PowerShell

```
Get-ExecutionPolicy -List
Set-ExecutionPolicy AllSigned
# Otras opciones de políticas de ejecución son:
# - Restricted: Los scripts no correrán.
# - RemoteSigned: Los scripts que se hayan descargado sólo correrán si han sido firmados por un editor de confianza.
# - AllSigned: Los scripts requieren ser firmados por un editor de confianza.
# - Unrestricted: Ejecuta cualquier script.
help about_Execution_Policies # para obtener más ayuda sobre políticas de ejecución.

# Versión instalada de PowerShell:
$PSVersionTable

# Si necesita encontrar algún comando
Get-Command about_* # tiene por abreviación (o alias): gcm
Get-Command -Verb Add # lista todos los comandos que tienen por verbo 'Add'
Get-Alias ps # Lista el alias de ps
Get-Alias -Definition Get-Process

Get-Help ps | less # alias: help
ps | Get-Member # alias: gm

Show-Command Get-EventLog # Muestra un formulario para llenar los parámetros del comando Get-EventLog

Update-Help # Actualiza la ayuda (debe ser ejecutado en una consola elevada como admin)

# Como ya lo notó, los comentarios empiezan con #

# Ejemplo de un simple hola mundo:
Write-Output Hola mundo!
# echo es el alias del comando Write-Output (a los comandos también se les dice cmdlets)
# La mayoría de los cmdlets y funciones siguen la convención de llamarse de la forma: Verbo-Sustantivo

# Cada comando inicia en una nueva línea, o después de un punto y coma:
Write-Output 'Esta es la primer línea'; echo 'Esta es la segunda'

# La declaración de una variable se ve así:
$unaCadena ="Algún texto"
# O así:
$unNumero = 5 -as [double]
$unaLista = 1,2,3,4,5
$unaCadena = $unaLista -join '--' # también existe el parámetro -split
$unaTablaHash = @{nom1='val1'; nom2='val2'}

# Uso de variables:
echo $unaCadena
echo "Interpolación: $unaCadena"
echo "`$unaCadena tiene longitud de $($unaCadena.Length)"
echo '$unaCadena'
echo @"
Esta es una Here-String
$otraVariable
"@
# Note que una ' (comilla simple) no expande las variables!
# Las Here-Strings también funcionan con comilla simple

# Variables Automáticas:
# Hay algunas variables previamente definidas en el ambiente que le pueden servir, tales como
echo "Booleanos: $TRUE y $FALSE"
echo "Valor vacío: $NULL"
echo "Valor de retorno del último programa: $?"
echo "Código de salida del último programa en Windows: $LastExitCode"
echo "El último token en la última línea de la sesión activa: $$"
echo "El primer token: $^"
echo "PID del script: $PID"
echo "Ruta completa del directorio dónde está el script actual: $PSScriptRoot"
echo 'Ruta completa de script actual: ' + $MyInvocation.MyCommand.Path
```

```powershell
echo "Ruta completa de directorio actual: $Pwd"
echo "Argumentos pasados a la invocación de una función, script o bloque de código: $PSBoundParameters"
echo "Argumentos no predefinidos: $($Args -join ', ')."
# Para saber más sobre variables automáticas: `help about_Automatic_Variables`

# Para enlazar otro archivo (operador punto)
. .\otroNombreDeScript.ps1


### Control de Flujo
# Tenemos la estructura de if como es usual:
if ($Edad -is [string]) {
  echo 'Pero... si $Edad no puede ser una cadena de texto!'
