POWERSHELL – APPUNTI

Ver. 0.0.3

Prof. Fischetti Pietro

Powershell (PS) e' la shell evoluta di Microsoft per i Sistemi Windows, che affiancano la storica shell command.com. PowerShell, noto inizialmente come Microsoft Shell, MSH è una shell caratterizzata dall'interfaccia a riga di comando (CLI) e da un linguaggio di scripting, sviluppata da Microsoft, basato sulla programmazione a oggetti e sul framework Microsoft .NET [1]. Si avvia dal prompt dei comandi digitando powershell oppure powershell ISE per avere anche l'interfaccia grafica (UI). E' case-insensitive.I comandi vengono anche chiamati cmdlets (serie di comandi). La differenza fondamentale tra l'approccio Unix e quello di PowerShell risiede nel fatto che piuttosto che creare una "pipeline" (lett. tubo) basata su input ed output testuali, PowerShell fa passare i dati da una cmdlet all'altra come oggetti (dati dotati di una struttura ben precisa). L'output di una cmdlet, se si tenta di accedervi tramite riga di comando, viene automaticamente convertito in testo; se, invece, si tenta di utilizzarlo in un'altra cmdlet, esso verrà convertito nell'oggetto più appropriato per l'input di quell'altra cmdlet.

Hello World

```
PS C:\> Write-Output "hello world"
hello world
PS C:\> write-host 'Hello World'
Hello World
PS C:\> echo "Hello World"
Hello World
PS C:\> "hello world"
hello world
```

Read

```
PS C:\> Read-Host -prompt "tuo nome?"
tuo nome?: Albert
Albert
```

Variabili

Le variabili devono avere il nome preceduto dal carattere $.

Esempio:

```
PS C:\> $test='Hello World'
PS C:\> echo $test
Hello World
PS C:\> $nome=(Read-Host -prompt "tuo nome?")
joe
PS C:\> $nome.length
3
```

Oltre alle variabili che posso definire esistono delle variabili predefinite in PS

Variabili Builtin

```
$?          :    Oggetto Corrente
$Error:          Collezione di Errori
$Home:           La directory di profilo
$LastExitCode:   Codice numerico ritornato dall'ultimo programma eseguito (in C e' il return 0; del main())
$PID:            Un numero che identifica un processo su una macchina
$PsHome:         dir di installazione di PS.
$Pwd:            Stampa la directory corrente di lavoro
$null:           se una variabile e' stata o meno inizializzata
```

Ad esempio con $null possiamo sapere se una variabile contiene un valore o meno prima di operare su di essa.

```
PS:\> if ($s -ne $null) {$s.length}
```

Help

Per avere l'Help dei comandi e' disponibile il cmdlet get-Help.
Ad esempio per ottenere informazioni circa il cmdlet Get-Process

```
PS C:\> Get-Help Get-Process
```

Con -Detailed otteniamo informazioni dettagliate che includano i parametri e esempi, mentre con -Full vengono aggiunti informazioni di tipo su input and output e note addizionali. input and output object types, and additional notes. input e output object types, and additional notes.
Per individuare la lista dei comandi disponibili nella shell si ulizza il cmdlet Get-Command. L'esecuzione di `Get-Command` senza parametri restituisce un elenco di tutti i comandi nel sistema, con l'alias, il nome la versione e il modulo dove si trova il cmdlet.

```
PS C:\> Get-command
CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Alias           Add-AppPackage                      2.0.1.0    Appx
Alias           Add-AppPackageVolume                2.0.1.0    Appx
Alias           Add-AppProvisionedPackage           3.0        Dism
Alias           Add-ProvisionedAppPackage           3.0        Dism
Alias           Add-ProvisionedAppxPackage          3.0        Dism
Alias           Add-ProvisioningPackage             3.0        Provisioning
Alias           Add-TrustedProvisioningCertificate  3.0        Provisioning
……
```

Esempio :

```
PS C:\> Get-Command Get-Process
```

E' possibile specificare il tipo nella ricerca:
Get-Command -Name *service* -CommandType Cmdlet, Function, Alias

Get-Date e' il cmdlet utilizzato per sapere la data e l'ora corrente:

```
PS C:\> Get-command Get-Date

CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Cmdlet          Get-Date                            3.1.0.0    Microsoft.PowerShell.Utility
```

```
PS C:\> Get-Date
giovedì 18 febbraio 2021 11:08:44
```

I cmdLet ritornano oggetti creati sul framework MS .NET che contiene centinaia di classi.

Operatori logici
Dato che non si possono utilizzare i simboli di >,< in quanto utilizzati dal sistema operativo per la redirezione dell'IO si utilizzano delle abbreviazioni (tra parentesi il corrispondente operatore in C):
-eq (=) -ne(!=) -lt(<) -gt(>) -le(<=) -ge(>=)
es:

```
PS C:\> 123 -gt 12
```

operatori -like e- notlike per lavorare sulle stringhe di ricerca:

```
PS C:\> 'genova' -like 'ge*'
```

operatori booleani:
-and -ot -xor -not
Valori booleani:
$true $false
Gli operatori di verifica tipo -is e -isnot si utilizzaono per sapere che tipo di oggetto contiene una variabile

```
PS C:\> $dt=Get-Date
PS C:\> $dt -is [datetime]
```

```
PS C:\>  $n=12; $n -is [int32]
```

flusso di esecuzione:
if(...){} elseif, switch(){vl:{}default:{}}, while, do while, break, continue
Per l'esecuzione ripetuta un certo numero di volte esistono il for e il foreach.
Esempio scrivo 5 volte hello a video

```
PS C:\>  for (es:for($i=0;$i -lt 5; $i++){Write-host 'hello'})
```

la foreach si utilizza per iterare su una collezione di oggetti tipicamente liste se non voglio utilizzare la proprieta' che mi dice quanti ce ne sono:

foreach($<item> in <collection>){...}

foreach($proc in Get-process){echo $proc}

```
PS C:\> $a=Get-Process
PS C:\> for ($i=0;$i -lt ($a).length;$i++){echo ($a)[$i]}
```

oppure:

```
PS C:\> for ($i=0;$i -lt (Get-Process).length;$i++){echo (Get-Process)[$i]}
```

Esempio:
Voglio calcolare la lunghezza di una variabile che contiene una stringa evitando errori a Run-Time:

```
PS C:\> $test.Length
Impossibile chiamare un metodo su un'espressione con valore null.
In riga:1 car:1
+ $test2.tostring()
+ ~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
   + FullyQualifiedErrorId : InvokeMethodOnNull
Quindi verifico che la variabile $test contenga un oggetto e che questo sia una stringa.
if($test -ne $null -and $test -is [String]){$test.length}
```

Script

Gli script di PS sono file di testo, con estensione .PS1, che se richiamati dalla shell portano all'esecuzione delle istruzioni che contiene.
Ad esempio creiamo uno script, che ritorni informazioni su un processo dato il suo PID:

```
PS C:\> type procid.ps1
param($procid)
write-host 'processing...."
Get-Process -Id $procid
```

Se lo avviamo con:

```
PS C:\> .\procid.ps1 19780
Ed tteniamo il seguente messaggio di errore:
.\procid.ps1 : Impossibile caricare il file C: \procid.ps1. L'esecuzione di script è disabilitata
nel sistema in uso. Per ulteriori informazioni, vedere about_Execution_Policies all'indirizzo
https://go.microsoft.com/fwlink/?LinkID=135170.
In riga:1 car:1
+ .\procid.ps1 19780
+ ~~~~~~~~~~~~~
   + CategoryInfo          : Errore di protezione: (:) [], PSSecurityException
   + FullyQualifiedErrorId : UnauthorizedAccess
The Default Execution Policy is set to restricted, you can see it by running Get-ExecutionPolicy:
```

Controlliamo i permessi impostati sulla macchina per l'esecuzione dello script:

```
PS C:\> Get-ExecutionPolicy -List

        Scope ExecutionPolicy
```

```
      ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process       Undefined
  CurrentUser       Undefined
 LocalMachine       Restricted
```

In questo caso l'esecuzione di script nella macchina locale e' disabilitato. Per consentirlo digitiamo:

```
PS C:\> Set-ExecutionPolicy unrestricted
Esempio script che visualizza i processi lanciati nell'ultima ora:
foreach($proc in Get-process){
    if($proc.starttime){# alcuni processi es system e idle non danno questa info
        $procAge=(get-date)-$proc.StartTime
        if($procAge.TotalHours -le 1){#mettere le parentesi graffe anche nel caso di 1 istruz.
            $proc
        }
    }
}
```

Oggetti

Ogni informazione gestita da PS e' rappresentata da un oggetto .NET.
Ricordiamo che un oggetto e' un'istanza di un certo tipo (classe). Per accedere a metodi e proprieta' di un oggetto si utlizza il nome del cmdlet racchiuso tra parentesi e seguto dal punto.
Ad esempio per avere informazioni sulla console PowerShell utilizzata:

```
PS C:\> Get-Host
Name             : ConsoleHost
Version          : 5.1.19041.610
InstanceId       : 2ca0faea-6c18-43c2-be66-ae05e1ae5e6f
UI               : System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture   : it-IT
CurrentUICulture : it-IT
PrivateData      : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
DebuggerEnabled  : True
IsRunspacePushed : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace
```

Se vogliamo conoscere il solo numero di versione di PS, non possiamo semplicemente fare cosi'

```
PS C:\> Get-Host.Version
Get-Host.Version : Termine 'Get-Host.Version' non riconosciuto come nome di cmdlet, funzione,
programma eseguibile ….
```

Il messaggio di errore indica che si e' richiesto il cmdlet Get-Host.Version che non e' un cmdlet di PS
La sintassi corretta prevede di racchiudere il cmdlet fra parentesi tonde in questo modo ottengo un riferimento all'oggetto e il nome della proprieta' richiesta preceduta dal punto:

```
PS C:\> (Get-Host).Version

Major  Minor  Build  Revision
-----  -----  -----  --------
5      1      19041  610
```

Ma come faccio a sapere la classe (il tipo) di un oggetto restituito da un cmdLet?
Ad esempio il tipo restituito dal cmdlet Get-Date, si utilizza il metodo GetType():

```
PS C:\> (Get-Date).GetType()
IsPublic IsSerial Name                                     BaseType
```

```
-------- -------- ----                    --------
True     True     DateTime                System.ValueType
```

Notare il Tipo Base da cui deriva l'oggetto in questione.
Oppure:

```
PS C:\> (Get-Date).GetType().FullName
System.DateTime
```

Seguendo la filosofia che tutto e' un oggetto in .NET anche i numeri sono oggetti:

```
PS C:\> (123).GetType().FullName
System.Int32
PS C:\> (12.3).GetType().FullName
System.Double
```

Con il cmdLet New-Object possiamo costruire oggetti:

```
PS C:\> $obj=new-Object System.Int32
PS C:\> $obj.gettype()

IsPublic IsSerial Name                    BaseType
-------- -------- ----                    --------
True     True     Int32                   System.ValueType
```

Creari oggetti COM quali ad esempio WORD

```
# Create Word Object
$wrd = new-object -com "word.application"
# Make Word Visible
$wrd.visible = $true
# Open a document
$doc = $wrd.documents.open("C:\temp\myDocument.doc")
```

Oppure di creare dei tipi personalizzati come nella classica Programmazione a oggeti ([2]):

```
# Class Syntax
class CyberNinja
{
}
```

Posso convertire da un tipo ad un altro:

```
$str="123"
$intero=(Int32)$str
```

Le informazioni di un oggetto in PS si ottengono con il cmdlet Get-Member.
Per conoscere le proprieta' di un oggetto si utilizza il cmdlet Get-Member:
Get-Member [-MemberType Property|Method|Aliases]

```
PS C:\> Get-Member -InputObject (Get-Service) -MemberType Property


   TypeName: System.Object[]
```

Se voglio sapere il tipo degli elementi contenuti in questo array:

```
PS C:\> Get-Member -InputObject (Get-Service)[0] -MemberType Property


   TypeName: System.ServiceProcess.ServiceController
```

```
Name              MemberType Definition
----              ---------- ----------
CanPauseAndContinue Property   bool CanPauseAndContinue {get;}
CanShutdown       Property   bool CanShutdown {get;}
CanStop           Property   bool CanStop {get;}
……..
```

Quindi Get-Service ritorna un array di oggetti di tipo System.ServiceProcess.ServiceController

Si possono ottenere le stesse informazioni con le pipeline:

```
PS C:\ > Get-Service | Get-Member -MemberType Property


   TypeName: System.ServiceProcess.ServiceController

Name              MemberType Definition
----              ---------- ----------
CanPauseAndContinue Property   bool CanPauseAndContinue {get;}
CanShutdown       Property   bool CanShutdown {get;}
……
```

Occorre sottolineare che Get-Member ritorna solo il nome delle proprieta' di un oggetto. Se voglio conoscerne anche il valore posso utilizzare il cmdlet Select-Object:

```
PS C:\>  Get-Service -ServiceName 'w3svc' | Select-Object -Property 'StartType'

StartType
---------
Automatic
```

Alcune proprieta' hanno degli Alias:

```
PS C:\> Get-Service | Get-Member -MemberType 'AliasProperty'


   TypeName: System.ServiceProcess.ServiceController

Name              MemberType    Definition
----              ----------    ----------
Name              AliasProperty Name = ServiceName
RequiredServices AliasProperty RequiredServices = ServicesDependedOn
```

Se voglio conoscere imembri statici utilizzo l'opzione -Static
Esempio

```
PS C:\> Get-member -InputObject (Get-Date) -static
Oppure:
PS C:\> [System.Datetime] | Get-Member -static
Oppure:
PS C:\> Get-Date | Get-Member -Static
   TypeName: System.DateTime

Name          MemberType Definition
----          ---------- ----------+
```

```
Compare        Method     static int Compare(datetime t1, datetime t2)
DaysInMonth    Method     static int DaysInMonth(int year, int month)
Equals         Method     static bool Equals(datetime t1, datetime t2), static bool Equals(System...
….
```

La Pipeline

Come si sa la tecnica delle pipe nei sistemi operativi esiste da parecchio tempo, e rende gli script piu' semplici evitando l'introduzione di variabili e file temporanei. La riga di comando costituita da piu' blocchi separati dal carattere di pipe (|) prende il nome di pipeline e viene eseguita dalla shell da sinistra verso destra: ogni risultato fornito da ogni blocco diventa l'input per il blocco successivo .
Esempio, si vuole chiudere tutte le istanze del processo notepad in esecuzione; di seguito vengono visualizzati tre modi equivalenti ma l'ultimo che utilizza la pipeline risulta piu' semplice e chiaro:

```
PS C:\> $prcs=(Get-Process notepad)
PS C:\> Stop-Process -InputObject $procs

PS C:\> Stop-Process -InputObject (Get-Process notepad)

PS C:\> Get-Process notepad | Stop-Process
```

Ma come fa la pipeline ad associare (binding) gli oggetti ai parametri di input?.I parametri che accettano oggetti oggetti dalla pipeline ammettono due diverse modalita' di associazione: il passaggio per valore (ByValue) dove PS verifica che il tipo fornito sia del tipo ammesso dal parametro stesso o che sia possibile una conversione. Il passaggio per nome di prorieta' (ByPropertyName) la shell ricava il nome del parametro dalla prorieta' oppurtuna dell'oggetto. Ad esempio supponiamo che un processo abbia PID=1231 e che lo voglia interrompere, con lo script che segue non riuscirei:

```
PS C:\> 1231 | stop-process
stop-process : Impossibile associare l'oggetto di input a qualsiasi parametro del comando. Il
comando non accetta l'input da pipeline oppure l'input e le relative proprietà non corrispondono ad
alcun parametro che accetta l'input da pipeline.
In riga:1 car:8
+ 4364 | stop-process
+        ~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (1231:Int32) [Stop-Process], ParameterBindingException
    + FullyQualifiedErrorId : InputObjectNotBound,Microsoft.PowerShell.Commands.StopProcessCommand
```

Vediamo con Get-Help l'Help di Stop-Process, e focalizziamo l'attenzione sulla dicitiura "Accettare input da pipeline?" presente nell'help dei parametri con valore diverso da false:

```
PS C:\> get-help stop-process -Full
    -Id <int[]>
        Accettare input da pipeline?    true (ByPropertyName)

    -InputObject <Process[]>
        Accettare input da pipeline?    true (ByValue)
…..
```

Questo vuol dire che se voglio interrompere un processo per PID devo creare un oggetto che abbia una prorieta' di nome Id, nello script seguente ci sono due modi per fare cio':

```
class myDummy
{
[Int] $Id
}
$myObj = [myDummy]::new()
$myObj.Id=4364
```

```
$myObj | Stop-Process

#Oppure:
$myObj=new-object PSObject
$myObj |Add-Member -Value 4364 -Name "Id" -MemberType NoteProperty
```

Questo fa capire perche' il seguente script va in errore:

```
PS C:\> Get-Date | Stop-Process
```

Esiste inoltre un'utile comando per il debug e il tracing.
Facciamo il debug con:

```
PS C:\> trace-command -name parameterbinding -Expression{get-date|stop-process} -PSHost
.....
DEBUG: ParameterBinding Information: 0 :              ERROR: ERROR: COERCE FAILED: arg [09/02/2021
20:19:01] could not be converted to the parameter type [System.Diagnostics.Process]
.....
```

E' proprio l'impossibilita' di convertire il tipo datetime nel tipo Process che riporta il cmdlet.
I cmdlet di seguito si usano spesso nelle pipeline e facilitano l'interrogazione con una sintassi simile a SQL.
where-object
Esempio recupero i processi avviati nelle ultime 2 ore, Utilizzo la variabile automatica $_ che rappresenta l'oggetto corrente:
Get-process | where-object {$_.StartTime -gt (Get-Date).AddHours(-2)}

Al posto di where-object si puo' utilizzare l'alias (?):
Get-process | ? {$_.StartTime -gt (Get-Date).AddHours(-2)}
oppure:
ps| ? {$_.StartTime -gt (Get-Date).AddHours(-2)}

sort-object (alias:sort)
Esempio restituisce i processi che nelle ultime 2 ore hanno consumato + memoria

```
PS C:\> Get-process | where-object {$_.StartTime -gt (Get-Date).AddHours(-2)} | Sort-object WorkingSet
descending
```

Esempio:
Contenuto del file movies.csv che voglio importare con il cmdlet import_csv

```
"Title","ReleaseDate","Comments","Rating"
"Star Trek Beyond","7/22/2016","must see","PG-13"
"Jason Bourne","7/29/2016",,"PG-13"
"Patient Zero","9/2/2016","Horror","NR"
"The Magnificent Seven","9/23/2016","PG-13"
"Doctor Strange","11/4/2016","Marvel","NR"
"Fantastic Beasts and Where to Find Them","11/18/2016","Harry Potter related","NR"
"Rogue One","12/16/2016","Star Wars","NR"
"The Dark Tower","2/17/2017","Stephen King","NR"
"Ghost in the Shell","3/31/2017","SciFi","NR"
"Spectral","8/12/2016","Supernatural thriller","PG-13"
"The Space Between Us","8/19/2016","space adventure","PG-13"
"Miss Peregrine's Home for Peculiar Children","9/30/2016","Tim Burton","PG-13"
"Arrival","11/11/2016","sci-fi","NR"
"Moana","11/25/2016","Disney animated","NR"
"Passengers","12/21/2016","sci-fi","NR"
"Assassin's Creed","12/23/2016",,"NR"
```

```
"Sing","12/23/2016","animated","NR"
"John Wick: Chapter Two","2/10/2017",,"NR"
"Wonder Woman","6/2/2017","comic book","NR"
"Justice League","11/17/2017","","NR"

$data = Import-CSV .\movies.csv
($data) | sort {$_.releaseDate}
```

Se voglio ordinare per data si nota che l'elenco non viene ordinato correttamente in quanto il campo che contiene la data in realta' e' stato importato come stringa. Occorre convertire la colonna releaseData

```
($data) | sort {[Datetime]$_.releaseDate}
```

select-object

con -skip limita il numero di oggetti restituito, con -first e -last specifico il numero di elementi da recuperare.
Esempio processo avviato nelle ultime 2 ore e che occupa + memoria

```
PS C:\> Get-process | where-object {$_.StartTime -gt (Get-Date).AddHours(-2)} | Sort-object WorkingSet descending | select-object -First 1
```

con -property e excludeproperty dico quali sono le proprieta' da visualizzare
Esemio cerco le sole prop. description e starttime

```
PS C:\> Get-process|select-object -property description, starttime
```

con -expandproperty si chiede di ritornare la prop. specificata anche in caso di collezione di oggetti.
Esempio cerco il modulo (eseguibile e librerie) del processo + esoso di memoria:

```
PS C:\> Get-process|Sort-object workingset -Descending|select-object -First 1 -Expandproperty Modules
```

ForEach-Object

Foreach, presente in molti linguaggi odierni eper certi aspetti alternativo al ciclio for, permette la scansione in un contenitore di oggetti.
ForEachObject -Process <script> [-Begin <script>] [-End <script>]
Alias:foreach oppure %

```
PS C:\> foreach($item in $test) {$item}
#oppure:
PS C:\> foreach($_ in $test) {$_}
```

Esempio chiedo la lista dei processi in esecuzione in un pc remoto e recupera il nome e il percorso dell'eseguibile, salvando su file di testo.

```
PS C:\> $log=''
PS C:\> Get-process -ComputerName 10.0.1.23 | ForEach-Object
{$log+=$_.MainModule.FileName+[Environment]::NewLine}
$log | Set-Content ProcessLog.Txt
```

Oppure utilizzando Begin e End:

```
PS C:\> Get-process -ComputerName 10.0.1.23 | ForEach-Object
{$log+=$_.MainModule.FileName+[Environment]::NewLine}
-Begin {$log=''} -End{$log}|Set-content ProcessLog.txt
```

Group-Object

Raggruppa in base a una o piu' proprieta':
group-object -property <nome> [-casesensitive]

Esempio raggruppo per estensione l'elenco degli elementi della cartella c:\windows

```
PS C:\> Get-childitem c:\windows |Group-object Extension
```

Array

Ci sono vari modi per inizializzare una variabile di tipo array, come sequenza di valori separate da virgola:

```
PS C:\> $a=1,2,3,5,7,8,9
PS C:\> $a.getType()
IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

Per definire un Array che contiene un solo elemento:

```
PS C:\> $a1=,7
#oppure
PS C:\> $a1=@(5)
```

Una variabile di tipo array puo' contenere dati di tipo diverso:

```
PS C:\> $test=123,(Get-process | select -first 1),'joe'
```

Si possono definire array specificando un range numerico

```
PS C:\> $r=2..5
PS C:\> $n=-3..1
```

E creare un nuovo array come unione:

```
PS C:\> $r+$n
2
3
4
5
-3
-2
-1
0
1
```

Non esiste rimuovi elementi ma si utilizzano le tecniche degli array.
Es. rimuovo il 3 elemento.

```
PS C:\> $test='a','b','c','d'.'e','f','g'
PS C:\> $test=$test[0..2+4..6]
PS C:\> $test
a
b
c
f
g
```

Filtrare gli elementi: su un array si possono usare gli operatori di confronto:

```
PS C:\> $a=0..9
PS C:\> $a -lt 5
0
1
2
3
```

```
4
```

Ma non e' possibile filtrare su 2 o piu' condizioni, occorre usare Where-Object.

```
PS C:\> $test=-9..9
$test | where-object {$_ -gt 0 -and $_ %2 -eq 0}

#oppure:
for($i=0; $i -lt $test.length;$i++){
        $item=$test[$i]
        if($item -gt 0 -and $item %2 -eq 0){
                $item
        }
 }
```

Stampare gli elementi di un array

```
PS C:\> foreach($item in 1..5) {$item}
#oppure:
foreach($_ in 1..5) {$_}
#oppure:
1..5 | ForEach-Object {$_}
#oppure:
1..5 | % {$_}
#oppure:
ForEach ($number in 1..5){$number}
1
2
3
4
5
```

Verificare se un array contiene un valore

```
PS C:\> $test =1..5
PS C:\> $test -contains 3
True
PS C:\ > $test -contains 32
False
PS C:\ > $test -contains $null
False
PS C:\ > $test -notcontains $null
True
```

Vediamo i membri statici degli array

```
PS C:\ > @().gettype()|get-member -static
.....
FindLast        Method     static T FindLast[T](T[] array, System.Predicate[T] match)
FindLastIndex   Method     static int FindLastIndex[T](T[] array, System.Predicate[T] match), stati...
ForEach         Method     static void ForEach[T](T[] array, System.Action[T] action)
IndexOf         Method     static int IndexOf(array array, System.Object value), static int IndexOf...
LastIndexOf     Method     static int LastIndexOf(array array, System.Object value), static int Las...
new             Method     System.Object[] new(int )
ReferenceEquals Method     static bool ReferenceEquals(System.Object objA, System.Object objB)
Resize          Method     static void Resize[T]([ref] T[] array, int newSize)
Reverse         Method     static void Reverse(array array), static void Reverse(array array, int i...
```

```
Sort          Method     static void Sort(array array), static void Sort(array keys, array items)...
TrueForAll    Method     static bool TrueForAll[T](T[] array, System.Predicate[T] match)
...
$test=1..5
PS C:\ > [array]::indexof($test,3)
3
#oppure:
PS C:\ > ($test).indexof(3)
3
#oppure:
PS C:\ > [array]::indexof($test,31)
-1
#oppure:
PS C:\ > [array]::reverse($test)
PS C:\ > $test
5
4
3
2
1
```

Array Associativi

Per la definizione si usa il carattere @ seguito dalle parentesi graffe. In alcuni linguaggi vengono anche definiti dictionary. E' un contenitore di coppie chiave, valore

```
PS C:\> $hash=@{}
PS C:\> $hash=@{conto='cento';'data'=(Get-Date);'Processes'=(Get-Process)}
$hash.count
PS C:\> $hash['data']

mercoledì 10 febbraio 2021 22:59:19

PS C:\> $hash['data']=123
PS C:\> $hash['data']
123

$hash['duecento']=200
$hash.Remove('duecento')
```

Esempio: carica un Array Associativo da file:

File:machines.txt

```
SRV01=10.0.1.1
SRV02=10.0.1.2
SRV03=10.0.1.3
```

```
PS C:\> $machines=get-content -raw .\machines.txt | ConvertFrom-StringData
PS C:\ > $machines

Name                   Value
----                   -----
SRV01                  10.0.1.1
SRV03                  10.0.1.3
```

```
SRV02                    10.0.1.2

PS C:\ > $machines['SRV02']
10.0.1.2
PS C:\ > $machines['srv02']
10.0.1.2
#si puo usare anche il punto in caso di chiave string
PS C:\> $machines.SRV02
10.0.1.2

#accesso alle chiavi
PS C:\> $machines.keys
SRV01
SRV02
SRV03

#accesso ai valori
PS C:\ > $machines.values
10.0.1.1
10.0.1.3
10.0.1.2
```

