

SUPSI

Ambienti Operativi: Windows Powershell

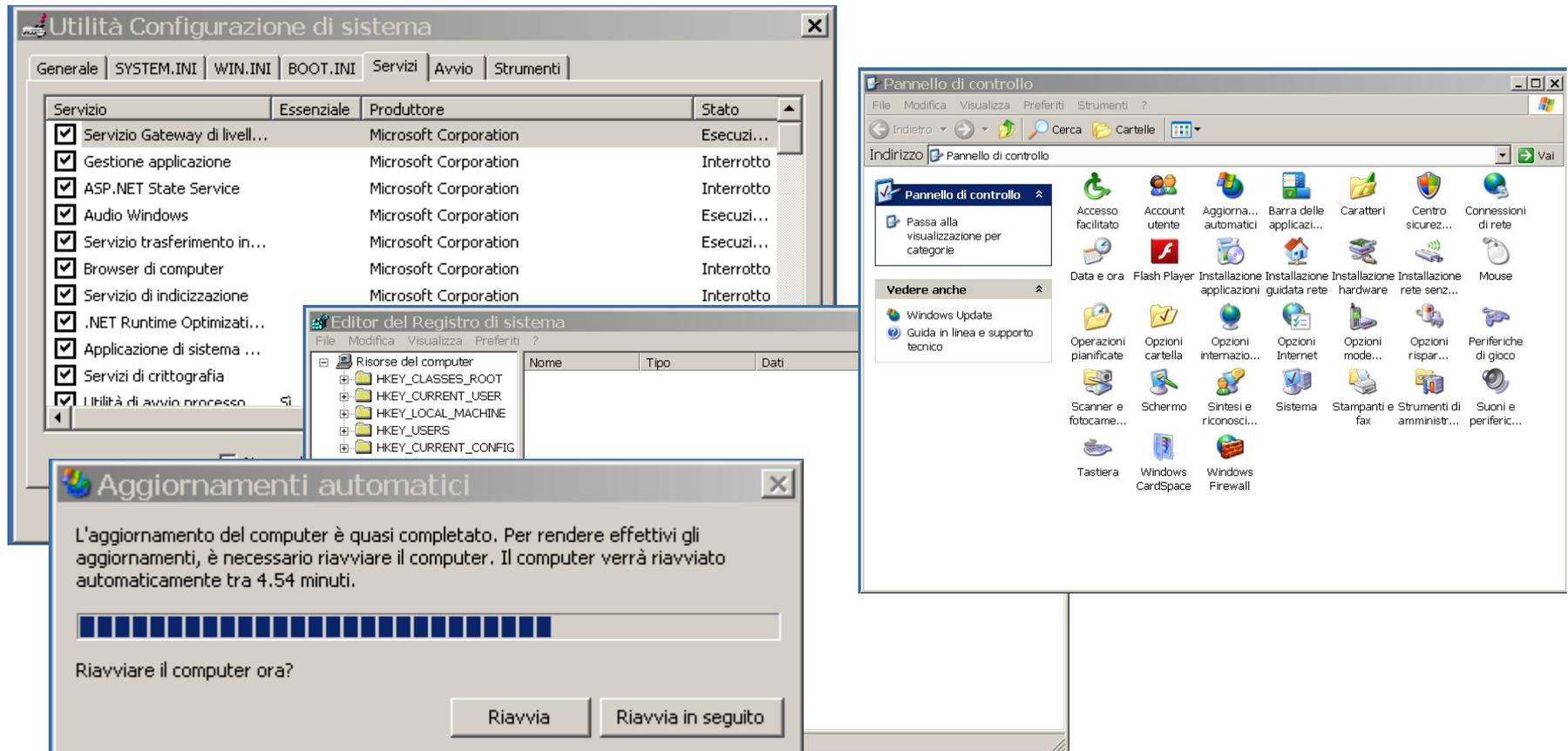
Amos Brocco, Ricercatore, DTI / ISIN

Amministrazione di un sistema Windows

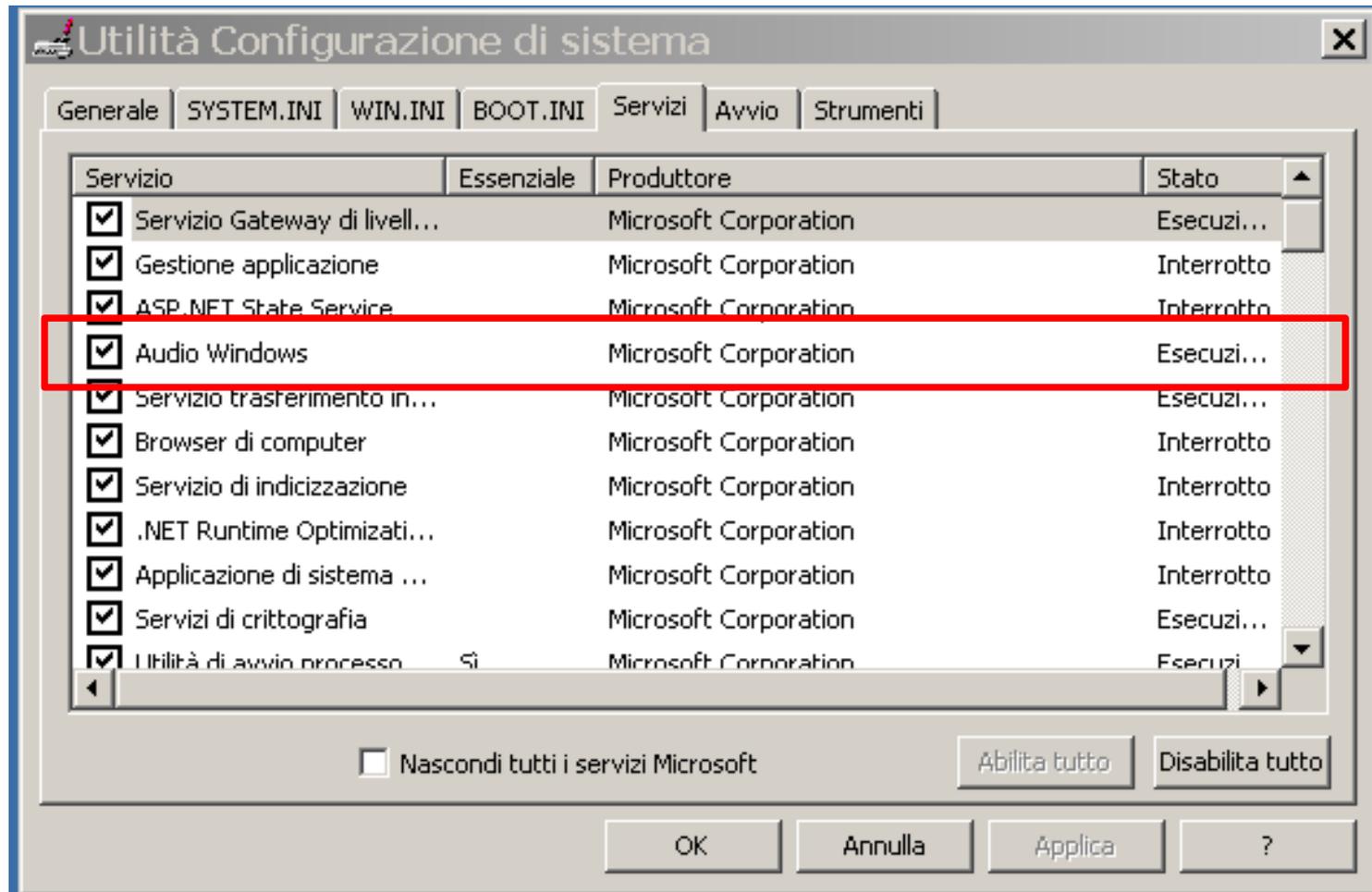
- Gestire gli utenti e i permessi
- Modificare dei file o la configurazione di un applicazione
- Copiare / Spostare / Cancellare dei files
- Modificare il registro
- Installare nuove applicazioni o driver
- Aggiornare il sistema
- Configurare i servizi in esecuzione

Amministrazione di un sistema Windows

- ...tipicamente un utente Windows utilizza dei tool grafici



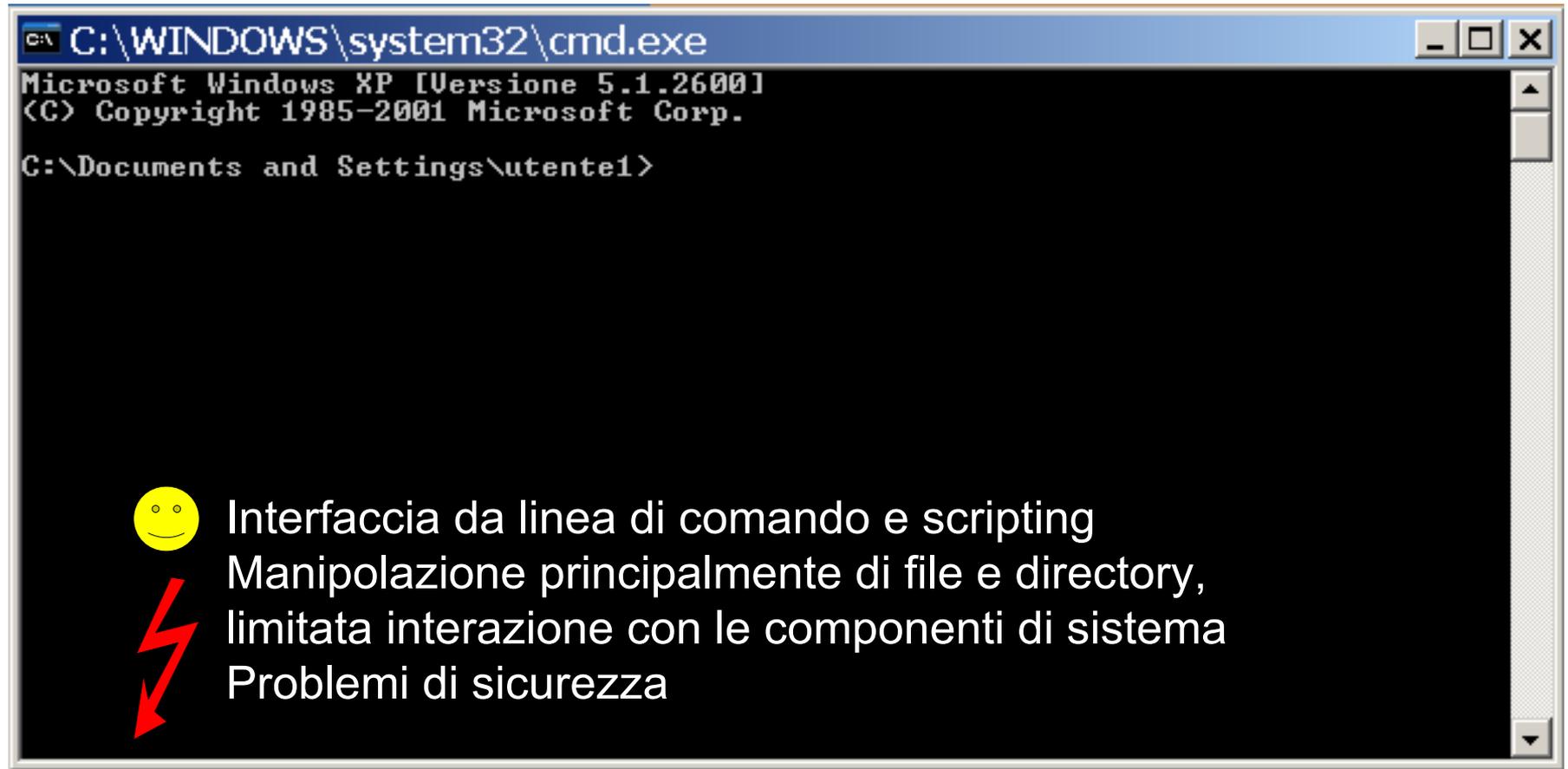
Esempio: disabilitare il server audio



Limiti delle interfacce grafiche

- Interfaccia grafica (GUI)
 - 😊 va bene per configurare un solo PC e per operazioni da eseguire una volta sola
 - ⚡ non ottimale per
 - configurare decine o centinaia di PC, in remoto
 - automatizzare operazioni simili e ripetitive

