

- Lista de comandos

Get-Command

Get-Command -Verb Format

Get-Command -Noun Random

Get-Command -Noun *user*

Alias de Get-Command

gcm -Noun item, content

- Ayudas:

Update-Help (como administrador)

Help Get-Date

Get-Help Get-Date -Full

Get-Help Get-LocalUser **-Online**

- Uso de parámetros:

Get-Command **-CommandType** Alias

Get-Process **-Name** powershell

Get-Command -CommandType Alias **-Name** gci

Alias -Name f*

gci **-Directory**

Get-Random -Minimum 8 -Maximum 20 -OutVariable n > \$null

- Canaliza la salida del comando Get-Command hacia Get-Member para mostrar sus propiedades y métodos (estructura del objeto o miembros)

Get-Command -CommandType Alias | **Get-Member**

Get-Date | **gm** ← muestra los miembros del cmdlet Get-Date

(Get-Date).GetType()

(Get-Date).DayOfWeek

Get-LocalUser -Name alberto | gm

- Canaliza la salida del comando hacia una ventana externa, una impresora o un fichero.

Get-Command | **Out-GridView**

Get-Process | Out-GridView

Get-Process | Out-Host **-paging** ← paginando la salida

Get-Process | **Out-File** -FilePath E:\informes\procesos.txt

- Comandos de formato para cambiar la vista de salida

get-item . | **Format-Wide**

get-item . | Format-Wide **-Property** Name

get-item . | **Format-List**

get-item * | **fl**

alias | **ft** -Property Name, ResolvedCommand

Get-LocalUser -Name alberto | **fl -Property** Lastogon

Get-Process -Name iexplore | Format-List -Property ProcessName,FileVersion,StartTime,Id

Get-Service -Name win* | **Format-Table** -Property Name, Status, StartType, DisplayName, DependentServices **-Wrap**

Get-Service -Name win* | **Sort-Object** StartType | Format-Table **-GroupBy** StartType

- Lista de elementos del contenedor actual (directorio o carpeta)

Get-Childitem

dir, ls o **gci** son alias de *Get-ChildItem*

```
gci env:
gci variable:
$env:SYSTEMROOT
```

- Muestra información sobre un elemento

Get-Item *elemento*

Get-Item . ← muestra información sobre el directorio actual

(Get-Item .).LastAccessTime ← obtiene el valor de la propiedad

- Muestra información de usuarios/grupos locales

Get-LocalUser -Name Alberto | **format-list**

Get-LocalGroup

Get-LocalGroup -Name Administradores | fl

- Unidades de disco, Recursos compartidos

Get-Disk

Get-Partition

Get-SmbShare

- Información de red

Get-NetAdapter

- Eventos

Get-EventLog -List ←tipos de eventos

Get-EventLog -LogName System -Index nnn | fl

*Get-WinEvent -Listlog ** ← todos los tipos de registro

- *Invoke-Item* ejecuta la acción predeterminada para un archivo/carpeta

Invoke-Item C:\WINDOWS

Invoke-Item prueba.txt

- Seleccionar partes de objetos (*más adecuado si se va a canalizar la respuesta*)

alias | **Select-Object** -Property Name, ReferencedCommand

- Filtrar objetos de la canalización

Get-Service | Where Status -eq "Stopped"

alias | **Select-Object** -Property Name, ReferencedCommand | **Where-Object**

{\$_ReferencedCommand -like "Get-ChildItem"}

➔ *\$_ enumera cada uno de los elementos del resultado del comando previo*

➔ *-like operador de comparación*

➔ *en este caso, se ha utilizado Select-Object en lugar de Format-Table porque con este último no se obtiene una colección de elementos enumerables*

Get-Alias | ? -Property ReferencedCommand -Match 'Format'