ATTENZIONE un hash non e' un array!!! Quindi non funziona correttamente l'ordinamento

```
$noblegas=@{elio=2;neon=10;argon=18}
PS C:\ > $noblegas.GetType()

IsPublic IsSerial Name                     BaseType
-------- --------     ----                     --------
True     True     Hashtable                System.Object


PS C:\ > $noblegas.GetType().FullName
System.Collections.Hashtable

PS C:\> $noblegas

Name                Value
----                -----
argon                  18
neon                   10
elio                    2

#ordinare per nome (NB. Non va)
PS C:\> $noblegas|sort-object name

Name                Value
----                -----
argon                  18
neon                   10
elio                    2

#Si utilizza allora per ordinare il metodo GetEnumerator()
```

```
PS C:\> $noblegas.GetEnumerator() | sort-object name

Name                   Value
----                   -----
argon                  18
elio                   2
neon                   10
```

Esempio

```
PS C:\> $ageList = @{}
PS C:\> $ageList.Kevin = 35
PS C:\> $ageList.Alex = 9
PS C:\> $agelist.GetEnumerator() | ForEach-Object{
      $message = '{0} is {1} years old!' -f $_.key, $_.value
      Write-Output $message
 }
Kevin is 35 years old!
Alex is 9 years old!
```

Per effttuare ricerche in un array associativo, si utilizza il metodo ContainsKey() per cercare una spcifica chiave mentre ContainsValue per cercare uno specifico valore

```
PS C:\> $noblegas.containsKey('Neon')
PS C:\> $noblegas.containsValue(3)
```

Stringhe

Le stringhe si definiscono con i dpooi apici o con apici singoli.
N.B.!Se uso i doppi apici le variabili precedute dal $ sono sostuite, mentre con apici singoli no:

```
PS C:\> $a="uno"
PS C:\> $b="ciao $a a tutti"
PS C:\> $b
ciao uno a tutti
PS C:\> $b='ciao $a a tutti'
PS C:\> $b
ciao $a a tutti
```

I Caratteri di escape si specificano con l'apice inverso (ALT+96):

```
PS C:\> $s="ciao a `n tutti"
PS C:\> $s
ciao a
 tutti

PS C:\> "$x `$x $x"
123 $x 123

PS C:\> $stars=@{
AlphaCentauri=4.3;
Barnard=5.9;
Wolf359=7.8
}
PS C:\>  $stars.GetEnumerator()| %{"$($_.name)`t$($_.Value)"}
Barnard 5.9
Wolf359 7.8
```

Posso accedere ad una stringa tramite numero intero o sequenza

```
PS C:\ > $s="welcome world !"
write-host $s[2]
write-host $s[3..5]

l
c o m
```

join di piu' stringhe

```
PS C:\ > "uno", "due", "tre" -join ';'
uno;due;tre
```

Split di una stringa:

```
PS C:\> '1,2,3,4,5,7' -split ','
1
2
3
4
5
7
```

Si puo' applicare lo split puo' specificare il numero di sottostringhe da restituire

```
PS C:\> '1,2,9,4,5,7' -split ',',3
2
9,4,5,7
```

Si puo' anche considerare come delimitatore uno script al quale la shell fornisce, per mezzo della variabile automatic $_, ciascun carattere della stringa di partenza.
Esempio considero come delimitatori il solo carattere '0':

```
PS C:\>   '1;3,0:7/11|13' -split {$_ -eq '0'}
1;3,
:7/11|13
```

Esempio sono considerati come delimitatori di stringa tutti i caratteri non compresi nell'intervallo riservato alle cifre

```
PS C:\> '1;3,4:7/11|13' -split {$_ -gt '9' -or $_ -lt '0'}
1
3
4
7
11
13
```

Replace:

```
PS C:\ > 'castello' -replace 'ste', 'va'
cavallo
```

NB! Una stringa e' immutabile

```
PS C:\> $R="unoduetre"
PS C:\> $R[1]='G'
Impossibile eseguire l'indicizzazione in un oggetto di tipo System.String.
In riga:1 car:1
+ $R[1]='G'
+ ~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : CannotIndex
```

Con il metodo ToCharArray, che crea un array di byte a partire da una stringa,ho una copia modificabile della stringa:

```
PS C:\> $w="unoduetre".tochararray()
$w[7]='Q'
```

oppure:

```
PS C:\> $U=$R[0..($R.length-1)]
$U[3]='F'
```

Altre metodi dell'oggetto stringa:
toupper(),tolower()
substring(),Remove(),Trim(),TrimStart(),TrimEnd(),IndexOf(), LastIndexOf()

WildCard e Regular Expressions
Il primo wildcard e' il carattere asterisco * che indica Zero o + caratteri

```
PS C:\temp> get-childitem *.txt|select name
    Directory: C:\temp

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        13/11/2020     09:32             21 a.txt
-a----        13/11/2020     09:32             21 a1.txt
-a----        13/11/2020     09:32             21 aa.txt
-a----        16/11/2020     12:03              6 b.txt
-a----        04/02/2021     11:20           1179 List.txt
-a----        10/02/2021     23:06             48 MACHINES.txt
-a----        13/11/2020     08:58             21 Nuovo documento di testo.txt
-a----        10/01/2021     15:08            967 x.txt
```

Il wildcard punto interrogativo ? indica un solo carattere

```
PS C:\ temp> ls a?.txt


    Directory: C:\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
```

```
-a----        13/11/2020    09:32             21 a1.txt
-a----        13/11/2020    09:32             21 aa.txt
```

Diverso dall'utilizzo con l'asterisco

```
PS C:\ temp> ls a*.txt


    Directory: C:\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        13/11/2020    09:32             21 a.txt
-a----        13/11/2020    09:32             21 a1.txt
-a----        13/11/2020    09:32             21 aa.txt
```

Il Wildcard parentesi quadra [] in dica i valori ammessi per il carattere

```
PS C:\temp> ls a[1a].txt


    Directory: C:\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        13/11/2020    09:32             21 a1.txt
-a----        13/11/2020    09:32             21 aa.txt
```

PS C:\temp> ls *.txt

```
    Directory: C:\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        13/11/2020    09:32             21 a.txt
-a----        13/11/2020    09:32             21 a1.txt
-a----        13/11/2020    09:32             21 a2.txt
-a----        13/11/2020    09:32             21 a3.txt
-a----        13/11/2020    09:32             21 a4.txt
-a----        13/11/2020    09:32             21 a5.txt
-a----        13/11/2020    09:32             21 aa.txt
-a----        16/11/2020    12:03              6 b.txt
-a----        04/02/2021    11:20           1179 List.txt
-a----        10/02/2021    23:06             48 MACHINES.txt
-a----        13/11/2020    08:58             21 Nuovo documento di testo.txt
-a----        10/01/2021    15:08            967 x.txt
```

```
PS C:\temp> ls a[1-3].txt


    Directory: C:\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        13/11/2020     09:32             21 a1.txt
-a----        13/11/2020     09:32             21 a2.txt
-a----        13/11/2020     09:32             21 a3.txt
```

Regular Expression: (System.Text.RegularExpressions.Regex)si usa gli operatori -match -nomatch e -replace
Pattern . qualsiasi carattere eccetto il newline

```
PS C:\ > "power" -match "po.er"
True
PS C:\ > "potwer" -match "po.er"
False
PS C:\ > "poster" -match "po.er"
False
PS C:\ > "poer" -match "po.er"
False
```

pattern [] Serie di car ammessi o non ammessi con ^ su un unico carattere:

```
PS C:\ > "power" -match "po[a-z]er"
True
PS C:\ > "power" -match "po[^0-9]er"
True
PS C:\ > "HAL9000" -replace "[0-9]","x"
HALxxxx
PS C:\ > "poster" -match "po[a-z]er"
False
```

PUNTI DI ANCORAGGIO (^ e $ esterni a [])
^ Inizio Riga, $ Fine Riga

```
PS C:\ > "the power" -match "po[a-z]er"
True
PS C:\ > "the power is here" -match "po[a-z]er"
True
PS C:\ > "power to me" -match "po[a-z]er"
True
True
False
PS C:\ > "the power" -match "po[a-z]er$"
True
PS C:\ > "power to me" -match "po[a-z]er$"
False
PS C:\ > "power" -match "^po[a-z]er$"
True
```

```
PS C:\ > "the power" -match "^po[a-z]er$"
False
PS C:\ > "the power" -notmatch "^po[a-z]er$"
True
PS C:\ > "power to me" -notmatch "^po[a-z]er$"
True
```

Quantificatori ?, *, +
? rende la sequenza che lo precede opzionale

Esempio cercare i processi explorer e iexplore

```
PS C:\ > get-process  | select name|? {$_ -match 'i?explorer?'} | SELECT NAME

Name
----
explorer
iexplore
iexplore
```

usando una coppia di parentesi tonde si impone di considerare piu' di un carattere

```
PS C:\ > "powershell" -match "^po(wer)?sh(ell)?$"
True
PS C:\ > "posh" -match "po(wer)?sh(ell)?$"
True
PS C:\ > "posh e' un acronimo di PowerShell" -replace 'po(wer)?sh(ell)?','xyz'
xyz e' un acronimo di xyz
```

Il quantificatore *rende la sequenza che lo precede ripetibile in modo indefinito

```
PS C:\ > "unoduetre" -match "uno*"
True
PS C:\ > "unoduetre" -match "unoo*"
True
PS C:\ > "unoduetre" -match "unooo*"
False
```

```
PS C:\temp > ls t*.txt
   Directory: C:\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        13/11/2020     09:32             21 Test.txt
-a----        13/11/2020     09:32             21 Test9x.txt
-a----        13/11/2020     09:32             21 Test9x9x.txt
-a----        13/11/2020     09:32             21 Test9x9x9x.txt

PS C:\ > Get-ChildItem |?{$_ -match '^Test(9x)*.txt$'}|select name

Name
```

```
----
Test.txt
Test9x.txt
Test9x9x.txt
Test9x9x9x.txt
```

Con il simbolo + la seq. che lo precede deve apparire almeno una volta nella stringa.

```
PS C:\ > Get-ChildItem |?{$_ -match '^Test(9x)+.txt$'}|select name

Name
----
Test9x.txt
Test9x9x.txt
Test9x9x9x.txt
```

Usando le parentesi {} il match si ha solo al verificarsi di un determinato numero di ripetizioni della sequenza della string da ricercare.

```
PS C:\ > 'abc123' -match '^[a-z]{5}[0-9]{3}$'
False
PS C:\ > 'abcdef123456' -match '^[a-z]{5}[0-9]{3}$'
False
PS C:\ > 'abcde123' -match '^[a-z]{5}[0-9]{3}$'
True
```

Se si vuole il num di ripetizioni in un range aggiungere nelle parentesi graffe la virgola,

```
PS C:\ > 'abc123' -match '^[a-z]{3,5}[0-9]{3,4}$'
True
PS C:\ > 'abcdefg123456' -match '^[a-z]{3,5}[0-9]{3,4}$'
False
PS C:\ > 'abcde123' -match '^[a-z]{3,5}[0-9]{3,4}$'
True
PS C:\ > 'abcdefg123456' -replace '[a-z]{3,5}[0-9]{3,4}','xxx'
abxxx56
```

CLASSI DI CARATTERE

\w	alfanum
\W	non alfanum
\d	cifra
\D	no cifra
\s	spazio bianco
\S	NO spazio bianco

Esempio check PIVA

```
PS C:\ > '01234567890' -match '^\d{11}$'
True
```

LE ALTERNANZE (|)

```
PS C:\ > 'http://www.powershell.it' -match '^(http|https|ftp)://.*$'
True
```

```
PS C:\ > 'ftp://10.0.1.26/xyz/abc' -match '^(http|https|ftp)://.*$'
True
PS C:\ > 'ztp://10.0.1.26/xyz/abc' -match '^(http|https|ftp)://.*$'
False
```

IL CARATTERE DI ESCAPE (\)
Esemio cercare i files di word e di excel:

```
PS C:\ > Get-ChildItem | ?{$_ -match '\.(doc|docx|xsl|xlsx)$'}|select name
```

LE SEQUENZE DI ESCAPE

\n	nuova riga
\t	tab
\r	ritorno a capo
\f	avanzam linea
\b	backsace
\0	$null
\a	avviso sonoro
\v	tab verticale

Esempio cerca il ritorno a capo:

```
PS C:\ > @"
>> 1234567
>> abcde
>> "@ -match "^\d{7}\n\w*$"
True
```

```
PS C:\ > @"
>> 1234567
>> abcde
>> "@ -match "\n"
True
```

I GRUPPI
Sono insiemi di caratteri racchiusi da parentesi tonde. il sistema estrae dalla stringa il testo di ciascun gruppo assegnandone un numero in base alla posizione del gruppo nell'espressione a partire da sinistra verso destra.
il gruppo numero zero corrisponde all'intera stringa. Ogni gruppo riconosciuto e' aggiunto all'array $matches.

```
PS C:\ > 'http://www.powershell.it/Forum.aspx' -match '^(http|ftp)://(.*)$'
True
PS C:\ > $matches

Name                           Value
----                           -----
2                              www.powershell.it/Forum.aspx
1                              http
0                              http://www.powershell.it/Forum.aspx
```

un'importante caratteristica dei gruppi e' la possibilita' di usare i valori recuperati dalla stringa da ricercare direttamente all'interno del testo di rimpiazzo con l'operatore -replace: ogni riferimento a questi valori e' composto dal dollaro ($)

seguito dal numero di gruppo desiderato.
Esempio manipolare una lista di url http, in maniera tale da ottenere una seconda lista in cui ciascun elemento faccia capo alla porta 8080

```
PS C:\ > $urls='http://example.com/x','http://example.com:9876','http://www.example.com/x/y/z'
PS C:\ > $urls -replace '^http://([^/:]*)(:\d*)?(/.*)?$','http://$1:8080$3'
http://example.com:8080/x
http://example.com:8080
http://www.example.com:8080/x/y/z
```

Acquisizioni denominate

Per impostazione predefinita, le acquisizioni o gruppi vengono archiviate in ordine numerico crescente, da sinistra a destra. È anche possibile assegnare un **nome** a un gruppo di acquisizione. Questo **nome** diventa una chiave nella `$Matches` variabile automatica **Hashtable** .

All'interno di un gruppo di acquisizione, usare `?<keyname>` per archiviare i dati acquisiti con una chiave denominata.

Esempio:

```
PS> $string = 'The last logged on user was CONTOSO\jsmith'
PS> $string -match 'was (?<domain>.+)\\(?<user>.+)'
True

PS> $Matches

Name                           Value
----                           -----
domain                         CONTOSO
user                           jsmith
0                              was CONTOSO\jsmith

PS> $Matches.domain
CONTOSO

PS> $Matches.user
jsmith
```

Select-String

Assomiglia alla findstr di Windows o a grep di Unix.
Tramite il parametro -Pattern il testo da ricercare e' una regex, mentre con -SimpleMatch no.
il contenuto da esaminare puo' essere una stringa (o un array di stringhe) da passare via pipeline oppure si esaminano piu' file usando eventualmente il parametro -Path.

```
PS C:\ > type test.txt
uno due
tre
quattro

PS C:\ > type test9x.txt
uno due
tre
quattro
```

```
tre

tre
fine


PS C:\ > Select-string 'tre' .\t*.txt  -Simplematch

Test.txt:2:tre
Test9x.txt:2:tre
Test9x.txt:4:tre
Test9x.txt:6:tre
Test9x9x.txt:2:tre
Test9x9x9x.txt:2:tre

#con -list ci limitiamo al primo match:
PS C:\ > Select-string 'tre' .\t*.txt  -Simplematch -list

Test.txt:2:tre
Test9x.txt:2:tre
Test9x9x.txt:2:tre
Test9x9x9x.txt:2:tre
```

FUNZIONI E FILTRI

E' possibile visualizzare a video le funzioni disponibili:

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

Definire Funzioni

```
function is-even($value){
        (%value %2) -eq 0
}

#oppure:
function is-even(){
        param($value)

($value %2) -eq 0
}
```

Chaiamare un funzione:

```
PS C:\ > is-even 12
True
PS C:\ > is-even
True
PS C:\ > is-even 5
False
```

Notare la chiamata senza argomenti. in questo caso il parametro mancante e' $null che viene automaticamente convertito a 0.
il parametro e' automaticamente denominato

```
PS C:\ > is-even -value 5
```

```
False
```

Tipizzazione dei parametri.

modifichiamo per un attimo la funzione eliminando il calcolo:

```
function is-even($value){

}
```

posso chiamare con un parametro carattere:
is-even u
se vogliamo tipizzare il parametro di input affinche' accetti solo interi:

```
function is-even([int32]$value){

}
```

```
is-even -value u
is-even : Impossibile elaborare la trasformazione degli argomenti nel parametro 'value'. Impossibile
convertire il valore "u" nel tipo "System.Int32". Errore: "Formato della stringa di input non
corretto."
In riga:1 car:16
+ is-even -value u
+              ~
   + CategoryInfo          : InvalidData: (:) [is-even], ParameterBindingArgumentTransformationExcep
  tion
   + FullyQualifiedErrorId : ParameterArgumentTransformationError,is-even
```

un parametro puo' essere preceduto dal termine [switch] per indicare nella funzione se il parametro e' stato passato

```
function myFunc([switch]$help){
        if($help){
                "help"
        }else{
                "quiet"
        }
}

PS C:\ > myfunc
quiet
PS C:\ > myfunc -help
help
```

un parametro puo avere un valore predefinito:

```
function myHello($to='world'){
        'hello ' + $to
}
PS C:\ > myhello
hello world
```

```
PS C:\ > myhello "my friends"
hello my friends
```

I Parametri posizionali:

```
function myTest(){
        foreach($arg in $args){
                echo $arg
        }
}
PS C:\ > mytest uno due "tre three" quattro
uno
due
tre three
quattro
```

Parametri obbligatori con la decorazione Mandatory

```
PS C:\ >  function sum(){
    Param
    (
        [Parameter(Mandatory=$true,HelpMessage="Inserire Il Primo operando")]
        [int] $a,
        [Parameter(Mandatory=$true,HelpMessage="Secondo Operando")]
        [int] $b
    )
 return $a+$b}
PS C:\ > sum

Cmdlet sum nella posizione 1 della pipeline dei comandi
Specificare i valori per i seguenti parametri:

Digitare !? per la Guida.
a: !?
Inserire Il Primo operando
a:

PS C:\ > sum 3 2
5
PS C:\ > sum 3
Cmdlet sum nella posizione 1 della pipeline dei comandi
Specificare i valori per i seguenti parametri:
b:
```

Parametri che possono accettare valori $null

```
function quad([parameter(Mandatory=$true)]$a){$a$a}
PS C:\ > quad $null
test : Impossibile associare l'argomento al parametro 'a' perché è null.
```

```
function quad([parameter(Mandatory=$true)][AllowNull()][int]$a){$a*$a}
PS C:\ > quad $null
```

```
0
```

Nel caso di parametri stringa posso specificare se il parametro puo’ essere una Stringa Vuota:

```
function StrStr([parameter(Mandatory=$true)][string]$a){$a + $a}
PS C:\ > StrStr ''
StrStr : Impossibile associare l'argomento al parametro 'a' perché è una stringa vuota.
```

```
PS C:\ > function StrStr([parameter(Mandatory=$true)][AllowEmptyString()][string]$a){$a + $a}
PS C:\ > strstr ''
```

Nel caso di parametri di tipo array si puo'specificare un minimo e un massimo di oggetti
function aTest([parameter(Mandatory=$true)][ValidateCount(3,5)]$a){$a}

```
PS C:\ > aTest 3,2,5
3
2
5
PS C:\ > aTest 3,2
aTest : Impossibile convalidare l'argomento sul parametro 'a'. Il numero di argomenti specificati (2)
```

Set di Parametri accettabili:
function aTest([ValidateSet('impiegato','titolare','contabile')]$a){$a}

```
PS C:\ > atest Titolare
titolare
PS C:\ > atest Boss
aTest : Impossibile convalidare l'argomento sul parametro 'a'. L'argomento "Boss" non appartiene al
set "impiegato;titolare;contabile" specificato dall'attributo ValidateSet. Fornire un argomento
incluso nel set ed eseguire di nuovo il comando.
```

Vincolo sulla lunghezza del parametro stringa.

```
PS C:\ > function sTest([ValidateLength(2,5)]$a){$a}
PS C:\ > sTest u
sTest : Impossibile convalidare l'argomento sul parametro 'a'. Il numero di caratteri (1)
dell'argomento è insufficiente. Specificare un argomento la cui lunghezza sia maggiore o uguale a "2"
ed eseguire di nuovo il comando.
In riga:1 car:7
+ sTest u
+       ~
    + CategoryInfo          : InvalidData: (:) [sTest], ParameterBindingValidationException
    + FullyQualifiedErrorId : ParameterArgumentValidationError,sTest

PS C:\ > stest uno
uno
```

Parametro stringa vincoli con la regex.

```
PS C:\ > function PIVA([ValidatePattern('^\d{11}$')]$a){$a}
function PIVA([ValidatePattern('^\d{11}$')]$a){$a}
PS C:\ > PIVA 122334
PIVA : Impossibile convalidare l'argomento sul parametro 'a'. L'argomento "122334" non corrisponde al
modello "^\d{11}$". Fornire un argomento che corrisponda a "^\d{11}$" ed eseguire di nuovo il comando.
```

```
PS C:\ > PIVA 12345678900
12345678900
```

Verifica su range:

```
PS C:\ > function TTest([ValidateRange(15,20)]$a){$a}
PS C:\ > TTest 3
TTest : Impossibile convalidare l'argomento sul parametro 'a'. L'argomento 3 è minore dell'intervallo
minimo consentito di 15. Fornire un argomento maggiore o uguale a 15 ed eseguire di nuovo il comando.
In riga:1 car:7
+ TTest 3
+       ~
   + CategoryInfo          : InvalidData: (:) [TTest], ParameterBindingValidationException
   + FullyQualifiedErrorId : ParameterArgumentValidationError,TTest

PS C:\ > TTest 19
19
```

Verifica con script d'espressione: Il primo parametro deve essere positivo e il secondo pari:

```
function Test([ValidateScript({$_ -gt 0})][int]$a,[ValidateScript({($_ %2) -eq 0})][int]$b){$a,$b}

PS C:\ > Test 5 6
5
6
PS C:\ > Test 5 7
test : Impossibile convalidare l'argomento sul parametro 'b'. Lo script di convalida "($_ %2) -eq 0"
per l'argomento con valore "7" non ha restituito il risultato True. Determinare il motivo per cui lo
script di convalida non è riuscito ed eseguire di nuovo il comando.
In riga:1 car:8
+ Test 5 7
+       ~
   + CategoryInfo          : InvalidData: (:) [test], ParameterBindingValidationException
   + FullyQualifiedErrorId : ParameterArgumentValidationError,test

PS C:\ > Test -5 6
test : Impossibile convalidare l'argomento sul parametro 'a'. Lo script di convalida "$_ -gt 0" per
l'argomento con valore "-5" non ha restituito il risultato True. Determinare il motivo per cui lo
script di convalida non è riuscito ed eseguire di nuovo il comando.
In riga:1 car:6
+ Test -5 6
+      ~~
   + CategoryInfo          : InvalidData: (:) [test], ParameterBindingValidationException
   + FullyQualifiedErrorId : ParameterArgumentValidationError,test
```

Esempio

```
function somma {
        Param ([int]$a,[int]$b)
        $c = $a + $b
        return $c
}
somma 3 2
```

INTERAGIRE CON LA PIPELINE

La lettura dei dati dalla pipeline e piu' complessa rispetto alla scrittura.
Gli oggetti presenti nella pipeline vengono forniti alla funzione nella variabile $input di tipo array.

Esempio Calcolo Radice Quadrata

```
PS C:\ > function sqr(){foreach($x in $input){[Math]::Sqrt($x)}}
PS C:\ > 4, 25, 33 | sqr
2
5
5,74456264653803
```

Le funzioni si possono suddividere in 3 blocchi begin,process,End che vengono richiamati :
Begin all'inizio dell'interazione con la pipeline
Process contiene l'algoritmo di elaborazione dei dati di input
End blocco da eseguire al termine dell'elaborazione della pipeline
Esempio Somma di oggetti provenienti dalla pipeline ($_ e' l'oggetto corrente presente nella pipeline):

```
function sum-object{
        Begin{
                $sum=$null
        }

        Process{
                $sum += $_
        }

        End{
                $sum
        }
}

PS C:\ > 3, 5, 10|sum-object
18
```

I Provider

In Powershell un provider fornisce un'interfaccia comune di accesso a determinati dati, e relativi contenitori in questo modo posso con gli stessi strumenti sia al file system oiuttosto che il registro ad esempio. Un modello provider estendibile fornisce la possibilità di accedere e manipolare non solo il file system, ma anche altre strutture dati gerarchiche. Ad esempio, PowerShell integra un provider per il Registro di Windows che consente l'accesso agli alberi "HKLM" (HKEY_LOCAL_MACHINE) e "HKCU" (HKEY_CURRENT_USER). In questo modo, il registro può essere visualizzato con comandi quali "dir HKLM:\SOFTWARE\Microsoft" dal prompt della shell. PowerShell fornisce provider per la libreria di certificati di sicurezza, le variabili d'ambiente e di shell, le funzioni e gli alias; gli utenti possono creare loro propri provider e integrarli in PowerShell.