cmd.exe



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\utente1>
```



Interfaccia da linea di comando e scripting
Manipolazione principalmente di file e directory,
limitata interazione con le componenti di sistema
Problemi di sicurezza

Windows Script Host (WSH)

- Disponibile a partire da Windows 98
- Supporta più linguaggi (Windows 2000+): VBScript (Visual Basic Script), Javascript, Perl,...



Permette di manipolare gli oggetti di sistema



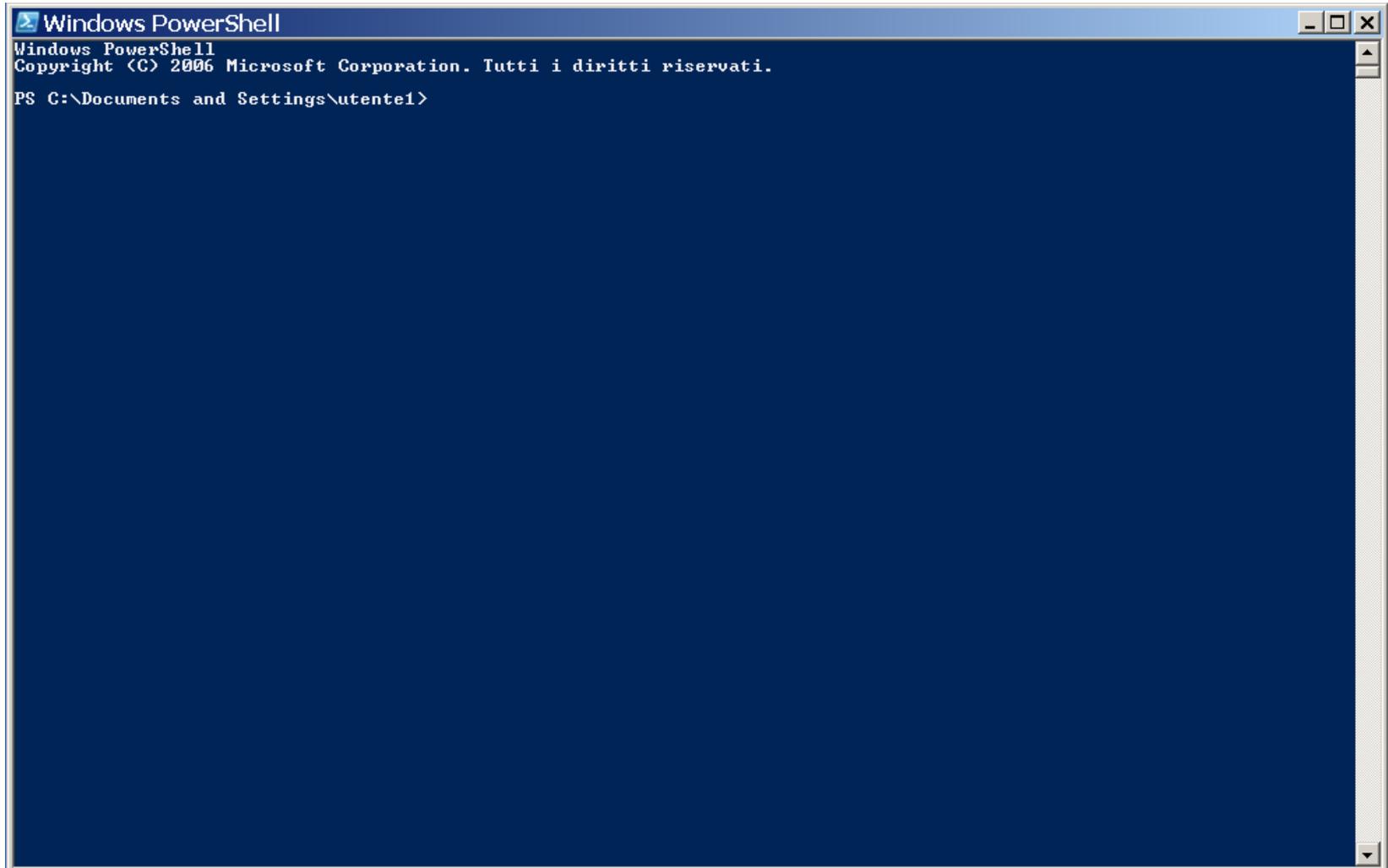
Progettato per lo scripting, non come interfaccia da linea di comando

Problemi di sicurezza

- Esempio:

```
<job>
<script language="VBScript">
  MsgBox "Ciao mondo"
</script>
<script language="JScript">
  WSH.echo("Ciao mondo");
</script>
</job>
```

PowerShell

A screenshot of a Windows PowerShell console window. The window title is "Windows PowerShell". The text inside the window reads: "Windows PowerShell", "Copyright (C) 2006 Microsoft Corporation. Tutti i diritti riservati.", and "PS C:\Documents and Settings\utente1>". The window has a dark blue background and a light blue title bar with standard Windows window controls (minimize, maximize, close) in the top right corner. There is also a vertical scrollbar on the right side of the window.

```
Windows PowerShell
Copyright (C) 2006 Microsoft Corporation. Tutti i diritti riservati.
PS C:\Documents and Settings\utente1>
```

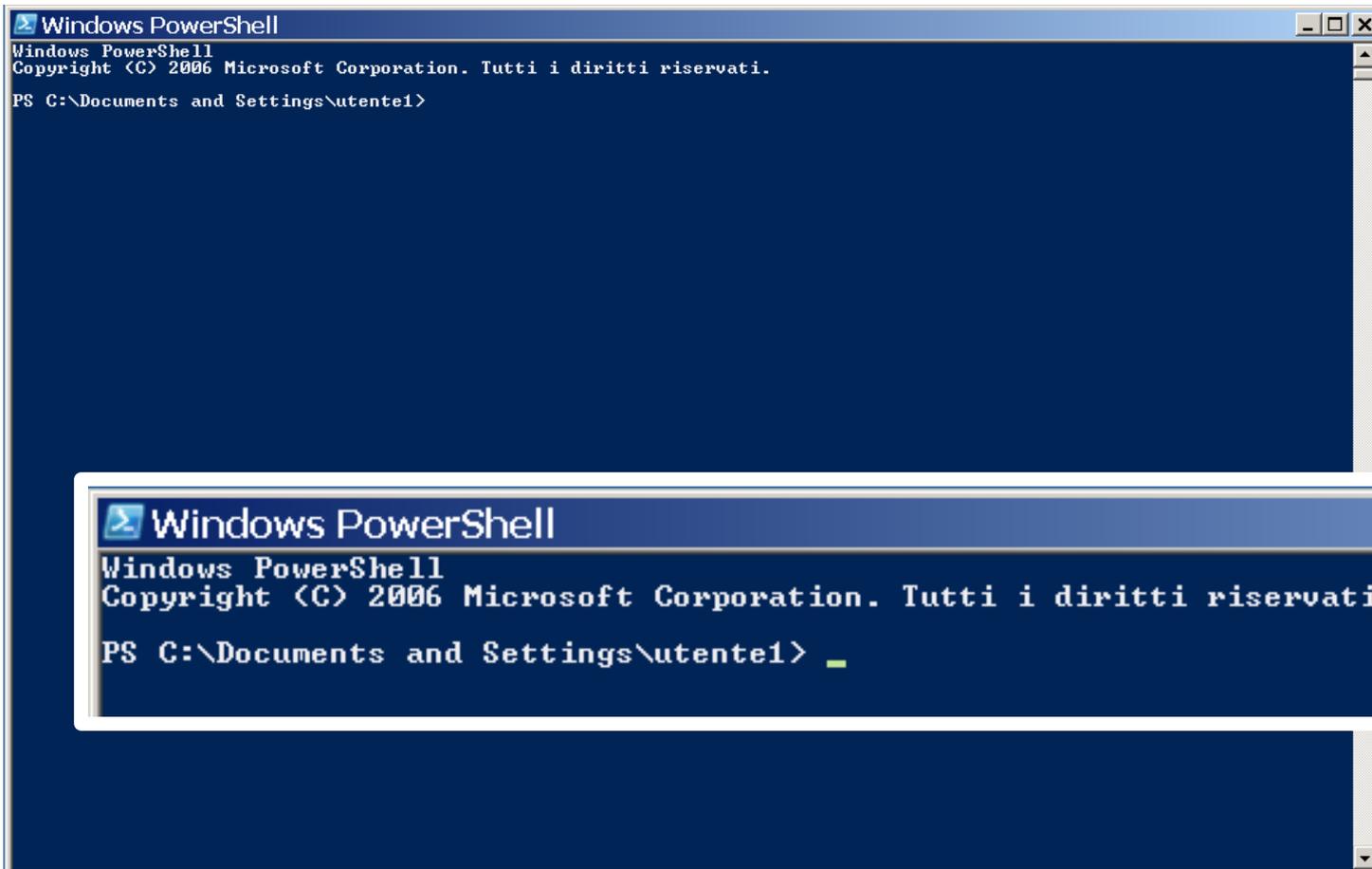
Cos'è Windows Powershell?

- Ambiente di scripting avanzato basato sulla piattaforma .NET
 - Dispone di un **interfaccia da linea di comando** e supporta lo **scripting**
 - Ispirato alle shell UNIX
 - Possibilità di interagire nativamente con le componenti di sistema di Windows (o le applicazioni)
 - Automazione
 - Elaborazione dei dati avanzata

Cos'è Windows Powershell?

- Sicuro
 - possibilità di impedire l'esecuzione di script la cui origine non è verificata (dall'amministratore di sistema)
- Orientato ad oggetti
 - I comandi (chiamati **cmdlet**, *commandlet*), invece di ritornare flussi di testo (come in Bash), ritornano oggetti

Il prompt dei comandi



The image shows a Windows PowerShell console window. The title bar reads "Windows PowerShell". The main content area displays the following text:

```
Windows PowerShell  
Copyright (C) 2006 Microsoft Corporation. Tutti i diritti riservati.  
PS C:\Documents and Settings\utente1>
```

A smaller, semi-transparent inset window is overlaid on the bottom left of the main window, showing the same text but with a green cursor at the end of the prompt:

```
Windows PowerShell  
Windows PowerShell  
Copyright (C) 2006 Microsoft Corporation. Tutti i diritti riservati.  
PS C:\Documents and Settings\utente1> _
```

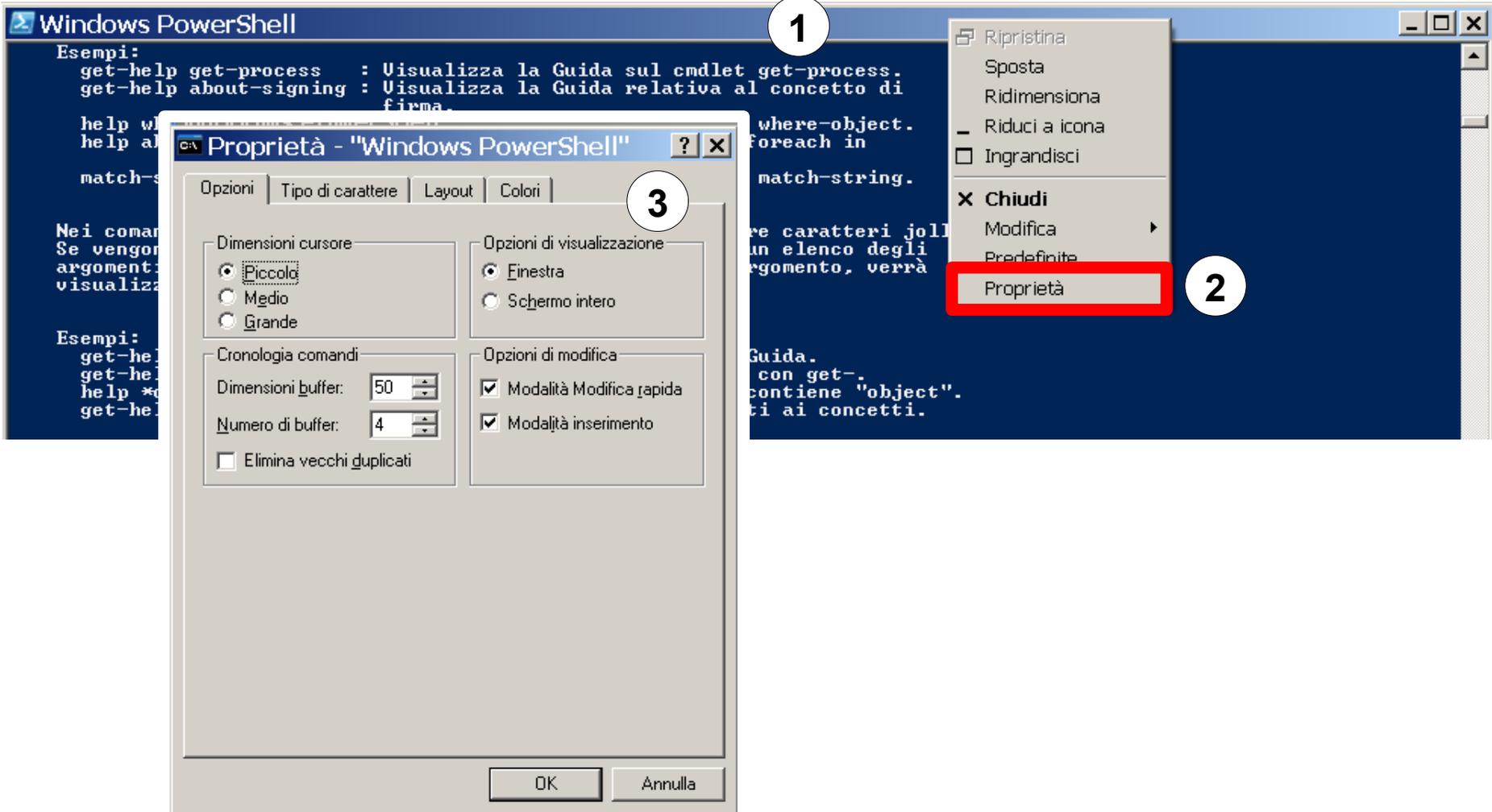
Disabilitare il server audio (con PowerShell)

DisabilitaServizi.ps1

```
$computer = get-content c:\listapc.txt
foreach($pc in $computer) {
    $servizio = Get-WmiObject Win32_Service -ComputerName $pc -Filter "name='AudioSrv'"
    if ($servizio.started -eq $true) {
        $servizio.StopService()
    }
}
```

Cambiare le impostazioni della shell

click con tasto destro



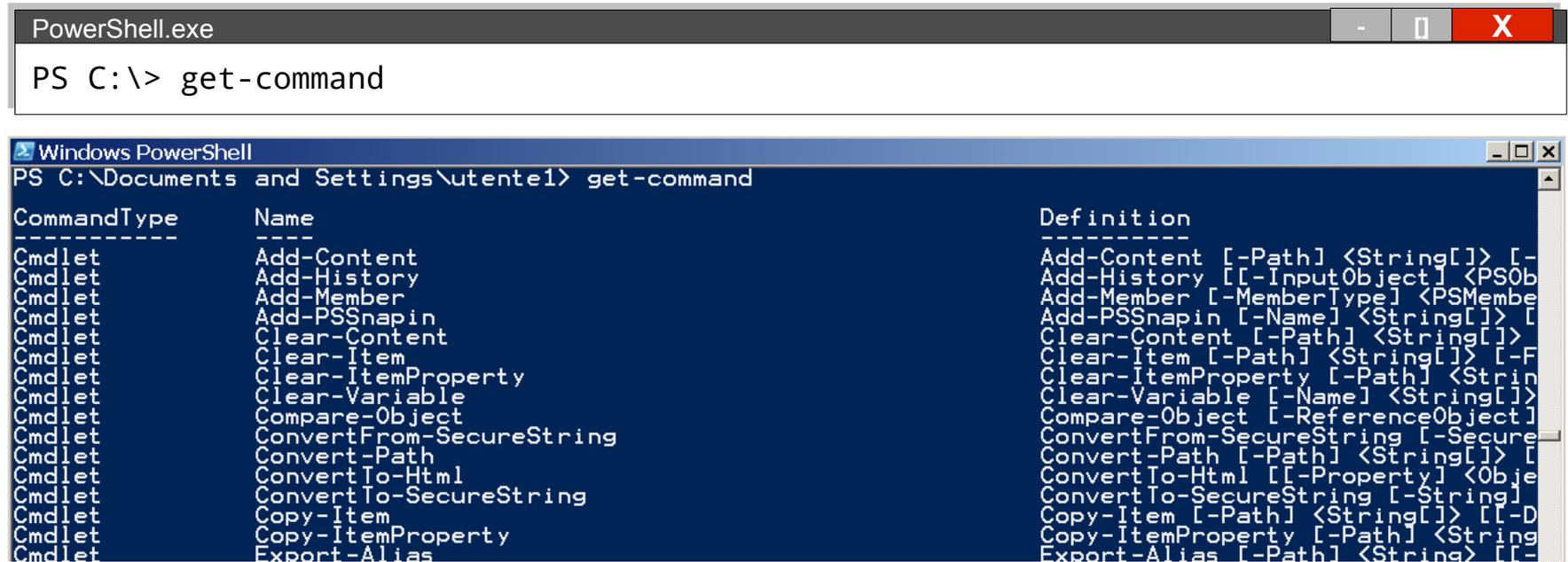
Comandi

- Ogni comando è case-insensitive (può essere scritto maiuscolo o minuscolo) e ha la forma
 - **verbo-nome**
 - Esempi:
 - get-help
 - get-process
 - get-member
 - get-childitem
 - i parametri iniziano con -

Autocompletamento

- Con il tasto *TAB* posso richiedere l'autocompletamento della linea di comando corrente
 - completamento dei percorsi
 - completamento dei nomi dei comandi

Lista dei comandi disponibili



The image shows two overlapping PowerShell windows. The top window, titled 'PowerShell.exe', shows the command 'PS C:\> get-command' entered. The bottom window, titled 'Windows PowerShell', shows the same command executed, resulting in a table of available commands.

```
PS C:\> get-command
```

CommandType	Name	Definition
Cmdlet	Add-Content	Add-Content [-Path] <String[]> [-
Cmdlet	Add-History	Add-History [[-InputObject] <PSOb
Cmdlet	Add-Member	Add-Member [-MemberType] <PSMembe
Cmdlet	Add-PSSnapin	Add-PSSnapin [-Name] <String[]> [
Cmdlet	Clear-Content	Clear-Content [-Path] <String[]>
Cmdlet	Clear-Item	Clear-Item [-Path] <String[]> [-F
Cmdlet	Clear-ItemProperty	Clear-ItemProperty [-Path] <Strin
Cmdlet	Clear-Variable	Clear-Variable [-Name] <String[]>
Cmdlet	Compare-Object	Compare-Object [-ReferenceObject]
Cmdlet	ConvertFrom-SecureString	ConvertFrom-SecureString [-Secure
Cmdlet	Convert-Path	Convert-Path [-Path] <String[]> [
Cmdlet	ConvertTo-Html	ConvertTo-Html [[-Property] <Obje
Cmdlet	ConvertTo-SecureString	ConvertTo-SecureString [-String]
Cmdlet	Copy-Item	Copy-Item [-Path] <String[]> [[-D
Cmdlet	Copy-ItemProperty	Copy-ItemProperty [-Path] <String
Cmdlet	Export-Alias	Export-Alias [-Path] <String> [[-

Alias

- Sono disponibili delle scorciatoie (alias) per i comandi più utilizzati
 - **get-alias** per vedere gli alias attuali
 - **new-alias** per definire un nuovo alias

```
Windows PowerShell
PS C:\Documents and Settings\utente1> get-alias

CommandType Name Definition
-----
Alias ac Add-Content
Alias asnp Add-PSSnapin
Alias clc Clear-Content
Alias cli Clear-Item
Alias clp Clear-ItemProperty
Alias clv Clear-Variable
Alias cpi Copy-Item
Alias cpp Copy-ItemProperty
Alias cvpa Convert-Path
```

alias

cmdlet corrispondente

Esempi di alias

```
PS C:\Documents and Settings\utente1> get-alias ls
```

CommandType	Name	Definition
Alias	ls	Get-ChildItem

```
PS C:\Documents and Settings\utente1> get-alias cp
```

CommandType	Name	Definition
Alias	cp	Copy-Item

```
PS C:\Documents and Settings\utente1> get-alias rm
```

CommandType	Name	Definition
Alias	rm	Remove-Item

Ottenere aiuto: **get-help**



Equivale a 'man' di Unix/Linux

```
Windows PowerShell

PS C:\Documents and Settings\utente1> get-help
ARGOMENTO
    Get-Help

DESCRIZIONE BREVE
    Visualizza la Guida sui cmdlet e i concetti relativi a PowerShell.

DESCRIZIONE DETTAGLIATA

SINTASSI
    get-help {<NomeCmdlet> | <NomeArgomento>}
    help {<NomeCmdlet> | <NomeArgomento>}
    <NomeCmdlet> -?