} elseif ($Edad -lt 12 -and $Edad -gt 0) {
  echo 'Niño (Menor de 12. Mayor que 0)'
} else {
  echo 'Adulto'
}

# Sentencias switch de PS son más poderosas comparadas con otros lenguajes
$val = "20"
switch($val) {
  { $_ -eq 42 }           { "La respuesta es 42"; break }
  '20'                    { "Exactamente 20"; break }
  { $_ -like 's*' }       { "No distingue entre mayúsculas/minúsculas"; break }
  { $_ -clike 's*'}       { "clike, ceq, cne para ser diferenciar el caso entre mayúsculas/minúsculas"; break }
  { $_ -notmatch '^.*$'}  { "Emparejamiento de expresiones regulares. cnotmatch, cnotlike, ..."; break }
  { 'x' -contains 'x'}    { "FALSO! -contains es para listas!"; break }
  default                 { "Otros" }
}

# El for clásico
for($i = 1; $i -le 10; $i++) {
  "Número de ciclo $i"
}
# O más corto
1..10 | % { "Número de ciclo $_" }

# PowerShell también incluye
foreach ($var in 'valor1','valor2','valor3') { echo $var }
# while () {}
# do {} while ()
# do {} until ()

# Manejo de excepciones
try {} catch {} finally {}
try {} catch [System.NullReferenceException] {
 echo $_.Exception | Format-List -Force
}


### Proveedores
# Lista de archivos y directorios en la ubicación actual
Get-ChildItem # o el alias `dir`
Set-Location ~ # ir al directorio principal del usuario

Get-Alias ls # -> Get-ChildItem
# ¿¡Eh!? Estos cmdlets tienen nombres genéricos porque a diferencia de otros lenguajes de scripting,
# PowerShell no opera únicamente en el directorio actual.
Set-Location HKCU: # se dirige a la rama HKEY_CURRENT_USER del registro de Windows

# Para hacer un listado de todos los proveedores disponibles
Get-PSProvider


### Tuberías
# Los Cmdlets tienen parámetros que controlan su ejecución:
Get-ChildItem -Filter *.txt -Name # Se obtiene sólo el nombre de todos los archivos txt
# Sólo se necesita escribir caracteres de un parámetro hasta que deja de ser ambiguo
ls -fi *.txt -n # -f no se puede porque también existe -Force
# Use `Get-Help Get-ChildItem -Full` para un tratado más completo
```

```powershell
# Los results del cmdlet anterior se le pueden pasar como entrada al siguiente.
# `$_` representa el objeto actual en el objeto de tubería.
ls | Where-Object { $_.Name -match 'c' } | Export-CSV exportado.txt
ls | ? { $_.Name -match 'c' } | ConvertTo-HTML | Out-File exportado.html

# Si se confunde con la tubería use `Get-Member` para revisar
# los métodos y propiedades de los objetos de la tubería:
ls | Get-Member
Get-Date | gm

# ` es el caracter de continuación de línea. O termine la línea con un |
Get-Process | Sort-Object ID -Descending | Select-Object -First 10 Name,ID,VM `
 | Stop-Process -WhatIf

Get-EventLog Application -After (Get-Date).AddHours(-2) | Format-List

# Use % como una abreviación de ForEach-Object
(a,b,c) | ForEach-Object `
 -Begin { "Iniciando"; $counter = 0 } `
 -Process { "Procesando $_"; $counter++ } `
 -End { "Terminando: $counter" }

# El siguiente comando ps (alias de Get-Process) devuelve una tabla con 3 columnas
# La tercera columan es el valor de memoria virtual en MB y usando 2 dígitos decimales
# Las columnas calculadas pueden escribirse más extensamente como:
# `@{name='lbl';expression={$_}`
ps | Format-Table ID,Name,@{n='VM(MB)';e={'{0:n2}' -f ($_.VM / 1MB)}} -autoSize