Per ottenere l'array dei provider disponibili:

```
PS C:\ > get-psprovider

Name             Capabilities                   Drives
----             ------------                   ------
Registry         ShouldProcess, Transactions    {HKLM, HKCU}
Alias            ShouldProcess                  {Alias}
Environment      ShouldProcess                  {Env}
```

```
FileSystem          Filter, ShouldProcess, Credentials      {C, D, E}
Function            ShouldProcess                           {Function}
Variable            ShouldProcess                           {Variable}
WSMan               Credentials                             {WSMan}
```

Determiniamo il tipo restituito:

```
PS C:\ > (get-psprovider)[0].GetType().FullName
System.Management.Automation.ProviderInfo
```

L'output di Get-PsProvider include la proprieta' Name, che indica il nome del provider,
e la proprieta Drives, che contiene una collezione di drive gestiti da ciascun provider.

```
PS C:\ > Get-PSProvider | Select Name,PSSnapIn

Name       PSSnapIn
----       --------
Registry   Microsoft.PowerShell.Core
Alias      Microsoft.PowerShell.Core
Environment Microsoft.PowerShell.Core
FileSystem  Microsoft.PowerShell.Core
Function   Microsoft.PowerShell.Core
Variable   Microsoft.PowerShell.Core
WSMan
```

Quando PS gestisce un percorso assoluto ne determina il drive e da questo il provider da impiegare
Per recuperare la lista dei drive disponibili nel sistema si puo' usare il cmdlet Get-PSDrive:

```
PS C:\ > Get-PSDrive

Name         Used (GB)     Free (GB) Provider      Root                 CurrentLocation
----         ---------     --------- --------      ----                 ---------------
Alias                                  Alias
C              756,14       174,26 FileSystem    C:\
Cert                                   Certificate   \
D                                      FileSystem    D:\
E                0,00         0,00 FileSystem    E:\
Env                                    Environment
Function                               Function
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
Variable                               Variable
WSMan                                  WSMan
```

Drive Nativi di PS:

Drive	*Provider*	*Descrizione*
Alias	Alias	Catalogo degli alias
cert	Certificate	Catalogo dei certificati X.509 installati nel sistema
Function	Function	Function
HKCU	Registry	Registry di Windows, hive HKEY_CURRENT_USER
HKLM	Registry	Registry di Windows, hive HKEY_LOCAL_MACHINE

Variable	Variable	Catalogo delle variabili
WSMan	WSMan	Crdatalogo di configurazione di WS_Management
*:(A:,C:,...)	FileSystem	Unita' corrispondente del FileSystem

ogni drive dispone di un percorso corrente nella proprieta' CurrentLocaltion.

Il Provider FileSystem

per vedere i drive per le unita' disponibile nel file system

```
PS C:\ > Get-PSDrive | ? {$_.Provider.Name -eq 'FileSystem'}

Name         Used (GB)     Free (GB) Provider     Root                      CurrentLocat
                                                                            ion
----         ---------     --------- --------     ----                      ------------
C              756,14       174,26 FileSystem    C:\                       temp
D                          FileSystem    D:\
E               0,00         0,00 FileSystem    E:\
```

Il provider FileSystem gestisce inoltre l'accesso diretto ai percorsi di rete.
Il provider Registry non supporta l'accesso a un registro remoto. Ma per oviiare a cio'
e' possibile utilizzare il metodo statico OpenRemoteBaseKey() della classe Win32.RegistryKey
Il provider Certificate gestisce i certificati X.509 installati nella macchina locale
e permette il recupero immediato di tutti i dettagli, come gli enti emettitori e le
date di validita'. Il drive cert:, governato da questo provider, espone una struttura
gerarchica simile a quella esibita nella console MMC dallo snap-in Certificati.

Get-Location - Ritorna il percorso di lavoro corrente

```
PS C:\ > (Get-Location).gettype().Fullname
System.Management.Automation.PathInfo
```

Esempi

```
PS C:\temp> Get-Location

Path
----
C:\temp
```

```
PS C:\ > (Get-Location).Provider

Name             Capabilities                    Drives
----             ------------                    ------
FileSystem       Filter, ShouldProcess, Credentials      {C, D, E}
```

Set-Location - Imposta il percorso di lavoro corrente
Imposta il percorso di lavoro corrente, che puo' essere relativo (.,..,\)

```
PS C:\temp> set-location c:\windows
PS C:\windows>
```

In quest'esempio, invece, la posizione di lavoro corrente e' una chiave registry di Windows:

```
PS C:\windows> set-Location HKLM:\SYSTEM\CurrentControlSet
```

```
PS HKLM:\SYSTEM\CurrentControlSet>
```

Noare la posizione impostata dal provider nel prompt.

Get-ChildItem - Recuperare gli elementi
Sintassi
Get-ChildItem <Percorso> [-Force][-Recurse]
Questo cmdlet ritorna tipi di oggetto diversi a seconda del provider utilizzato e dell'elemento ritornato.
Ad esempio il Provider di tipo System.IO.FileSystem ritorna oggetti FileInfo o System.IO.DirectoryInfo,
Il provider Restry ritorna oggetti Microsoft.Win32.RegistryKey, mentre il provider Certificate ritorna
elementi di tipo X509Certificate2 per i certificati e X509Store per i contenitori, entrambi appartenenti
al namespace Microsoft.WSMAn.Management. Per i provider non gerarchici come Variable, Alias, Function
e Environment,e' sufficiente specificare il percorso del drive desiderato.
Esempi:

```
PS C:\ > Get-ChildItem -Path C:\ -Filter pagefile.sys -Force
PS C:\ > Get-ChildItem C:\Windows\System32
PS C:\ > Get-ChildItem HKLM:\SYSTEM\CurrentControlSet\Control
PS C:\ > Get-ChildItem env:
PS C:\ > Get-ChildItem Function:
```

Get-ChildItem prevede il parametro -Force per ritornare gli elementi nascosti. Mentre
con il parametro -Recurse si chiede un recupero ricorsivo degli elementi.
Esempio nello script che segue si recuperano tutti i certificati e gli store X.509 della macchina locale:

```
PS C:\ > Get-ChildItem cert:\LocalMachine -Recurse

Name : TestSignRoot

Subject      : CN=Microsoft ECC Testing Root Certificate Authority 2017, O=Microsoft Corporation,
               L=Redmond, S=Washington, C=US
Issuer       : CN=Microsoft ECC Testing Root Certificate Authority 2017, O=Microsoft Corporation,
               L=Redmond, S=Washington, C=US
Thumbprint   : A4B37F4F6DE956922273D5CB8E7E0AAFB7033B90
FriendlyName : Microsoft ECC Testing Root Certificate Authority 2017
NotBefore    : 09/11/2017 20:53:14
NotAfter     : 09/11/2042 21:01:55
Extensions   : {System.Security.Cryptography.Oid, System.Security.Cryptography.Oid,
               System.Security.Cryptography.Oid, System.Security.Cryptography.Oid...}
.....................
```

Get-ChildItem ha l'alias ls per similitudine con Linux

Get-Item E' un cmdlet utilizzato per recuperare il singolo elemento.
Esempio

```
PS C:\ > Get-Item c:\windows\System32\Notepad.exe


    Directory: C:\windows\System32


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        15/01/2021     12:14         202240 Notepad.exe
```

Manipolare gli Elementi_

New-Item, Remove-Item, Rename-Item, Copy-Item, Move-Item, Set-Item, Invoke-Item
Sintassi e Esempi:
New-Item <Percorso> [-ItemType <Tipo>][-Value <Valore>][-Force]

```
PS C:\ > New-Item .\TestDir -ItemType Directory
PS C:\ > New-Item .\TestDir\Test.txt -ItemType File -Value "Hello, world"
PS C:\ > New-Item HKCU:\TestKey -Value "Hello, World"

PS C:\ > Remove-Item <Percorso> [-ItemType <Tipo>][-Recurse][-Force]
PS C:\ > Remove-Item HKCU:\TestKey
PS C:\ > Remove-Item .\TestDir -Recurse
PS C:\ > Rename-Item <Percorso> -NewName <Nome> [-Force]
PS C:\ > Rename-Item .\Test.txt -NewName HelloWorld.txt
PS C:\ > $x=123
PS C:\ > Rename-Item Variable:\x -NewName "y"
PS C:\ > $x
PS C:\ > $y
123
```

Copy-Item <Sorgente> <Destinazione>[-Recurse][-Force]

```
PS C:\ > Copy-Item .\Test.txt .\TestDir
```

Move-Item <Sorgente> <Destinazione>[-Force]
In questo cmdlet -Force forza la sovrascrittura senza generare errori

```
PS C:\ > New-Item HKCU:\Software\TestKey1\Xyz -Value "test" -Force
PS C:\ > New-Item HKCU:\Software\TestKey2 -Type Directory
PS C:\ > Move-Item HKCU:\Software\TestKey1\* HKCU:Software\TestKey2
```

Set-Item <Percorso> <Valore> [-Force]
In questo cmdlet -Force forza la sovrascrittura senza generare errori
In questo script Set-Item modifica il corpo della funzione C:

```
PS C:\ > Set-Item Function:C: {Write-Host "hello world"}
PS C:\ > c:
```

hello world

Invoke-Item <Percorso>
Invoke-item esegue l'operazione predefinita associata al percorso specificato
(nello Script seguente si aprirebbe Notepad)

```
PS C:\ > Invoke-Item .\Test.txt
```

Proprieta' degli Elementi
E possibile leggere e modificare le proprieta' degli elementi forniti dai provider.
Get-ItemProperty
  [-Path] <String[]>
  [[-Name] <String[]>]
  [-Filter <String>]
Ad esempio si recupera la versione di Windows installata.

```
PS C:\ > Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\'ProductName
PS C:\ > Get-ItemProperty -Path C:\ -Filter pagefile.sys LastWriteTime
```

Set-ItemProperty
Sintassi:

Set-ItemProperty <Percorso> <Proprieta><Valore> [-Force]
Esempio: Imposto a ReadOnly il File c:\temp\test.txt

```
PS C:\ temp> ls Test.txt
   Directory: C:\temp
Mode            LastWriteTime       Length Name
----            -------------       ------ ----
-a----      13/11/2020    09:32           21 Test.txt

PS C:\temp> set-itemproperty -path c:\temp\test.txt -name IsReadOnly -value $true
PS C:\temp> ls Test.txt


   Directory: C:\temp


Mode            LastWriteTime       Length Name
----            -------------       ------ ----
-ar---      13/11/2020    09:32           21 Test.txt
```

New-ItemProperty – Crea nuovi elementi
New-ItemProperty
  [-Path] <String[]>
  [-Name] <String>
  [-PropertyType <String>]
  [-Value <Object>]
  [-Force]
  [-Filter <String>]

Esempio

```
PS C:\ > New-ItemProperty -Path "HKLM:\Software\MyCompany" -Name "NoOfEmployees" -Value 822
```

Esempio abilita il log del caricamento degli assembly di Microsoft .NEY,
aggiungendo una specifica voce al registry di Windows:

```
PS C:\ > New-ItemProperty HKLM:\Software\Microsoft\Fusion EnableLog -Value 1
```

Remove-ItemProperty – Rimuove elementi
Remove-ItemProperty
    [-Path] <String[]>
    [-Name] <String[]>
    [-Force]
    [-Filter <String>]
Esempio rimuove la voce EnableLog dal Registro di Windows:

```
PS C:\ > Remove-ItemProperty HKLM:\Software\Microsoft\Fusion EnableLog
```

Rename-ItemProperty Rinomina elementi
Rename-ItemProperty
    [-Path] <String>
    [-Name] <String>
    [-NewName] <String>
    [-PassThru]
    [-Force]
    [-Filter <String>]
Esempio rinomina una chiave del registro

```
PS C:\ > Rename-ItemProperty -Path HKLM:\Software\SmpApplication -Name config -NewName oldconfig
```

Copy-ItemProperty – Copia elementi
Copy-ItemProperty
[-Path] <String[]>
[-Name] <String>
[-Destination] <String>
[-PassThru]
[-Force]
[-Filter <String>]
La destinazione deve esistere
Esempio copiare il nome del driver della porta seriale di Windows dalla sua chiave del registry a un'altra:

```
PS C:\ > Copy-ItemProperty HKLM:\SYSTEM\CurrentControlSet\ServicesSerial HKCU:\TestKey DisplayName
```

Move-ItemProperty -
Move-ItemProperty
[-Path] <String[]>
[-Name] <String[]>
[-Destination] <String>
[-PassThru]
[-Force]
[-Filter <String>]

Esempio spostare un paio di elementi della lista dei comandi avviati di recente (MRU) tramite la finestra di dialogo Esegui di Windows dalla loro posizione nel registry in un'altra:

```
PS C:\ > Move-ItemProperty HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU
HKCU:\Test\MRUList
```

Switch WhatIf
Quando si specifica questo switch il cmdlet non viene eseguito, ma viene mostrato a video la descrizione dell'operazione che il comando avrebbe altrimenti effettuato
Esempio

```
PS C:\ > Copy-Item c:\temp\a.txt c:\temp\my -WhatIf
```

WhatIf: Esecuzione dell'operazione "Copia file" sulla destinazione "Elemento: C:\temp\a.txt Destinazione: C:\temp\my\a.txt".

Switch -Confirm
funziona come lo switch WhatIf ma chiede conferma all'utente prima di proseguire:

```
PS C:\ > Copy-Item c:\temp\a.txt c:\temp\my -Confirm

Conferma
Eseguire l'operazione?
Esecuzione dell'operazione "Copia file" sulla destinazione "Elemento: C:\temp\a.txt Destinazione:
C:\temp\my\a.txt".
[S] Sì  [T] Sì a tutti  [N] No  [U] No a tutti  [O] Sospendi  [?] Guida
(il valore predefinito è "S"):
```

WMI

WMI (Windows Management Instrumentation) e' l'implementazione MS di WBEM x l'automazione e la gestione di Windows. DEVE essere attivo il servizio WinMgmt, e i componenti DCOM e RPC e che le relative

porte non siano bloccate da Firewall. Da remoto occorre essere ammn sulla macchina di destinazione. Ogni modulo in grado di supportare qualche classe WMI prende il NOME di provider WMI e fa capo ad una dll memorizzata nella cartella %SystemRoot%System32\wbem. Il "Common Information Model" (CIM) è uno standard open source per l'accesso e la visualizzazione delle informazioni su un computer. È uno standard del settore che esiste da molti anni, ma non include alcun metodo per accedere ai dati su un computer remoto. WMI è la versione Microsoft di CIM. Microsoft ha aggiunto DCOM e RPC al framework di gestione CIM insieme ad altre piccole modifiche e lo ha chiamato Windows Management Interface. WMI era la soluzione di Microsoft per come utilizzare CIM su computer remoti su una rete. Il problema principale contro WMI è che non è molto compatibile con i firewall. Devi fare un sacco di grossi buchi in un firewall per farlo funzionare.

Visualizzare i Namespace WMI

```
PS C:\> Get-WmiObject -Namespace Root -Class __Namespace | Select-Object -Property Name
#oppure:
PS C:\> Get-CimInstance -Namespace Root -ClassName __Namespace
```

Il namespace di default e' ROOT\CIMV2 il contenuto puo' essere visualizzato nel modo seguente:

```
PS C:\ > Get-WmiObject -Namespace "ROOT\CIMV2" -List

   NameSpace: ROOT\CIMV2

Name                          Methods           Properties
----                          -------           ----------
__SystemClass                 {}                {}
__thisNAMESPACE               {}                {SECURITY_DESCRIPTOR}
__Provider                    {}                {Name}
__Win32Provider               {}                {ClientLoadableCLSID, CLSID, Concurrency,...
__ProviderRegistration        {}                {provider}
……..
```

Esempio quanta RAM fisica e' installata su una macchina?:
Cerco la Win32_ che si occupa della memoria tramite l'operatore like:

```
PS C:\ > Get-WmiObject -Namespace root\CIMV2 -List | Where-Object {$_.name  -Like
"Win32_*Memory*"}

   NameSpace: ROOT\cimv2

Name                          Methods           Properties
----                          -------           ----------
Win32_CacheMemory             {SetPowerState, R... {Access, AdditionalErrorData, Associativi...
Win32_SMBIOSMemory            {SetPowerState, R... {Access, AdditionalErrorData, Availabilit...
Win32_MemoryArray             {SetPowerState, R... {Access, AdditionalErrorData, Availabilit...
Win32_MemoryDevice            {SetPowerState, R... {Access, AdditionalErrorData, Availabilit...
Win32_SystemMemoryResource    {}                {Caption, CreationClassName, CSCreationCl...
Win32_DeviceMemoryAddress     {}                {Caption, CreationClassName, CSCreationCl...
Win32_PhysicalMemory          {}                {Attributes, BankLabel, Capacity, Caption...
Win32_PhysicalMemoryArray     {IsCompatible}    {Caption, CreationClassName, Depth, Descr...
Win32_AssociatedProcessorMemory {}              {Antecedent, BusSpeed, Dependent}
Win32_MemoryDeviceLocation    {}                {Antecedent, Dependent}
……..
```

A questo punto posso cooscere la capacita' della memoria fisica.

```
PS C:\ > get-wmiobject win32_physicalmemory|select capacity
```

```
 capacity
 --------
8589934592
8589934592
#oppure:
PS C:\ > get-wmiobject win32_physicalmemory|measure-object Capacity -sum

Count    : 2
Average  :
Sum      : 17179869184
Maximum  :
Minimum  :
Property : Capacity
Oppure:
PS C:\ > (Get-WmiObject Win32_PhysicalMemory | measure-object Capacity -sum).sum/1gb
16
```

Ottenere informazioni sui Dischi
Get-wmiObject Win32_DiskPartition
#oppure remoto:
Get-wmiObject Win32_DiskPartition -ComputerName [<IP>|NETBIOS NAME\FQDN]

```
PS C:\ > Get-wmiObject Win32_DiskPartition


NumberOfBlocks   : 204800
BootPartition    : True
Name             : Disco #0, partizione #0
PrimaryPartition : True
Size             : 104857600
Index            : 0

NumberOfBlocks   : 1951186944
BootPartition    : False
Name             : Disco #0, partizione #1
PrimaryPartition : True
Size             : 999007715328
Index            : 1

NumberOfBlocks   : 2097152
BootPartition    : False
Name             : Disco #0, partizione #2
PrimaryPartition : False
Size             : 1073741824
Index            : 2

PS C:\ > (Get-wmiObject Win32_DiskPartition).__SERVER
LAPTOP-POJRGMSE
LAPTOP-POJRGMSE
LAPTOP-POJRGMSE
oppure:
Get-wmiObject Win32_DiskPartition | select-object __server
```

```
__SERVER
--------
LAPTOP-POJRGMSE
LAPTOP-POJRGMSE
LAPTOP-POJRGMSE

PS C:\ > (Get-wmiObject Win32_DiskPartition).__namespace
root\cimv2
root\cimv2
root\cimv2
PS C:\> (Get-wmiObject Win32_DiskPartition).__class
Win32_DiskPartition
Win32_DiskPartition
Win32_DiskPartition
PS C:\ > (Get-wmiObject Win32_DiskPartition).__PATH
\\LAPTOP-POJRGMSE\root\cimv2:Win32_DiskPartition.DeviceID="Disk #0, Partition #0"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_DiskPartition.DeviceID="Disk #0, Partition #1"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_DiskPartition.DeviceID="Disk #0, Partition #2"
```

Ottenere informazioni sulle Stampanti

```
PS C:\ > Get-wmiobject Win32_printer

PS C:\ > Get-wmiobject Win32_printer | Select-object __PATH

__PATH
------
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="OneNote for Windows 10"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="OneNote (Desktop)"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="Microsoft XPS Document Writer"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="Microsoft Print to PDF"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="HPCE57F8 (HP Officejet Pro X476dw MFP)"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="HP Laser 103 107 108"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="Fax"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="ele2-p1"

#oppure:
PS C:\ > Get-wmiobject Win32_printer | Select __PATH

__PATH
------
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="OneNote for Windows 10"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="OneNote (Desktop)"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="Microsoft XPS Document Writer"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="Microsoft Print to PDF"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="HPCE57F8 (HP Officejet Pro X476dw MFP)"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="HP Laser 103 107 108"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="Fax"
\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="ele2-p1"
```

Ottenere informazioni sulla CPU

```
PS C:\ > switch((get-wmiobject Win32_processor).Architecture )
    {
```

```
      0       {'x86'}
      1       {'MIPS'}
      2       {'Alpha'}
      3       {'PowerPC'}
      6       {'ia64'}
      9       {'x64'}
      default   {"{$_}"}
    }
x64
```

Ottenere informazioni sulle Schede di rete
Esempio vediamo i nomi delle interfacce di rete:

```
$interfaces = Get-WmiObject Win32_NetworkAdapter
$interfaces | foreach {
   $friendlyname = $_ | Select-Object -ExpandProperty NetConnectionID
   $name = $_.GetRelated("Win32_PnPEntity") | Select-Object -ExpandProperty Name
   "$friendlyname is $name"
}

echo "oppure:`n"
$interfaces = Get-WmiObject Win32_NetworkAdapter
$Outputs = @()
foreach ($Interface in $Interfaces) {
   $Output = New-Object -TypeName System.Object
   $Output | Add-Member -MemberType NoteProperty -Name FriendlyName -Value
$Interface.NetConnectionID
   $Output | Add-Member -MemberType NoteProperty -Name Name -Value $Interface.Name
   $Outputs += $Output
}

echo $Outputs

echo "oppure:`n"
$interfaces = Get-WmiObject Win32_NetworkAdapter
$interfaces | ForEach{
   $friendlyname = $_ | ForEach-Object { $_.NetConnectionID }
   $name = $_.GetRelated("Win32_PnPEntity") | Select-Object -ExpandProperty Name
   # This tests to ensure friendlyname isn't null
   if($friendlyname){
      "$friendlyname is $name"
   }
}

is Microsoft Kernel Debug Network Adapter
Wi-Fi is Qualcomm Atheros QCA9377 Wireless Network Adapter
Ethernet is Realtek PCIe GBE Family Controller
 is Microsoft Wi-Fi Direct Virtual Adapter #6
Connessione di rete Bluetooth 2 is Bluetooth Device (Personal Area Network) #2
 is Microsoft Wi-Fi Direct Virtual Adapter #7
 is WAN Miniport (SSTP)
……
```

Solo la lista dei dispositivi fisici

```
$query = "SELECT * FROM Win32_NetworkAdapter WHERE Manufacturer != 'Microsoft' AND NOT
PNPDeviceID LIKE 'ROOT\\%'"
$interfaces = Get-WmiObject -Query $query | Sort index
$interfaces | ForEach{
   $friendlyname = $_ | ForEach-Object { $_.NetConnectionID }
   $name = $_.GetRelated("Win32_PnPEntity") | Select-Object -ExpandProperty Name
   # This tests to ensure friendlyname isn't null
   if($friendlyname){
      "$friendlyname is $name"
   }
}
```

Le schede di rete di tipo ethernet

```
PS C:\ > get-wmiobject Win32_NetworkAdapter |?{$_.AdapterType -like 'eth*'}
```

IL filtro sulle richieste viene applicato a valle. se voglio applicare il filtro a monte
posso usare un linguaggio WQL SQl like

```
PS C:\ > Get-wmiObject -Query "select Name FROM Win32_UserAccount WHERE Disabled=FALSE"
```

Il type accelerator [WMI] converte un path che identifica un'istanza di una classe WMI nell'oggetto .NET che la rappresenta.
Esempio Formati di carta supportati da una stampante:

```
PS C:\ > $prt=[WMI]'\\LAPTOP-POJRGMSE\root\cimv2:Win32_Printer.DeviceID="HP Laser 103 107 108"'
PS C:\ > $prt.printerPaperNames
A4
A3
A5
B4 JIS
B5 JIS
Busta C5
Busta C4
Busta DL
Busta n. 10
Busta Monarch
Lettera
11x17
Legale
Executive
Oficio 8.5x13
Oficio 216x340 mm
4x6
Formato personalizzato
```

```
PS C:\ > [WMI]

IsPublic IsSerial Name                          BaseType
-------- -------- ----                          --------
True     True     ManagementObject              System.Management.ManagementBaseObject


PS C:\ > [WMISearcher]
```

```
IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    ManagementObjectSearcher                 System.ComponentModel.Component
```

Esempio Servizi automatici ma non avviati

```
$search=[WMISearcher] 'Select * from Win32_service WHERE StartMode="Auto" AND State!="Running"'
$search.Get()
```

Metodi d'istanza

```
$printer=[WMI]'Win32_Printer.DeviceID="HP Printer LaserJet 1000"'
$printer.PrintTestPage()
```

Metodo statico

```
([WMIClass] 'Win32_Process').create('calc.exe',$null,$null)
```

Per invocare un metodo independemente che sia d'istanza o statico si usa Invoke-wmiMethod
Esempio

```
Invoke-wmiMethod -Path 'Win32_Process' -Name Create -ArgumentList 'calc.exe'
```

Remove-WmiObject
Esempio termina tutti i processi associati alla calcolatrice:

```
PS C:\ > Get-WmiObject -Query "SELECT * from win32_process  where name='calculator.exe'" | remove-wmiobject
```

Set-WmiInstance
Modifica proprieta' di un oggetto:
Esempio modificare l'etichetta di un HD:

```
PS C:\ > Set-WmiInstance -Path 'Win32_Volume.DeviceID="\\\\?\\Volume(12345678-1122-4433-a344-121323432}\\"' -Arguments @{Label='Disco Rigido'}
```

Oppure in modo + semplice, prendendo il volume del disco:

```
PS C:\ > Get-WmiObject -Query 'SELECT * FROM Win32_Volume WHERE DriveLetter="C:"' | Set-WmiInstance -Arguments @{Label="Disco Rigido"}
```

Lista delle codifiche supportate dal sistema:

```
PS C:\ > [Text.Encoding]::GetEncodings() | select name,displayname

Name              DisplayName
----              -----------
IBM037            IBM EBCDIC (Stati Uniti-Canada)
IBM437            OEM Stati Uniti
IBM500            IBM EBCDIC (internazionale)
ASMO-708          Arabo (ASMO 708)
DOS-720           Arabo (DOS)
.....

#Unicode
function Get-EncodedBytes([String]$encoding,[string]$value){
        $bytes = [Text.encoding]::GetEncoding($encoding).GetBytes($value)
        ($bytes | % {'{0:x2}' -f $_}) -join ' '
```

```
}

Get-EncodedBytes 'ISO-8859-1' 'cioè'
Get-EncodedBytes 'UTF-8' 'cioè'
Get-EncodedBytes 'Unicode' 'cioè'

3 69 6f e8          #NB il carattere è non compreso nell'ascii a 7 bit viene usato anche l'ottavo bit
63 69 6f c3 a8    #usa 2 bytes per codificare è
63 00 69 00 6f 00 e8 00 #usa 16 bit, 2 bytes per ogni carattere
```

I FILE (di testo e binari)

Lettura da File: Get-Content
Esempio legge un file ed estrae le sole righe che terminano per elle

```
PS C:\ > Get-content .\test.txt -match 'elle\S?$
```

Esempio legge le prime 3 righe:

```
PS C:\ > Get-content .\test.txt -TotalCount 3
```

Differenze file di testo e binario

```
PS C:\ > 1,2,3|set-content a
PS C:\ > type a
1
2
3
PS C:\ > [byte]1, [byte]2, [byte]3 |set-content aa -Encoding Byte
PS C:\ > type aa
😊☻❤️
```

Aggiungere elementi ad iun file:Add-Content
Svuotare un file:Clear-Content
il cmdlet out-file e' uno dei comandi di out (solo su file testuali) della pipeline l'uscita del comando utilizza una rappresentazione testuale identica a quanto visualizzato sulla console dell'host

```
PS C:\ > ps |out-file aaa
```

confrontare con:

```
PS C:\ > ps |select-object name |set-content aaa
PS C:\ > ps |select-object -Expand name |set-content aaa
```