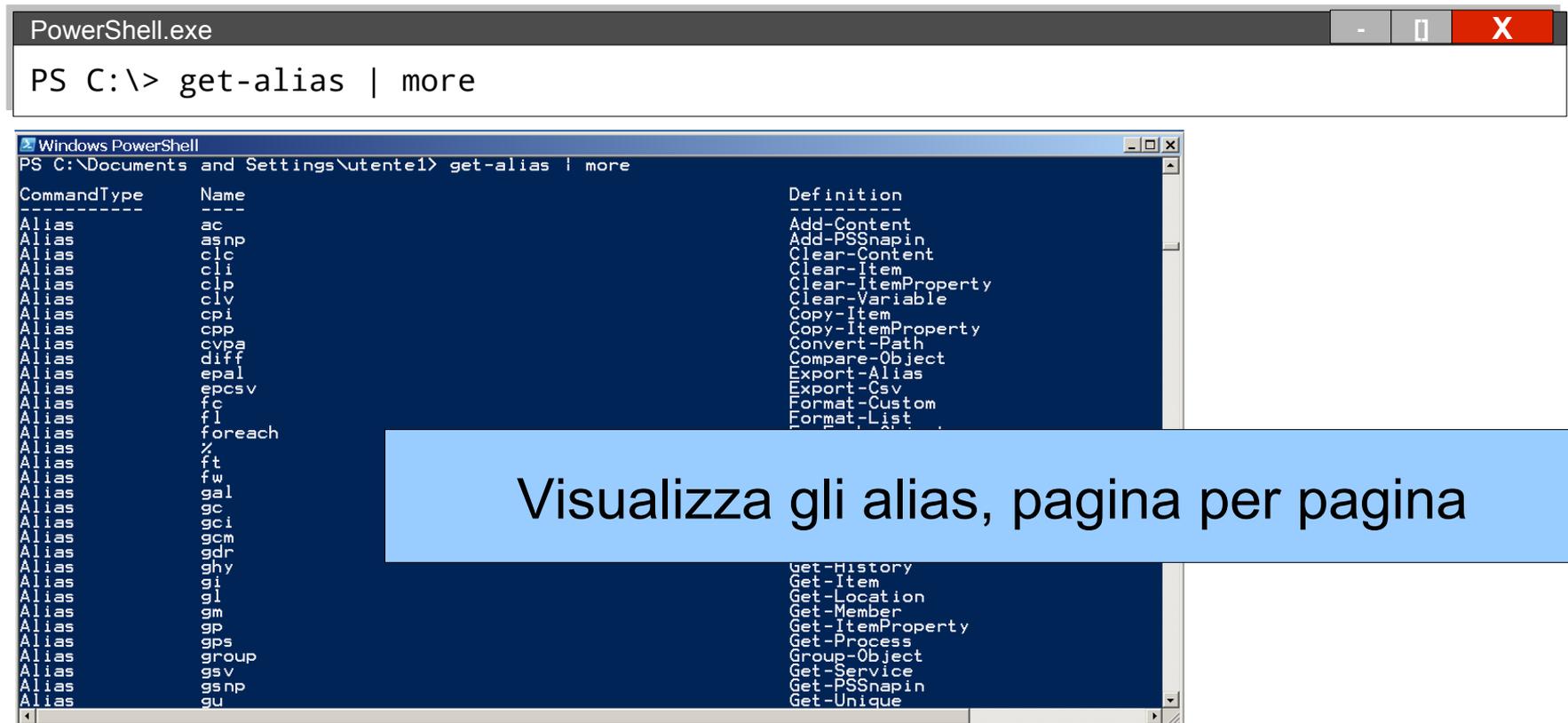
I comandi "Get-help" e "-?" consentono di visualizzare le informazioni della
Guida in un'unica pagina.
Il comando "help" consente di visualizzare la Guida in più pagine.

Esempi:
    get-help get-help
    get-help about_arrays
```

Visualizza l'aiuto per i comandi della shell

Pipe

- Come in Bash, con il carattere **|** (pipe, **Alt Gr + 7**) posso concatenare più comandi e redirigere l'output



```
PowerShell.exe
PS C:\> get-alias | more

Windows PowerShell
PS C:\Documents and Settings\utente1> get-alias | more

CommandType      Name                Definition
-----
Alias             ac                  Add-Content
Alias             asnp               Add-PSSnapin
Alias             clc                Clear-Content
Alias             cli                Clear-Item
Alias             clp                Clear-ItemProperty
Alias             clv                Clear-Variable
Alias             cpi                Copy-Item
Alias             cpp                Copy-ItemProperty
Alias             cvpa               Convert-Path
Alias             diff               Compare-Object
Alias             epal               Export-Alias
Alias             epcsv              Export-Csv
Alias             fc                 Format-Custom
Alias             fl                 Format-List
Alias             foreach
Alias             %
Alias             ft                 Get-Content
Alias             fw                 Get-File
Alias             gal                 Get-AppLockerPolicy
Alias             gc                 Get-Content
Alias             gci                 Get-ChildItem
Alias             gcm                 Get-Command
Alias             gdr                 Get-Date
Alias             ghy                 Get-Event
Alias             gi                 Get-EventLog
Alias             gl                 Get-EventLogEntry
Alias             gm                 Get-EventLogEntryType
Alias             gp                 Get-GroupPolicy
Alias             gps                 Get-Process
Alias             group              Group-Object
Alias             gsv                 Get-Service
Alias             gsnv               Get-PSSnapin
Alias             gu                 Get-Unique
```

Visualizza gli alias, pagina per pagina

Redirezione dell'output

- Possiamo anche redirigere l'output dei comandi su un file

>	Redirige l'output nel file specificato. Se il file esiste già il suo contenuto viene sovrascritto
>>	Redirige l'output nel file specificato. Se il file esiste già aggiunge il nuovo contenuto alla fine
2>	Redirige gli errori nel file specificato. Se il file esiste già il suo contenuto viene sovrascritto
2>>	Redirige gli errori nel file specificato. Se il file esiste già aggiunge il nuovo contenuto alla fine
2>&1	Redirige gli errori nello standard output

Altri modi per redirigere l'output

- Oppure utilizzare i comandi **out-...** per scrivere l'output di un comando in un file o una stampante
 - fare riferimento alle pagine di aiuto con **get-help out***



```
Windows PowerShell
PS C:\Documents and Settings\utente1> get-help out-*

Name                Category          Synopsis
----                -
Out-Null            Cmdlet            Elimina l'output anziché inviarlo al...
Out-Default         Cmdlet            Invia l'output al formattatore e al ...
Out-Host            Cmdlet            Invia l'output alla riga di comando.
Out-File            Cmdlet            Invia l'output a un file.
Out-Printer        Cmdlet            Invia l'output a una stampante.
Out-String          Cmdlet            Invia gli oggetti all'host come seri...
```

Redirezione dell'input



- Non possiamo usare il carattere `<` per redirigere l'input

- Alternative

- **echo**

```
Windows PowerShell
PS C:\Documents and Settings\utente1> echo "Ciao Mondo" | sort-object
Ciao Mondo
PS C:\Documents and Settings\utente1> _
```

- **get-content**

```
Seleziona Windows PowerShell
PS C:\Documents and Settings\utente1> Get-Content "C:\Documents and Settings\utente1\Desktop\Testo.txt" | sort-object
Alberto
Aldo
Giacomo
Giovanni
Paolino
PS C:\Documents and Settings\utente1>
```

Esportare l'output

- Con i comandi **export-...** posso scrivere l'output di un comando in un file formattandolo
 - CSV (**Export-Csv**)
 - XML (**Export-Clixml**)
 - ...
- Fare riferimento alle pagine di aiuto con **get-help export-***

```
PS C:\Documents and Settings\utente1> get-help export-*
```

Name	Category	Synopsis
Export-Console	Cmdlet	Esporta la configurazione della cons...
Export-Csv	Cmdlet	Crea un file con valori delimitati d...
Export-Alias	Cmdlet	Esporta in un file informazioni sugl...
Export-Clixml	Cmdlet	Crea una rappresentazione basata su ...

Un primo esempio...

Voglio ottenere la lista dei processi
(programmi) in esecuzione,
ordinata secondo l'utilizzo della CPU...

`get-process`

`sort-object`

Soluzione



Equivale a 'ps' di Unix/Linux

PowerShell.exe

```
PS C:\> get-process | sort-object CPU
```

Windows PowerShell

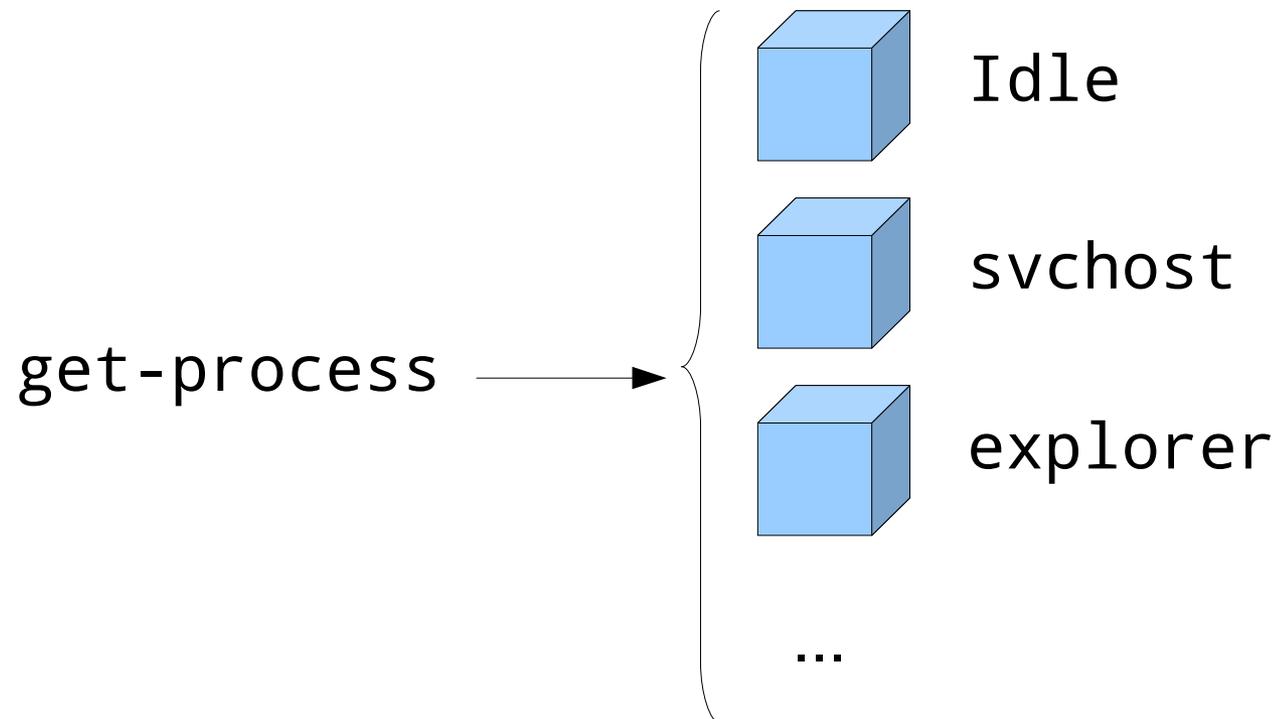
```
PS C:\Documents and Settings\utente1> get-process | sort-object CPU
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
0	0	0	16	0		0	Idle
107	0	1276	3756	36	0.01	488	svchost
170	0	1492	3904	34	0.01	1324	svchost
106	0	936	3292	25	0.02	824	VBoxService
117	0	2204	4060	36	0.03	3068	wuaucnt
55	0	784	2720	32	0.03	1880	VBoxTray
77	0	912	3128	30	0.03	1904	ctfmon
108	0	1140	3612	32	0.03	2272	alg
120	0	3060	4812	41	0.05	1732	spoolsv
201	0	2996	4888	60	0.07	868	svchost
19	0	172	424	4	0.07	372	smss
73	0	1232	3516	29	0.07	1160	svchost
238	1	1648	4196	34	0.21	956	svchost
241	0	5408	10052	58	0.21	1888	msseces
352	0	3824	6404	41	0.32	668	lsass
504	4	8132	6936	61	0.39	612	winlogon
261	0	1612	3432	20	0.66	656	services
365	1	9276	16844	90	0.92	1624	explorer
364	8	32060	30272	145	1.23	2260	powershell
374	0	2476	4332	24	2.32	588	csrss
262	0	0	212	2	2.76	4	System
1400	34	23476	34484	156	5.90	1088	svchost
344	14	88460	49332	148	8.91	1048	MsMpEng

```
PS C:\Documents and Settings\utente1>
```

