Intro Powershell

Introdución

É o novo shell en liña de comandos de Windows pensado para substituír o shell clásico cmd.

As principais diferencias son:

- **Orientación a obxectos**:
  - **CMD:** Traballa con texto plano. Os comandos producen saídas como liñas de texto, e para procesalas, necesitas ferramentas externas.
  - **PowerShell:** Traballa con obxectos, Isto significa que os comandos devolven estruturas de datos (obxectos) que podes manipular directamente sen converter todo a texto.
- **Comandos (Cmdlets):**
  - **CMD:** Utiliza comandos básicos e sinxelos como `dir`, `copy`, `del`, etc. que proveñen do antigo MS-DOS.
  - **PowerShell:** Introduce cmdlets, que son comandos específicos cunha nomenclatura uniforme (verbo-nome) como `Get-Command`, `Set-Item`, `New-Object`
- **Pipelines mellorados:**
  - **CMD:** Permite redirixir texto entre comandos usando |, pero só funciona con datos en formato de texto
  - **PowerShell**: Os pipelines poden pasar obxectos entre comandos, o que é moito máis poderoso e flexible.
- **Programación e scripts:**
  - **CMD:** Pode executar scripts .bat ou .cmd, pero as capacidades de programación son limitadas.
  - **PowerShell:** Pode executar scripts .ps1
- **Multiplataforma**
  - **CMD:** Só funciona en sistemas Windows.
  - **PowerShell:** Desde PowerShell Core (versión 6 en adiante), é multiplataforma e funciona en Windows, macOS e Linux.

Exemplo: Listar os procesos ordenados por consumo de memoria

- **Powershell:**

```
Get-Process | Sort-Object -Property WorkingSet -Descending | Select-Object -First
10
```

- **CMD**

```
tasklist | sort
```

Podemos ordenar alfabeticamente por unha columna pero non conseguiremos o que queremos

Cmdlets

- Un cmdlet é un comando compilado.
- Pódense invocar como un comando dende PowerShell.
- Hai miles de cmdlets dispoñibles.
- Os cmdlets teñen o formato verbo-nome. (Get-Help)

Os seguintes cmdlets axúdanos a atopar o cmdlet que precisemos

- **Get-Command**: Lista todos os cmdlets dispoñibles no sistema.
  - Podemos filtrar a súa saída con -Noun ou -Verb
- **Get-Help:** Amosa a axuda dun cmdlet. Tamén podemos empregar o alias help.
  - `Get-Help -Name 'name-of-command'`
- **Get-Member**: Os cmdlets devolven obxectos e propiedades. Por exemplo para ver os métodos dunha cadea de texto

```
PS /home/rojastic> "a string" | Get-Member

   TypeName: System.String

Name                 MemberType            Definition
----                 ----------            ----------
Clone                Method                System.Object Clone(), System…
CompareTo            Method                int CompareTo(System.Object v…
Contains             Method                bool Contains(string value),
```

Exemplos:

- Atopa cmdlets que o seu nome comece por File

```
PS /home/rojastic> Get-command -Noun File*

CommandType     Name                                               Version
-----------     ----                                               ------
Function        Export-File                                        0.9.3
Cmdlet          Get-FileHash                                       7.0.0…
Cmdlet          Out-File                                           7.0.0…
Cmdlet          Unblock-File                                       7.0.0…
```

- Atopa cmdlets que o verbo sexa get e o seu nome comece por File

```
PS /home/rojastic> Get-command -Verb Get -Noun File*

CommandType     Name                                               Version
-----------     ----                                               ------
Cmdlet          Get-FileHash                                       7.0.0…
```

- Examinando a axuda dun comando

```
PS /home/rojastic> Get-Help -Name Get-FileHash

NAME
    Get-FileHash
```

```
SYNOPSIS
    Computes the hash value for a file by using a specified hash
    algorithm.
```

- Para ver só os exemplos dun comando

```
help Get-FileHash -Examples
```

Traballando con obxectos

- Cada vez que executamos un cmdlet, devólvese un obxecto
- **Exemplo:**
  - Obtemos os procesos executándose na nosa máquina

```
PS /home/rojastic> get-process

 NPM(K)    PM(M)      WS(M)     CPU(s)      Id  SI ProcessName
 ------    -----      -----     ------      --  -- -----------
      0     0.00      83.59       1.95      20   1 node
      0     0.00     268.91       5.69     134 131 pwsh
      0     0.00       3.18       0.00     133 131 runuser
      0     0.00       0.68       0.00       1   1 sh
      0     0.00       3.20       0.00       6   1 startNode.sh
      0     0.00       3.13       0.00     131 131 startPwsh.sh
```

  - Obtemos información dun proceso concreto

```
PS /home/rojastic> get-process -Name node

 NPM(K)    PM(M)      WS(M)     CPU(s)      Id  SI ProcessName
 ------    -----      -----     ------      --  -- -----------
      0     0.00      87.77       2.08      20   1 node
```