Reindirizzare l'output
operatore >
operatore >>
operator 2>
operatore 2>>
operatore 2>&1

```
PS C:\ > remove-item xxxx >z
remove-item : Impossibile trovare il percorso 'C: \xxxx' perché non esiste.
In riga:1 car:1
+ remove-item xxxx >z
+ ~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (C: \xxxx:String) [Remove-Item], ItemNotFoundExcep
   tion
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.RemoveItemCommand
```

```
PS C:\ > remove-item xxxx 2>z
PS C:\ > type z
remove-item : Impossibile trovare il percorso 'C: \xxxx' perché non esiste.
In riga:1 car:1
+ remove-item xxxx 2>z
+ ~~~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : ObjectNotFound: (C:\\xxxx:String) [Remove-Item], ItemNotFoundExcep
  tion
   + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.RemoveItemCommand
```

PROCESSI E SERVIZI

Avviare un processo: start-process(start)
Esempio

```
PS C:\ > Start-process notepad
#oppure per associazione:
PS C:\ > Start-process test.txt
```

Ottenere informazioni sui processi

```
PS C:\ > new-object system.diagnostics.processstartinfo arduinopwmfreq.docx
PS C:\ > get-process (ps | gps)
PS C:\ > get-process notepad
PS C:\ > get-process -id 1231,2342 -ComputerName 10.0.1.2
PS C:\ > get-process notepad|select  cpu,workingset,mainwindowtitle
#Lista i Processi con UI bloccata:
PS C:\ > get-process | ? {$_.Responding -eq $false}

PS C:\ > (get-process notepad)[0].gettype().fullname
System.Diagnostics.Process
```

Alcune Proprieta' dell'oggetto System.Diagnostics.Process:
ExitCode:Codice uscita processo
ExitTime:Data e Ora termine processo
ID:PID
MainModule:Modulo principale del processo
Modules:Collection dei moduli del processo
ProcessName:Nome del processo
ProcessorAffinity:Mask di affinita' con la CPU
Responding:UI bloccata o no
StandardError,StandardInput,StandardOutput:rif ai canali std
StartInfo:Contiene Info sul lancio e la configurazione del processo
Threads:collezioni dei Threads del processo
WorkingSet:Dimensione totale della memoria impegnata

Alcune proprieta' di System.Diagnostics.ProcessModule
FileName:Il nome del file del modulo
FileVersionInfo:Dettagli sulla versione e produttore del modulo
ModuleMemorySize:memoria occupata dal modulo
ModuleName:Solitamente il nome del file senza path.

Alcune proprieta' di System.Diagnostics.FileVersionInfo

CompanyName:Nome del produttore
FileVersion:Versione del File
FileName:Nome del file
Language:la lingua delle risorse ccontenute nel file
ProductName:Nome del prodotto associato al file
ProductVersion:Versione del prodotto associato al file
Esempio seleziona i soli processi creati da Microsoft

```
PS C:\ > get-process -FileVersionInfo|?{$_.CompanyName -like 'Microsoft*'}|select filename >z
```

Esempio visualizzare i moduli (dll) che la calcolatrice di windows utilizza

```
PS C:\ > get-process calculator  -module | select filename
FileName
--------
C:\Program
Files\WindowsApps\Microsoft.WindowsCalculator_10.2012.21.0_x64__8wekyb3d8bbwe\Calculato...
C:\WINDOWS\SYSTEM32\ntdll.dll
C:\WINDOWS\System32\KERNEL32.DLL
C:\WINDOWS\System32\KERNELBASE.dll
C:\WINDOWS\System32\combase.dll
C:\WINDOWS\System32\ucrtbase.dll
C:\WINDOWS\System32\RPCRT4.dll
C:\WINDOWS\System32\OLEAUT32.dll
.....
```

Fermare uno o piu' processo stop-process (alias kill):
Esempio termina i notepad:

```
PS C:\ > get-process notepad | stop-process
```

Attesa wait-process
La shell attende la terminazione di uno o piu' processi locali identificati da -id o -Name
Si puo' usare -Timeout.
Esempio si attende indefinitamente la chiusura di tutte le istanze di notepad

```
PS C:\ > Get-process notepad | wait-process
```

I SERVIZI

Ottenere info su uno o piu' servizi locali o remoti Get-Service
Synthax:
Get-Service [<Nome>] [-ComputerName <Computer>]
Get-Service -DisplayName[<NomeVisualizzato>] [-ComputerName <Computer>]
Esempio si recuperano tutti i servizi il cui nome visualizzato inizi con windows:

```
PS C:\ > Get-Service -DisplayName Windows*

Status   Name               DisplayName
------   ----               -----------
Running  IpOverUsbSvc       Windows Phone IP over USB Transport...
Stopped  MixedRealityOpe... Windows Mixed Reality OpenXR Service
Running  mpssvc             Windows Defender Firewall
Stopped  msiserver          Windows Installer
Running  SDRSVC             Windows Backup
Stopped  WaaSMedicSvc       Windows Update Medic Service
Stopped  wcncsvc            Windows Connect Now - Registro conf...
```

```
Running  WSearch          Windows Search
Stopped  wuauserv         Windows Update
```

Esempio ottenere informazioni sul servizio w3svc (servizio di pubblicazione sul web) di una macchina remota (via DCOM/RPC):

```
PS C:\ > Get-Service w3svc -ComputerName 192.168.0.41|select Name,MachineName,Status
```

Gli oggetti ritornati da get-service sono di tipo System.ServiceProcess.ServiceController.
Alcune proprieta' di System.ServiceProcess.ServiceController:
DisplayName: Nome descrittivo del servizio, come viene visualizzato in services.msc
MachineName: Nome del PC in cui il servizio e' installato.
ServiceName: Nome identificativo del servizio
Status: stato corrente del servizio

Possibili valori di Status:
Stopped:Servizio arrestato
StartPending:Servizio sta per essere avviato
StopPending: il servizio sta per essere arrestato
Running:Servizio avviato
ContinuePending:
PausePending:Il servizio sta per essere messo in pausa
Paused: il servizio e' in Pausa

Alcuni methodi di System.ServiceProcess.ServiceController
Continue: Continua l'esecuzione del processo (in seguito a una pausa)
Pause: mette in pausa il servizio
Refresh: Aggiorna le prop. dell'oggetto in base allo stato attuale del servizio
Start: avvia il servizio
Stop:arresta il servizio
WaitForStatus($Status):Sospende l'esecuzione dello script fino a quando il servizio non e' nello stato indicato

Dipendenze del servizio:

```
PS C:\ > get-service w3svc -RequiredServices

Status   Name           DisplayName
------   ----           -----------
Running  WAS            Servizio Attivazione processo Windows
Running  HTTP           Servizio HTTP
```

Esempio servizi che dipendono dal servizio HTTP:

```
PS C:\ > get-service HTTP -DependentServices

Status   Name           DisplayName
------   ----           -----------
Stopped  WMPNetworkSvc     Servizio di condivisione in rete Wi...
Stopped  WinRM             Gestione remota Windows (WS-Managem...
Stopped  Wecsvc            Raccolta eventi Windows
Running  W3SVC             Servizio Pubblicazione sul Web
Stopped  w3logsvc          Servizio di registrazione W3C
Stopped  upnphost          Host di dispositivi UPnP
Running  SSDPSRV           Individuazione SSDP
Stopped  Fax               Fax
```

```
Running  Spooler          Spooler di stampa
Stopped  RemoteAccess     Routing e Accesso remoto
Stopped  FDResPub         Pubblicazione risorse per individua...
Stopped  fdPHost          Host provider di individuazione fun...
```

start-service
Esempio avvio lo spooler di stampa sul PC remoto (bisogna essere amministratori):

```
PS C:\ > get-service spooler -ComputerName 192.168.1.43|start-service\
```

stop-service
Esempio si recuperano i servizi che dipendono dallo spooler di stampa (es serviio FAX) e si arrestano:

```
PS C:\ > get-service spooler -DependentService | stop-service
```

suspend-service
resume-service
Esempio si mette in pausa il servizio Strumentazione gestione windows(Winmgmt) per poi continuarne l'esecuzione:

```
PS C:\ > get-service Winmgmt | suspend-service
PS C:\ > get-service Winmgmt | resume-service
```

restart-service
arresta e riavvia uno o piu' servizies.
Esempio Riavvia il servizio w3svc

```
PS C:\ > get-service w3svc | restart-service
```

set-service
modifica qlcune info sul servizio ed eventualmente cambiarne lo stato
Esempio si rende manuale la modalita' di avvio di windows update(wuauserv):

```
PS C:\ > get-service wuauserv | set-service -startupType Manual

Automatic
Manual
Disabled
```

IL REGISTRO EVENTI DI WINDOWS

Get-EventLog (classic)
Get-winEvent (new)

```
PS C:\ > Get-EventLog -List

  Max(K) Retain OverflowAction       Entries Log
  ------ ------ --------------       ------- ---
     512      7 OverwriteOlder            0 ACEEventLog
  20.480      0 OverwriteAsNeeded    13.566 Application
  20.480      0 OverwriteAsNeeded         0 HardwareEvents
     512      7 OverwriteOlder            0 Internet Explorer
  20.480      0 OverwriteAsNeeded         0 Key Management Service
     128      0 OverwriteAsNeeded        85 OAlerts
                                    Security
  20.480      0 OverwriteAsNeeded    10.415 System
     512      7 OverwriteOlder            0 Windows Azure
```

```
 15.360     0 OverwriteAsNeeded       955 Windows PowerShell
```

get-eventlog ritorna la lista dei registri disponibii sulla macchina sotto forma di oggetti System.Diagnostics.EventLogQuindi si puo' interrogare un registro ed ottenere oggetti evento di tipo System.Diagnostics.EventLogEntry.
Esempio si recuperano tutti gli eventi del registro applicazione

```
PS C:\ > get-eventlog application

  Index Time        EntryType   Source              InstanceID Message
  ----- ----        ---------   ------              ---------- -------
  13566 feb 17 09:22  Information gupdate                    0 Impossibile trovare la descri...
  13565 feb 17 09:17  Information gupdate                    0 Impossibile trovare la descri...
  13564 feb 17 09:03  Information edgeupdate                 0 Service stopped.
  13563 feb 17 08:58  Information edgeupdate                 0 Service stopped.
  13562 feb 17 08:52  Information edgeupdate                 0 Service stopped.
  13561 feb 17 08:21  Information gupdate                    0 Impossibile trovare la descri...
  13560 feb 17 08:18  Information gupdate                    0 Impossibile trovare la descri...
  13559 feb 17 08:12  Information Software Protecti...  1073758208 Pianificazione del riavvio de...
  13558 feb 17 08:11  Information Software Protecti...  3221241866 Migrazione offline di livello...
  13557 feb 17 00:04  Information HHCTRL                 1904 Impossibile trovare la descri...
.....................
```

E' possibile filtrare con where-object ma e' possibile filtrare all'origine dei dati per performance.
Get-EventLog <Nome Registro> [-Before <Data>][-After <Data>][-Newest <tot>][-EntryType <TipoEvento>]
[-IstanceId <identificativo istanza>][-Message <Testo>][-Source <sorgente>][-UserName <utente>]
Valori ammessi per -EntryType
Error:identifica un errore
FailureAudit:Identifica un tentativo di accesso con esito negativo a una risorsa controllata
Information:Identifica un'informazione generica o un'operazione avvenuta con successo
SuccessAudit: identificaun tentativo di accesso con esito positivo ad una risorsa controllata
Warning:identifica una situazione potenzialmente problematica

Cerco nel reg system gli eventi che contengono un messaggio SQL server

```
PS C:\ > Get-EventLog System -Message "SQL Server*"
```

Altro Esempio

```
PS C:\ > Get-EventLog -LogName Application | Where-Object Source -Match defrag
```

esempio si recuperano i primi 3 tentativi di accesso fallito a una risorsa controllata avvenuta dopo una certa data nel registro security in locale (run as admin):

```
PS C:\ > Get-EventLog Security -EntryType FailureAudit -after ([datetime]'2020-01-01') -Newest 3

  Index Time        EntryType   Source              InstanceID Message
  ----- ----        ---------   ------              ---------- -------
 127051 feb 16 10:46  FailureA... Microsoft-Windows...       4625 Accesso di un account non riuscito....
 124602 feb 15 10:56  FailureA... Microsoft-Windows...       4625 Accesso di un account non riuscito....
 115426 feb 12 10:46  FailureA... Microsoft-Windows...       4625 Accesso di un account non riuscito....
```

Se devo operare in remoto nelle macchine di destinazione deve essere avviato il servizio di registro remoto (Remote Registry) e tra la macchina sorgente e destinazione deve essere consentito il traffico RPC.

Esempio Recupero tutti gli eventi legati all'id di istanza 4624 (l'evento di logon con esito positivo nel sistema) dal registro di sicurezza :

```
PS C:\ > Get-EventLog Security -InstanceId 4624 -Newest 3

  Index Time        EntryType   Source              InstanceID Message
  ----- ----        ---------   ------              ---------- -------
 128686 feb 17 10:27  SuccessA... Microsoft-Windows...       4624 Accesso di un account riuscito....
 128684 feb 17 10:24  SuccessA... Microsoft-Windows...       4624 Accesso di un account riuscito....
 128670 feb 17 10:02  SuccessA... Microsoft-Windows...       4624 Accesso di un account riuscito....
```

Tramite gli oggetti System.Diagnostics.EventLogEntry e' possibile recuperare i dettagli associati ad un evento: la prop MachineName ritorna il nome del PC in cui l'evento e' stato prodotto, message il testo completo eTimeGenerated la data di generazione. La prop ReplacementStrings contiene un array di stringhe (la cui strutturazione dipende dall'evento vedi sito Microsoft Technet) utilizzate come valori per il modello dell'evento.

```
PS C:\ > Get-EventLog Security -InstanceId 4624 -Newest 2|foreach-object{
$data=@{}
$data.TimeGenerated=$_.TimeGenerated
$data.SecurityId=$_.ReplacementStrings[4]
$data.UserName=$_.ReplacementStrings[5]
$data.Domain=$_.ReplacementStrings[6]
$data.IPAddress=$_.ReplacementStrings[18]
$data.getEnumerator()
}


Name                           Value
----                           -----
UserName                       SYSTEM
Domain                         NT AUTHORITY
TimeGenerated                  17/02/2021 10:42:19
SecurityId                     S-1-5-18
IPAddress                      -
UserName                       SYSTEM
Domain                         NT AUTHORITY
TimeGenerated                  17/02/2021 10:33:30
SecurityId                     S-1-5-18
IPAddress                      -
```

Il cmdLet Get-WinEvent
Sintassi
Get-WinEvent -LogName Application | Where-Object { $_.ProviderName -Match 'defrag' }

```
PS C:\ > Get-WinEvent -ListLog *

LogMode   MaximumSizeInBytes RecordCount LogName
-------   ------------------ ----------- -------
Circular          15728640         970 Windows PowerShell
Circular           1052672           0 Windows Azure
Circular          20971520       10445 System
Circular          20971520       29155 Security
Circular           1052672          85 OAlerts
Circular          20971520           0 Key Management Service
```

```
Circular         1052672         0 Internet Explorer
Circular        20971520         0 HardwareEvents
Circular        20971520     13600 Application
.....................
```

FILTRARE GLI EVENTI TRAMITE ARRAY ASSOCIATIVO
Si possono filtrare le interrogazioni a get-WinEvent utilizzando un hash table,
all'interno del quale sono memorizzate coppie chiave-valore relative ai criteri di uguaglianza

Esempio recuperare eventi critici e di errore (Level pari a 1 e 2) generati in System nelle ultime 12 ore

```
PS C:\ > Get-WinEvent -FilterHashtable @{logname='system'; level=1,2;StartTime=(Get-Date).AddHours(-
12)} -MaxEvents 50


   ProviderName: Microsoft-Windows-DNS-Client

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
17/02/2021 17:01:45           1014 Avviso           Timeout della risoluzione dei nomi per il nome
js.amniscontentd...


   ProviderName: Microsoft-Windows-DistributedCOM

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
17/02/2021 15:49:18          10016 Avviso           Le impostazioni delle autorizzazioni impostazioni predefinite
d...


   ProviderName: Ntfs

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
17/02/2021 13:39:09            130 Avviso           La struttura del file system sul volume C: è stata ripristinata.


   ProviderName: Microsoft-Windows-DistributedCOM

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
17/02/2021 12:39:00          10016 Avviso           Le impostazioni delle autorizzazioni impostazioni predefinite
d...
17/02/2021 10:46:03          10016 Avviso           Le impostazioni delle autorizzazioni impostazioni predefinite
d...
17/02/2021 09:53:11          10016 Avviso           Le impostazioni delle autorizzazioni impostazioni predefinite
d...
```

Principali chiavi utilizzabili con il parametro -FilterHashTable:
LogName: Il Nome di uno o piu log degli eventi. accetta caratteri wildcard.
es:logname='application','system'
ProviderName:il nome di uno o piu' provider di log di cui filtrare gli eventi (wildcard)

ID: Uno o piu' identificativi numerici degli eventi desiderati
Level: Uno o piu' livelli di criticita' con cui effettuare la ricerca.
(1=FATAL, 2=ERROR, 3=Warning, 4=Information, 5=DEBUG, 6=TRACE)
StartTime: Data e ora di inizio dell'intervallo entro il quale comprendere la ricerca
EndTime: Data e ora di fine dell'intervallo entro il quale comprendere la ricerca
UserID: il SID (Security Identifier) dell'utente Windows cui gli eventi da ricercare sono correlati.
accetta anche una stringa con il nome dell'utente, convertita in automatico in SID dal cmdlet

Recuperare i Provider Disponibili:
Tipo di oggetto restituito

```
PS C:\ > (get-WinEvent -ListProvider *| select-object -first 1)[0].GetType().FullName
System.Diagnostics.Eventing.Reader.ProviderMetadata
```

Tramite il parametro -ListProvider il cmdlet get-WInEvent e' in grado di recuperare i provider registrati.
ad esempio

```
PS C:\ > get-WinEvent -ListProvider * | select-object -first 5


Name     : PowerShell
LogLinks : {Windows PowerShell}
Opcodes  : {}
Tasks    : {Integrità modulo
         , Integrità comando
         , Integrità provider
         , Ciclo di vita modulo
         ...}

Name     : Windows Azure Runtime 2.9.0.0
LogLinks : {Windows Azure}
Opcodes  : {}
Tasks    : {None
         }

Name     : Workstation
LogLinks : {System}
Opcodes  : {}
Tasks    : {}

Name     : WMIxWDM
LogLinks : {System}
Opcodes  : {}
Tasks    : {}

Name     : WinNat
LogLinks : {System}
Opcodes  : {}
Tasks    : {}
```

Il tipo ritornato nel precedente ha alcune proprieta' quali ad esempio LogLinks contiene
un riferimento per ogni log degli eventi su cui il provider puo' scrivere,
mentre Events contiene una collezione di modelli di evento utilizzati dal provider

Ad Esempio si recuperano tutti i modelli di evento creati dal provider Microsoft-Windows-Security-Auditing cui e' affidata la gestione degli eventi di audit del sistema operativo, e se ne visualizza a video l'identificativo:

```
PS C:\ > Get-WinEvent -ListProvider Microsoft-Windows-Security-Auditing|
>> Select-object -Expand Events | Select-Object Id | Format-Wide -Column 20

4608 4609 4610 4611 4612 4614 4615 4616 4616 4618 4621 4622 4624 4624 4624 4625 4626
4627 4634 4646
4647 4648 4649 4650 4651 4652 4653 4654 4654 4655 4656 4656 4657 4658 4659 4660 4661
4661 4662 4663
4663 4664 4665 4666 4667 4668 4670 4671 4672 4673 4674 4675 4688 4688 4688 4689 4690
4691 4692 4693
4694 4695 4696 4697 4697 4698 4698 4699 4699 4700 4700 4701 4701 4702 4702 4703 4704
4705 4706 4707
4709 4710 4711 4712 4713 4714 4715 4716 4717 4718 4719 4720 4722 4723 4724 4725 4726
4727 4728 4728
.....
```

RECUPERARE GLI EVENTI DA FILE

Oltre al recupero dai log degli eventi, Get-WinEvent e' anche in grado di ottenere informazioni dai file memorizzati tramite Event Viewer (estensione .evt e .evtx oppure .etl)

Esempio:

```
PS C:\ > Get-WinEvent -Path .\Archive.evtx
```

New-EventLog

Consente di creare un nuovo log degi eventi oppure di registrare una nuova sorgente associata ad un log.

Esempio registra la sorgente powershell.it:

```
PS C:\ > New-EventLog -Log Application -Source myPS
```

Write-EventLog

Si puo' scrivere su un log esistente

sintassi:

Wite-EventLog <Nome Log> <Nome sorgente> <ID evento> [Tipo Voce] <Messaggio> [-ComputerName <PC1>,[,<PC2>,...]]

Esempio si aggiunge un nuovo evento al log Application registrandolo tramite la sorgente myPS creato nell'esempio precedente.

```
PS C:\ > Write-EventLog Application 12345 Information 'Evento di Test'
```

Limit-EventLog

-LogName si indicano uno o piu' log su cui agire.

-MaximumSize dimensione massima del log

-RetentionDays indica il numero minimo di giorni di permanenza di un evento nel log..

-OverflowAction modo con cui i nuovi eventi sono aggiunti al log una volta che questo ha raggiuntola dimensione massimaa stabilita.

I Valori che puo' assumere il parametro -OveflowAction

OverwriteAsNeeded: E' il valore + comune. Impone al log di sovrascrivere gli eventi vecchi con quelli nuovi, partendo dal meno recente.

OverwriteOlder: Impone al log di sovrascrivere gli eventi vecchi con quelli nuovi, a patto che i primi siano rimasti nel log almeno per il numero di giorni specificato da -RetentionDays; nel caso non vi siano elementi da sovrascrivere, i nuovi eventi vengono ignorati.

DoNotOverwite: Impone al log di non sovrascrivere mai gli eventi passati e di ignorare i nuovi.

Ad esempio: si reimposta il log Application affinche' abbia una dimensione massima di 20 MB e sovrascriva i vecchi eventi con i nuovi:
Limit-EventLog Application -MaximumSize 20MB -OverflowAction OverwriteAsNeeded

Remove-EventLog
Esempio per eliminare la sorgente myPS definita in precedenza:

```
PS C:\ > Remove-EventLog -Source myPS
```

INTERNE E LA SHELL

PING:
Occorre utilizzare il metodo d'istanza send() della classe System.Net.NetworkInformation.Ping
tramite ip address o nome DNS.

```
PS C:\ > $pinger=New-Object.Net.NetworkInformation.Ping
$pinger.Send("192.168.1.44")
```

Il metodo send() ritorna un oggetto System.Net.NetworkInformation.PingReply da cui rilevo il risultato e la latenza. Esempio

```
$pingReply=$pinger.Send('www.google.com', 1000)
if($pingReply.Status -eq 'Success'){
        "www.google.com ha risposto al ping in {0}ms" -f $pingReply.RoundtripTime
}else{
        "www.google.com Non ha risposto al ping, causa: {0}" -f $pingReply.Status
}
```

NB: puo' essere che il servizio ICMP e' disabilitato.

UTILIZZARE IL PROTOCOLLO HTTP
Download di file via HTTP
Allo scopo si puo' utilizzare la classe System.Net.WebClient e il metodo DownloadFile()

```
PS C:\ > $wc=New-object Net.WebClient
$wc.Downloadfile("https://www.google.it/",'c:\Temp\goo.html')
#Oppure:
PS C:\ > Invoke-WebRequest -Uri "https://www.repubblica.it" -OutFile "file.txt"
```

Posso leggere il contenuto:

```
PS C:\ > Get-Content .\file.TXT
```

XML - Lettura di Feed RSS
I feed RSS sono documenti XML per contenuti sul Web che si possono recuperare via HTTP.
Per evitare di memorizzare su disco i documenti scaricati ma elaborarli in memoria e' possibile usare il metodo DownLoadString() di System.Net.WebClient

```
$uri="https://www.newegg.com/d/Product/RSS?Submit=RSSDailyDeals&Depa=0"
$str=(New-Object System.Net.WebClient).DownloadString($uri)
$str
$feed=[xml](New-Object System.Net.WebClient).DownloadString($uri)
$feed.rss.channel

#oppure altro sito
$feed=[xml](New-Object
System.Net.WebClient).DownloadString("http://channel9.msdn.com/Events/MIX/MIX11/RSS/wmvhigh")
```

```
foreach($i in $feed.rss.channel.item)
{
   $url = New-Object System.Uri($i.enclosure.url)
   $file = $url.Segments[–1]

   if (! (Test-Path $file))
   {
      $url.ToString()
      #(New-Object System.Net.WebClient).DownloadFile($url, $file)
   }
}
```

WEB SERVICE

Un Web Service e' sostanzialmente una interfaccia software che contiene funzioni richiamabili dalla rete, tramite protocollo HTTP. Inoltre sono auto-contenuti ed auto-descrittivi. cioè è in grado di farci sapere che funzioni mette a disposizione (senza bisogno di conoscerle a priori) e ci permette inoltre di capire come vanno utilizzate tramite l'architettura UDDI.

Ad esempio si vuole ottenere le informazioni dell' indirizzo pubblico IP del mio PC:

```
$ip=(Invoke-WebRequest -uri "http://ifconfig.me/ip").Content
$uri="http://ip-api.com/json/$ip"
(New-Object System.Net.WebClient).DownloadString($uri)
PS C:\ > echo "MY IP:$ip"

{"status":"success","country":"Italy","countryCode":"IT","region":"25","regionName":"Lombardy","city":"M
ilan","zip":"2
0121","lat":45.4642,"lon":9.18998,"timezone":"Europe/Rome","isp":"Vodafone","org":"","as":"AS30722
Vodafone Italia S.p
.A.","query":"5.90.198.95"}
MY IP:5.90.198.95

PS C:\ > Invoke-RestMethod $uri

status      : success
country     : Italy
countryCode : IT
region      : 25
regionName  : Lombardy
city        : Milan
zip         : 20121
lat         : 45,4642
lon         : 9,18998
timezone    : Europe/Rome
isp         : Vodafone
org         :
as          : AS30722 Vodafone Italia S.p.A.
query       : 5.90.198.95
```

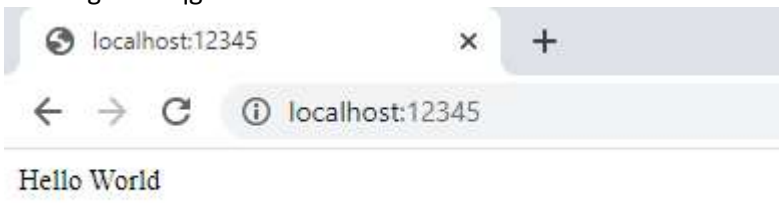
Notare le informazioni ritornate dal Web Service in formato Json.

SERVER HTTP (System.Net.HttpListener)

E’ possibile creare un server http in ascolto su una determinata porta:

```
E’ possibile creare un Server in PowerShell
#Listener HTTP on Port 12345
$httpListener = New-Object Net.HttpListener
$httpListener.Prefixes.Add("http://localhost:12345/")
$httpListener.Start()
#Ciclo ricezione connessioni entrata
while($true){#$HttpListener.IsListening){
#Recupero del contesto HTTP
$httpContext = $httpListener.GetContext()
$HttpRequest = $HttpContext.Request

$httpResponse = $httpContext.Response

#Aggiunta degli Header allo stream di risposta HTTP

$httpResponse.headers.Add("Content-Type","text/html")
$httpResponse.Headers.Add("Server","PowerShell/1.0")

#Scrittura del corpo dell'output

$writer = New-Object IO.StreamWriter($httpResponse.OutputStream,[Text.Encoding]::UTF8)
$writer.Write('<html><body>Hello World</body></html>')
$writer.close()
}
#Chiusura del listener HTTP (comunque forzata da CTRL+BREAK)
$HttpListener.Stop()
```

Nella figura seqguente la chiamata da Browser



Interrogare il DNS
Si utilizza la classe System.Net.Dns in grado di effettuaRE INTERROGAZIONI dns LIMITATE AI RECORD a, cname e AAAA.
Esempio

```
PS C:\ > [System.Net.Dns]::GetHostEntry("www.microsoft.com")

HostName                Aliases AddressList
--------                ------- -----------
e13678.dscb.akamaiedge.net {}     {2.22.33.235}
```

Nel Caso l'Host Specificato sia relativo ad un record CNAME, la proprieta' HostName ritorna il nome DNS del relativo record A (o AAAA)

Combinando le funzionalita' DNS e Ping e' possibile calcolare la latenza minima, massima e media di tutti gli indirizzi IP di un particolare host:

```
$pinger=New-Object Net.NetworkInformation.Ping
```

```
[Net.Dns]::GetHostEntry("www.google.com")|
select-object -Expand AddressList | ForEach-Object {$pinger.Send($_)}|
Measure-Object -Max -Min -Ave RoundTripTime
```

Per convertire un file da xml a json:

```
function ConvertFrom-Xml {
 param([parameter(Mandatory, ValueFromPipeline)] [System.Xml.XmlNode] $node)
 process {
  if ($node.DocumentElement) { $node = $node.DocumentElement }
  $oht = [ordered] @{}
  $name = $node.Name
  if ($node.FirstChild -is [system.xml.xmltext]) {
   $oht.$name = $node.FirstChild.InnerText
  } else {
   $oht.$name = New-Object System.Collections.ArrayList
   foreach ($child in $node.ChildNodes) {
    $null = $oht.$name.Add((ConvertFrom-Xml $child))
   }
  }
  $oht
 }
}

PS C:\ > [xml[]] (Get-Content -Raw C:\\temp\\test.xml) | ConvertFrom-Xml | ConvertTo-Json -Depth 5
```

INVIARE E-MAIL

Di seguito un esempio di script per l'invio di email da PowerShell

```
$emailSmtpUser="xxxx.yyyy@fastwebnet.it"
$credentials=$Host.UI.PromptForCredential("Enter your Email address and
password","",$emailSmtpUser,"")
$pwd=$credentials.getnetworkcredential().password
$secpasswd = ConvertTo-SecureString $pwd -AsPlainText -Force
$from=$emailSmtpUser
$cred = New-Object System.Management.Automation.PSCredential ($from,$secpasswd)
$Subject='Test3'
$Body='Hello'
Send-MailMessage -SmtpServer 'smtp.fastwebnet.it' -Port '587' -Credential $cred -UseSsl -From $from -To
'TTTTTTTT@MMMMM.it' -Subject $Subject -body $msg
```

JOB

PS e' in grado di gestire piu' attivita' in parallelo e in background in sessioni indipendenti da quella corrente, ma comunque comunicanti.

L'infrastruttura dei Job si basa sul servizio WinRM (Windows Remote Management) che gestisce job remoti e locali.

In PS per configurare WinRM (chiamato anche WSMan sotto win) e' sufficiente utilizzare SetWSManQuickConfig.

WinRM utilizza le seguenti porte:

TCP/UDP 5985, wsman (HTTP)

TCP/UDP 5986, wsmans (HTTPS)
Per controllare la configurazione di WinRM sia corretta si utilizza il cmdlet:

```
PS C:\ > Test-wsMan
wsmid           : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0
```

CICLO DI VITA DI UN JOB
In PS un jobcorrisponde ad un oggetto di tipo Management.Automation.PSRemotingJob.
Il ciclo dei job termina alla chiusura della sessione corrente o alla chiusura di PS.

Tutti i Job Creati nella sessione corrente sono inseriti in una lista.

AVVIO DI UN JOB Start-Job
Il cmdlet utilizzato e' Start-Job la cui sintassi e':
Start-Job[<comandi>]
oppure
Start-job <script-esterno> [-ArgumentList >Arg1>,<Arg2>,...]
Ad esempio

```
PS C:\ > Start-Job {Write-Host "Hello World";Read-Host -prompt "tuo nome?"}

Id     Name            PSJobTypeName   State         HasMoreData     Location             Command
--     ----            -------------   -----         -----------     --------             -------
1      Job1            BackgroundJob   Running       True            localhost            Write-Hos...
```

Le principali prop. della classe System.Management.Automation.PSRemotingJob
Command:Il testo del comando fornito
HasMoreData:Indica che il job puo' restituire nuovi dati prodotti dallo script
Id:L'identificativo numerico univoco del job all'interno della sessione corrente.
Location:il nome o l'indirizzo della macchina presso la quale il job e' in esecuzione
Name:Il nome descrittivo del job, utilizzabile per facilitarne la gestione
State:Indica lo stato corrente del Job

I Valori che puo' assumere la proprieta' state
NotStarted:>Il job non e' stato ancora avviato
Running:Il job e' in esecuzione
Completed:L'esecuzione del job e' stata portata a termine
Failed:Il job e' terminato con un errore
Stopped: l'esecuzione del job e' stata interrotta
Blocked:L'esecuzione del job e' bloccata, in attesa di input da parte dell'utente.

Il Recupero dei Risultati Receive-Job
Receive-Job <Oggetto job1>[,<Oggetto Job2>,...]
Receive-Job <Id1>[,<Id2>,...]
Receive-Job [Name1][,<Name2>,...]
<Oggetto job1> [,<Oggetto job2>,...]|Wait-job

Data la natura asincrona di un job, ogni informazione prodotta da un job e' inviata a WinRM,
che la memorizza in buffer di memoria. Con il cmdLet Receive-Job e' possibile
dalla sessione corrente contattare WinRM per prelevare i dati dal buffer.
Questo ciclo di recupero dati puo' continuare fino a quando il job ha completato le proprie

attivita' o non viene terminato.
E' possibile indicare al cmdLet Receive-Job il job di interesse tramite i parametri -Job, -Id, -Name Ritornati dalla chiamata precedente a Start-Job.

Dato che come si e' detto il flusso di informazioni e' asincrono l'istante in cui i risultati di un job sono disponibili non e' esattamente prevedibile, e' possibile verificare la proprieta' HasMoreData per sapere se il job ha o meno dei dati da restituire.
Esempio attendo che il cmdlet Get-Process abbia dei dati prima di stamparli a video

```
PS C:\ > $job = Start-job{Get-Process}
While($job.HasMoreData){$job | Receive-Job}
```

Attesa del completamento
Per Attendere il completamento di un Job si puo' usare il cmdLet Wait-Job specificando i job tramite i parametri -Job -Id -Name, oppure utilizzando la pipeline come da sintassi:
Wait-Job <Oggetto job1>[,<Oggetto Job2>,...]
Wait-Job <Id1>[,<Id2>,...]
Wait-Job [Name1][,<Name2>,...]
<Oggetto job1> [,<Oggetto job2>,...]|Wait-job

Esempio:

```
PS C:\ > $job = Start-job{Get-Process}
>> $job | wait-Job

Id    Name          PSJobTypeName   State       HasMoreData     Location          Command
--    ----          -------------   -----       -----------     --------          -------
1     Job1          BackgroundJob   Completed     True          localhost         Get-Process


PS C:\ > $job | Receive-Job

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
   914      10    19672       9644           4008   0 AdminService
   429      26    14028      33444      10,25   8148   1 ApplicationFrameHost
   358      12     2544      10784           4880   1 atieclxx
   147       8     1632       5440          1772   0 atiesrxx
   544      30    22820       2560      1,14   4364   1 Calculator
          ...............................
```

Il cmdLet accetta inolter il parametro -Timeout per indicare il tempo massimo di attesa previsto espresso in secondi.
Il cmdLet ritorna al prompt allo scadere di questo intervallo o al completamento dei job indicati.
Nel caso in cui tutti i job indicati siano completati nell'intervallo specificato da -Timeout il cmdLet Wait-Join ne emette i relativi oggetti nella Pipeline, in caso contrario il cmdlet non emette nulla.

```
PS C:\ > $job1 = Start-job {Start-Sleep 2}
PS C:\ > $job2 = Start-job {Start-Sleep 3}
PS C:\ > $job1,$job2 | Wait-Job -Timeout 10

Id    Name          PSJobTypeName   State       HasMoreData     Location          Command
```

```
--  ----        -------------  -----      -----------  --------        -------
3   Job3        BackgroundJob  Completed  False        localhost       Start-Sl...
5   Job5        BackgroundJob  Completed  False        localhost       Start-Sl...
```

In questaltro script si porta la sospensione del secondo job a 30 secondi.
Notare Wait-job non ritorna in questo caso nessun oggetto nonostante il primo sia completato:

```
PS C:\ > $job1 = Start-job {Start-Sleep 2}
PS C:\ > $job2 = Start-job {Start-Sleep 30}
PS C:\ > $job1,$job2 | Wait-Job -Timeout 10
PS C:\ > $job1.State, $Job2.State
Completed
Running
```

L'esecuzione di Wait-Job senza parametri porta all'attesa del completamento di tutti i job creati nella sessione corrente.

DESERIALIZZAZIONE
Per passare da una sessione all'altra gli oggetti vengono serializzati all'origine
e deserializzati alla destinazione con Receive-Job tramite protocollo XML WinRM.
Dato che alcune proprieta' di un oggetto complesso non hanno senso nella sessione di destinazione
a differenza degli oggetti primitivi alcune proprieta' non vengono serializzate, e non presentano i metodi.

Ad esempio si confronti i membri ritornati da questo script con la Get-Member su Get-Process.
Inoltre si noti che il tipo restituito da Get-Process non e' piu' System.Diagnostics.Process
ma Deserialized.System.Diagnostic.Process

```
PS C:\ > $job = Start-job {Get-Process expl*}
PS C:\ > Receive-Job $job | Get-Member

  TypeName: Deserialized.System.Diagnostics.Process

Name                 MemberType   Definition
----                 ----------   ----------
GetType              Method       type GetType()
ToString             Method       string ToString(), string ToString(string format, System.I...
Company              NoteProperty string Company=Microsoft Corporation
CPU                  NoteProperty double CPU=2464,25
Description          NoteProperty string Description=Esplora risorse
FileVersion          NoteProperty string FileVersion=10.0.19041.860 (WinBuild.160101.0800)
Handles              NoteProperty int Handles=3544
Name                 NoteProperty string Name=explorer
NPM                  NoteProperty long NPM=176592
Path                 NoteProperty string Path=C:\Windows\explorer.exe
PM                   NoteProperty long PM=118607872
Product              NoteProperty string Product=Sistema operativo Microsoft® Windows®
ProductVersion       NoteProperty string ProductVersion=10.0.19041.860
PSComputerName       NoteProperty string PSComputerName=localhost
PSShowComputerName   NoteProperty bool PSShowComputerName=False
RunspaceId           NoteProperty guid RunspaceId=437c6224-ea92-4d22-8952-4a6ca1d80c00
.......
```

I tipi primitivi deserializzati invece non subiscono il processo di ricostruzione descritto

per i tipi complessi e mantengono sia il namespace originale sia i propri membri e i metodi

```
PS C:\ > $job = Start-job {[Math]::Sin([Math]::PI * 2)}
PS C:\ > Receive-Job $job |Get-Member

   TypeName: System.Double

Name             MemberType   Definition
----             ----------   ----------
CompareTo        Method       int CompareTo(System.Object value), int CompareTo(double value), i...
Equals           Method       bool Equals(System.Object obj), bool Equals(double obj), bool IEqu...
GetHashCode      Method       int GetHashCode()
GetType          Method       type GetType()
.....................
```

Sia per i tipi primitivi che per gli altriPS aggiunge ad ogni oggetto le proprieta' PSComputerName e RunSpaceId
utili per stabilire il PC e la sessione di provenienza.
Visualizzazione job
il cmdLet Get-Job Visualizza i Job Creati nella sessione corrente.
Get-Job -Id <Id1>[,<Id2>,...]
Get-Job -Name <Nome11>[,<Nome2>,...]

Esempio:

```
PS C:\ > $job1 = Start-Job -Name Test {Write-Host "Hello"}
PS C:\ > $job2 = Start-Job -Name Test {Write-Host "World"}
PS C:\ > $job3 = Start-Job {Write-Host "Bye"}
PS C:\ > Get-Job -Name Test

Id     Name            PSJobTypeName   State         HasMoreData     Location             Command
--     ----            -------------   -----         -----------     --------             -------
9      Test            BackgroundJob   Running       True            localhost            Write-Ho...
11     Test            BackgroundJob   Running       True            localhost            Write-Ho...
```

Per Controllare l'ampiezza delle colonne del risultato si puo' usare il cmdLet Format-Table

```
PS C:\ > Get-Job -Name Test | Format-Table -AutoSize

Id Name PSJobTypeName State     HasMoreData Location  Command
-- ---- ------------- -----     ----------- --------  -------
9  Test BackgroundJob Completed True        localhost Write-Host "Hello"
11 Test BackgroundJob Completed True        localhost Write-Host "World"
```

Se non si specifica alcun parametro get-job ritornatutti i job creati nella sessione corrente.

```
PS C:\ > Get-Job | Format-Table -AutoSize

Id Name  PSJobTypeName State     HasMoreData Location  Command
-- ----  ------------- -----     ----------- --------  -------
1  Job1  BackgroundJob Completed False       localhost Start-Sleep 2
3  Job3  BackgroundJob Completed False       localhost Start-Sleep 30
5  Job5  BackgroundJob Completed False       localhost Get-Process expl*
7  Job7  BackgroundJob Completed False       localhost [Math]::Sin([Math]::PI * 2)
9  Test  BackgroundJob Completed True        localhost Write-Host "Hello"
```

```
11 Test  BackgroundJob Completed True       localhost Write-Host "World"
13 Job13 BackgroundJob Completed True       localhost Write-Host "Bye"
```

INTERRUZIONE ED ELIMINAZIONE
il cmdLet Stop-Job interrompe l'esecuzione di un job
specificando i job tramite i parametri -Job -Id -Name, oppure utilizzando la pipeline come da sintassi:
Stop-Job <Oggetto job1>[,<Oggetto Job2>,...]
Stop-Job <Id1>[,<Id2>,...]
Stop-Job [Name1][,<Name2>,...]
<Oggetto job1> [,<Oggetto job2>,...]|Stop-job

Esempi:

```
PS C:\ > Stop-Job -Name "Job1"
PS C:\ > Stop-Job -Id 1, 3, 4
#Ferma tutti i Job di background Bloccati
PS C:\ > Stop-Job -State Blocked
#Interrompe l'esecuzione di tutti i job di background creati nella sessione corrente
PS C:\ > Get-Job | Stop-Job
```

Di default questo comando non emette alcun oggetto nella pipeline; per forzare l'emissione di oggetti forniti in input (pratica utile nell'eliminazione dei job) e' necessario specificare lo switch -PassThru.
L'interruzione di un job d'altra parte non ne elimina ne' il relativo processo ne' la relativa sessione, gestita da WinRM;
per eliminare del tutto il job e liberare le risorseimpegnate e' necessario usare il cmdLet Remove-Job.
La sintassi e' la solita:
Remove-Job <Oggetto job1>[,<Oggetto Job2>,...]
Remove-Job <Id1>[,<Id2>,...]
Remove-Job [Name1][,<Name2>,...]
<Oggetto job1> [,<Oggetto job2>,...]|Remove-job

Ad esempio nello script che segue si creano tre diversi job (sopprimendo l'output con ilcmdLet Out-Null per una maggiore leggibiita').
Attraverso la pipeline prodotta da Get-Job, Stop-Job e Remove-Job si recuperanotuttii job prodotti nella sessione, si interromponi ed infine si eliminano.

```
Start-Job {Start-Sleep 100} | Out-Null
Start-Job {Start-Sleep 200} | Out-Null
Start-Job {Start-Sleep 300} | Out-Null
Get-Job|Stop-Job -PassThru | Remove-Job

PS C:\ > Get-Job|Stop-Job -PassThru | Remove-Job
PS C:\ > get-job
PS C:\ > (get-job).count
0
```

Ulteriore esempio che illustra la Read-Host in un Job (ovviamente non si usano Letture da tastiera nei Job per ovvi motivi):

```
PS C:\ > Start-Job {Read-Host -prompt "tuo nome?"}

Id    Name          PSJobTypeName   State       HasMoreData     Location          Command
--    ----          -------------   -----       -----------     --------          -------
15    Job15         BackgroundJob   Running     True            localhost         Read-Hos...
```

```
PS C:\ > Receive-Job -Id 15
tuo nome?: joe
PS C:\ > get-job

Id    Name         PSJobTypeName   State       HasMoreData    Location          Command
--    ----         -------------   -----       -----------    --------          -------
15    Job15        BackgroundJob   Completed    True          localhost         Read-Hos...


PS C:\ > Receive-Job -Id 15
joe
PS C:\ > get-job

Id    Name         PSJobTypeName   State       HasMoreData    Location          Command
--    ----         -------------   -----       -----------    --------          -------
15    Job15        BackgroundJob   Completed    False         localhost         Read-Hos...
```

ESEGUIRE TASK IN REMOTO

Per eseguire attivita' in remoto all'interno della shell, e' necessario che sul sistema di destinazione siano installati e correttamente configurati sia PowerShell sia WinRM (cioe' il servizio Windows Remote Management avviato), cosi' come il Firewall di Windows utilizzando il cmdlet Set-WSManQuickConfig.
Come detto per i job, il servizio rimane di default in ascolto sulla porta TCP/UDP 5985 con protocollo HTTP.
Utilizzando lo switch -UseSSL di Set-WSManQuickConfig, invece, WinRM rimane in ascolto sulla porta TCP/UDP 5986 e dialoga tramite protocollo HTTPS.
Utilizzando il provider WSMan si possono analizzare le impostazioni di configurazioni di WinRM.
La configurazione dwgli endpoint in ascolto, per esempio e' disponibile nella cartella WSMan:\localhost\Listener.
Nello script che segue, per esempio, si puo' verificare come nel PC utilizzato sia attivo un endpoint HTTP associati a tutti gli indirizzi suppportati (NB Per navigare all'interno del provider WSMAN e' necessario eseguire PowerShell come amministratore):

```
PS C:\ > set-location wsman:
PS WSMan:\> get-childitem


  WSManConfig:

ComputerName                  Type
------------                  ----
localhost                     Container


PS WSMan:\> cd .\Localhost
PS WSMan:\localhost> Get-ChildItem


  WSManConfig: Microsoft.WSMan.Management\WSMan::localhost

Type          Name                  SourceOfValue   Value
----          ----                  -------------   -----
System.String   MaxEnvelopeSizekb                    500
System.String   MaxTimeoutms                        60000
```

```
System.String   MaxBatchItems                   32000
System.String   MaxProviderRequests                4294967295
Container       Client
Container       Service
Container       Shell
Container       Listener
Container       Plugin
Container       ClientCertificate


PS WSMan:\localhost> cd .\Listener
PS WSMan:\localhost\Listener> Get-ChildItem


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Listener

Type            Keys                        Name
----            ----                        ----
Container       {Transport=HTTP, Address=*}       Listener_1084132640

PS WSMan:\localhost\Listener> cd .\Listener_1084132640
PS WSMan:\localhost\Listener\Listener_1084132640> Get-ChildItem


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Listener\Listener_1084132640

Type            Name                    SourceOfValue   Value
----            ----                    -------------   -----
System.String   Address                                 *
System.String   Transport                               HTTP
System.String   Port                                    5985
System.String   Hostname
System.String   Enabled                                 true
System.String   URLPrefix                               wsman
System.String   CertificateThumbprint
System.String   ListeningOn_1770022257                  127.0.0.1
System.String   ListeningOn_1747200100                  169.254.46.133
System.String   ListeningOn_181116157                   169.254.46.152
System.String   ListeningOn_519189856                   169.254.66.9
System.String   ListeningOn_443392242                   192.168.43.35
System.String   ListeningOn_1414502903                  ::1
System.String   ListeningOn_1295126491                  fe80::14dc:3fbe:3663:2e85%23
System.String   ListeningOn_365313423                   fe80::18dd:f89:b284:2e98%25
System.String   ListeningOn_1860798422                  fe80::5046:969c:a289:4209%9
System.String   ListeningOn_1438144256                  fe80::d0dc:ad79:5c3d:4d75%7
```

WinRM accetta solo le connessioni provenienti da una list di host considerati attendibili,e, per impostazione predefinita non accetta inizialmente alcuna connessione.Prima di eseguire comandi in remoto presso una particolare macchina, dunque e' necessario aggiungere l'hostche origina la richiesta alla lista degli host attendibili, memorizzati nell'elemento TrustedHosts all'interno di WSMan:\localhost\Client.
Nello script che segue che continua il precedente, si vede come l'elemento TrustedHosts non contenga alcun valore (quindi non permetta alcuna connessione remota):

```
PS WSMan:\localhost\Listener\Listener_1084132640> cd ..\..\
PS WSMan:\localhost> cd Client
PS WSMan:\localhost\Client> Get-ChildItem


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Client

Type          Name                    SourceOfValue   Value
----          ----                    -------------   -----
System.String   NetworkDelayms                         5000
System.String   URLPrefix                           wsman
System.String   AllowUnencrypted                     false
Container       Auth
Container       DefaultPorts
System.String   TrustedHosts
```

Per modificare la lista degli host attendibili e' sufficiente intervenire sull'elemento TrustedHosts, impostando la lista di nomi NetBIOS o indirizzi IP, separando ogni voce con una virgola.
Il simbolo di asterisco (*), indica che e' ammesso qualsiasi host.
Nello script che segue si impostano come host attendibili due hosts

```
PS WSMan:\localhost\Client> Set-Item .\TrustedHosts "PC01, 192.168.1.43"
```

Nello script seguente invece si permette l'impiego di WinRM da parte di qualsiasi macchina remota:

```
PS WSMan:\localhost\Client> Set-Item .\TrustedHosts "*"
```

Per verificare che la macchina corrente possa comunicare correttamente con un'istallazione remota di WInRM e' possibile utilizzare il cmdlet Test-WSMan.
Ad esempio per verificare la possibilita' di connettersi a un'istallazione WinRM tramite IP:

```
PS C:\ > Test-WSMan 192.168.43.35


wsmid           : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0

Invoke-command
Questo cmdlet permette di eseguire comandi in uno o piu' PC remoti.
Sintassi:
Invoke-command <Host1> [,<Host2>,...+{comando}
esempio:
invoke-command 192.168.43.25 {Get-Date}
```

Esempio ritornare gli elementi della radice del disco C:\ dell'host 192.168.43.21 la cui dimensione sia superiore a 1GB invoke-command 192.168.43.21 {Get-ChildItem C:\ -Force | ?{$_.lENGTH -GT 1gb}}
Come per i job in background anche le informazioni ritornate dai comandi eseguiti in remoto subiscono il processo di serializzazione e deserializzazione. Per impostazione predefinita Invoke-Command si collega al servizio WinRM utilizzando il protocollo HTTP sulla porta 5985 del PC di destinazione. Con lo switch -UseSSL, e' possibile forzare il comando aa impiegare il protocollo HTTPS sulla porta 5986, che puo' essere cambiata con lo switch -Port.

Sintassi:
Invoke-Command <Host> [-Port <Porta>] [-UseSSL]
Quando si usa Invoke-Command il sistema di origine si autentica su quello di destinazione utilizzando il protocollo Kerberos e le credenziali dell'utente corrente, che pertanto devono corrispondere a un utente recuperabile dalla macchina di destinazione (un utente locale o un utente di un dominio riconosciuto dalla macchina). Per fornire al cmdlet credenziali alternative rispetto a quelle dell'utente corrente eì necessario impiegare il parametro -Credential, specificando il nome dell'utente (dominio incluso) da usare: prima di procedere il sistema richiede la password; l'utente cosi' specificato deve comunque appartenere al gruppo Administrators della macchina remota.

```
PS C:\ > Invoke-Command 192.168.178.25 {$env:COMPUTERNAME} -Credential Test
```

La richiesta delle credenziali per l'utente Test specificato avviene con la visualizzazione di un DialogBox.
Inoltre Invoke-Command permette di eseguire su un PC remoto uno script.
Esempio:

```
PS C:\ > Invoke-Command 192.168.78.25 -FilePath .\sum.PS1 -ArgumentList 9,7,83
```

L'esecuzione di Invoke-Command richiede i privilegi di amministratore.

Esecuzione su piu' Macchine
E' possibile con il cmdlet Invoke-Command specificando piu' PC nel parametro -ComputerName eseguire la stessa attivita' su queste macchine. Per sapere in fase di raccolta dati quale sia il PC che ha prodotto quei risultati si utilizzala proprieta' PSComputerName aggiunta da PS in fase di serializzazione.
Esempio:

```
PS C:\ > Invoke-Command 192.168.178.25, PC001 {Get-Process expl*}
```

ESECUZIONE IN BACKGROUND
L'esecuzione di un'attivita' in remoto puo' essere convertita in un job, grazie allo switch -AsJob del cmdlet Invoke-Command: in tal caso il comando non ritorna direttamente il risultato ma un oggetto job creato all'interno della sessione locale.
In questo script si crea un job a partire da un'attivita' eseguita in remoto; l'impiego del job permette di liberare la sessione corrente dall'attesa del termine dell'operazione:

```
PS C:\ > $job = Invoke-Command192.168.178.25 {Get-ChildItem C:\ -Recurse} -AsJob
```

Nel caso in cui lo switch -AsJob sia usato a fronte di piu' host remoti, il cmdlet Invoke-Command ritorna un insieme di job subordinati a un job principale, che funge da collettore. I job subordinati sono tanti quanti gli host specificati e sono disponibili attraverso la proprieta' ChildJobs del Job principale. Nello script che segue illustra alcune informazioni sui job creati a fronte dell'esecuzione remota di un'attivita' in background su host multipli.

