



# Sistemes Operatius

"Mini-manual" de PowerShell

## Pràctiques

Curs 2021-22

David Martínez

Josep Blanes

Pere Vilà

Teo Jové

PowerShell és el “nou” entorn de comandes de Microsoft inclòs al sistema des de les versions Windows 7 i Windows Server 8. Era possible també descarregar-lo i instal·lar-lo en el Windows XP. És el sistema per defecte per treballar amb les darreres versions de Windows Server ja que és possible instal·lar un Windows server sense entorn gràfic (com? ostres!).

La principal característica és tot un nou conjunt de comandes que s’anomenen “Comand Lets” o **cmdlets** i que estan pensades com un Verb-Nom (per exemple la comanda **get-help**). La principal potència però ve perquè el PowerShell està lligat amb l’entorn .NET de forma que es poden utilitzar totes les funcions de les llibreries .NET directament des de la línia de comandes. Això inclou per exemple totes les funcions de les següents DLL:

- |  |                                    |
|--|------------------------------------|
| • Microsoft.CSharp.dll                         | • System.Data.dll                  |
| • Microsoft.Management.Infrastructure.dll      | • System.DirectoryServices.dll     |
| • Microsoft.PowerShell.Commands.Management.dll | • System.dll                       |
| • Microsoft.PowerShell.Commands.Utility.dll    | • System.Dynamic.dll               |
| • Microsoft.PowerShell.ConsoleHost.dll         | • System.Management.Automation.dll |
| • Microsoft.PowerShell.Security.dll            | • System.Management.dll            |
| • mscorlib.dll                                 | • System.Numerics.dll              |
| • System.Configuration.Install.dll             | • System.Transactions.dll          |
| • System.Core.dll                              | • System.Xml.dll                   |

Per exemple, es poden tractar excepcions, parsejar documents XML, etc. Una de les principals diferències amb els Shells de UNIX és que no està orientat a text, sinó que està orientat a objectes. Això xoca una mica ja que en el PowerShell el format de sortida de les comandes ja no és tan important.

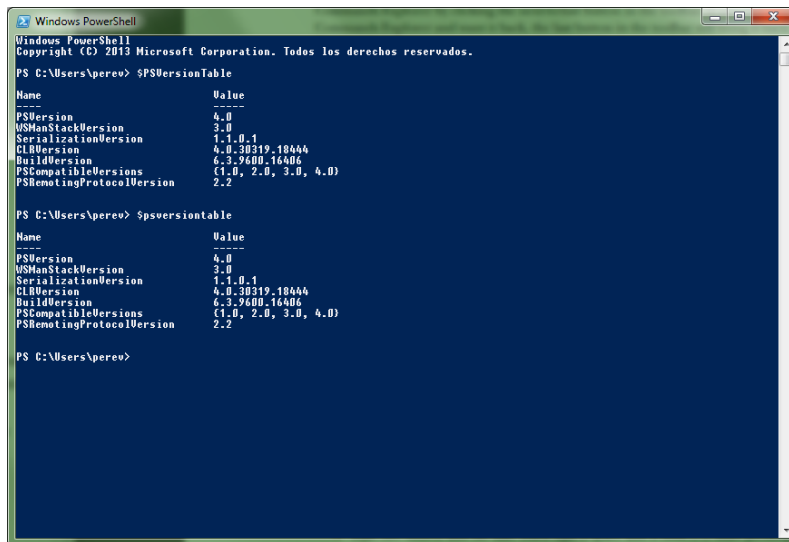
Un dels Cmdlets més importants, apart del **Get-Help**, és el **Get-Member**. Aquest Cmdlet et retorna la informació dels atributs i mètodes d’un objecte. Simplement s’agafa qualsevol variable (objecte) i es fa una pipe amb Get-Member i obtenim tota la informació d’aquella variable.

Les paraules clau del PowerShell són molt poques comparades amb un llenguatge de programació. Per obtenir ajuda sobre qualsevol paraula clau només cal fer Get-Help del “about” adequat (per exemple **Get-Help about\_switch**). Si vols conèixer la llista completa d’ajudes “about” pots executar **Get-Help about\_\***.