- Alias | Select-Object -Property Name, ReferencedCommand | Where-Object {\$_ReferencedCommand.ToString() -eq "Get-ChildItem"}
- Alias | Select-Object -Property Name, ReferencedCommand | Where-Object {\$_ReferencedCommand.Name -eq "Get-ChildItem"}
- Get-LocalUser | Where-Object {\$_Enabled} | Select -First 1 -Property Name, PasswordExpires

```
? Where-Object
$_ <elemento enumerado>
% Foreach-Object
```

- Asignar un valor a una variable
\$var=64
\$var2="sesenta y cuatro"
\$var3=Get-Childitem
[String]\$cadena="abcd"
- Mostrar el contenido de una variable → \$var
Write-Host \$var ← escribe directamente en la consola
Write-Output \$var ← se puede redirigir su salida por defecto (la consola)
- Declarar y poblar un array
\$lista=1,2,3,4,5
\$lista2=@(6,7,8,9,10)
\$lista3="Uno","dos","tres"
- Mostrar el contenido de un array
\$lista | ?{\$_%2 -eq 0} *para guardar el resultado→* \$pares=\$lista |?{\$_%2 -eq 0}
Write-Host "números pares: " (\$lista | ? {\$_%2 -eq 0})
\$lista | %{Write-Host "raíz cuadrada de "\$_ = ([Math]::Sqrt(\$_))}
\$lista | ForEach-Object -Begin {\$n=1} -Process {write-host "(\$n+\$_)=" (\$n+\$_); \$n++}
- Interactuar con cada elemento del array
\$lista | **ForEach-Object** {write " '\$_'" }
\$lista | **ForEach-Object** -Begin {\$n=1} -Process {write "\$n-\$_" ; \$n++}
foreach (\$e in \$lista) {write \$e} ← sintaxis de C#
- Declarar y poblar una *Hashtable* (Diccionario o Array Asociativo)
\$tabla=@{"VA"="Valladolid" ; "P"="Palencia"}
- Mostrar contenido del hashtable
\$tabla
\$tabla | gm ← muestra los miembros de la hashtable
\$tabla.Keys ← muestra todas las claves
\$tabla.Values ← muestra todos los valores

- Acceder a un elemento del *hashtable*

```
$tabla.Va  
$tabla.P  
$tabla["VA"]
```

- Añadir un elemento
\$tabla.Add("BU","Burgos")

- Eliminar un elemento
\$tabla.Remove("P")

- Modificar un elemento
\$tabla.VA="Pucela"

- Crear *hashtable* desde un archivo csv

```
$usuarios=Import-Csv -Path usuarios.csv  
Import-Csv -Path usuarios.csv | %{write $_.nombre} ← muestra los nombres de los usuarios
```

(Si se ha creado con este encabezado)

- Diferencia entre Write-Host y Write-Output

```
$wh="Host: uno dos tres cuatro" | Write-Host  
$wo="Output: uno dos tres cuatro" | Write-Output  
$wh,$wo  
Write-Output "1 2 3 4 5" | %{$_.Split()}  
Write-Host "1 2 3 4 5" | %{$_.Split()}
```

- Equivalencias con comandos del 'Símbolo del Sistema':

- *Get-ChildItem* (Dir, Ls) Muestra el contenido de una carpeta
- *Set-Location* (Cd) Cambia de carpeta
- *Move-Item* (Move) Mueve un archivo/carpeta
- *Rename-Item* (Ren) Cambia su nombre
- *Remove-Item* (Del, Rmdir) elimina un archivo/carpeta
- *Copy-Item* (Copy) Copia un archivo/carpeta
- *Get-Content* (Type) Muestra el contenido de un archivo
- *New-Item* (Mkdir) Crea un archivo/carpeta (*-ItemType {File | Directory}*)
 New-Item -Path .\prueba.txt -ItemType File
 Add-Content prueba.txt -Value (Get-Date).ToShortDateString()