### Funciones
# El atributo [string] es opcional.
function foo([string]$nombre) {
 echo "Hey $nombre, aquí tiene una función"
}

# Llamando una función
foo "Diga mi nombre"

# Funciones con parámetros nombrados, atributos de parámetros y documentación analizable
<#
.SYNOPSIS
Establecer un nuevo sitio web
.DESCRIPTION
Crea todo lo que su sitio necesite
.PARAMETER siteName
El nombre para el nuevo sitio web
.EXAMPLE
Crear-SitioWeb -Nombre SitioBonito -Po 5000
Crear-SitioWeb SiteWithDefaultPort
Crear-SitioWeb nombreSitio 2000 # ERROR! No se pudo validar arguemento de puerto
('nombre1','nombre2') | Crear-SitioWeb -Verbose
#>
function Crear-SitioWeb() {
 [CmdletBinding()]
 param (
  [Parameter(ValueFromPipeline=$true, Mandatory=$true)]
  [Alias('nombre')]
  [string]$nombreSitio,
  [ValidateSet(3000,5000,8000)]
  [int]$puerto = 3000
 )
 BEGIN { Write-Verbose 'Creando nuevo(s) sitio(s) web' }
 PROCESS { echo "nombre: $nombreSitio, puerto: $puerto" }
 END { Write-Verbose 'Sitio(s) web creado(s)' }
}


### Todo es .NET
# Una cadena PS es, de hecho, una cadena tipo System.String de .NET
# Todos los métodos y propiedades de .NET están disponibles
```

```powershell
'cadena'.ToUpper().Replace('E', 'eee')
# O más powershellezco
'cadena'.ToUpper() -replace 'E', 'eee'

# ¿No recuerda cómo es que se llama cierto método .NET?
'cadena' | gm

# Sintaxis para ejecutar métodos .NET estáticos
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic')

# Nótese que cualquier función que proviene de .NET Framework REQUIERE paréntesis para ser invocada
# al contrario de las funciones definidas desde PS, las cuales NO PUEDEN ser invocadas con paréntesis.
# Si se invoca una función/cmdlet de PS usando paréntesis,
# es equivalente a que le estuviera pasando un parámetro de tipo lista
$writer = New-Object System.IO.StreamWriter($ruta, $true)
$writer.Write([Environment]::NewLine)
$writer.Dispose()

### Entrada/Salida
# Leyendo una variable
$Nombre = Read-Host "¿Cómo se llama?"
echo "¡Hola $Nombre!"
[int]$Edad = Read-Host "¿Cuál es su edad?"

# Test-Path, Split-Path, Join-Path, Resolve-Path
# Get-Content filename # devuelve un string[]
# Set-Content, Add-Content, Clear-Content
Get-Command ConvertTo-*,ConvertFrom-*


### Material útil
# Actualizar la ruta de ejecuciones (PATH)
$env:PATH = [System.Environment]::GetEnvironmentVariable("Path", "Machine") +
 ";" + [System.Environment]::GetEnvironmentVariable("Path", "User")

# Encontrar Python en el path
$env:PATH.Split(";") | Where-Object { $_ -like "*python*"}

# Cambiar el directorio de trabajo sin tener que memorizar la ruta anterior
Push-Location c:\temp # se cambia el directorio de trabajo a c:\temp
Pop-Location # revierte el cambio y se devuelve a donde estaba al principio
# Los aliases son : pushd y popd

# Desbloquear un archivo después de descargarlo de Internet
Get-ChildItem -Recurse | Unblock-File

# Abre Windows Explorer en la ruta actual (usando el alias ii de Invoke-Item)
ii .

# Pulse cualquier tecla para salir
$host.UI.RawUI.ReadKey()
return

# Para crear un acceso directo
$WshShell = New-Object -comObject WScript.Shell
$Shortcut = $WshShell.CreateShortcut($link)
$Shortcut.TargetPath = $file
$Shortcut.WorkingDirectory = Split-Path $file
$Shortcut.Save()

# $Profile es la ruta completa para su `Microsoft.PowerShell_profile.ps1`
# Todo el código alojado allí será ejecutado cuando se ejecuta una nueva sesión de PS
if (-not (Test-Path $Profile)) {
 New-Item -Type file -Path $Profile -Force
 notepad $Profile
}
# Más información en: `help about_profiles`
# Para un shell más productivo, asegúrese de verifivar el proyecto PSReadLine descrito abajo
```

Enlaces de interés

- https://learn.microsoft.com/es-es/powershell/
- https://learn.microsoft.com/es-es/powershell/scripting/learn/ps101/00-introduction?view=powershell-7.3
- https://www.maquinasvirtuales.eu/curso-basico-de-powershell-introduccion/
- https://gist.github.com/pcgeek86/336e08d1a09e3dd1a8f0a30a9fe61c8a
- https://github.com/janikvonrotz/awesome-powershell

Licencia

Esta obra está bajo una licencia de Creative Commons Reconocimiento-Compartir Igual 4.0 Internacional.