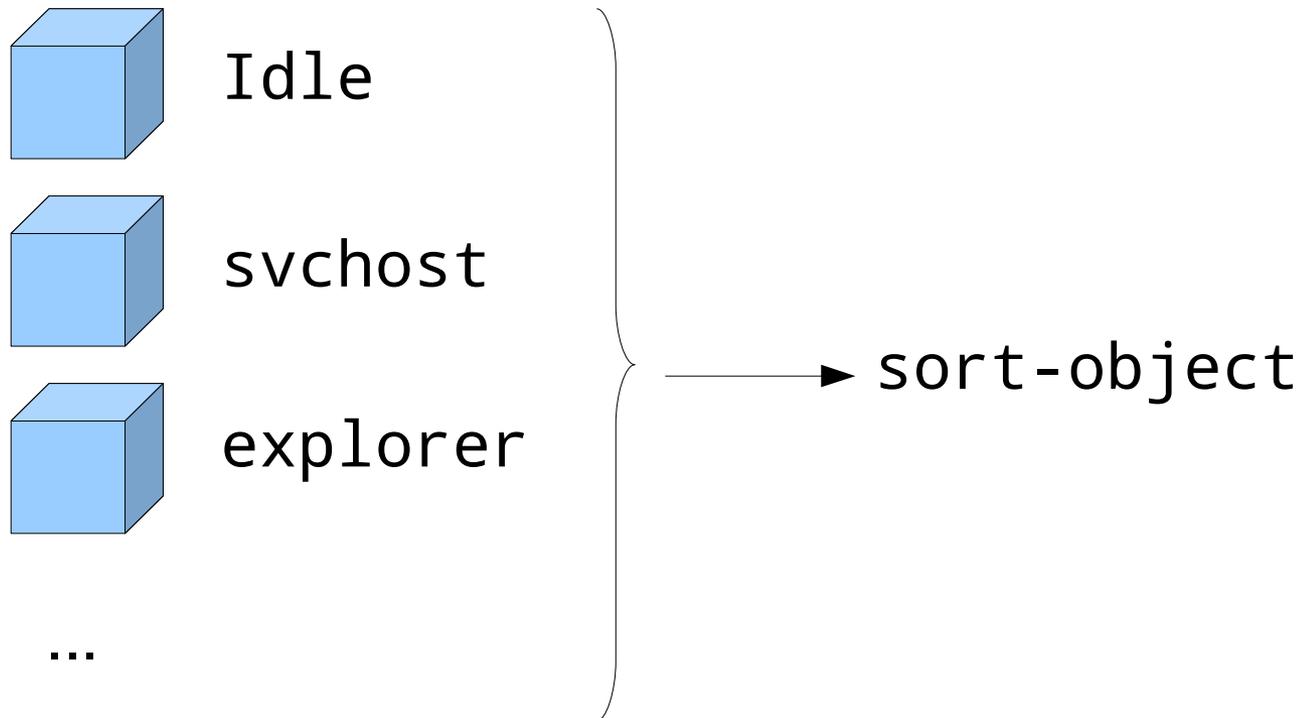
Oggetti e PowerShell

- A differenza di Bash, l'output di un comando non è una sequenza (stream) di caratteri ma di oggetti



Oggetti come input

- Allo stesso modo, un cmdlet si aspetta uno o più oggetti come input

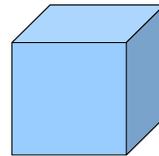


Oggetti

- Ogni oggetto dispone di
 - **proprietà** (es. nome, colore, numero di elementi,...)
 - **metodi** (cosa possiamo chiedere di fare all'oggetto, es. scrivi sul terminale, corri, abbaia,...)

Proprietà:

nome
colore
larghezza
altezza
...



Metodi:

apri
chiudi
...

Proprietà e membri di un oggetto

- Con **get-member** posso visualizzare le proprietà e i metodi di un oggetto (o più oggetti dello stesso tipo)
 - il parametro **-type** permette di specificare quale informazione vogliamo
 - **property**
 - **method**

```
PS C:\Documents and Settings\utente1> get-process | get-member -type property
```

```
TypeName: System.Diagnostics.Process

Name      MemberType Definition
-----
BasePriority Property System.Int32 BasePriority {get;}
Container Property System.ComponentModel.IContainer Container {get;}
EnableRaisingEvents Property System.Boolean EnableRaisingEvents {get;set;}
ExitCode Property System.Int32 ExitCode {get;}
ExitTime Property System.DateTime ExitTime {get;}
Handle Property System.IntPtr Handle {get;}
HandleCount Property System.Int32 HandleCount {get;}
```

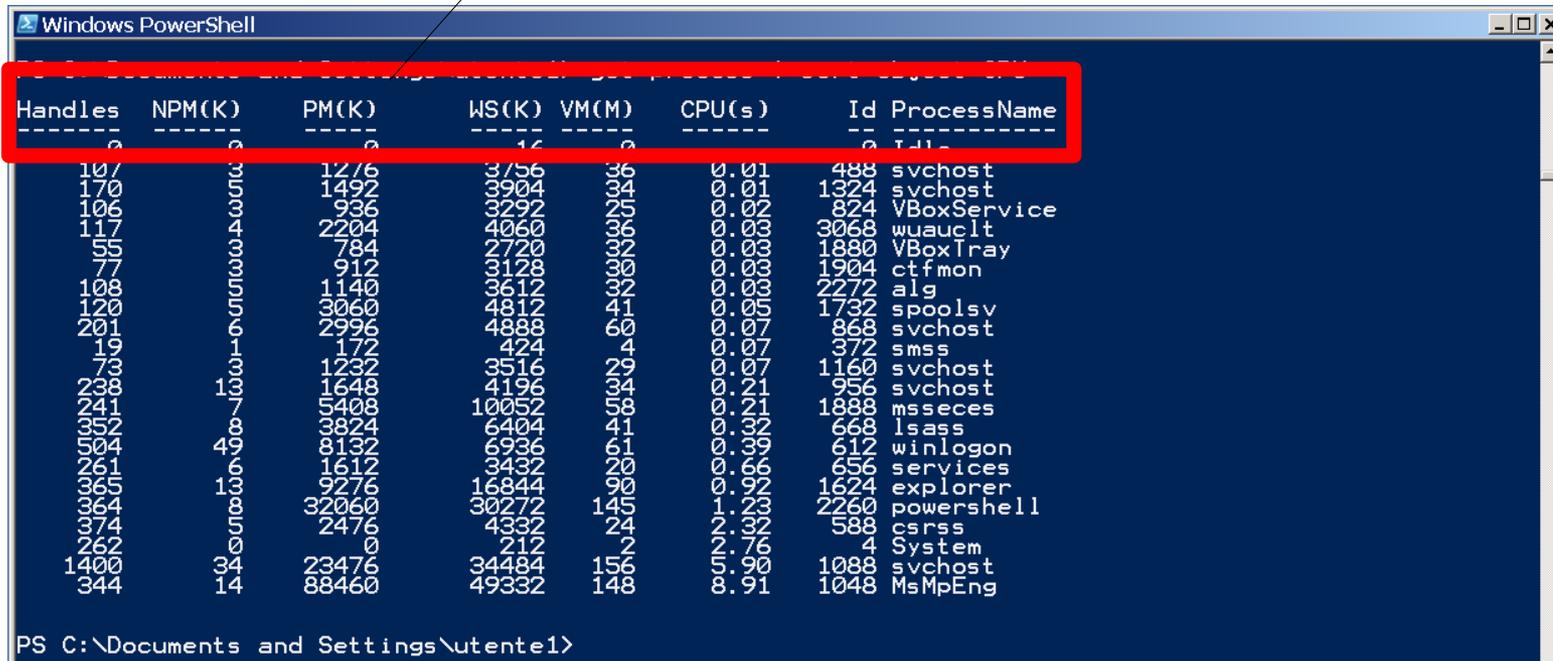
```
PS C:\Documents and Settings\utente1> get-process | get-member -type method
```

```
TypeName: System.Diagnostics.Process

Name      MemberType Definition
-----
add_Disposed Method System.Void add_Disposed(EventHandler value)
add_ErrorDataReceived Method System.Void add_ErrorDataReceived(DataReceivedEventHandler value)
add_Exited Method System.Void add_Exited(EventHandler value)
add_OutputDataReceived Method System.Void add_OutputDataReceived(DataReceivedEventHandler value)
BeginErrorReadLine Method System.Void BeginErrorReadLine()
BeginOutputReadLine Method System.Void BeginOutputReadLine()
CancelErrorRead Method System.Void CancelErrorRead()
CancelOutputRead Method System.Void CancelOutputRead()
Close Method System.Void Close()
```

Proprietà degli oggetti ritornati da **get-process**

Proprietà degli oggetti
Process



```
Windows PowerShell

Handles      NPM(K)      PM(K)      WS(K)      VM(M)      CPU(s)      Id ProcessName
-----
0           0           0           16          0           0.00        0 Idle
107         100         1276        3756        36          0.01        488 svchost
170         1492        1492        3904        34          0.01        1324 svchost
106         936         936         3292        25          0.02        824 VBoxService
117         2204        2204        4060        36          0.03        3068 wuauclt
55          784         784         2720        32          0.03        1880 VBoxTray
77          912         912         3128        30          0.03        1904 ctfmon
108         1140        1140        3612        32          0.03        2272 alg
120         3060        3060        4812        41          0.05        1732 spoolsv
201         2996        2996        4888        60          0.07        868 svchost
19          172         172         424         4          0.07        372 smss
73          1232        1232        3516        29          0.07        1160 svchost
238         1648        1648        4196        34          0.21        956 svchost
241         5408        5408        10052       58          0.21        1888 mssecex
352         3824        3824        6404        41          0.32        668 lsass
504         8132        8132        6936        61          0.39        612 winlogon
261         1612        1612        3432        20          0.66        656 services
365         9276        9276        16844       90          0.92        1624 explorer
364         32060       32060       30272       145         1.23        2260 powershell
374         2476        2476        4332        24          2.32        588 csrss
262         0           0           212         2          2.76        4 System
1400        34          23476       34484       156         5.90        1088 svchost
344         14          88460       49332       148         8.91        1048 MsMpEng

PS C:\Documents and Settings\utente1>
```