**ATENCIÓ** Abans de poder utilitzar el **Get-Help** a les versions més noves, cal descarregar l’ajuda amb la comanda **Update-Help**. Però per instal·lar l’ajuda que es baixa on-line cal executar el powershell amb permisos d’administrador (igual com s’explica més avall quan parla de canviar la política de seguretat per a l’execució de scripts): per tant cal obrir el PowerShell des del “menú principal” → “Accessoris” → “PowerShell” → “Power Shell” (fer clic amb el botó dret del mouse) → i al menú contextual triar l’opció “Executar com a administrador”.

Abans de començar a treballar amb PowerShell cal tenir present que no s’executa en l’entorn de comandes “normal” del Windows (l’aplicació “cmd”) sinó que hi ha dues possibilitats per executar-lo:

- 1) L’entorn PowerShell (o consola) que és un entorn de comandes normal i corrent. Permet entrar comandes i executar-les en mode text. Cada consola té un abast o “scope” particular. Per exemple:



```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\perev> $PSVersionTable

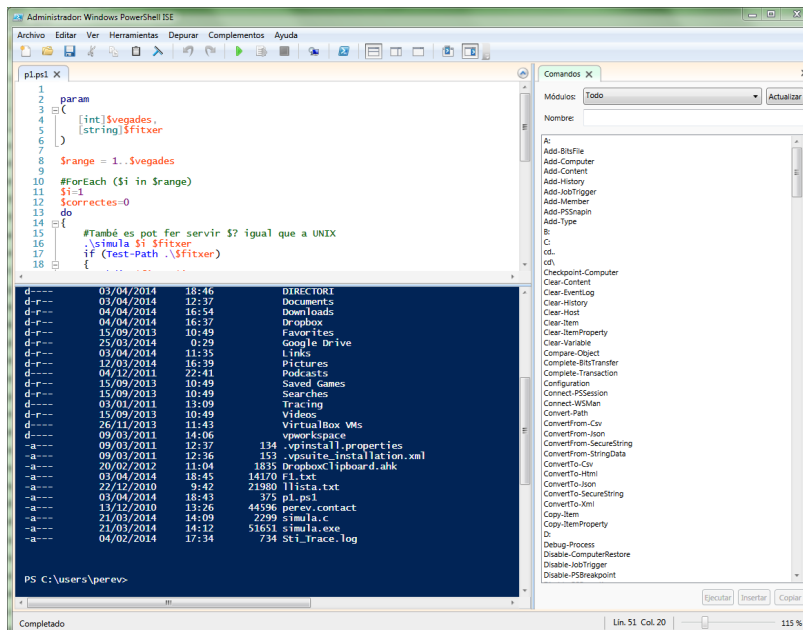
Name                           Value
----                           -
PSVersion                      4.0
VSMANStackVersion              3.0
SerializationVersion           1.1.0.1
CLRVersion                     4.0.30319.18444
BuildVersion                   6.3.9600.16406
PSCompatibleVersions           (1.0, 2.0, 3.0, 4.0)
PSRemotingProtocolVersion      2.2

PS C:\Users\perev> $psversiontable

Name                           Value
----                           -
PSVersion                      4.0
VSMANStackVersion              3.0
SerializationVersion           1.1.0.1
CLRVersion                     4.0.30319.18444
BuildVersion                   6.3.9600.16406
PSCompatibleVersions           (1.0, 2.0, 3.0, 4.0)
PSRemotingProtocolVersion      2.2

PS C:\Users\perev>
```

2) L'entorn de desenvolupament PowerShell ISE (Integrated Scripting Environment). Permet editar scripts i anar-los provant. A més té una ajuda integrada per tots els cmdlets i ajuda d'autoescriptura mentre es va escrivint. Es requereix tenir l'entorn gràfic instal·lat. En aquest entorn les execucions en fan en un abast o "scope" global. Per exemple:



```
1 param
2 {
3     [int]$vegades,
4     [string]$fitxer
5 }
6
7 $range = 1..$vegades
8
9
10 #ForEach ($i in $range)
11 $i=1
12 $correctes=0
13 do
14 {
15     #També es pot fer servir $? igual que a UNIX
16     \simula $i $fitxer
17     if (Test-Path .\$fitxer)
18     {
```