```
PS C:\ > $job = Invoke-Command 192.168.43.22, PC003, EX001 {pwd} -AsJob
$job
$job | Select-Object -Expand ChildJobs
```

Sessioni Remote
Ogni chiamata al cmdlet Invoke-Command crea una sessione separata dalle altre, le cui variabili definite in una sessione non rimangono in vita in un'altra.
Ad esempio

```
PS C:\ > Invoke-Command 192.168.43.21 {$test ='Hello World'; $test}
Hello World
Invoke-Command 192.168.43.21 {$test}
```

La seconda chiamata a Invoke-Command non ritorna nulla.
Per far fronte a questo problema PS permette la condivisione delle sessioni remote.

New-PSSession
il cmdlet New-PSSession crea una nuova sessione remota verso un particolare host e
la ritorna nella pipeline. In modo simile ai job, inoltre le sessioni possono avere un nome
descrittivo da fornire tramite il parametro -Name.
Sintassi:
New-PSSession <Host> [-Credential <Utente>][-UseSSL][-Port <Porta>][-Name <Nome descrittivo>]

```
PS C:\ > New-PSSession 192.168.178.25 -Name "Sessione di Test"
```

Gli oggetti ritornati da New-PSSession sono di tipo System.Management.Automation.Runspaces.PSSession
e
sono generalmente usati dagli altri cmdlet per operare nella sessione individuata da ciascuna istanza.

Principali Proprieta' della classe System.Management.Automation.Runspaces.PSSession
Availability: Indica la disponibilita' della sessione ad accettare comandi
ComputerName:Ritorna il nome dell'host con cui la sessione e' stata stabilita
Id:identificativo numerico univoco della sessione remota
Name:Il nome descritiivo della sessione

I principali valori della proprieta' Availability
Available:Indica la disponibilita' della sessione ad accettare comandi
Busy:Indica che la sessione sta eseguendo un comado e non ne puo' al momento accettare di nuovi
None:Sessione non disponibile perche' disconnessa.

Get-PSSession
Consente di recuperare le sessioni remote create all'interno della sessione corrente.
Sintassi
Get-PSSession
Get-PSSession <Host1>[<Host2>,...]
Get-PSSession <Id1> [<Id2>,...]
Get-PSSession -Name <Nome descrittivo1> [,<Nome descrittivo2>,...]

Ad esempio si cercano le sessioniremote di un host:

```
PS C:\ > Get-PSSession 192.168.43.21
```

Oppure con filtro:
Get-PSSession -Name *test | Format-List

Utilizzare la sessione in Invoke-Command
Una volta che la sessione remota e' stata creata ed e' disponibile per l'esecuzione dei comandi,
e' possibile includerla nelle chiamate a Inoke-Command attraverso il parametro -Session,
in maniera tale da far condividere agli script desiderati lo stesso ambiente remoto.
Quando si usa -Session tutti i parametri menzionatiin precedenza relativi alla configurazione
della sessione devono essere omessi con la seguente sintassi:
Invoke-Command <Sessione> {<Comandi>}
Invoke-Command <Sessione1>[,<Sessione2>,...]{<Comandi>}
Invoke-Command <Sessione> -FilePath <Percorso Script> -ArgumentList <Parametro1>[,<Parametro2>,...]
Invoke-Command <Sessione1> [,<Sessione2>,...] -FilePath <Percorso script> -ArgumentList
<Parametro1>[,<Parametro2>,...]
Nel caso si specifichi piu' di una sessione, Invoke-Command esegue le istruzioni impartite in ognuno
degli elementi indicati e ritorna un aggregato dei risultati.
Nello script che segue si crea dapprima una sessione remota e la si utilizza per eseguire piu' comandi
tramite Invoke-Command. A differenza dei casi precedenti qui gli script condividono l'ambiente di

esecuzione remoto e possono accedere, per esempio, alle stesse variabili remote:

```
PS C:\ > $session = New-PSSession SERVER01
Invoke-Command $session { $test = 'google'}
Invoke-Command $session { $test + 'it'}
google.it
```

Remove-PSSession
Le sessioni remote impegnano risorse sia nella macchina che origina i comandi sia, soprattutto, in quella che li esegue. Nonostante esista un sistema di eliminazione automatica che interviene in caso di disconnessione, una volta terminato di utilizzare una sessione conviene sempre procedere all'eliminazione della stessa mediante il cmdlet Remove-PSSession.
Sintassi :
Remove-PSSession <Host1> [,<Host2>,...]
Remove-PSSession <Id1> [<Id2>,...]
Remove-PSSession -Name <Nome descrittivo1> [,<Nome descrittivo2>,...]
Remove-PSSession -Session <Sessione1> [<Sessione2>,...]
<Sessione1> [<Sessione2>,...] | Remove-PSSession

Esempio si eliminano tutte le sessioni remote su un certo host:

```
PS C:\ > Remove-PSSession 192.168.43.25
```

Oppure:
$session | Remove-PSSession

Sessioni Remote Interattive
Per Sessione remota interattiva si inntende una sessione remota (o anche locale ma comunque gestita da WinRM) che permette uno scambio diretto di informazioni tra l'host remoto e quello locale, consentendo l'esecuzione di flussi di comandiccon un'interfaccia pressoche' identicaa quella fisica dell'host di destinazione. La differenza tra una sessione remota e una sessione remota interattiva e' il modo con cui la shell consente lo scambio di informazioni: nel primo caso l'esecuzione di comandi avviene solo tramite Invoke-Command , mentre nel secondo c'e' una totale trasparenza e le istruzioni sono eseguite come se si stesse operando in locale.

Enter-PSSession
Il cmdlet Enter-PSSession consente di avviare una sessione remota interattiva operando all'interno di una sessione esistente oppure creandone una ex novo. Per utilizzare una sessione esistente e' sufficiente fornire al comando l'elemento di interesseattraverso il parametro -Session, direttamente tramite la pipeline oppure con il parametro -id.
Enter-PSSession <Sessione>
Enter-PSSession <Identificativo sessione>
<Sessione> | Enter-PSSession
Enter-PSSession <Host>
New-pSSession <Host>[-Credential <Utente>][-UseSSL][-Port <Porta>]

Una volta che la shell ha avviato una sessione remota interattiva, il prompt riporta automaticamente l'informazione sull'host in uso, facendo precedere il nome alla classica sigla iniziale PS.
Nello script che segue si avvia una nuova sessione interattiva verso un particolare host e si eseguono alcuni comandi in remoto.

```
Enter-PSSession SERVER01
[server01]:PS C:\ > Get-Process | Sort-Object WS -Descending | Select-Object -First 3
[server01]:PS C:\ > $env:COMPUTERNAME
```

```
SERVER01
```

In questo esempio si crea una sessione interattiva a partire da una sessione remota creata
in precedenza e gia' utilizzata da Invoke-Command.

```
PS C:\ >$Session = New-PSSession 192.168.43.25
PS C:\ >Invoke-Command $session {Get-Service w3svc}
PS C:\ >Enter-PSSession $session
[192.168.43.25]PS C:\ > Start-Service w3svc
[192.168.43.25]PS C:\ > Get-Service w3svc
```

Nelle sessioni remote interattive le informazioni subiscono
un processo di serializzazione e deserializzazione ridotto, che
porta gli oggetti a mantenere le proprieta' e i metodi originali e
la propria gerarchia di classe. Infine, agli oggetti gestiti durante
le sessioni remote interattive non sono aggiunte le proprieta'
che permettono di recuperare la macchina (e la sessione) di provenienza.

Exit-PSSession
Il cmdlet Exit-PSSession termina una sessione remota interattiva.
Se la sessione remota era gia' esistente al momento della chiamata di Enter-PSSession,
questo comando si limita a uscire dalla modalita' di esecuzione interattiva,
ma non agisce sulla sessione remota, che puo' essere riutiizzata. Mel caso in cui,
invece, la sessione remota sia stata creata direttamente da Enter-PSSession, il
cmdlet Exit-PSSession si occupa anche della sua chiusura. Una volta terminata
la sessione remota interittiva, il prompt di PowerShell ritorna automaticamente
allo stato assunto in precedenza. Lo script che segue, continuazione di quello precedente,
dimostra come la sessione remota sia ancora utilizzabile dopo il terminedella modalita'
interattiva e come sia necessario chiuderla esplicitamente con Remove-PSSession:

```
[192.168.43.25]PS C:\ > Exit-PSSession
PS C:\ > Invoke-Command $session {[Environment]::OSVersion.VersionString}
Microsoft Windows NT 10.0.19041.0
PS C:\ > $session | Remove-PSSession
```

Nelle sessioni remote interattive la keyword exit richiama automaticamente il cmdlet Exit-PSSession

LA GESTIONE DEGLI ERRORI

Tipi di errori:
sintattici:

```
PS C:\ > $a=3+hello
In riga:1 car:6
+ $a=3+hello
+     ~
Specificare un'espressione di valore sul lato destro dell'operatore '+'.
In riga:1 car:6
+ $a=3+hello
+     ~~~~~
Token 'hello' imprevisto nell'espressione o nell'istruzione.
   + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
   + FullyQualifiedErrorId : ExpectedValueExpression
```

Runtime:
diversamente dagli errori sintattici gli errori a runtime non sono rilevati immediatamente dalla shell

ma si presentano solo al verificarsi di determinate condizioni.
Esempi

```
PS C:\ > $n=[int](Read-Host -Prompt '###')
###: n
Impossibile convertire il valore "n" nel tipo "System.Int32". Errore: "Formato della stringa di
input non corretto."
In riga:1 car:1
+ $n=[int](Read-Host -Prompt '###')
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (:) [], RuntimeException
    + FullyQualifiedErrorId : InvalidCastFromStringToInteger
```

Errore Classico tentativo di accedere ad una variabile non inizializzata

```
PS C:\ > $cf.tostring()
Impossibile chiamare un metodo su un'espressione con valore null.
In riga:1 car:1
+ $scv.tostring()
+ ~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : InvokeMethodOnNull
```

Logici:
Sono i piu' difficili da individuare.
Non comportano la visualizzazione di messaggi, ma danno un risultato diverso da quello atteso
Esempio nello script seguente che calcola la media c'e' la condizione -gt mentre avrebbe dovuto essere -lt:

```
$a=4,5,3,6,7
$s=0
for($i=0; $i -gt $a.length;$i++){
 $s+=$a[$i]
}
$vm=$s/$a.length
echo $vm
```

ECCEZIONI

In informatica, la gestione delle eccezioni (exception handling) e' una tecnica che consente di organizzare efficacemente il codice, concentrando la logica di risoluzione degli errori in un'unica posizione.Generalmente, un'eccezione sollevata in un particolare strato di codice puo' essere gestita dalla logica di risoluzione applicata allo strato stesso. Nel caso questa logica manchi oppure non sia stata concepita per gestire la tipologia di problema emerso, l'eccezione viene automaticamente propagata allo strato di codice chiamante, dove il sistema verifica nuovamente la disponibilita' di un blocco di gestione. Il processo si ripete fino a quando non viene trovato un blocco di codice in grado di gestire l'eccezione. Se la ricerca non va a buon fine, l'errore causa tipicamente il termine dell'attivita' e la visualizzazione di un messaggio appropriato.
PS gestisce gli errori in modo molto simile alle eccezioni in .NET.
THROW
Per sollevare un'eccezione in PS si utilizza l'istruzione throw che genera un'eccezione di tipo System.Management.Automation.RuntimeExceotion e a impostare un messaggio nella prop. Message.
Sintassi throw

```
throw <Eccezione>
```

throw <Oggetto>

Esempio eccezione generica:

```
PS C:\ > throw 'aia'
aia
In riga:1 car:1
+ throw 'aia'
+ ~~~~~~~~~~~~
   + CategoryInfo          : OperationStopped: (aia:String) [], RuntimeException
   + FullyQualifiedErrorId : aia

Sollevare un'eccezione di tipo System.IO.IOException:
PS C:\temp> throw New-Object IO.IOException 'Unita Z: non presente'
Unita Z: non presente
In riga:1 car:1
+ throw New-Object IO.IOException 'Unita Z: non presente'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : OperationStopped: (:) [], IOException
   + FullyQualifiedErrorId : Unita Z: non presente
```

Viene usata questa tecnica nel caso di parametri obbligatori:

```
PS C:\ > function Greet($name=$(throw 'Parametro $name obbligatorio!')        ){
        "Ciao, $Name"
}
Greet

Parametro $name obbligatorio!
In riga:1 car:24
+ function Greet($name=$(throw 'Parametro $name obbligatorio!')    ){
+                        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : OperationStopped: (Parametro $name obbligatorio!:String) [], RuntimeException
   + FullyQualifiedErrorId : Parametro $name obbligatorio!


PS C:\ > Greet Spider
Ciao Spider
```

Perche' le eccezioni sollevate tramite l'istruzione throw son considerate da PS errori fatali, la loro comparsa pregiudica l'esecuzione del resto dello script o della funzione

Trap
Non trattata qui

Try/catch/finally
Deriva dal C++.
La sintassi e' la seguente

```
try{
        <Comandi>
}
catch{
        <Gestione Errore>
```

```
}
oppure:
try{
        <Comandi>
}
finally{
        <Codice Finale>
}
oppure:
try{
        <Comandi>
}
catch{
        <Gestione Errore>
}
finally{
        <Codice Finale>
}
```

Esempio

```
PS C:\ > 3/0

Tentativo di divisione per zero.
In riga:1 car:1
+ 3/0
+ ~~~
   + CategoryInfo          : NotSpecified: (:) [], RuntimeException
   + FullyQualifiedErrorId : RuntimeException

try {
   3/0
}
catch{
"Hey c'e' un Errore"
}

#Viene stampato:
Hey c'e' un Errore
```

E' possibile limitare il tipo di eccezione gestito dal blocco catch specificando la classe di interesse. In questo modo e' possibile impiegare piu' blocchi catch all'interno dello stesso costrutto; tuttavia la shell, al verificarsi di un errore fatale, utilizza il primo blocco catch compatibile con il tipo dell'eccezione sollevata, in ordine di dichiarazione. La sintassi puo' essere schematizzata cosi':

```
[<Blocco Try>]
catch [<Tipo Eccezione1>]{
        Gestione Errore
}
catch [<Tipo Eccezione2>]{
        Gestione Errore
}
.....
catch {
        Gestione Errore
```

}
[<Blocco Finally>]

Nello script seguente si impiegano tre blocchi catch, in grado di operare con tipi differenti di eccezione.
Al verificarsi dell'errore fatale, la shell utilizza il primo blocco compatibile con l'eccezione.

```
try {
   "Prima riga"
   3/0
   "Terza Riga"
}
catch [IO.IOException]{
   "Catch IO"
}
catch [DivideByZeroException]{
   "Catch Divisione x zero"
}
catch {
   "Catch Generico"
}
Prima riga
Catch Divisione x zero
```

Il blocco Finally e' usato spesso per liberare le risorse eventualmente impegnate dal codice del blocco try,
a prescindere dal verificarsi di un errore fatale (cioe' il blocco finally viene sempre eseguito anche se
non ci sono stati errori).
Esempio nello script seguente il blocco finally e' utilizzato per eliminare il file temporaneo creato nel try:

```
$tempFileName=[IO.Path]::GetTempFileName()
try{
   "Creazione di $tempFileName..."
   "Test File" > $tempFileName
   3/$null
   #....
}
catch{
   Write-Host " Catch Generico"
}
Finally{
   "Rimozione di $tempFileName..."
   Remove-Item $tempFileName
}
Creazione di C:\ Users\acer\AppData\Local\Temp\tmpFF79.tmp...
 Catch Generico
Rimozione di C:\Users\acer\AppData\Local\Temp\tmpFF79.tmp...
```

nel blocco catch la variabile automatica $_ restituisce il riferimento all'errore fatale corrente attraverso
un'istanza della classe System.Management.Automation.ErrorRecord.

Cause degli errori fatali
Glierrori fatali possono essere generati da uno script quando:
*) Lo scipt contiene errori di sintassi; in questo caso la shell solleva automaticamente un'eccezione
di tipo System.Management.Automation.ParseException

*)lo script o funzione sollevano un'eccezione mediante l'istruzione throw.

Errori non fatali
Gli errori non fatali non interrompono il flusso di esecuzione di uno script.
A differenza di quelle sollevate dagli script e dai cmdlet, le eccezioni sollevate direttamente dai metodi e dalle proprieta' delle classi del Framework MS .NET sono considerate dalla shell errori non fatali.

```
PS C:\ > 1,2,3|ForEach-Object {$_;if($_ -eq 2){throw}}
1
2
ScriptHalted
In riga:1 car:39
+ 1,2,3|ForEach-Object {$_;if($_ -eq 2){throw}}
+                               ~~~~~
   + CategoryInfo          : OperationStopped: (:) [], RuntimeException
   + FullyQualifiedErrorId : ScriptHalted
```

Notiamo invece come nello script seguente l'eccezione dell'input non corretto non impedisca il flusso di esecuzione (si noti l'ultima riga do output)

```
 PS C:\ > #1,2,3|ForEach-Object {$_;if($_ -eq 2){throw}}
1,'a',3|ForEach-Object {[int]::Parse($_)}
1
Eccezione durante la chiamata di "Parse" con "1" argomento/i: "Formato della stringa di input non
corretto."
In riga:2 car:25
+ 1,'a',3|ForEach-Object {[int]::Parse($_)}
+                         ~~~~~~~~~~~~~~~~
   + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
   + FullyQualifiedErrorId : FormatException

3
```

Diversamente dagli errori fatali, quelli non fatali non sono gestibili attraverso i costrutti try/catch/finally.

Write-Error
il cmdlet Write-Error consente a uno script di generare errori non fatali. In tal caso la shell genera un'eccezione di tipo Microsoft.Powershell.Commands.WriteErrorException.
Sintassi:
Write-Error <Messaggio>
Write-Error -Exception <Eccezione>

Esempio, si noti che non viene interrotto il flusso dello script:

```
 PS C:\ > 1,2,3|ForEach-Object {$_;if($_ -eq 2){Write-Error "Boom"}}
#1,'a',3|ForEach-Object {[int]::Parse($_)}
1
2
1,2,3|ForEach-Object {$_;if($_ -eq 2){Write-Error "Boom"}}
#1,'a',3|ForEach-Object {[int]::Parse($_)} : Boom
   + CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorException
```

```
   + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException

3
```

Variare il comportamento nei cmdlet
La shell e' stata progettata ritenedo opportuno evitare l'interruzione del flusso
a causa di un problema di entita' minore.
Nello script che segue per esempio si creano dapprima due file di testo e poi
si chiama il cmdlet Remove-Item per eliminare quelli piu' un file inesistente.
Il risultato e'0 che Remove-Item fgenera un errore non fatale per
quest'ultimo file, ma elimina gli altri elementi, di fatto proseguendo con l'esecuzione:

```
PS C:\ > "Test" > "DeleteMe.txt"
"Test" > "DeleteMe2.txt"
Remove-Item DeleteMe.txt, dummy.txt, DeleteMe2.txt

Remove-Item : Impossibile trovare il percorso 'C:\temp\dummy.txt' perché non esiste.
In riga:3 car:1
+ Remove-Item DeleteMe.txt, dummy.txt, DeleteMe2.txt
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : ObjectNotFound: (C:\temp\dummy.txt:String) [Remove-Item],
ItemNotFoundException
   + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.RemoveItemCommand

PS C:\ > @(Get-childItem DeleteMe?.txt).Count
0
```

Inoltre mediante il parametro -ErrorAction (o -EA) e' possibile scegliere come qualsiasi comando debba rispondere
alla presenza di un errore non fatale:
I valori ammessi dal parametro -ErrorAction
Valore Descrizione

Continue: Valore predefinito, visualizza il messaggio di errore e prosegue con l'esecuzione dell'attaivita'
SilentlyContinue: Prosegue con l'esecuzione dell'attivita' senza visualizzare il messaggio di errore
Stop: Visualizza il messaggio di errore e arresta l'esecuzione dell'attivita', in maniera simile agli errori fatali
Inquire: Visualizza il messaggio di errore, ma chiede all'utente se proseguire con l'attivita' o terminarla.
Nel blocco che sehue lo script precedente e' rivisto affinche' richieda conferma prima di proseguire in seguito
all'errore non fatale:

```
PS C:\ > "Test" > "DeleteMe.txt"
>> "Test" > "DeleteMe2.txt"
>> Copy-Item DeleteMe.txt, dummy.txt, DeleteMe2.txt C:\Temp -EA inquire
>>

Conferma
Impossibile sovrascrivere l'elemento C: \DeleteMe.txt su se stesso.
[S] Sì  [T] Sì a tutti  [I] Interrompi comando  [O] Sospendi  [?] Guida
(il valore predefinito è "S"):
```

Riassumendo gli errori non fatali possono essere generati da uno script quando:

*)lo script richiama un metodo o una proprieta' del Framework MS .NET che solleva internamente un'eccezione
*)lo script richiama il cmdlet Write-Error.

Come detto i cmdlet sono progettati per generare errori non fatali in tutte le circostanze che lo richiedono.

Strutture degli Errori
Come detto le eccezioni catturate all'interno degli script sono restituite dalla shell come oggetti della classe System.Management.Automation.ErrorRecord che contiene alcune proprieta' illustrate di seguito:
Proprieta' Descrizione

CategoryInfo: Contiene informazioni strutturate sulla tipologia dell'errore
Exception: Ritorna l'eccezione originale, alla base dell'errore
InvocationInfo: Ritorna informazioni dettagliate sul comando e sulla riga dove l'errore si e' verificato
TargetObject: Ritorna l'oggetto che il comando stava elaborando quando si e' verificato l'errore.

Ogni oggetto relativo a un'eccezione della classe System.Exception a sua volta contiene alcune informazioni utili.
Proprieta' Descrizione

Message: Ritorna il messaggio testuale dell'eccezione
Source: Ritorna il nome del modulo in cui si e' verificato il problema
InnerException: Ritorna l'eventuale eccezione interna (le eccezioni possono essere strutturate una dentro l'altra su piu' livelli)

Esempio nello script che segue si utilizza la proprieta' InvocationInfo:

```
try {
3/$null
}
catch {
   "Errore provocato da:{0}" -f $_.InvocationInfo.Line

}
Errore provocato da:3/$null
```

VARIABILi automatiche e preferenze

$Error
La variabile automatica $Error contiene la collezione deglli errori generati durante la sessione corrente, ordinati
a partire dal piu' recente.

$MaximumErrorCount
Si utilizza per impostare il amssimo numero di elementi contenuto in $Error, che per default vale 256.

$LastExitCode
E' pari al codice numerico di uscita di un programma lanciato nella Shell
Esempio
```
PS C:\ > cmd.exe /C "DIR Z:\"
Impossibile trovare il percorso specificato.
```

```
PS C:\ > ECHO $LastExitCode
1
```

$?
Contiene un valore logico che varia in base allo stato dell'ultima operazione eseguita:
un valore pari a $true indica che il comando precedente e' andato a buon fine, mentre un valore $false segnala l'opposto. La shell assegna il valore $false alla variabile $? appena viene rilevato un errore fatale o meno da parte dello script, del cmdlet chiamato oppure di un oggetto del Framework .NEY; nel caso di eseguibili a console inoltre la shell imposta automaticamente il valore $? a $false se questi ritornano un codice di uscita numerico diverso da 0.

Esempio:

```
PS C:\ > Remove-Item fake.txt -EA SilentlyContinue
PS C:\ > if (-not $?) {"Rimozione Fallita!"}
```

Rimozione Fallita!

Diversamente da quanto avviene con l'omonima istruzione presente nella shell bash questa variabile non contiene il codice numerico di uscita dell'ultimo eseguibile a console lanciato.
Per risalire a tale valore e' sufficiente utilizzare la variabile automatica $LastExitCode.

$ErrorActionPreference
Modifica la gestione predefinita degli errori non fatali all'interno dei cmdlet. Il valore contenuto in questa variabile fornisce automaticamente ai cmdlet la preferenza per il parametro -ErrorAction nel caso quest'ultimo non sia esplicitamente fornito dall'utente. Quindi questa variabile puo' assumere gli stessi valori del parametro -ErrorAction.
Esempio

```
PS C:\ > $ErrorActionPreference='Stop'
PS C:\ > "Test">"DeleteMe.txt"
PS C:\ > "Test">"DeleteMe2.txt"
Remove-Item deleteMe.txt, Fake.txt, DeleteMe.txt
Remove-Item : Impossibile trovare il percorso 'C: \Fake.txt' perché non esiste.
In riga:3 car:1
+ Remove-Item deleteMe.txt, Fake.txt, DeleteMe.txt
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : ObjectNotFound: (C: \Fake.txt:String) [Remove-Item], ItemNotFoundException
   + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.RemoveItemCommand

Remove-Item : Impossibile trovare il percorso 'C: \DeleteMe.txt' perché non esiste.
In riga:3 car:1
+ Remove-Item deleteMe.txt, Fake.txt, DeleteMe.txt
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : ObjectNotFound: (C: \DeleteMe.txt:String) [Remove-Item], ItemNotFoundException
   + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.RemoveItemCommand
PS C:\ > @(Get-ChildItem DeleteMe?.Txt).Count
1
```

SCHEDULED TASKS
Con Powershell e' possibile gestire le operazioni pianificate tramite il modulo ScheduledTasks che contiene (vedere l'url: https://docs.microsoft.com/it-it/powershell/module/scheduledtasks/new-scheduledtasksettingsset?view=win10-ps per la sintassi) i cmdlet seguenti:
Disable-ScheduledTask: Disabilita un Task Schedulato

```
PS C:\temp> Get-ScheduledTask -TaskName Google*  | Disable-ScheduledTask

TaskPath                                 TaskName                          State
--------                                 --------                          -----
\                                        GoogleUpdateTaskMachineUA         Disabled
\                                        GoogleUpdateTaskMachineCore       Disabled
```

Enable-ScheduledTask: Abilita un Task Schedulato

Export-ScheduledTask: Esporta un Task come stringa XML

Get-ScheduledTask: Elenca i task schedulati e registrati sul computer locale

```
PS C:\temp> Get-ScheduledTask -TaskName Google*

TaskPath                                 TaskName                          State
--------                                 --------                          -----
\                                        GoogleUpdateTaskMachineCore       Ready
\                                        GoogleUpdateTaskMachineUA         Ready
```

Get-ScheduledTaskInfo: Restituisce informazioni sui task schedulati.