  - Comprobamos a información devolta é un obxecto,
    - É un obxecto de tipo Process
    - Ten propiedades, eventos e métodos

```
PS /home/rojastic> get-process -Name node | Get-Member

   TypeName: System.Diagnostics.Process

Name                        MemberType     Definition
----                        ----------     ----------
Handles                     AliasProperty  Handles = Handlecount
```

  - Para obter outros cmdlets que traballan co obxecto process

```
PS /home/rojastic> Get-Command -ParameterType Process

CommandType     Name                                               Versio
                                                                   n
-----------     ----                                               ------
Cmdlet          Debug-Process                                      7.0.0…
Cmdlet          Enter-PSHostProcess                                7.2.0…
Cmdlet          Get-Process                                        7.0.0…
Cmdlet          Get-PSHostProcessInfo                              7.2.0…
Cmdlet          Stop-Process                                       7.0.0…
Cmdlet          Wait-Process                                       7.0.0…
```

Traballando coa saíd dun CMDLETS como obxecto

- Examinamos a información obtida do proceso notepad.

```
PS C:\Users\rojas> get-process -Name notepad*

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    667      31    71448     124456       0,63   3236   1
```

- A saída do anterior comando é un obxecto da clase Process e podemos examinar as súas propiedades e métodos.

```
PS C:\Users\rojas> (get-process -Name notepad*) | get-member


   TypeName: System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
NPM                        AliasProperty  NPM = NonpagedSystemMemorySize64
PM                         AliasProperty  PM = PagedMemorySize64
SI                         AliasProperty  SI = SessionId
VM                         AliasProperty  VM = VirtualMemorySize64
WS                         AliasProperty  WS = WorkingSet64
Disposed                   Event          System.EventHandler Disposed(System.Object,
System.EventArgs)
ErrorDataReceived          Event          System.Diagnostics.DataReceivedEventHandler
ErrorDataReceived(System.Object, System.Diagnostics.DataReceivedEventArgs)
Exited                     Event          System.EventHandler Exited(System.Object,
System.EventArgs)
OutputDataReceived         Event          System.Diagnostics.DataReceivedEventHandler
OutputDataReceived(System.Object, System.Diagnostics.DataReceivedEventArgs)
BeginErrorReadLine         Method         void BeginErrorReadLine()
BeginOutputReadLine        Method         void BeginOutputReadLine()
CancelErrorRead            Method         void CancelErrorRead()
CancelOutputRead           Method         void CancelOutputRead()
Close                      Method         void Close()
```

- Para tratar a saída como un obxecto e acceder as súas propiedades e métodos poñémola entre parénteses.

```
PS C:\Users\rojas> (get-process -Name notepad*).Name
Notepad
PS C:\Users\rojas> (get-process -Name notepad*).Path
C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2410.21.0_x64__8wekyb3d8bbwe\
Notepad\Notepad.exe
```

- Rematamos o proceso

```
PS C:\Users\rojas> (get-process -Name notepad*).kill()
```

Examinando as propiedades dun listado de obxectos

- Ordenando un listado de obxectos

```
Get-Process  | Sort-Object cpu
```

Get-Process devolve un listado de obxectos da clase Process.

Ese listado de obxecto pode ser manipulado por Sort-Object para ordenalos por un determinado campo.

Exercicio: Como cambiaríamos o criterio de ordenación para que o faga en orde descendente.

- **Select-Object**

  Permite seleccionar só unhas propiedades dun obxecto ou quedarnos só con un determinado de obxectos

  O listado de procesos ordenados podémosllo pasar a outro cmd-let para quedarnos só con un determinado número de parámetros.

  Exercicio: Amosa os 5 procesos que consuman máis RAM.

  Exercicio: Amosa dos 5 procesos que consuman máis RAM só as propiedades nome e consumo de cpu.

Formateando a saída dos CMDLETS

Por defecto, cando executamos un comando a súa saída vai á pantalla (Out-Default). Como a saída é un obxecto, se hai unha vista rexistrada para ese tipo de obxecto, o amosarse por pantalla formatease dun determinado xeito.
As tuberías permiten cambiar isto e por exemplo, executar uns comandos sobre o resultados dos comandos anteriores.

Podemos empregar Select-Object para modificar o comportamento por defecto, escollendo só as propiedades que nos interesen

- Amosamos só as columnas que nos interesan

```
Get-Process zsh | Select-Object -Property Id, Name, CPU
```

- Amosamos só as columnas que nos interesan en orde descendente

```
Get-Process | Sort-Object -Descending -Property Name, CPU
```

- Cambiamos o título dunha columna polo que nos interese

```
Get-Process 'some process' | Sort-Object -Property @{Expression = "Name"; Descending =
$True}, @{Expression = "CPU"; Descending = $False}
```

- Podemos probar distintos formatos de saída

```
 Get-Process -Name sh | Format-List

Id      : 1
Handles :
CPU     : 0.01
SI      : 1
Name    : sh
```

```
 Get-Process -Name sh | Format-Table

 NPM(K)    PM(M)      WS(M)     CPU(s)      Id  SI
ProcessName
 ------    -----      -----     ------      --  --
-----------
      0     0.00       0.73       0.01       1   1 sh
```

- Se non queremos encabezado

```
(Get-Process).Name

Get-Process | Select-Object Name | ft -hide
```

Filtrar as propiedades dun obxecto - Instrospección – Introspection

Sérvenos para obter información dos obxectos
Por exemplo sabemos que un obxecto almacena unha determinada información, pero non sabemos o campo onde a almacena
Para elo podemos empregar **Get-Member**.