The screenshot shows the PowerShell ISE interface with a script editor on the left, a console window at the bottom showing the output of the script, and a command window on the right. The script defines a function 'param' with two parameters: '[int]\$vegades' and '[string]\$fitxer'. It then iterates through a range of values from 1 to \$vegades, and for each value, it runs the command '\simula \$i \$fitxer'. The console window shows the output of the script, which is a list of files and folders in the current directory, including 'DIRECTORI', 'Documents', 'Downloads', 'Dropbox', 'Favorites', 'Google Drive', 'Links', 'Pictures', 'Podcasts', 'Saved Games', 'Searches', 'Tracing', 'Videos', 'VirtualBox VMs', 'vpowerdspace', 'vpsuite\_installation.xml', '14170 Fl.txt', '21980 llista.txt', '376 pl.ps1', '44596 perev.contact', '2299 simula.c', '51631 simula.exe', and '734 Sti\_trace.log'. The command window on the right shows a list of commands and their descriptions, such as 'Add-Content', 'Clear-Content', 'Clear-EventLog', 'Clear-Host', 'Clear-Item', 'Clear-ItemProperty', 'Clear-Variable', 'Compare-Object', 'Complete-Transaction', 'Configuration', 'Connect-PSession', 'Connect-WiFi', 'Convert-Path', 'ConvertFrom-Csv', 'ConvertFrom-Json', 'ConvertFrom-SecureString', 'ConvertFrom-StringData', 'ConvertTo-Csv', 'ConvertTo-Html', 'ConvertTo-Json', 'ConvertTo-SecureString', 'ConvertTo-Xml', 'Copy-Item', 'Copy-ItemProperty', 'Debug-Process', 'Disable-ComputerRestore', 'Disable-JobTrigger', and 'Disable-PSBreakpoint'.

La darrera versió de PowerShell és la 7.2 (LTS) però és molt recent (2021). La majoria de llibres i documentació que es pot trobar fa referència a les versions 7.0, 6.2, 5.0, 4.0, 3.0 i anteriors. Consulta la versió instal·lada mostrant la variable **\$PSVersionTable**.

Podeu trobar l'ajuda (manual d'usuari, etc) oficial de Microsoft a aquí:

<https://docs.microsoft.com/en-us/powershell/>

També hi ha una bona comunitat de gent treballant amb PowerShell i fins i tot una revista:

<https://powershellmagazine.com/> → Revista

<https://poshcode.org/> → PowerShell Code Repository, Fòrum, xat, ...

<https://powershell.org/> → PowerShell Community

Per defecte en els entorns de PowerShell o PS-ISE **NO** es poden executar Scripts per una qüestió de seguretat. La política d'execució per defecte és "Restricted". Es pot consultar la política d'execució amb el cmdlet **Get-ExecutionPolicy** que pot valer el següent:

- Restricted → Does not load configuration files or run scripts. "Restricted" is the default execution policy
- AllSigned → Requires that all scripts and configuration files be signed by a trusted publisher, including scripts that you write on the local computer.
- RemoteSigned → Requires that all scripts and configuration files downloaded from the Internet be signed by a trusted publisher.
- Unrestricted → Loads all configuration files and runs all scripts. If you run an unsigned script that was downloaded from the Internet, you are prompted for permission before it runs.
- Bypass → Nothing is blocked and there are no warnings or prompts.
- Undefined → Removes the currently assigned execution policy from the current scope. This parameter will not remove an execution policy that is set in a Group Policy scope.

Per tant per poder executar els nostres propis scripts cal posar la política d'execució a "UnRestricted" amb la comanda **Set-ExecutionPolicy**. **ATENCIÓ**, això només es pot fer amb permisos d'administrador, per tant cal obrir el PowerShell des del "menú principal" → "Accessoris" → "PowerShell" → "Power Shell" (fer clic amb el botó dret del mouse) → i al menú contextual triar l'opció "Executar com a administrador". (PERILL: No us baixeu scripts que no sabeu què fan).