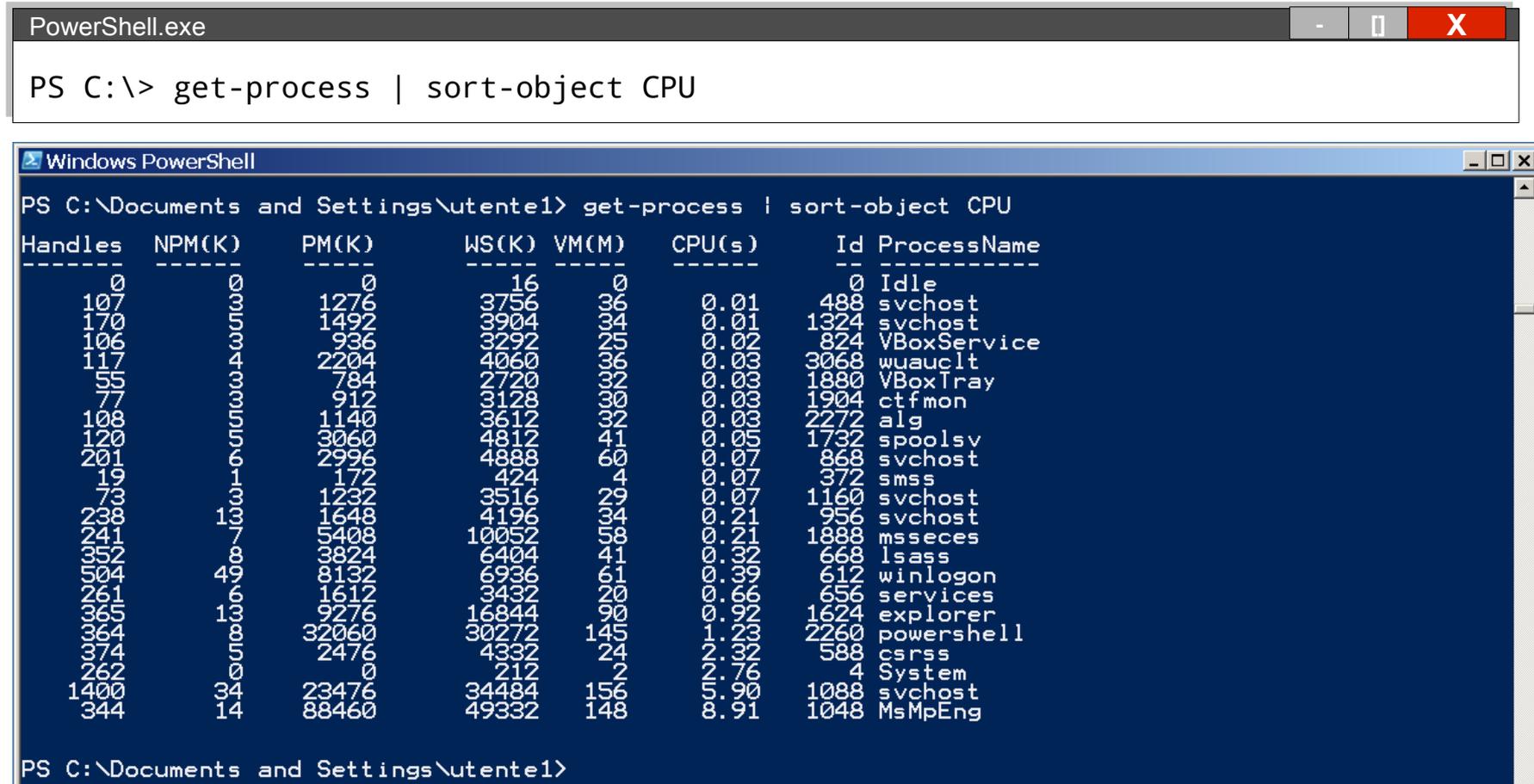
Ordinare l'output con **sort-object**



Simile a 'sort' di Unix/Linux

- Con **sort-object** posso ordinare l'output di un comando
 - I parametri che posso utilizzare sono
 - **-descending**
 - **-ascending** (default)
 - Come argomento devo passare **la proprietà dell'oggetto che voglio usare per l'ordinamento**

Lista dei processi, ordinata per tempo CPU



```
PowerShell.exe
PS C:\> get-process | sort-object CPU
```

```
Windows PowerShell
PS C:\Documents and Settings\utente1> get-process | sort-object CPU
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
0	0	0	16	0	0.00	0	Idle
107	336	1276	3756	36	0.01	488	svchost
170	1492	1492	3904	34	0.01	1324	svchost
106	936	936	3292	25	0.02	824	VBoxService
117	2204	2204	4060	36	0.03	3068	wuauclt
55	784	784	2720	32	0.03	1880	VBoxTray
77	912	912	3128	30	0.03	1904	ctfmon
108	1140	1140	3612	32	0.03	2272	alg
120	3060	3060	4812	41	0.05	1732	spoolsv
201	2996	2996	4888	60	0.07	868	svchost
19	172	172	424	4	0.07	372	smss
73	1232	1232	3516	29	0.07	1160	svchost
238	1648	1648	4196	34	0.21	956	svchost
241	5408	5408	10052	58	0.21	1888	msseces
352	3824	3824	6404	41	0.32	668	lsass
504	8132	8132	6936	61	0.39	612	winlogon
261	1612	1612	3432	20	0.66	656	services
365	9276	9276	16844	90	0.92	1624	explorer
364	32060	32060	30272	145	1.23	2260	powershell
374	2476	2476	4332	24	2.32	588	csrss
262	0	0	212	2	2.76	4	System
1400	34	23476	34484	156	5.90	1088	svchost
344	14	88460	49332	148	8.91	1048	MsMpEng

```
PS C:\Documents and Settings\utente1>
```

Un altro esempio...

Voglio ottenere la lista dei servizi,
ordinati per nome, e scegliere i primi 5

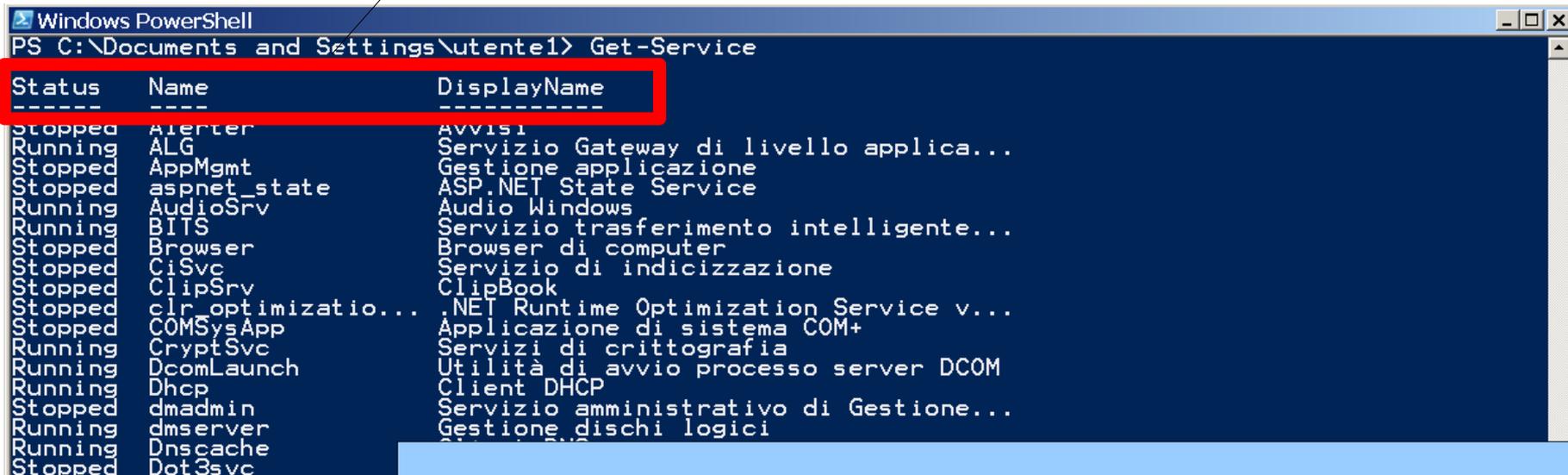
`get-service`

`sort-object`

`select-object`

Ottenere la lista dei servizi: **get-service**

Proprietà di Service



```
Windows PowerShell
PS C:\Documents and Settings\utente1> Get-Service

Status Name DisplayName
-----
Stopped Alerter Avviso
Running ALG Servizio Gateway di livello applica...
Stopped AppMgmt Gestione applicazione
Stopped aspnet_state ASP.NET State Service
Running AudioSrv Audio Windows
Running BITS Servizio trasferimento intelligente...
Stopped Browser Browser di computer
Stopped CiSvc Servizio di indicizzazione
Stopped ClipSrv ClipBook
Stopped clr_optimizatio... .NET Runtime Optimization Service v...
Stopped COMSysApp Applicazione di sistema COM+
Running CryptSvc Servizi di crittografia
Running DcomLaunch Utilità di avvio processo server DCOM
Running Dhcp Client DHCP
Stopped dmadmin Servizio amministrativo di Gestione...
Running dmserver Gestione dischi logici
Running Dnscache
Stopped Dot3svc
```

Lista dei servizi di sistema e il loro stato

Lista dei servizi, ordinata per nome completo

```
PowerShell.exe  
PS C:\> Get-Service | sort-object DisplayName
```

```
Windows PowerShell  
PS C:\Documents and Settings\utente1> Get-Service | sort-object DisplayName  
Status      Name                DisplayName  
-----  
Stopped     clr_optimizatio...  .NET Runtime Optimization Service v...  
Stopped     HidServ             Accesso periferica Human Interface  
Stopped     Netlogon            Accesso rete  
Running     seclogon            Accesso secondario  
Stopped     stisvc              Acquisizione di immagini di Windows...  
Stopped     napagent            Agente protezione accesso alla rete  
Running     wuauserv            Aggiornamenti automatici  
Stopped     COMSysApp           Applicazione di sistema COM+  
Stopped     NtmsSvc             Archivi rimovibili  
Running     ProtectedStorage    Archiviazione protetta  
Stopped     aspnet_state        ASP.NET State Service  
Running     AudioSrv            Audio Windows  
Stopped     RasAuto             Auto Connection Manager di Accesso ...  
Stopped     Alerter             Avvisi  
Stopped     Cryptui              Avvisi e notifiche di protezione
```

Limitare l'output con **select-object**



Simile a 'head' e 'tail'

- Con **select-object** posso limitare l'output di un comando
 - I parametri principali che posso utilizzare sono
 - **-first *n*** (seleziona i primi *n* elementi)
 - **-last *n*** (seleziona gli ultimi *n* elementi)

Lista dei servizi, ordinata per nome completo

PowerShell.exe

```
PS C:\> Get-Service | sort-object DisplayName | select-object -first 5
```

Windows PowerShell

```
PS C:\Documents and Settings\utente1> Get-Service | sort-object DisplayName | select-object -first 5
```

Status	Name	DisplayName
Stopped	clr_optimizatio...	.NET Runtime Optimization Service v...
Stopped	HidServ	Accesso periferica Human Interface
Stopped	Netlogon	Accesso rete
Running	seclogon	Accesso secondario
Stopped	stisvc	Acquisizione di immagini di Windows...

