lt	Menor que
-le	Menor que o igual que
-gt	Mayor que
-ge	Mayor que o igual que
-eq	Igual a
-ne	No igual a
-like	Like (usa comodines para la coincidencia de patrones)
-match	Uso de expresiones regulares para encontrar…
-contains	Usado para ver si un conjunto de ítems contienen un ítem concreto.

- Operadores matemáticos:

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Modulo

- La orden Read-Host

  - Mediante la orden read-host podemos leer valores de la entrada estándar para guardarlos en la variable o variables que se indiquen. Los datos introducidos por la entrada estándar deben distinguirse por separadores (espacio o tabulador).

Ejemplo:

```
$respuesta = Read-Host "Escribe tu nombre:"

$respuesta → Sergio (introducido antes)
```

CONTROL DE FLUJO

El shell posee sentencias para el control del flujo similares a las existentes en otros lenguajes de programación

If…then

```
If (condición)
{
  Orden 1
  Orden 2
   …….
  Orden n
}
```

Ejemplo

```
if ($a -eq "rojo")

{

   Write-Host "El color es: ROJO"

}
```

If…else

```
If (condicion)
{
  Orden 1
  Orden 2
    ……
  Orden n
}
else
{
   Orden 1
   Orden 2
   ……..
   Orden n
}
```

Ejemplo

```
if ($a -eq "red")
{
   Write-Host "El color es: ROJO"
}
else
{
   Write-Host “Estamos dentro de ELSE”
}
```

If…elseif…..else

```
If (condición)
{
  Orden 1
  Orden 2
   …….
  Orden n
}
elseif (condición)
{
   Orden 1
   Orden 2
    ……
   Orden n
}
else
{
   Orden 1
   Orden 2
    ……
    Orden n
}
```

Ejemplo

```
$a = "blanco"
if ($a -eq "rojo")
{
   Write-Host "El color es: ROJO"
}
elseif ($a -eq "blanco")
{
   Write-Host "El color es: BLANCO"
}
else
{
   Write-Host "Otro color…"
}
```

Bifurcación múltiplie switch

```
Switch ($variable)
{
  “opcion1” { orden 1
              orden 2
               …….
              orden n}
  “opcion2” { orden 1
              orden 2
               ……..
              orden n}
  Default {  orden 1
             orden 2
              ……..
             orden n}
}
```

Ejemplo

```
$a = "rojo"
switch ($a)
{
"red" {Write-Host "El color es ROJO"}
"white"{"El color es BLANCO"}
default{"Otro color"}
 }
```

El bucle for

Version 1:

```
For (variable1=valor_inicio;condicion_de_parada;increment)
{
   Orden 1
   Orden 2
    ……..
   Orden n
}
```

Ejemplo

```
For ($a=1; $a –le 10; $a++)
{
   Write-Host “$a”
}
```

Version 2:

```
Foreach (var in Grupo_Objeto)
{
   Orden 1
   Orden 2
    ……..
   Orden n
}
```

Ejemplo

```
Foreach ($i in Get-Childitem c:\windows)
{
   $i.name; $i.creationtime
}
```

El bucle do…while

```
Do
{
   Orden 1
   Orden 2
    …….
   Orden n
}
```

Ejemplo

```
$a=1
Do
{
  $a; $a++
}
While ($a –lt 10)
```

El bucle do…until

```
Do
{
   Orden 1
   Orden 2
    …….
   Orden n
}
```

Ejemplo

```
$a=1
Do
{
   $a; $a++
}
Until ($a –gt 10)
```

Las ordenes TRUE y FALSE

La orden TRUE siempre devuelve un valor “verdadero”.
La orden FALSE siempre devuelve un valor “falso”.
Puede combinarse con Do-While y Do-Until para formar bucles infinitos.
Ejemplo

```
$a=1
Do
{
   $a; $a++
   Write-Host “$a”
}
Until ($a –gt 10)
```

Funciones

El lenguaje de programación de PowerShell permite crear funciones para realizar tareas repetitivas fácilmente. El funcionamiento es parecido al que posee cualquier lenguaje de programación, en el cual se agrupan conjunto de comandos y se los llama por un nombre. El formato de las funciones es el siguiente:

```
function nomfunc (paramtero1, parámetro 2,….,parámetro n)
{
   orden1
   orden 2
      …
   ordenN
}
```

Ejemplo

```
function sum ([int]$a,[int]$b)
{
   return $a + $b
}
sum 4 5
```

Las características principales de las funciones son:

- Pueden definirse en cualquier lugar
- Para invocar la función hay que escribir el nombre como si fuera un comando.
- Podemos pasarle parámetros.