New-ScheduledTask: Crea un task schedulato. NB. Questo cmdlet non registra automaticamente il task nel servizio Task Scheduler di Windows.

New-ScheduledTaskAction: Indica un'azione o comando che il task deve eseguire Quando il Task Scheduler di Windows lo avvia.

New-ScheduledTaskPrincipal: Permette di eseguire un Task da eseguire nel contesto di sicurezza di un account specificato indipendemente se l'account e' loggato o meno.

New-ScheduledTaskSettingsSet: Crea un oggetto contenente impostazioni per il task.

New-ScheduledTaskTrigger: Crea un oggetto Trigger che indica quando avviare il task. Il trigger puo' essere di tipo temporale o a evento e avviera' il task. Il trigger Time-based avvia un task ad un specifico istante o piu' volte giornalmente o settimanalmente. Il trigger Event-Based avvia un Task quando il Sistema si avvia o quando un utente si logga. I trigger possono esesre piu' di uno , in questo caso Task Scheduler avviera' il task ogni volta che il trigger si attiva.
Creates a scheduled task trigger object.

Register-ClusteredScheduledTask: Registers a scheduled task on a failover cluster.

Register-ScheduledTask: Registra la definizione tramite un nome del Task Scheduled sul computer locale

```
$run = (Get-Date).AddMinutes(3);
$action = New-ScheduledTaskAction -Execute 'c:\\windows\\system32\\Notepad.exe' -Argument
'c:\\temp\\test.txt' -WorkingDirectory 'C:\\temp'
$trigger = New-ScheduledTaskTrigger -Once -At $run

$settings = New-ScheduledTaskSettingsSet
#Register-ScheduledTask -Action $action -Trigger $trigger -TaskName "MYTASK" -Description "Task di
Prova" -User "system" -Settings $settings

$V=3
$taskName = "MYTASK$V"
```

```
# construct task
$inputObject = New-ScheduledTask -Action $action -Trigger $trigger -Description "Task di Prova$V" -
Settings $settings

$user = "$env:USERDOMAIN\$env:USERNAME"
$password = "Password"
# Register the task in the folder
Register-ScheduledTask $taskName -InputObject $inputObject -TaskPath "myTasks" -User "system"
#Register-ScheduledTask $taskName -InputObject $inputObject -TaskPath "myTasks" -User $user -
Password $password
```

Set-ClusteredScheduledTask: Changes settings for a clustered scheduled task.

Set-ScheduledTask: Modifica le impostazioni di un Task, anche se questo e' in esecuzione.

Start-ScheduledTask: esegue in modo asincrono un task registrato

Stop-ScheduledTask: ferma tutte le istanze di un task.

Unregister-ClusteredScheduledTask: Removes a scheduled task from a failover cluster.

Unregister-ScheduledTask: De-registra un scheduled task.

Si noti che sono necessari diritti elevati per disabilitare o abilitare le attività su un sistema. Ciò significa che è necessario eseguire PowerShell da un prompt dei comandi con privilegi elevati.

SICUREZZA
La sicurezza (Role-based) in windows si basa su due concetti:
Identity: Un oggetto identity rappresenta un singolo utente ed espone proprieta' come il nome dell'utente, se l'utente e' stato autenticato e il security provider utilizzato per autenticare l'utente. (gli utenti ASP.NET possono essere anonimi, cioe' utenti che non sono stati autenticati. Un oggetto identity viene utilizzato nella fase di autenticazione del login

Principal: Un oggetto principal e' una combinazione di una user identity piu' tutti i ruoli assegnati all'utente. Questi ruoli sono i gruppi Windows a cui l'utente appartiene, se l'utente e' l'utente Windows interattivo o e' un utente ASP.NET autenticato con l'autenticazione Windows. Un oggetto principal viene utilizzato nel decidere se una o piu' risorse deve essere concessa a un utente (fase di autorizzazione).
Le classi e le interfacce inerenti la sicurezza role-based si trovano nel namespace System.Security.Principal.

Tutte le classi identity implementano l'interfaccia Iidentity, che espone le seguenti properieta' RadOnly:
IsAuthenticated restituisce True se l'utente e' stato autenticato
AuthenticationType Restituisce una stringa che indica quale metodo di autenticazione e' stato utilizzato puo' essere NTLM, Basic, Forms, Passport o quanto restituito dal provider di autenticazione (Kerberos ad esempio) .
Name – restituisce il nome utente con cui l'utente ha effettuato il login. Per un utente Windows, e' una stringa nel formato nomedominio\nomeutente o nomesistema\nomeutente; per un utente web autenticato con l'autenticazioneForms, e' il nome dell'utente digitato nella pagina di login.

Con il metodo GeCurrent della classe .net Security.Principal.WindowsIdentity si recupera l'oggetto WindowsIdentity che fornisce informazioni anche sul token di autenticazione.

```
PS C:\> [Security.Principal.WindowsIdentity]::GetCurrent()
```

```
AuthenticationType : NTLM
ImpersonationLevel : None
IsAuthenticated    : True
IsGuest            : False
IsSystem           : False
IsAnonymous        : False
Name               : LAPTOP-POJRGMSE\acer
Owner              : S-1-5-21-2019186598-2351530761-1348927116-1001
User               : S-1-5-21-2019186598-2351530761-1348927116-1001
Groups             : {S-1-5-21-2019186598-2351530761-1348927116-513, S-1-1-0,
                     S-1-5-21-2019186598-2351530761-1348927116-1006,
                     S-1-5-21-2019186598-2351530761-1348927116-1003...}
Token              : 2680
AccessToken        : Microsoft.Win32.SafeHandles.SafeAccessTokenHandle
UserClaims         : {http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name:
                     LAPTOP-POJRGMSE\acer,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/primarysid:
                     S-1-5-21-2019186598-2351530761-1348927116-1001,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/primarygroupsid:
                     S-1-5-21-2019186598-2351530761-1348927116-513,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid:
                     S-1-5-21-2019186598-2351530761-1348927116-513...}
DeviceClaims       : {}
Claims             : {http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name:
                     LAPTOP-POJRGMSE\acer,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/primarysid:
                     S-1-5-21-2019186598-2351530761-1348927116-1001,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/primarygroupsid:
                     S-1-5-21-2019186598-2351530761-1348927116-513,
                     http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid:
                     S-1-5-21-2019186598-2351530761-1348927116-513...}
Actor              :
BootstrapContext   :
Label              :
NameClaimType      : http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name
RoleClaimType      : http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid
```

Per visualizzare l'elenco dei SID dei gruppi di appartenenza in modo piu' leggibile:

```
PS C:\> [System.Security.Principal.WindowsIdentity]::GetCurrent().Groups | Format-Table -auto
```

Il SID, security Identifier, e' un numero (compresa tra 8 e 68 bytes)usato per identificare utenti, gruppi e computer in Windows [3].
Esempio stampa il nome descrittivo di alcuni SID:

```
PS C:\> $sids = 'S-1-5-32-544','S-1-2-1'
foreach($sid in $sids){
$ntAccount =
[System.Security.Principal.SecurityIdentifier]::new($sid).Translate([System.Security.Principal.NTAccount]).V
alue
echo $ntAccount
}

BUILTIN\Administrators
```

ACCESSO CONSOLE

Esempio da nome a SID

```
PSC:\>([System.Security.Principal.NTAccount]"acer").Translate([System.Security.Principal.SecurityIdentifier)
```

Per elencare tutti i gruppi di appartenenza:

```
PS C:\> $groups=@{}
([System.Security.Principal.WindowsIdentity]::GetCurrent()).Groups | %{
   $groups.$_ =$_.Translate([System.Security.Principal.NTAccount])}

([System.DirectoryServices.AccountManagement.UserPrincipal]::Current).GetGroups().SID | %{
   $groups.$_ =$_.Translate([System.Security.Principal.NTAccount])}
echo $groups
```

Verifico se lo script gira o meno come administrator

```
$currentPrincipal = New-Object
Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurrent())
$currentPrincipal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)



# verifica run whit elevate privilege
$currentPrincipal = New-Object
Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurrent())

$isElevate = $currentPrincipal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)

If($isElevate) {
  Write-Host "Lo script sta girando con elevati privilegi" -ForeGround Green
} else {
  Write-Host "Attenzione lo script richiede elevati privilegi" -ForeGround Red
}
```

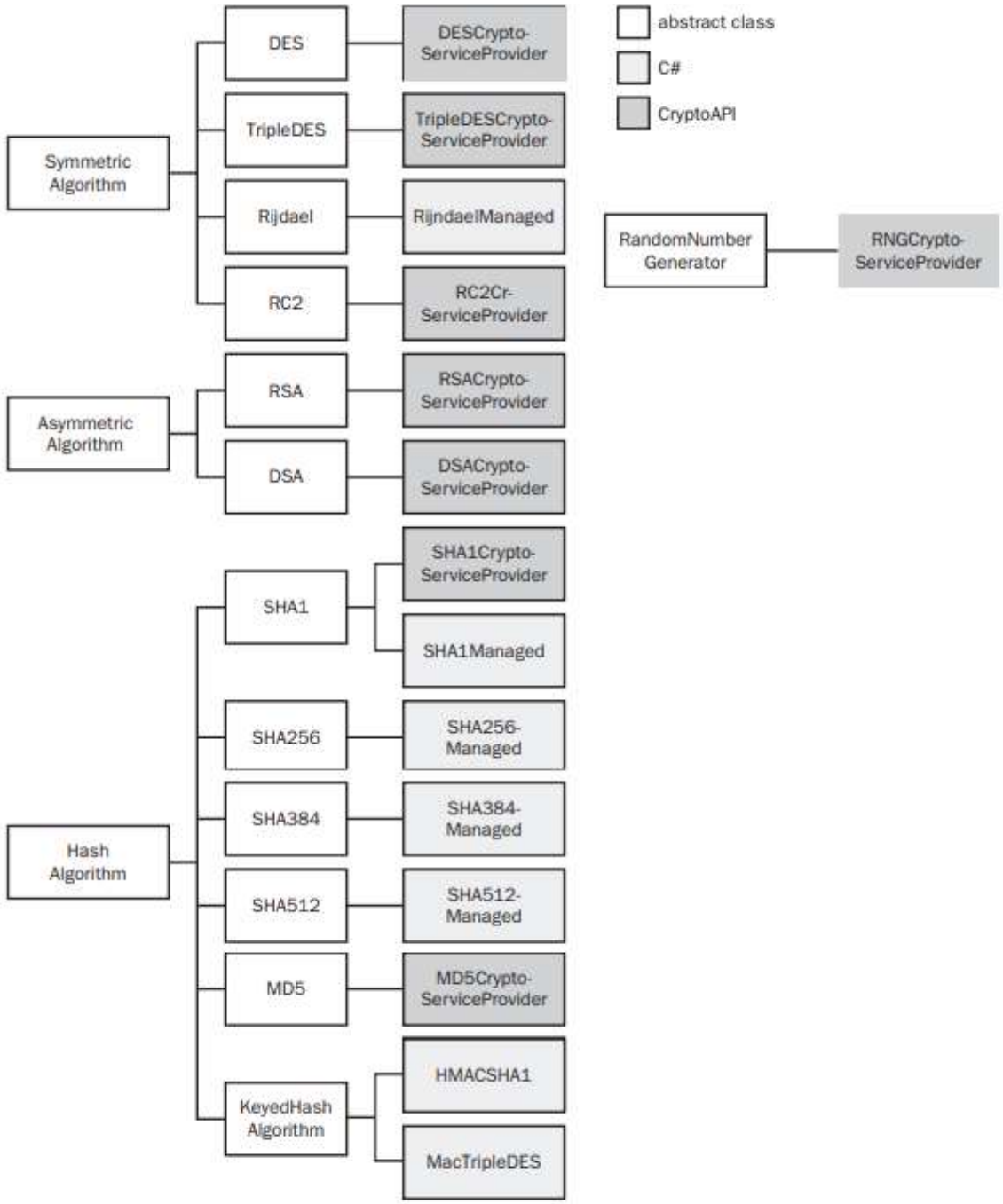
Funzione che mi dice se un utente appartiene al gruppo degli amministratri:

```
function Test-LocalAdminGroupMembership {
  param([string] $user)

  # Load the required assembly (a no-op if already loaded).
  Add-Type -AssemblyName System.DirectoryServices.AccountManagement

  # Obtain the specified user as a UserPrincipal instance.
  $up = try {
    if (-not $user) { # default to current user

      [System.DirectoryServices.AccountManagement.UserPrincipal]::Current
    } else {
      [System.DirectoryServices.AccountManagement.UserPrincipal]::FindByIdentity(
        [System.DirectoryServices.AccountManagement.UserPrincipal]::Current.Context,
        $user
      )
    }
  } catch {
    Throw
  }
```

```
  # See if the well-known SID of the local Administrators group
  $up.GetGroups().SID.Value -contains 'S-1-5-32-544'
}
```

CRITTOGRAFIA

Il .NET Frameqork contiene molte classi appartenenti al namespace System.Security.Cryptography che permettono di crittografare e decrittografare i dati, firmare i dati con codice hash, e autenticare i dati per dimostrarne la provenienza. Nella Figura seguente viene riportata la gerarchia delle piu' importanti classi nel namespace System.Security.Cryptography.



Dall'ereditarieta' delle classi si puo' rendere il codice generico, considerando che le classi astratte espongono un metodo statico Create().

Crittografia Simmetrica

Gli algoritmi per la crittografia simmetrica sono contenuti nel namespace System.Security.Cryptography. Devono essere asseggnate le proprieta' Key e IV prima di crittografare e decrittografare i dati. La proprieta' Key e' la password che l'algoritmo utilizza nel processo crittografico e deve essere assegnato ad un array di byte di una determinata lunghezza. Ciascun algoritmo supporta chiavi di differenti lunghezze ad esempio 128, 192 o 256 bit nel caso del Rijandael o 192 bit per il TripleDes. Per default viene utilizzata per la sicurezza la chiave piu' lunga, che sara' random se non si assegna un valore specifico alla proprieta' Key. La proprieta' IV e' il cosiddetto vettore di inizializzazione che viene utilizzato per aumentare la casualita' dei blocchi crittografati. A differenza della proprieta' Key non e' necessario memorizzarne il valore in una posizione sicura, ma e' essenziale utilizzare i valori random per i relativi byte. Se non siassegna esplicitamente questa proprieta' viene generato automaticamente un array Byte random quando si avvia il processo di crittografazione. E' cruciale che si riutilizzi lo stesso valore nel decrittografare i dati.

Ad esempio voglio conoscere la classe di default (che puo'essere modificato aggiungendo un elemento <cryptographySetting> nel file machine.config) impostata sulla mia macchina per la crittografia simmetrica:

```
PS C:\> Remove-Variable encr
$encr = [System.Security.Cryptography.SymmetricAlgorithm]::Create()
echo $encr.GetType().Name $encr.KeySize

RijndaelManaged         256
```

Si possono creare le classi specificandone il nome nella funzione Create():

```
PS C:\ >
[System.Security.Cryptography.SymmetricAlgorithm]::Create("System.Security.Cryptography.Rijndael").GetType().Name

RijndaelManaged


[System.Security.Cryptography.SymmetricAlgorithm]::Create("System.Security.Cryptography.DES").GetType().Name

DESCryptoServiceProvider
```

Oppure istanziando la classe concreta:

```
PS C:\ > $hash = [System.Security.Cryptography.SHA1]::Create();

echo $hash.GetType().Name

SHA1CryptoServiceProvider
```

Esempio di seguito uno script per criptare/decriptare una stringa

```
using namespace System.IO
using namespace  System.Security.Cryptography

Function InitializeSymmetricAlgorithm( [SymmetricAlgorithm]$encr, [string]$fileName){
   $fs=$null
   $ivLength = $encr.IV.Length

   try{
        $fs = New-Object FileStream ($fileName, [FileMode]::OpenOrCreate)
```

```
            if ($fs.Length -eq 0){
                    $fs.Write($encr.Key, 0, $encr.Key.Length)
                    $fs.Write($encr.IV, 0, $ivLength)
            }
            else{
                    #Altrimenti, legge la chiave dal file
                    ##$ek=$encr.Key | % { [System.BitConverter]::ToString($_) }
                   $bytes = [byte[]]::new($encr.Key.Length)
                    $br = $fs.Read($bytes,0,$encr.Key.Length)

                    $encr.Key = $bytes;
                    #Legge il vettore di inizializzazione
        $bytes = [byte[]]::new($ivLength)
                    $br=$fs.Read($bytes,0,$ivLength)
                    $encr.IV = $bytes
            }
  }

  finally{
            if ($fs -ne $null){
                    $fs.Close()
     }
  }
}

 Function SymCrypt([String]$str,$sPathKey,$sPathEncrypt){
          $data=[system.text.encoding]::ASCII.GetBytes($str)
           #$encr =[.RijndaelManaged]::Create()
          $encr = [SymmetricAlgorithm]::Create("System.Security.Cryptography.Rijndael")
          InitializeSymmetricAlgorithm $encr $sPathKey
          $fs=$null
          $cs=$null
          try{
                    $fs = New-Object FileStream ($sPathEncrypt, [System.IO.FileMode]::Create)
                    $cs = New-Object CryptoStream($fs, $encr.CreateEncryptor(), [CryptoStreamMode]::Write)
                    $cs.Write($data, 0, $data.Length)
                    $cs.FlushFinalBlock()
          }
          finally{
                    if ($cs -ne $null){
                              $cs.Close()
                    }
                    elseif ($fs -ne $null){
                              $fs.Close()
                    }
          }

}
Function SymDecrypt($sPathKey,$sPathEncrypt){
  #$encr =[RijndaelManaged]::Create()
  $encr = [SymmetricAlgorithm]::Create("Rijndael")
          InitializeSymmetricAlgorithm $encr $sPathKey
          $fs = $null
```

```
        $cs = $null
        $data2 = $null
        try{
                $fs = New-Object FileStream ($sPathEncrypt, [System.IO.FileMode]::Open)
                $cs = New-Object CryptoStream($fs, $encr.CreateDecryptor(), [CryptoStreamMode]::Read)
                $data = [byte[]]::new($fs.Length)
                $br=$cs.Read($data, 0, $data.Length)
                $ret = [System.Text.Encoding]::ASCII.GetString($data)
                return $ret
        }
        finally{
                if ($cs -ne $null){
                        $cs.Close()
                 }
                elseif ($fs -ne $null){
                        $fs.Close()
                }
        }
}

SymCrypt "hello world vuol dire Salve Mondo!!!"  "C:\\temp\\key.dat" "c:\\temp\\encrypt.dat"
$r= SymDecrypt "C:\\temp\\key.dat" "c:\\temp\\encrypt.dat"
echo $r
```

Esempio di seguito uno script per criptare/decriptare un File

```
Function SymEncryptFile([System.Security.Cryptography.SymmetricAlgorithm]$encr,
$sourceFile,$destFile,$includeIV){
        $souFs=$null
   $desFs=$null
        $cs=$null
        $BUFFERSIZE = 4096
        try{
                $souFs = New-Object System.IO.FileStream($sourceFile, [System.IO.FileMode]::Open)
                $desFs =  New-Object System.IO.FileStream($destFile, [System.IO.FileMode]::Create)
                if ($includeIV){
                        $desFs.Write($encr.IV,0,$encr.IV.Length)
     }
     $cs = New-Object System.Security.Cryptography.CryptoStream($desFs, $encr.CreateEncryptor(),
[System.Security.Cryptography.CryptoStreamMode]::Write)

     $bytes = [byte[]]::new($BUFFERSIZE)
                while(1 -eq 1){
                        $bytesRead = $souFs.Read($bytes, 0, $bytes.Length)
                        if($bytesRead -eq 0){
                                break
       }
                        $cs.Write($bytes,0,$bytesRead)
                }
                $cs.FlushFinalBlock()
         }
catch{
Write-Error $Error[0]
}
```

```
        finally{
                if ($cs -ne $null){
                        $cs.Close()
    }
                elseif ($souFs -ne $null){
                        $souFs.Close()
    }
    elseif ($desFs -ne $null){
                        $desFs.Close()
    }
        }
}


 Function SymDecryptFile([System.Security.Cryptography.SymmetricAlgorithm ]$encr, $sourceFile,
$destFile, $includeIV){
        $souFs=$null
  $desFs=$null
        $cs=$null
        $BUFFERSIZE = 4096
        try{
                $souFs = New-Object System.IO.FileStream($sourceFile, [System.IO.FileMode]::Open)
                $desFs =  New-Object System.IO.FileStream($destFile, [System.IO.FileMode]::Create)
                if ($includeIV){
        $iv = [byte[]]::new($encr.IV.Length)
                        $souFs.Read($iv, 0, $iv.Length)
                        $encr.IV = $iv
                }
     $cs = New-Object System.Security.Cryptography.CryptoStream($souFs, $encr.CreateDecryptor(),
[System.Security.Cryptography.CryptoStreamMode]::Read)
     $bytes = [byte[]]::new($BUFFERSIZE)
                while(1 -eq 1){
                        $bytesRead = $cs.Read($bytes, 0, $bytes.Length)
                        if ($bytesRead -eq 0){
                                break
        }
                        $desFs.Write($bytes, 0, $bytesRead)
                }
        }
catch{
Write-Error $Error[0]
}
        finally{
                if ($cs -ne $null){
                        $cs.Close()
    }
                elseif ($souFs -ne $null){
                        $souFs.Close()
    }
    elseif ($desFs -ne $null){
                        $desFs.Close()
    }
        }
```

```
}

[System.Security.Cryptography.RijndaelManaged] $encr
=[System.Security.Cryptography.RijndaelManaged]::Create()

SymEncryptFile $encr "c:\\temp\\calc.exe" "c:\\temp\\calc.dat"
SymDecryptFile $encr "c:\\temp\\calc.dat" "c:\\temp\\calc2.exe"
```

Crittografia Asimmetrica

Dalla figura precedenta si vede che la classe astratta AsymmetricAlgorithm contiene due classi RSA e DSA. Gli algoritmi di crittografia asimmetrica operano su due chiavi distinte: una chiave pubblica e una chiave privata. Si puo' generare una coppia di chiavi e distribuire la chiave pubblica in modo che chiunque possa utilizzarla per crittografare un messaggio decifrabile solo dal proprietario della chiave privata. Analogamente a un algoritmo simmetrico un oggetto AsymmetricAlgorithm appena creato contiene chiavi generate in modo random. Si puo' esportarle in XML su file econ il metodo ToXmlString e importarle con il metodo FromXmlString. Il protocollo HTTPS utilizza la crittografia asimmetrica per codificare la chiave utilizzata dall'algoritmo simmetrico che codifica i dati effettivi.

Esempio stampa l'Algoritmo di default per la crittografia asimmetrica per la macchina locale

```
using namespace System.IO
using namespace  System.Security.Cryptography

Function prtDfltAsymAlgo(){
 $asym = [AsymmetricAlgorithm]::Create()
 echo ($asym.GetType().Name)


 [String]$publicKey = $asym.ToXmlString($false)
 echo $publicKey
 [String]$bothKeys = $asym.ToXmlString($true)
 $bothKeys | out-file "C:\\temp\\asym.xml"
}

prtDfltAsymAlgo
        RSACryptoServiceProvider
```

Nell'esempio seguente viene crittato e decrittato un messaggio tramite chiave asimmetrica.

```
Function InitializeAsymmetricAlgorithm($asym, $fileName, $includePrivateKey){

        $fs=$null
        try{
                $fs = New-Object System.IO.FileStream($fileName,
[System.IO.FileMode]::OpenOrCreate,[System.IO.FileAccess]::ReadWrite)

                if ($fs.Length -eq 0){
                        $sw = new-object System.IO.StreamWriter($fs)
                        $sw.Write($asym.ToXmlString($includePrivateKey))
                        $sw.Close()
                }
                else{
                        $sr = new-object System.IO.StreamReader($fs)
```

```
                        $asym.FromXmlString($sr.ReadToEnd())
                }
        }
        finally{
                if ($fs -ne $null){
                        $fs.Close()
    }
        }
}
Function AsymEncryptDecrypt(){
 $rsaCSP = new-Object    System.Security.Cryptography.RSACryptoServiceProvider

 InitializeAsymmetricAlgorithm $rsaCSP "C:\\Temp\\rsakey2.xml"  $false


 $data = [System.Text.Encoding]::ASCII.GetBytes("ciao a tutti amici come state?")
 $encryptedBytes = $rsaCSP.Encrypt($data,$false);

 $rsaCSP2 = new-Object System.Security.Cryptography.RSACryptoServiceProvider
 InitializeAsymmetricAlgorithm $rsaCSP2 "C:\\Temp\\rsakey2.xml" $false
 $data2 = $rsaCSP2.Decrypt($encryptedBytes, $false)

            $DecryptedContent = [system.text.encoding]::UTF8.GetString($data2)
              $DecryptedContent = $DecryptedContent.Trim()

 Write-Host $DecryptedContent
}
 AsymEncryptDecrypt
```

Cmdlet per i Certificati con Powershell

Quando si parla di Crittografia asimmetrica prima di tutto abbiamo bisogno di un certificato. Questo certificato includerà una chiave privata e una chiave pubblica. Con la chiave privata possiamo decriptare i dati. Con la chiave pubblica possiamo crittografare i dati. Ciò significa che se qualcuno ha la mia chiave pubblica (posso darla a qualcuno senza preoccupazioni) può crittografare i dati a me indirizzati. E io sono l'unico su questo pianeta che può decifrarlo. Perché sono l'unico che ha la chiave privata.

New-SelfSignedCertificate - Creazione di un certificato:

Il cmdlet New-SelfSignedCertificate crea un certificato autofirmato a scopo di test. Utilizzando il parametro CloneCert, è possibile creare un certificato di prova in base a un certificato esistente con tutte le impostazioni copiate dal certificato originale ad eccezione della chiave pubblica. Il cmdlet crea una nuova chiave con lo stesso algoritmo e lunghezza. Potrebbe essere necessaria la delega quando si utilizza questo cmdlet con il servizio remoto di Windows PowerShell e la modifica della configurazione utente.

Parametri:

-CertStoreLocation
Specifica l'archivio certificati in cui archiviare il nuovo certificato. Se il percorso corrente è Cert: \ CurrentUser o Cert: \ CurrentUser \ My, l'archivio predefinito è Cert: \ CurrentUser \ My. Se il percorso corrente è Cert: \ LocalMachine o Cert: \ LocalMachine \ My, l'archivio predefinito è Cert: \ LocalMachine \ My. In caso contrario, è necessario specificare Cert: \ CurrentUser \ My o Cert: \ LocalMachine \ My per questo parametro. Questo parametro non supporta altri archivi di certificati.
-DnsName

Specifica uno o più nomi DNS da inserire nell'estensione del nome alternativo del soggetto del certificato quando un certificato da copiare non viene specificato tramite il parametro CloneCert. Anche il primo nome DNS viene salvato come nome del soggetto. Se non viene specificato alcun certificato di firma, anche il primo nome DNS viene salvato come Nome emittente.
-KeyUsage
Specifica l'utilizzo della chiave.