```
PS C:\Documents and Settings\utente1> _
```

Filtrare l'output con **where**



Simile a 'grep'

- Con **where {condizione}** posso filtrare l'output di un comando
 - Nella condizione posso utilizzare i seguenti operatori
 - **-lt** Minore di
 - **-le** Minore o uguale ai
 - **-gt** Maggiore di
 - **-ge** Maggior o uguale a
 - **-eq** Uguale a
 - **-ne** Non uguale a
 - **-like** Uguale (con wildcard)

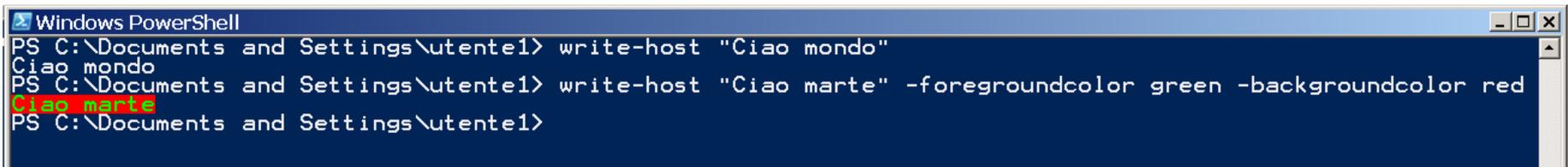
```
PS C:\Documents and Settings\utente1\Desktop> ls | where {$_ -like "P*.ps1"}
```

```
Directory: Microsoft.PowerShell.Core\FileSystem::C:\Documents and Settings\utente1\Desktop
```

Mode	LastWriteTime	Length	Name
-a---	22.11.2011 13:52	155	Prova.ps1
-a---	22.11.2011 13:53	176	Prova2.ps1
-a---	22.11.2011 13:54	183	Prova3.ps1

Scrivere sulla console con **write-host**

- Con **write-host [stringa]** posso scrivere sulla console (simile a 'echo' in bash)
 - Tra i parametri che posso utilizzare troviamo
 - **-foregroundcolor**
(cambia il colore di primo piano, **get-help write-host -detailed** per una lista dei colori disponibili)
 - **-backgroundcolor** (cambia il colore di sfondo)



```
Windows PowerShell
PS C:\Documents and Settings\utente1> write-host "Ciao mondo"
Ciao mondo
PS C:\Documents and Settings\utente1> write-host "Ciao marte" -foregroundcolor green -backgroundcolor red
Ciao marte
PS C:\Documents and Settings\utente1>
```

Metacaratteri / Globbing

- PowerShell supporta diversi metacaratteri, simili a quelli di bash

*	Zero o più caratteri	a*
?	Esattamente un carattere	b?llo
[]	Un carattere tra quelli definiti nell'insieme	brav[oaie]
[-]	Intervallo di caratteri	bing[a-m]

Metacaratteri

```
PS C:\Documents and Settings\utente1> ls D????????

Directory: Microsoft.PowerShell.Core\FileSystem::C:\Documents and Settings\utente1

Mode                LastWriteTime         Length Name
----                -
d-r--              21.09.2011   13:39             Documenti
```

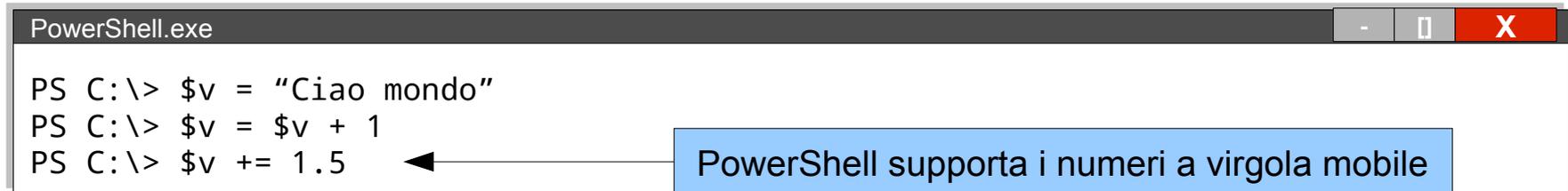
```
PS C:\Documents and Settings\utente1> ls D[oe]*

Directory: Microsoft.PowerShell.Core\FileSystem::C:\Documents and Settings\utente1

Mode                LastWriteTime         Length Name
----                -
d----              22.11.2011   10:05      Desktop
d-r--              21.09.2011   13:39      Documenti
```

Variabili

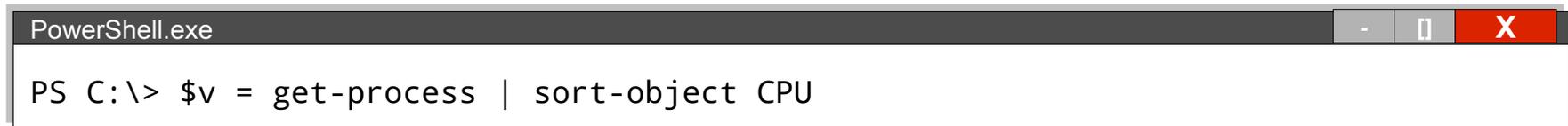
- Come in bash, è possibile definire delle variabili
 - il nome della variabile è sempre preceduto da \$



```
PowerShell.exe
PS C:\> $v = "Ciao mondo"
PS C:\> $v = $v + 1
PS C:\> $v += 1.5
```

PowerShell supporta i numeri a virgola mobile

- il valore è un oggetto, per esempio l'output di un comando



```
PowerShell.exe
PS C:\> $v = get-process | sort-object CPU
```

- posso visualizzare il valore semplicemente richiamando il nome della variabile



```
PowerShell.exe
PS C:\> $v
```

Nota: è possibile avere variabili 'tipate'... ma nell'ambito di questo corso non ci interessano

Accedere alle proprietà di un oggetto

- Ottenere il valore di una proprietà di un oggetto
 - da una variabile:

```
PS C:\Documents and Settings\utente1> $pallino = get-process | select-object -first 1
```

```
PS C:\Documents and Settings\utente1> $pallino.ProcessName
```

```
alg
PS C:\Documents and Settings\utente1> .
```

- da una serie di comandi (raggruppandoli con le parentesi)

```
PS C:\Documents and Settings\utente1> (get-process | select-object -first 1).ProcessName
```

```
alg
PS C:\Documents and Settings\utente1> .
```

Array

- Per definire un array specificando gli elementi (separati da una virgola) o utilizziamo il simbolo @

```
PowerShell.exe
PS C:\> $a = @"ciao","mondo","!"
PS C:\> $a = "ciao","mondo","!"
```

- La proprietà **Count** ritorna il numero degli elementi

```
PowerShell.exe
PS C:\> $a.Count
```

- Per accedere a un elemento dell'array utilizziamo la notazione **[n]**

```
PowerShell.exe
PS C:\> $a[0]
```

- Il primo elemento ha indice 0
- Posso specificare un intervallo con la forma **[n , m]**

Array

- Posso aggiungere elementi a un'array utilizzando **+**



```
PowerShell.exe
PS C:\> $a = @("ciao","mondo","!")
PS C:\> $a = $a + 5
```

- Con **+** posso anche concatenare due array



```
PowerShell.exe
PS C:\> $a = @("ciao","mondo","!")
PS C:\> $b = @("alpha","beta")
PS C:\> $c = $a + $b
```

Esempio di array

```
PS C:\Documents and Settings\utente1> $a = @("ciao", "mondo", "!")
PS C:\Documents and Settings\utente1> $a[0]
ciao
PS C:\Documents and Settings\utente1> $a[1]
mondo
PS C:\Documents and Settings\utente1> $a.Count
3
```

```
PS C:\Documents and Settings\utente1> $a = @("ciao", "mondo", "!")
PS C:\Documents and Settings\utente1> $a -contains "ciao"
True
PS C:\Documents and Settings\utente1> $a -contains "hello"
False
PS C:\Documents and Settings\utente1> █
```

```
PS C:\Documents and Settings\utente1> $a[1,2]
mondo
PS C:\Documents and Settings\utente1> $a = $a + 5
PS C:\Documents and Settings\utente1> $a
ciao
mondo
PS C:\Documents and Settings\utente1> █
```

Operare su una lista di oggetti

- Alcuni comandi ritornano più di un oggetto
 - es. `get-service`
- Con **ForEach-Object** possiamo eseguire dei comandi su ogni elemento di questa lista

ForEach-Object { comandi }

- La sequenza di comandi, separata da **;** deve essere inclusa tra parentesi graffe **{ }**
- L'oggetto corrente è assegnato alla variabile **\$_**

Esempio: ForEach-Object

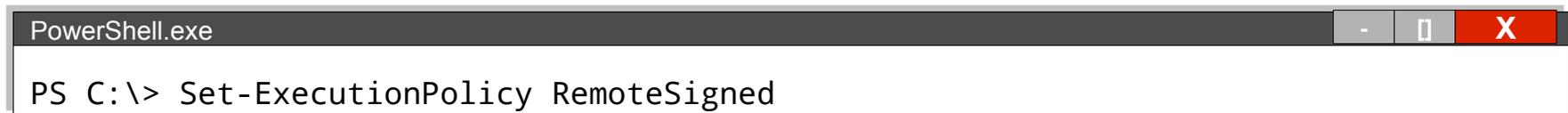
```
PS C:\Documents and Settings\utente1> ls | ForEach-Object { echo $_.Name $_.LastAccessTime }
Desktop
martedì, 22. novembre 2011 10:06:12
Documenti
martedì, 22. novembre 2011 08:36:11
Menu Avvio
martedì, 22. novembre 2011 08:36:23
Preferiti
giovedì, 3. novembre 2011 03:20:44
```