Els fitxers de script de PowerShell són fitxers de text normals i han de tenir l'extensió **".PS1"** (si no, no es poden executar). Per executar qualsevol shell o aplicació de PowerShell cal especificar el path (per exemple **.\test.ps1**). El resultat d'executar una comanda des de la línia de comandes o des de dins d'un script és el mateix però el format pot ser que sigui diferent.

Ús de punt i coma opcional. Es pot fer servir o no, però cal usar-lo si es posen varies comandes a la mateixa línia. Per exemple:

```
$s = "Hello";  
$s += " World"; $s += " !";  
$s;  
Hello World!
```

### Variables

Cal pensar en que una variable és com una capsa on s'hi poden posar varies coses, fins i tot coses de diferents tipus. El nom de la variable pot tenir fins i tot espais però no és recomanable. El nom de la variable fa referència a la "capsa" en si, i per accedir al seu contingut cal afegir al davant el caràcter "\$". El caràcter "\$" no forma part del nom de la variable.

```
$var = 'hello'  
$number = 1  
$numbers = 1,2,3,4,5,6,7,8,9  
${index de la taula} = 0
```

De fet la variable "numbers" és un array o una col·lecció d'ítems. "=" és l'operador d'assignació. Es pot utilitzar numbers[0] o numbers[1] per accedir al primer o al segon element o també numbers[-1] o numbers[-2] per accedir a l'últim o al penúltim element de numbers.

No cal declarar les variables ni els tipus, es fa de forma automàtica a l'utilitzar-les. De totes formes es pot fer si cal. Per exemple:

```
$a = "Hello"
$a #Prints Hello
$a = 1
$a += 1
$a #Prints 2
$a = 1,2,3,"a" #Create an array of different types
[int]$a = "Hello" #Error: Cannot convert to type "System.Int32"
```

#### Regles de cometes:

Normalment les cadenes van amb cometes senzilles, però també poden anar amb cometes dobles quan:

- 1) es vol incloure el valor d'una altra variable a la cadena:  

```
$name = 'Don'
$prompt = "My name is $name"
```
- 2) Utilitzar el caràcter d'escapament `"` (accent obert) per poder imprimir caràcters de "control" (`$` → per poder imprimir el caràcter `"$"` i ``` insertarà al text un tabulador):  

```
$debug = "El contingut de `$variable és: `t $variable"
```
- 3) Quan la cadena ha d'incloure cometes simples:  

```
$filter1 = "name='BITS'"
```

Atenció que l'accent obert és el caràcter d'escapament. Més exemples:

```
"A string"
A string
"A string with 'Quotes'"
A string with 'Quotes'
"A string with `Escaped Quotes`"
A string with "Escaped Quotes"
$s = "PowerShell"
"A string with a variable: $s"
A string with a variable: PowerShell
"A string with a quoted variable: '$s'"
A string with a quoted variable: 'PowerShell'
'Variables are not replaced inside single quotes: $s'
Variables are not replaced inside single quotes: $s
```

També existeix la possibilitat de definir cadenes (o textos) de més d'una línia, incloses variables i les dobles cometes no cal escaparles. Comencen amb `@` i acaben amb `@`

```
$name = "World"
$HereString = @"
This is a here-string
It can contain multiple lines
"Quotes don't need to be escaped"
Plus, you can include variables 'Hello $name'
"@

This is a here-string
It can contain multiple lines
"Quotes don't need to be escaped"
Plus, you can include variables 'Hello World'
```

### Atributs i mètodes d'un objecte

A PowerShell qualsevol cosa és un objecte. Un simple cadena “nom” és un objecte del tipus `System.String`. Utilitzant el cmdlet **Get-Member** es poden explorar els atributs i mètodes d'aquell objecte:

```
$var = 'Hello'
$var | Get-Member
```

S'accedeix als atributs i mètodes mitjançant un “.” (punt). Per exemple:

```
$svc = Get-Service
$svc[0].name
$name = $svc[1].name
$name.length
$name.ToUpper()
```

Les crides a les mètodes (no funcions o scripts) sempre han de dur els parèntesis, encara que no tinguin paràmetres.