EJEMPLOS

```
Get-ChildItem | ?{$_.Mode -match "d"} | Select-Object -Property FullName
Get-ChildItem | Where Mode -match "d" | Select-Object -Property FullName
Get-ChildItem | ?{$_.Mode -match "d"} | %{write $_.FullName}
(Get-ChildItem | ? mode -match "d").FullName
Write-Host "En esta carpeta hay" (Get-ChildItem | ? mode -match "d").Count "directorios"
Get-ChildItem -Directory | Select-Object -Property FullName
gci -Directory | Select-Object -Property FullName | Export-Csv -Path listaD.csv
(Get-Alias -Definition Get-ChildItem | select Name).Name
wget -OutFile gato.jpg -Uri http://placekitten.com/800/600 -PassThru | Select RawContentLength
Invoke-Item gato.jpg
```

```
alias | Export-csv alias.csv
ps | Export-csv procesos.csv
```

```
Get-Process -Name powershell | Format-Wide -Property Id
Get-Process | Select-Object -Property ProcessName,@{Name="Iniciado";Expression={$_.StartTime }}
Get-Process | ConvertTo-HTML | Out-File procesos.html
Invoke-Item procesos.html
```

```
Get-LocalUser | Get-Member ← muestra las "propiedades" del objeto LocalUser
Get-LocalUser | Where-Object {$_.Enabled -eq $true} | Select-Object -Property Name
Get-LocalUser | ft -Property SID, Name
```

```
Get-Random -minimum 1 -maximum 100
1,3,5,7 | Get-Random 'A','B','C' | Get-Random 1..100 | Get-Random -count 3
```

```
ls function:[d-z]: -Name | ?{!(Test-Path $_)} | Get-Random
ls function:[d-z]: -Name | ?{!(Test-Path $_)} | Select -First 1
```

```
[Environment]::ls64BitProcess
```

```
[Environment]::OSVersion
```

```
Write-Host "La versión de Windows es" ([Environment]::OSVersion).Version
```

```
Get-Host ←muestra información sobre la versión de PowerShell instalada
```

```
Add-Type -AssemblyName System.speech ←Incorpora la librería con las funciones del speaker
(New-Object System.Speech.Synthesis.SpeechSynthesizer).Speak("Hola seres humanos del
Gregorio Fernández")
```

```
$resultado=Write-Output "La versión de windows es " ([System.Environment]::OSVersion).Version
$voz=New-Object System.Speech.Synthesis.SpeechSynthesizer
$voz.Speak($resultado)
```

WMI - CIM

Windows Management Instrumentation (WMI) es una infraestructura que contiene datos y operaciones administrativas en sistemas operativos basados en Windows. Se pueden escribir *scripts* o aplicaciones de WMI para automatizar tareas administrativas en equipos remotos, pero WMI también proporciona datos de administración a otras partes del sistema operativo o aplicaciones.

p.e.: *Win32_diskdrive, Win32_LogicalDisk, Win32_OperatingSystem*

Get-WmiObject -List

Get-WmiObject -class **Win32_DiskDrive** | Format-Table -Property Index, Model

Get-WmiObject -class **Win32_LogicalDisk** | Select-Object -Property Name, FreeSpace

Los cmdlets de Modelo de información común (CIM) se incorporaron en la versión 3.0 de PowerShell. Los cmdlets de CIM están diseñados para usarse en equipos Windows y no Windows. Los cmdlets de WMI están en desuso, por lo que se recomienda usar los cmdlets de CIM en lugar de los más antiguos de WMI.

Todos los cmdlets de CIM están incluidos en un módulo. Para obtener una lista de los cmdlets de CIM, use Get-Command con el parámetro Module:

Get-Command -Module **CimCmdlets**

Lista de recursos de hardware y configuración del equipo:

Get-CimClass -Namespace **root/CIMV2**

Get-CimInstance -ClassName Win32_DiskDrive -Property *

Get-CimInstance -ClassName Win32_DiskDrive | Select-Object -Property Index, Model

Get-CimInstance -ClassName Win32_DiskDrive | Select-Object -Property Index, Model, Description, MediaType, Partitions | Format-Table

[Información relevante en Documentos de Microsoft](#)