```
PS C:\Documents and Settings\utente1> get-process | ForEach-Object { echo $_.ProcessName $_.TotalProcessorTime }
alg
Days                : 0
Hours               : 0
Minutes            : 0
Seconds            : 0
Milliseconds       : 30
Ticks              : 300432
TotalDays          : 3.47722222222222E-07
TotalHours         : 8.34533333333333E-06
TotalMinutes      : 0.00050072
TotalSeconds       : 0.0300432
TotalMilliseconds  : 30.0432

csrss
Days                : 0
Hours               : 0
Minutes            : 0
Seconds            : 9
Milliseconds       : 533
Ticks              : 95337088
TotalDays          : 0.000118910951095109E-06
```

Script e sicurezza

- Per ragioni di sicurezza, PowerShell non permette (di default) l'esecuzione di script, ma solo l'utilizzo in modalità interattiva
 - Per poter eseguire uno script, bisogna “firmarlo” (certificarne l'origine) → *troppo complicato!* *
 - Possiamo comunque abilitare l'esecuzione dei nostri script cambiando la policy di esecuzione:



```
PowerShell.exe  
PS C:\> Set-ExecutionPolicy RemoteSigned
```

* http://powershellscripts.com/article3_powershell_script_signing.html

Percorsi e back-slash

- In Windows il separatore utilizzato nei percorsi (path) è il back-slash '\'
- Il percorso corrente è **.**
- Se volessimo eseguire lo script Prova.ps1 dobbiamo fornire:
 - il percorso completo dello script
 - oppure, se ci troviamo già nella directory dello script, **.\Prova.ps1**

Argomenti passati a uno script

- La variabile **\$args** è un array contenente gli argomenti passati ad uno script
 - a differenza di bash, l'elemento 0 non è il nome dello script ma il primo argomento
 - per conoscere il numero di elementi (argomenti) si utilizza la proprietà **Count**

If

- `if (condizione) { comandi }`
`elseif (condizione) { comandi }`
`else { comandi }`
- Se ci sono più comandi bisogna separarli con `;` o mettere ogni comando su una nuova riga

Condizionali

-eq <i>oppure</i> -ieq	Uguaglianza (case-insensitive)	Pippo -eq PIPPO
-ceq	Uguaglianza (case-sensitive)	Pippo -ceq Pippo
-lt	Minore di	5 -lt 9
-gt	Maggiore di	8 -gt 3
-le	Minore o uguale	7 -le 7
-ge	Maggiore o uguale	9 -ge 4
-ne	Non uguale	11 -ne 59
-not <i>oppure</i> !	Negazione logica	!(9 -ge 4)
-and	And logico	(1 -lt 4) -and (3 -ge 3)
-or	Or logico	(9 -eq 2) -or (7 -le 11)

Condizionali (2)

-match	Match con espressione regolare	“pp?” -match “Pippo”
-like	Match con wildcard	“Pi*” -like “Pippo”
-contains	Elemento in array	\$array -contains “Pippo”
-notmatch	Nessun match	“pp?” -notmatch “Pippo”

Esempio If

Prova.ps1

```
if ($args.Count -lt 1) {  
    echo "Bisogna specificare almeno un argomento!";  
    exit;  
} else {  
    echo "Argomenti ricevuti "$args.Count  
}
```

```
PS C:\Documents and Settings\utente1\Desktop> .\Prova.ps1  
Bisogna specificare almeno un argomento!  
PS C:\Documents and Settings\utente1\Desktop> .\Prova.ps1 ciao mondo  
Argomenti ricevuti  
2
```

Switch

- ```
switch (espressione) {
 {condizione} { comandi }
 valore { comandi }
 default { comandi }
}
```
- Se ci sono più comandi bisogna separarli con **;** o mettere ogni comando su una nuova riga
- Switch funziona in modo case-insensitive
  - per avere case-sensitive usare l'opzione **-casesensitive**
- Vengono considerate sempre tutte le possibilità:
  - per interrompere prima usiamo **break**

## Esempio Switch

### Prova.ps1

```
switch($args[0]) {
 0 { echo "Zero!" }
 {($_ -gt 0) -and ($_ -lt 10)} { echo "Tra 1 e 9" }
 10 { echo "Dieci!" }
 default { echo "Più di dieci!" }
}
```

```
PS C:\Documents and Settings\utente1\Desktop> .\Prova.ps1 8
Tra 1 e 9
PS C:\Documents and Settings\utente1\Desktop> .\Prova.ps1 0
Zero!
PS C:\Documents and Settings\utente1\Desktop> .\Prova.ps1 10
Dieci!
PS C:\Documents and Settings\utente1\Desktop> .\Prova.ps1 11
Più di dieci!
PS C:\Documents and Settings\utente1\Desktop> .\Prova.ps1 5
Tra 1 e 9
```

## Esempio Switch

### Prova2.ps1

```
switch($args[0]) {
 0 { echo "Zero!" }
 8 { echo "Otto!" }
 {($_ -gt 0) -and ($_ -lt 10)} { echo "Tra 1 e 9" }
 10 { echo "Dieci!" }
 default { echo "Più di dieci!" }
}
```

### Prova3.ps1

```
switch($args[0]) {
 0 { echo "Zero!" }
 8 { echo "Otto!"; break }
 {($_ -gt 0) -and ($_ -lt 10)} { echo "Tra 1 e 9" }
 10 { echo "Dieci!" }
 default { echo "Più di dieci!" }
}
```

```
PS C:\Documents and Settings\utente1\Desktop> .\Prova2.ps1 8
Otto!
Tra 1 e 9
PS C:\Documents and Settings\utente1\Desktop> .\Prova3.ps1 8
Otto!
PS C:\Documents and Settings\utente1\Desktop>
```

## Esempio Switch

### Prova.ps2

```
switch($args[0]) {
 0 { echo "Zero!" }
 8 { echo "Otto!" }
 {($_ -gt 0) -and ($_ -lt 10)} { echo "Tra 1 e 9" }
 10 { echo "Dieci!" }
 default { echo "Più di dieci!" }
}
```

```
PS C:\Documents and Settings\utente1\Desktop> .\Prova2.ps1 8
Otto!
Tra 1 e 9
PS C:\Documents and Settings\utente1\Desktop> .\Prova3.ps1 8
Otto!
PS C:\Documents and Settings\utente1\Desktop>
```

## Switch con wildcard e regex

- Con l'opzione **-wildcard** possiamo effettuare la scelta utilizzando dei valori che includono dei metacaratteri
- `switch -wildcard (espressione) {  
    valore { comandi }  
    default { comandi }  
}`
- Con l'opzione **-regex** possiamo usare delle espressioni regolari
- `switch -regex (espressione) {  
    'regex' { comandi }  
    default { comandi }  
}`

## Esempio Switch con wildcard

### Wildcard.ps1

```
switch -wildcard ($args[0])
{
 A* { "Inizia con A" }
 Zup* { "Inizia con Zup" }
 default { "Altro" }
}
```

```
PS C:\Documents and Settings\utente1\Desktop> .\Wildcard.ps1 Zuppa
Inizia con Zup
PS C:\Documents and Settings\utente1\Desktop> .\Wildcard.ps1 Pesce
Altro
PS C:\Documents and Settings\utente1\Desktop> .\Wildcard.ps1 Amarena
Inizia con A
```

## Esempio Switch con regex

### Regex.ps1

```
switch -regex ($args[0])
{
 'A\d' { "Inizia con A e ha un numero alla seconda posizione" }
 '(Zup)+' { "Contiene una o più ripetizioni di Zup" }
 default { "Altro" }
}
```

```
PS C:\Documents and Settings\utente1\Desktop> .\Regex.ps1 Amarena
Altro
PS C:\Documents and Settings\utente1\Desktop> .\Regex.ps1 A10
Inizia con A e ha un numero alla seconda posizione
PS C:\Documents and Settings\utente1\Desktop> .\Regex.ps1 ZupZupZapZep
Contiene una o più ripetizioni di Zup
```

## Loop

- Abbiamo a disposizione diversi tipi di loop
  - **while**
  - **do ... while**
  - **do ... until**
  - **for**
  - **foreach**
- Nei loop **for**, **foreach**, e **while** possiamo usare:
  - **continue** : per “saltare” alla prossima iterazione
  - **break** : per uscire dal loop

## While / Do While

- `while (condizione) { comandi }`
- `do { comandi } while (condizione)`
- Con **while** il test viene effettuato all'inizio del ciclo
  - se il test è falso i comandi non vengono mai eseguiti
- Con **do - while** il test viene effettuato alla fine del ciclo
  - i comandi vengono eseguiti almeno una volta

## Esempio While / Do-While

### while.ps1

```
$a = 5

while ($a -gt 0) {
 echo $a;
 $a = $a - 1;
}

do {
 echo $a;
 $a = $a + 1;
} while ($a -le 5)
```

```
PS C:\Documents and Settings\utente1\Desktop> .\while.ps1
```

```
5
4
3
2
1
0
1
2
3
4
5
```

## Until

- `do { comandi } until (condizione)`
- Con **do - until** il test viene effettuato alla fine del ciclo
  - il ciclo viene ripetuto se la condizione ritorna falso
  - i comandi vengono eseguiti almeno una volta

## Esempio Until

until.ps1

```
$a = 0

do {
 echo $a;
 $a = $a + 1;
} until($a -gt 5)
```

```
PS C:\Documents and Settings\utente1\Desktop> .\until.ps1
0
1
2
3
4
5
```

## For

- `for (inizializzazione; condizione; incremento) { comandi }`

## Esempio For

for.ps1

```
for ($i=1; $i -le 6; $i++) {
 echo $i;
}
```

```
PS C:\Documents and Settings\utente1\Desktop> .\for.ps1
1
2
3
4
5
6
```

## Foreach

- `foreach ($<elemento> in $<lista>)  
{comandi}`
- Con **foreach** posso iterare su una lista, array, o insieme di elementi assegnando, ad ogni ripetizione, un nuovo valore alla variabile specificata

## Esempio Foreach

### foreach.ps1

```
$numeri = @(1, 2, 3, 4, 5)

foreach ($numero in $numeri) {
 echo $numero;
}

$files = ls

foreach ($file in $files) {
 echo ($file.Name + " --> " + $file.Length);
}
```

```
PS C:\Documents and Settings\utente1\Desktop> .\foreach.ps1
1
2
3
4
5
ProcessViewer -->
0 --> 10
for.ps1 --> 42
foreach.ps1 --> 176
Prova.ps1 --> 155
Prova2.ps1 --> 176
Prova3.ps1 --> 183
Regex.ps1 --> 190
until.ps1 --> 59
while.ps1 --> 112
Wildcard.ps1 --> 190
Windows PowerShell - EN.zip --> 2021236
Windows PowerShell.lnk --> 1979
```

## Funzioni

- `function nome(parametri) { ... }`
  - Possiamo definire una funzione sia all'interno di uno script che da linea di comando (nel qual caso, avremo definito un nuovo comando)

```
Function somma($a, $b) {
 $risultato = $a + $b;
 $risultato;
}

Somma 10 5
```

Quello che viene stampato su schermo diventa il valore di ritorno

## Filtri

- Con le funzioni è anche possibile creare dei filtri, per esempio:

```
Function Filtro
{
$input | where-object {$_.Name -eq "Prova"}
}
```

- La variabile **\$input** all'interno delle funzione permette di accedere all'input della pipe

```
ls | Filtro
```

## Caricare definizioni nella shell corrente

- Per caricare delle definizioni nella shell corrente (i.e source in bash) utilizziamo il '.'



```
PowerShell.exe - □ X
PS C:\> . .\script.ps1
```