### Parèntesis

A part d'utilitzar-los per les crides a mètodes, els parèntesis simplement indiquen (com el les matemàtiques) un ordre de precedència, o sigui, és el que s'ha d'executar primer. Per tant, si una cosa va entre parèntesis, primer s'executarà allò i tot el que hi havia entre parèntesis es substitueix pel seu resultat abans de continuar l'execució.

```
$name = (Get-Service)[0].name
Get-Service -computerName (Get-Content fitxer.txt)
```

### Abast (scope)

Un “scope” és tot l'entorn de variables i elements del sistema visibles o accessibles des d'un determinat entorn d'execució. PowerShell manté una jerarquia de “scopes” i hi ha una sèrie de regles per interactuar entre diferents “scopes”.

El shell per si mateix és un “scope” i s'anomena “global”. Quan s'executa un script, es crea un nou “scope” i el script s'executa allà dins. Qualsevol cosa que es crea dins del script queda dins d'aquest “scope” i no és accessible des de fora. Quan el script acaba, el “scope” i tot el que contenia desapareix. Com a excepció a això, els mòduls que es carreguen són globals i es mantenen carregats quan acaba el script que els ha carregat.

Quan en un “scope” s'intenta accedir a un objecte que no s'ha creat en aquell “scope”, llavors es busca si existeix al “scope” immediatament superior (o “scope” pare). En condicions normals no s'ha d'accedir a variables fora del “scope” actual però existeix una forma d'accedir al “scope” global. Cal afegir la modificació “global:” abans del nom de la variable \$global:variable.

### Arrays

Un array de PowerShell es correspon amb un `System.Array` de .NET. Interactuar amb arrays és molt senzill. Per exemple, per crear-los només cal posar els seus elements separats per comes:

```
$animals = "cat", "dog", "bat"
$animals.GetType()
IsPublic IsSerial      Name      BaseType
-----
True      True      Object[]  System.Array
$animals
cat
dog
bat
```

Per crear un array buit cal fer-ho amb l'expressió @(). Per exemple:

```
$animals = @()
$animals.Count
0
```

Es poden afegir elements a un array amb l'operador +=. Per exemple:

```
$animals = "cat", "dog", "bat"
$animals += "bird"
$animals
cat
dog
bat
bird
```

Per accedir a un element de l'array, es fa de la forma tradicional:

```
$animals = "cat", "dog", "bat"
$animals[1]
dog
$animals[0..1]
cat
dog
$animals[-1] # Get the last element
bat
```

Per buscar un element dins d'un array es poden fer servir els operadors de comparació:

```
$animals = "cat", "dog", "bat"
$animals -ne 'cat'
dog
bat
$animals = "cat", "dog", "bat"
$animals -like '*a*'
cat
bat
```

#### La sentència If

```
If ($this -eq $that) {
    # commands
} elseif ($those -ne $them) {
    # commands
} elseif ($we -gt $they) {
    # commands
} else {
    # commands
}
```

El valor 0 (zero) s'interpreta com a FALS i els valors diferents de zero com a TRUE. També existeixen les variables predeterminades \$True i \$False.

#### La sentència Do While

```
Do {
    # commands
} While ($this -eq $that)
```

També es pot posar el **While** al davant:

```
While (Test-Path $path) {
    # commands
}
```

En aquest cas no hi ha cap comparació ja que el cmdlet **Test-Path** ja retorna sempre o cert o fals.