- Obtemos a información do obxecto devolto por Get-Process

```
PS C:\Users\usuario\Desktop\05-Powershell> get-process | Get-Member -MemberType
Properties


   TypeName: System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
NPM                        AliasProperty  NPM = NonpagedSystemMemorySize64
PM                         AliasProperty  PM = PagedMemorySize64
SI                         AliasProperty  SI = SessionId
VM                         AliasProperty  VM = VirtualMemorySize64
WS                         AliasProperty  WS = WorkingSet64
__NounName                 NoteProperty   string __NounName=Process
BasePriority               Property       int BasePriority {get;}
Container                  Property       System.ComponentModel.IContainer Container
{get;}
```

Filtrar Saída dun cmdlet

Exemplo 1: Obter programas instalados

Get-Package obtén os programas instalados, actualizacións, etc en formato táboa.

```
PS D:> Get-Package

Name                           Version          ProviderName
------------                   -------          ------
7-Zip 21.07 (x64)              21.07            Programs
GIMP 2.10.30                   2.10.30          Programs
Mozilla Firefox (x64 es-ES)    99.0.1           Programs
Microsoft Visual C++ 2022 X... 14.31.311        msi
Microsoft Visual C++ 2013 x... 12.0.21005       msi
```

- Quedámonos só cos programas

```
PS D:> Get-Package | Where-Object { $_.ProviderName -eq "Programs"}

Name                           Version          ProviderName
----                           -------          ------
7-Zip 21.07 (x64)              21.07            Programs
GIMP 2.10.30                   2.10.30          Programs
Mozilla Firefox (x64 es-ES)    99.0.1           Programs
Mozilla Maintenance Service    96.0.3           Programs
Notepad++ (64-bit x64)         8.3.3            Programs
```

$_ É coñecida como Pipeline Variable, representa o obxecto
Where-object: Selecciona obxectos dunha colección baseándose nunha condición.

- Quedámonos só co nome dos programas e a versión

```
PS D:> Get-Package | Where-Object { $_.ProviderName -eq "Programs"} | Select-Object
Name,Version

Name                                Version
----                               -------
7-Zip 21.07 (x64)                  21.07
GIMP 2.10.30                       2.10.30
Mozilla Firefox (x64 es-ES)        99.0.1
Mozilla Maintenance Service        96.0.3
```

Contar o número de programas instalados

```
(Get-Package | Where-Object { $_.ProviderName -eq "Programs"} |  Group-object).Count
25
```

Saber se temos algún antivirus instalado

```
PS D:> Get-Package |  Where-Object { $_.Name -like "*virus*"} | Select-Object Name

Name
----
Actualización de inteligencia de seguridad para Microsoft Defender Antivirus - KB2267602
(Versión 1.363.1310.0)
Actualización de inteligencia de seguridad para Microsoft Defender Antivirus - KB2267602
(Versión 1.363.688.0)
```

Contar as ocurrencias dun caracter nunha cadea de texto

```
PS D:\> ("123451891".toCharArray() | where-object {$_ -eq  '1'}).Count
3
```

- Convertemos a cadea de texto a un array de caracteres e comparamos cada un deles.

```
rojas@debianRojas:~$
```

Variables

- As variables comenzan por $
- Powershell non diferencia as maiúsculas e minúsculas, no nome das variables tampouco
- Decláranse facendo a súa primeira asignación

```
$numero = 1
$nome = “Xiana”
```

Cadeas de Texto

- Podemos tratar os elementos dunha cadea de texto

```
$frase="Texto largo"
#Extraer primer elemento (elemento 0)
$frase[0]
#Extraer segundo elemento (elemento 1)
$frase[1]
#Extraer último elemento
$frase[10]

#Determinar la longitud de una frase
$frase="Texto largo"
$frase.Length
```

Scripts

Os scripts teñen a extensión .ps1: **Exemplo.ps1**

```
# Powershell como python ten conversión dinámica de tipos
# Indicamos que ten que almacenar o valor como número
[Int]$Number = Read-Host "Introduce un valor"
Introudce un valor: 5
$Square=$Number*$Number
Write-Host "O cadrado de $Number é $Square."
```

Por defecto e por razóns de seguridade a execución de scripts está deshabilitada.

```
PS C:\Users\rojas> powershell .\exemplo.ps1
.\exemplo.ps1 : No se puede cargar el archivo C:\Users\rojas\exemplo.ps1 porque la
ejecución de scripts está deshabilitada en este sistema.
```

Se queremos executar scripts temos explicitalo.

```
PS C:\Users\rojas> powershell.exe -ExecutionPolicy Bypass .\exemplo.ps1
Introduce un valor: 6
O cadrado de 6 é 36.
```