### La sentència **ForEach**

Serveix per enumerar els objectes d'un array o col·lecció d'un en un. És molt semblant al cmdlet **ForEach-Object**.

```
$services = Get-Service
    ForEach ($service in $services) {
        $service.Stop()
    }
```

### Altres sentències

Podeu buscar a l'ajuda com funcionen altres sentències com ara el **Switch** o el **For** i també s'inclouen exemples interessants. Feu **Get-Help about\_switch | more** o redirigit a un fitxer per llegir-ho en detall. Feu també **Get-Help about\_For** i és altament recomanable que també feu **Get-Help about\_Comparison\_Operators** on s'explica i hi ha exemples de tots els operadors de comparació (n'hi ha de molt interessants):

-eq	-le	-Contains
-ne	-Like	-NotContains
-gt	-NotLike	-In
-ge	-Match	-NotIn
-lt	-NotMatch	-Replace

### Funcions

Les funcions són scripts que es poden cridar amb un nom. Les funcions només es poden cridar des del "scope" on han estat creades.

```
function One {
    function Two {
        Dir
    }
    Two # Des d'aquí es pot cridar la funció Two.
}
One
Two   # Aquesta crida provoca un error ja que la funció Two està
      # definida dins la funció One i des d'aquí no és visible.
```

Qualsevol cosa entre claus { } és un sub-shell o una funció (pot tenir nom o no) amb la paraula clau **function**. Per exemple:

```
$n = "PowerShell"
$closure = {"Hello $n"}
& $closure
Hello PowerShell

function Suma5 ($num) {
    $num + 5
}
Suma5 5
10
```



### Paràmetres de les funcions (o scripts)

Els paràmetres es defineixen amb la paraula clau **param** al començament del script o funció. És obligatori que sigui el primer codi del script o funció.

```
param (  
    [string]$computername,  
    [string]$logfile,  
    [int]$attemptcount = 5  
)
```

En aquest exemple es defineixen tres paràmetres. A dins del script s'utilitzen com qualsevol altre variable i poden prendre valors per defecte. Els paràmetres, si no s'indica el contrari, són opcionals. Un script o funció amb aquests paràmetres es pot cridar de moltes formes diferents. Aquests són exemples correctes de passar aquests paràmetres:

```
./test -computername SERVER  
./test -comp SERVER -log err.txt -attempt 2  
./test SERVER err.txt 2  
./test SERVER 2  
./test -log err.txt -attempt 2 -comp SERVER
```

No s'han d'utilitzar parèntesis en les crides a les funcions per passar els paràmetres, ja que ho agafaria com un **array**. Per exemple:

```
function Test ($p, $x) {  
    "This is the value of p $p and the value of x $x"  
}  
  
Test (1, 2)  
This is the value of p 1 2 and the value of x
```

La variable p pren el valor de l'array (1,2) i la variable x queda buida. Si es posen els paràmetres sense parèntesis llavors s'obté el resultat esperat.

```
Test 1 2  
This is the value of p 1 and the value of x 2
```

Es pot declarar que un paràmetre sigui especial, anomenat **Cmdlet Binding**. Entre altres casos es pot fer que sigui obligatori o que accepti l'entrada des d'una **pipe**. Veure **Get-Help about\_functions\_advanced\_parameters** per a més opcions.

```
[CmdletBinding()]  
param (  
    [Parameter(Mandatory=$True)]  
    [string]$computername,  
  
    [Parameter(Mandatory=$True)]  
    [string]$logfile,  
  
    [int]$attemptcount = 5  
)
```

### Llistat de sortida en una finestra gràfica

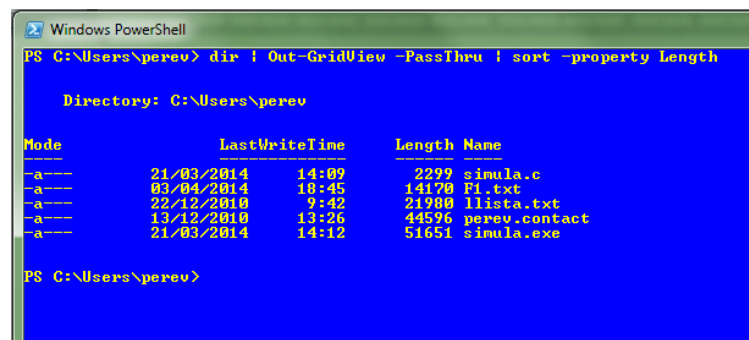
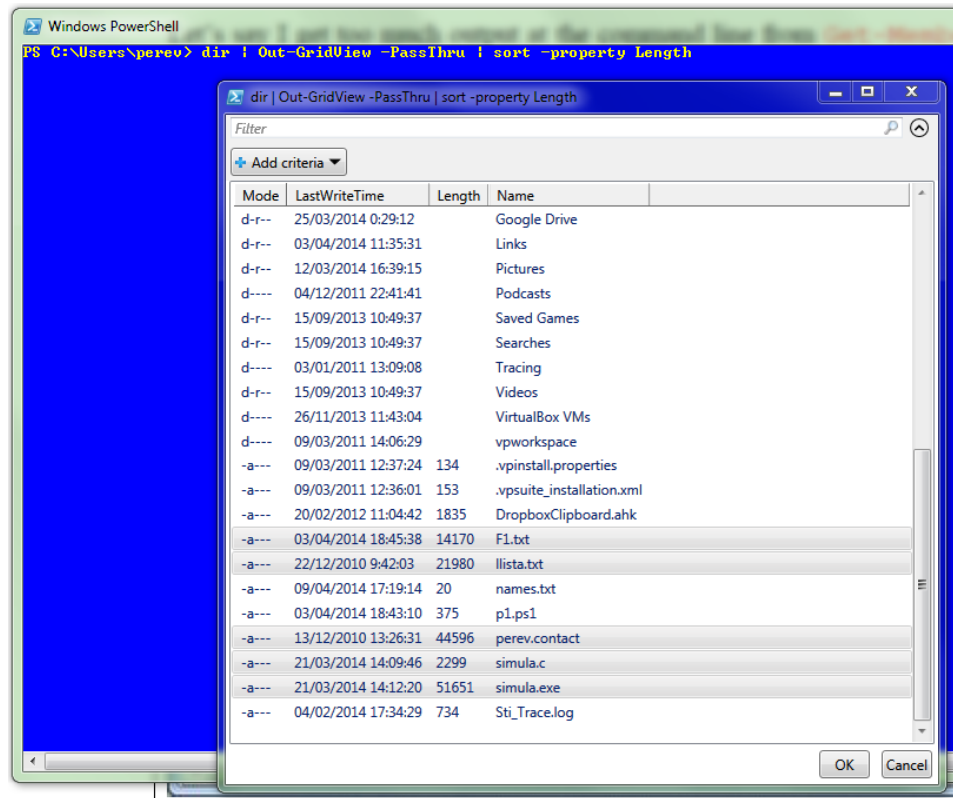
Qualsevol cmdlet que la seva sortida sigui un llistat, es pot fer que en lloc de sortir per la consola, surti per una finestra gràfica. Només cal fer una **pipe** amb **Out-GridView**.

```
dir | Out-GridView
```

Això genera el llistat de fitxers i carpetes però en una finestra gràfica on és possible ordenar per les diferents columnes, ascendent o descendent, etc. Fins i tot, amb l'opció **-PassThru** és possible seleccionar quelcom a la finestra gràfica i quan es prem el botó OK el resultat de la selecció es passa per la **pipe** a la següent comanda.

En el següent exemple (veure imatges) es mostra el contingut de la carpeta actual en mode gràfic, es seleccionen 5 fitxers i quan es prem OK es passen a la comanda sort que els ordena per mida.

```
dir | Out-GridView -PassThru | sort -property Length
```



En aquesta darrera comanda s'ha utilitzat **dir** i **sort** que no són cmdlets. De fet són àlies de cmdlets. Hi ha una gran quantitat d'àlies predefinits per tal de poder utilitzar, si hi estàs acostumat, les antigues comandes de MS-DOS (dir, ren, copy, cd, cls, del, ...) i també moltes de les comandes de UNIX (ls, mv, echo, grep, cp, clear, pwd, rm, ...). Tots aquests àlies es poden consultar amb el cmd-let **Get-Alias**.

Si detectes errors o coses per afegir o millorar, si et plau fes-nos arribar els comentaris en un correu electrònic. Gràcies.