-Type
Specifica il tipo di certificato creato da questo cmdlet.
I valori accettati per questo parametro sono:

CodeSigningCert
Custom
DocumentEncryptionCert
DocumentEncryptionCertLegacyCsp
SSLServerAuthentication (default)

Esempio:

```
PS C:\temp> New-SelfSignedCertificate -DnsName myTest -CertStoreLocation "Cert:\CurrentUser\My" -
KeyUsage KeyEncipherment,DataEncipherment, KeyAgreement -Type DocumentEncryptionCert


   PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My

Thumbprint                Subject

----------                -------

5A72520120DF95B1661C5603CB8FDF72623D5246  CN= myTest
```

Ricerca negli stores di un certificato per subject:

```
PS C:\temp> dir cert: -Recurse | Where-Object { $_.Subject -like "*myTes*" }
```

Ricerca dei certificati non validi

```
PS C:\temp>  dir cert: -Recurse | Where-Object { $_.NotAfter -lt (Get-Date 2018-12-31) }

PS C:\temp> dir cert: -Recurse | Where-Object { $_.NotAfter -gt (Get-Date) -and $_.NotAfter -lt (Get-
Date).AddYears(1) }
```

Nella crittografia a chiave pubblica, un'impronta digitale a chiave pubblica è una breve sequenza di byte utilizzata per identificare una chiave pubblica più lunga. Le impronte digitali vengono create applicando una funzione hash crittografica a una chiave pubblica. Poiché le impronte digitali sono più corte delle chiavi a cui si riferiscono, possono essere utilizzate per semplificare alcune attività di gestione delle chiavi. Nel software Microsoft, "identificazione personale" (thumbprint) viene utilizzata invece di "impronta digitale" (fingerprint).
Per verificare la posizione del certificato con la console di gestione di Windows avviamo: certmgr.msc
e ci posizionamo certificati- Utente Corrente>Personali>Certificati .
Per ottenere loa stessa informazione in Powershell usiamo il cmdlet:

```
PS C:\> Get-Childitem -Path Cert:\CurrentUser\My -DocumentEncryptionCert
```

Negli Script seguenti veiene illustrato la creazione di un sito Web su IIS che accetta connessioni https

```
#Create Web Site
$iissite="website2"
$app="myApp"
$siteLoc= "C:\Temp\$iissite"
$appLoc="$siteLoc\$app"

remove-website -Name $iissite
Remove-Item -Recurse -Force $siteLoc

new-Item -type Directory -PATH $siteLoc
new-Item -type Directory -PATH $appLoc
new-Item -type File  -path "$appLoc\index.html"
Set-Content "$appLoc\index.html" "<!DOCTYPE html>`n<html> <body><h1>Hello World</h1></body>
<html>"

<#
Set-Content "C:\Temp\$iissite\$app\web.config" @"
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.web>
      <authorization>
        <allow users="?"/>
      </authorization>
    </system.web>
</configuration>
"@
#>

New-Website -Name $iissite -Port 8081 -PhysicalPath $siteLoc
New-Item -Type Application -Path "IIS:\Sites\$iissite\$app" -physicalPath $appLoc

$path = $siteLoc
$acl  = Get-Acl -Path $path

$user = New-Object -TypeName 'System.Security.Principal.SecurityIdentifier' -ArgumentList
@([System.Security.Principal.WellKnownSidType]::BuiltinUsersSid, $null)
$rule = New-Object System.Security.AccessControl.FileSystemAccessRule($user, 'FullControl',
'ContainerInherit,ObjectInherit', 'None', 'Allow')
$acl.SetAccessRule($rule)
Set-Acl -Path $path -AclObject $acl

#Start-WebSite -Name "$iissite"

#http://laptop-pojrgmse:8081/$app/

<#creare altre app:
new-Item -type Directory -PATH "C:\Temp\$iissite\$app"
new-Item -type File  -path "C:\Temp\$iissite\$app\index.html"
Set-Content "C:\Temp\$iissite\$app\index.html" "<!DOCTYPE html>`n<html> <body><h1>Hello
World2</h1></body> <html>"
New-Item -Type Application -Path "IIS:\Sites\$iissite\$app" -physicalPath "C:\Temp\$iissite\$app"
```

```
#>
```

Creare un certificato

```
$hostname = "$((Get-WmiObject win32_computersystem).DNSHostName)"
$certFolder="My"  #"WebHosting"
$certLoc="cert:\LocalMachine\$certFolder"
New-SelfSignedCertificate -dnsname $hostname -KeyLength 2048 -CertStoreLocation $certLoc -NotAfter
(Get-Date).AddYears(20)
```

Binding del sito con il certificato

```
$httpsPort=443
$iisSite = "website2"
$hostname = "$((Get-WmiObject win32_computersystem).DNSHostName)"   #.$((Get-WmiObject
win32_computersystem).Domain)
$certFolder="My"  #"WebHosting"
$certLoc="cert:\LocalMachine\$certFolder"
$thumbprint = (Get-ChildItem -Path $certLoc | Where-Object {$_.Subject -match
"CN=$hostname"}).Thumbprint;
$cert = "$certLoc\$thumbprint"
#Remove Cert
#Get-ChildItem $cert | Remove-Item

#Remove Web Bind
#Get-WebBinding -Port $httpsPort -Name $iisSite | Remove-WebBinding
New-WebBinding -Name $iisSite -Protocol https -Port $httpsPort -HostHeader $hostname -SslFlags 1 # SNI
certificate
(Get-WebBinding -Name $iisSite -Port $httpsPort -Protocol "https" -HostHeader
$hostname).AddSslCertificate($thumbprint, $certFolder)
```

Crittografia dei dati
Crittografiamo un messaggio e archiviamolo in un file.

```
PS C:\temp>  "This is a secret message" | Protect-CmsMessage -To cn= myTest -OutFile C:\Temp\secret.txt

PS C:\temp> type secret.txt
-----BEGIN CMS-----
MIIBqwYJKoZIhvcNAQcDoIIBnDCCAZgCAQAxggFDMIIBPwIBADAnMBMxETAPBgNVBAMMCHBld2Ey
MzAzAhAZ0OV7F1OLj05E8CRDrmhFMA0GCSqGSIb3DQEBBzAABIIBAFmKhMZBg0usRSK0FnwGwCcB
sL7ZMFP1cbKzjOt67kPkhdSdfYm572jKVb4f7ryabl/+/aYk5kLL2cOm+EXoPHGciPHDGJY3mFV5
xtQshAK8+y+gSRcPZdoGEW1qTiJHJUnHFEjj3fGmG4WNeRY9mnsNCk1GN9Ampcgwcsvllkuics5N
KMvb5Y48TNinMWKBrEzEVbNy4ulM6VLj/8A2m81N0Pjol6ruYfpF51RrEKEj9YpdCUzi5moNpUFZ
mY8ovV10T0qB5qGtUXivLc/l3GKzu5JVzeS9aA1KrDqDjtxThmjUWQ3mzwhwOVi/N6HdmE9sDMeY
VOJ+zSit2k9Hy5wwTAYJKoZIhvcNAQcBMB0GCWCGSAFlAwQBKgQQ7rxtV1iqTynRvMUb50OrL4Ag
zcBwxe4IllUZ6uxIpdM6zumnrw3Z/71ZOopDthUBUIw=
-----END CMS-----
```

Decrittazione dei dati
Per decrittografare i dati crittografati eseguire Unprotect-CmsMessage. Assicurandosi di aver effettuato l'accesso con l'account utente che ha creato il certificato e che abbia la chiave privata.
PS C:\temp> Unprotect-CmsMessage -Path C:\Temp\secret.txt
This is a secret message

Se mi dovessi loggare alla macchina con un diverso account il comando precedente dare errore.
Verificare se un File e' crittografato:

```
PS C:\windows> dir *.exe | Get-AuthenticodeSignature | Format-Table -Auto

   Directory: C:\windows


SignerCertificate                  Status Path
-----------------                  ------ ----
A4341B9FD50FB9964283220A36A1EF6F6FAA7840 Valid  bfsvc.exe
A4341B9FD50FB9964283220A36A1EF6F6FAA7840 Valid  explorer.exe
A4341B9FD50FB9964283220A36A1EF6F6FAA7840 Valid  HelpPane.exe
A4341B9FD50FB9964283220A36A1EF6F6FAA7840 Valid  hh.exe
A4341B9FD50FB9964283220A36A1EF6F6FAA7840 Valid  notepad.exe
3BDA323E552DB1FDE5F4FBEE75D6D5B2B187EEDC Valid  PSEXESVC.exe
A4341B9FD50FB9964283220A36A1EF6F6FAA7840 Valid  regedit.exe
9A4AC4F7C818A9104A4BD9FAA5DB14F603CFD7C5 Valid  RtCRU64.exe
312860D2047EB81F8F58C29FF19ECDB4C634CF6A Valid  splwow64.exe
A4341B9FD50FB9964283220A36A1EF6F6FAA7840 Valid  winhlp32.exe
312860D2047EB81F8F58C29FF19ECDB4C634CF6A Valid  write.exeHASH
```

Le classi che derivano direttamente o indirittamente da HashAlgorithm permettono di analizzare un blocco di dati per calcolarne il valore hash, che puo' essere considerato come una firma dei dati. La funzione hash non puo'essere invertita quindi e' virtualmente impossibile creare un blocco di dati che corrisponda ad un valore hash noto, cosi' come due blocchi di dati producano lo stesso hash. Questa caratteristica puo' rilevare qualsiasi modifica a un file di cui sia noto il valore hash originale.

```
PS C:\>
function prtDfltHashAlgo(){
        $hash = [System.Security.Cryptography.HashAlgorithm]::Create()
        echo $hash.GetType().Name
 }

 prtDfltHashAlgo
SHA1CryptoServiceProvider
```

Esempio Calcola l'hash di una stringa o di un file

```
function ComputeHashFromFile($fileName){
 $md5StringBuilder = New-Object System.Text.StringBuilder 50
 $sha1 = [System.Security.Cryptography.SHA1]::Create("SHA1")
 $fs=$null
  try{
     $fs = new-Object System.IO.FileStream($fileName, [System.IO.FileMode]::Open)
      return $sha1.ComputeHash($fs)
  }
 finally{
        if ($fs -ne $null){
          $fs.Close()
         }
  }
}

 function ComputeHashFromString($text){
  $md5StringBuilder = New-Object System.Text.StringBuilder 50
```

```
  $ue = New-Object System.Text.UTF8Encoding
  $sha1 = [System.Security.Cryptography.SHA1]::Create("SHA1")

   $data=[system.text.encoding]::UTF8.GetBytes($text)
   return $sha1.ComputeHash($data)
}

$r1=ComputeHashFromFile "C:\\Temp\\hello.txt"
$rs1=$r1 | % { [System.BitConverter]::ToString($_) }
write-host $rs1

$r2= ComputeHashFromString "hello world"
#write-host $r2
$rs2=$r2 | % { [System.BitConverter]::ToString($_) }
write-host $rs2
```

Di seguito un esempio di script per l'hashing comprensivo di molti aspetti visti preso da:
http://dbadailystuff.com/2013/03/11/get-hash-a-powershell-hash-function

```
<#
.SYNOPSIS
Gets the hash value of a file or string

.DESCRIPTION
Gets the hash value of a file or string
It uses System.Security.Cryptography.HashAlgorithm (http://msdn.microsoft.com/en-
us/library/system.security.cryptography.hashalgorithm.aspx)
and FileStream Class (http://msdn.microsoft.com/en-us/library/system.io.filestream.aspx)
Based on: http://blog.brianhartsock.com/2008/12/13/using-powershell-for-md5-checksums/ and some
ideas on Microsoft Online Help

Be aware, to avoid confusions, that if you use the pipeline, the behaviour is the same as using -Text, not -
File

.PARAMETER File
File to get the hash from.

.PARAMETER Text
Text string to get the hash from

.PARAMETER Algorithm
Type of hash algorithm to use. Default is SHA1

.EXAMPLE
C:\PS> Get-Hash "hello_world.txt"
Gets the SHA1 from myFile.txt file. When there's no explicit parameter, it uses -File

.EXAMPLE
Get-Hash -File "C:\temp\hello_world.txt"
Gets the SHA1 from myFile.txt file

.EXAMPLE
```

```
C:\PS> Get-Hash -Algorithm "MD5" -Text "Hello Wold!"
Gets the MD5 from a string

.EXAMPLE
C:\PS> "Hello Wold!" | Get-Hash
We can pass a string throught the pipeline

.EXAMPLE
Get-Content "c:\temp\hello_world.txt" | Get-Hash
It gets the string from Get-Content

.EXAMPLE
Get-ChildItem "C:\temp\*.txt" | %{ Write-Output "File: $($_)   has this hash: $(Get-Hash $_)" }
This is a more complex example gets the hash of all "*.tmp" files

.NOTES
DBA daily stuff (http://dbadailystuff.com) by Josep Martínez Vilà
Licensed under a Creative Commons Attribution 3.0 Unported License

.LINK
Original post: https://dbadailystuff.com/2013/03/11/get-hash-a-powershell-hash-function/
#>
function Get-Hash
{
   Param
   (
      [parameter(Mandatory=$true, ValueFromPipeline=$true, ParameterSetName="set1")]
      [String]
      $text,
      [parameter(Position=0, Mandatory=$true,
      ValueFromPipeline=$false, ParameterSetName="set2")]
      [String]
      $file = "",
      [parameter(Mandatory=$false, ValueFromPipeline=$false)]
      [ValidateSet("MD5", "SHA", "SHA1", "SHA-256", "SHA-384", "SHA-512")]
      [String]
      $algorithm = "SHA1"
   )
   Begin
   {
      $hashAlgorithm = [System.Security.Cryptography.HashAlgorithm]::Create($algorithm)
   }
   Process
   {
      $md5StringBuilder = New-Object System.Text.StringBuilder 50
      $ue = New-Object System.Text.UTF8Encoding

      if ($file){
         try {
            if (!(Test-Path -literalpath $file)){
               throw "Test-Path returned false."
            }
         }
```

```
        catch {
          throw "Get-Hash - File not found or without permisions: [$file]. $_"
        }
        try {
          [System.IO.FileStream]$fileStream = [System.IO.File]::Open($file, [System.IO.FileMode]::Open);
          $hashAlgorithm.ComputeHash($fileStream) |
            % { [void] $md5StringBuilder.Append($_.ToString("x2")) }
        }
        catch {
          throw "Get-Hash - Error reading or hashing the file: [$file]"
        }
        finally {
          $fileStream.Close()
          $fileStream.Dispose()
        }
      }
      else {
        $hashAlgorithm.ComputeHash($ue.GetBytes($text)) |
          % { [void] $md5StringBuilder.Append($_.ToString("x2")) }
      }

      return $md5StringBuilder.ToString()
    }
}
```

```
Write-Output "`nSome examples how to call it:"
Get-Hash "c:\temp\myScriptFile.sql"
Get-Hash "c:\temp\br[a{ets.txt"
Get-Hash "c:\temp\node.exe"
Get-Hash -Algorithm "MD5" -Text "A MD5 checksum!"

Write-Output "`nFour hello world examples that return the same hash value:"
Get-Hash "c:\temp\hello_world.txt"
"Hello Wold!" | Get-Hash
Get-Hash -Text "Hello Wold!"
Get-Content "c:\temp\hello_world.txt" | Get-Hash
Get-ChildItem "C:\temp\*world*.txt" |
  %{ Write-Output "File: $($_)   has this hash: $(Get-Hash $_)" }
```

Si puo' invocare il metodo ComputeHash di un oggetto SHA1CryptoServiceProvider piu' volte, senza dover reinizializzare l'oggetto. Il metodo ComputeHash funziona bene se i dati di cui si vuole ottonere il valore hash sono in un buffer o in uno stream, ma non e' sufficientemente flessibile se si vuole ottenere il valore da piu' sorgenti, ad esempio per generare un valore hash di tutti i file di una directory. In questo caso si deve creare un CryptoStream che esegue il wrapping di un ulteriore stream e che calcola l'hash dei dati scritti su quest'ultimo. Se si associa un oggetto CryptoStream a un oggetto che deriva da HashAlgorithm, i byte non vengono affatto modificati mentre vengono letti dallo (o scritti nello) stream sottostante; invece il relativo valore hash viene calcolato e sara' disponibile al termine dell'operazione. Se il CryptoStream non deve inviare nessun dato, si puo' eseguire il wrapping su uno stream fittizio, come nell'esempio seguente nullStream. Nell'esempio seguente viene calcolato l'hash di una certa directory poi viene modificato un file nella stessa directory e si varifica che l'hash e' cambiato.

```
using namespace System.IO
using namespace  System.Security.Cryptography

```

```
class NullStream : System.IO.MemoryStream
{
        [Void] Write([Byte[]] $buffer, [int] $offset, [int] $count){

        }
}

Function ComputeHashFromDirectory([String] $path){
   $BUFFERSIZE = 4096
        $sha1 = [SHA1]::Create()
   $obj=new-object NullStream
        #$cs = new-object CryptoStream $obj $sha1
[System.Security.Cryptography.CryptoStreamMode]::Write
$cs = New-Object CryptoStream($obj, $sha1, [CryptoStreamMode]::Write);


        foreach ($file in [System.IO.Directory]::GetFiles($path)){
                $fs = $null;
                try{
                        $fs = new-object FileStream($file, [FileMode]::Open)
        $buffer = [byte[]]::new($BUFFERSIZE)
                        while($true){
                                $readBytes = $fs.Read($buffer, 0, $buffer.Length);
                                if ($readBytes -eq 0){
                                        break;
         }
                                $cs.Write($buffer, 0, $readBytes);
                        }
                }
                finally{
                 if ($fs -ne $null){
                        $fs.Close()
    }
                }
        }
        $cs.FlushFinalBlock()
        $cs.Close();
        return $sha1.Hash
}

$r=ComputeHashFromDirectory "C:\\temp\\AAAA"
$rs=$r | % { [System.BitConverter]::ToString($_) }
write-host $rs

C2 76 A4 BB 6A CC FC 27 60 A7 26 83 9B 76 0B D8 A6 1D B9 22
C6 CC 0E C5 80 DA 6C E3 EF 03 27 D3 11 96 E7 7B 79 96 76 F1
```

Algoritmi Hash Con Chiave

Le funzioni ComputerHashFromFile e ComputeHashFromDirectory precedentemente introdotte permettono di rilevare facilmente se un file e' stato manomesso, senza dover memorizzare una copia del file originale. Tuttavia, assumono che venga salvato il valore hash in una posizione sicura in modo che un intruso non possa ricalcolare semplicemente il valore hash della nuova versione del file e sostituire il valore hash originale con il valore ricacolato. Questo puo' non essere realistico se si stanno trasmettendo dati via

rete poiche' il valore hash deve viaggiare con i dati stessi. Si puo' risolvere questo problema con una delle due classi concrete che derivano dalla classe astratta KeyedHashAlgorithm cioe' HMACSHA1 e MacTripleDES. Un algoritmo con chiave e' simile ad un ordinario algoritmo hash, ad eccezione del dover inizializzare la funzione hash con una chiave segreta. Questo meccanismo impedisce a un intruso il ricalcolo del valore hash poiche' non conosce la chiave utilizzata dalla funzione hash. Nell'esempio che segue la funzione AppendHash valuta il valore hash di un file e accoda questo valore al file stesso.

```
using namespace System.IO
using namespace  System.Security.Cryptography
Function AppendHash([String]$fileName, [Byte[]]$key, $discardHash) {
 $BUFFERSIZE = 4096
        $keyhash = new-object HMACSHA1 #($key)
   $keyhash.key = $key
        $fs = $null
        $cs = $null
        try{
                $fs = new-object FileStream($fileName, [FileMode]::Open)
                $obj=new-object NullStream
                $cs = new-object CryptoStream($obj, $keyhash, [CryptoStreamMode]::Write)
                $bytesToRead = $fs.Length - $keyhash.HashSize/8
                $buffer = [byte[]]::new($BUFFERSIZE)
                while($true){
                        $bytesRead = $fs.Read($buffer, 0, [Math]::Min($bytesToRead, $buffer.Length))
                        if ($bytesRead -eq 0){
                                break
                        }
                        $bytesToRead -= $bytesRead
                        $cs.Write($buffer, 0, $bytesRead)
                }
                $cs.FlushFinalBlock()
                $realHash = $keyhash.Hash
                $storedHash=[byte[]]::new($keyhash.HashSize/8)
                $fs.Read($storedHash, 0, $storedHash.Length)
                if($discardHash){
                        $fs.SetLength($fs.Length - $storedHash.Length)
    }
                for($i=0; $i -lt $realHash.Length; $i++){
                        if($realHash[$i] -ne $storedHash[$i]){
                                return $false
       }
    }
                return $true
        }
        finally{
                if ($cs -ne $null){
                        $cs.Close()
    }
                if ($fs -ne $null){
                        $fs.Close()
    }
        }
}
```

```
[int32[]]$key = 3, 45, 78, 123, 9, 77
#$key=[Text.Encoding]::ASCII.GetBytes("8191a1b1c1d1e1.....")
AppendHash "c:\\temp\\a1.txt" $key $true
```

VerifyHash verifica un file firmato delle precedente routine e rimuove opzionalmente il valore hash.La procedura VerifyHash non puo' utilizzare un metodo ComputeHash sullo stream di Input poiche' questo metodo comprenderebbe il valore hash alla fine del file. Invece questa procedura utilizza un CryptoStream che esegue il wrapping di un oggetto NullStream per calcolare il valore hash dei dati man mano che

vengono letti dal file ma interrompe la lettura subito prima del codice hash in coda.

```
using namespace System.IO
using namespace  System.Security.Cryptography
class NullStream : System.IO.MemoryStream
{
        [Void] Write([Byte[]] $buffer, [int] $offset, [int] $count){

        }
}
#Return True if a file signed with AppendHash hasn't been modified,
#and optionally deletes the hash value.
Function VerifyHash([string]$filename,[Byte[]] $key,$discardHash){
  $BUFFERSIZE = 4096
  # Create a keyed hash object with given key.
  $keyhash = New-Object HMACSHA1($key)
  $fs = $null
  $cs = $null
  try{
    # Open the file.
    $fs = New-Object FileStream($filename, [FileMode]::Open)
    # Create a CryptoStream that just evaluates the hash value.
    $cs = New-Object CryptoStream(New-Object NullStream, $keyhash, [CryptoStreamMode]::Write)
    # The number of bytes in the file excluding hash at the end
    $bytesToRead = $fs.Length - $keyhash.HashSize / 8
    # Read the file contents while evaluating its hash code.
    $buffer = [byte[]]::new($BUFFERSIZE)
    while($true){
      $bytesRead = $fs.Read($buffer, 0, [Math]::Min($bytesToRead, $buffer.Length))
      if ($bytesRead -eq 0){
        break
      }
      $bytesToRead -= $bytesRead
      # Send data to null stream just to evaluate the hash code.
      $cs.Write($buffer, 0, $bytesRead)
    }
    # Evaluate the hash value of data read so far.
    $cs.FlushFinalBlock()
    $realHash = $keyhash.Hash
    # Read the hash value stored at the end of the file.
    $storedHash = [byte[]]::new($keyhash.HashSize / 8)
    $fs.Read($storedHash, 0, $storedHash.Length)
    # Discard hash value if so requested.
    if ($discardHash){
     $fs.SetLength($fs.Length - $storedHash.Length)
```

```
    }
    # Compare real and stored hash bytes, return False if they don't match.
    For ($i= 0; $i -lt $realHash.Length; $i++){
       If ($realHas[$i] -ne $storedHash[$i]){
          Return $false
       }
    }
    # All bytes match, hash is verified.
    Return $True
         }
         finally{
                  if ($cs -ne $null){
                           $cs.Close()
    }
                  if ($fs -ne $null){
                           $fs.Close()
    }
         }
}
```

Valori Random

Molte delle classi crittografiche viste richiedono una chiave che contiene byterandom. Per ovvi motivi piu' aleatoria e' la chiave piu' sicura e' la crittografia.Per un robusto meccanismo crittografico tuttavia non si puo' utilizzare semplicemente la classe System.Random per generare chiavi randompoiche' questa classe genera sequenze di valori riproducibili e non sufficientemente random per scopi crittografici. Invece si deve utilizzare la classe RNGCryptoServiceProvider che esegue il wrapping alla libreria CryptoAPI.

```
using namespace  System.Security.Cryptography
$rng = New-object RNGCryptoServiceProvider
$bytes = [byte[]]::new(64)
$rng.GetBytes($bytes)
$r=$bytes | % { [System.BitConverter]::ToString($_) }
Write-Host $r
AF 73 B3 BF BC 6B 49 4C 2A 2D 31 D2 BD E2 C4 75 E9 22 09 31 …..
```

Un ulteriore modo comune per generare una chiave da utilizzare per scopi crittografici e' derivarla da una password testuale. Ad esempio, si puo' richiedere all'utente una password leggibile e il programma generera' una sequenza di byte apparentemente randomutilizzando una sorta di trasformazione di ciascun carattere della stringa. Si puo' utilizzare un oggetto PasswordDerivedBytes il cui costruttore accetta una stringa e un array di byte noto come salt che ha lo stesso scopo dei vettori di inizializzazione negli algoritmi crittografici simmetrici: servono a nascondere qualsiasi pattern nei dati di input

```
 $password = "helloworld"
$salt = [int32[]]$ia = 3, 45, 78, 123, 9, 77
$pdb = New-Object PasswordDeriveBytes($password, $salt)
$r = $pdb.GetBytes(32)
$r=$r | % { [System.BitConverter]::ToString($_) }
write-host $r
92 64 7B 6C AB 81 16 74 7D EA ….
```

Questo codice crea la stessa sequenza di byte per ciascuna password fornita (purche' si utilizzi lo stesso array salt).

Fornire le credenziali

Tramite interfaccia CUI

```
PS C:\ > $user = Read-Host "Enter Username"

Enter Username: alice

PS C:\ > $pass = Read-Host "Enter Password" -AsSecureString

Enter Password: *********

PS C:\ > echo $user,$pass
alice
System.Security.SecureString
```

oppure tramite Fnsetra di dialog

```
PS C:\ > $MyCredential = Get-Credential
```

Notare che la password non viene mostrata in chiaro come il nome dell'utente.
Per automatizzare l'uso delle password si usano i cmdlet ConvertTo-SecureString e ConvertFrom-SecureString, in quanto come fa vedere l'esempio seguente si vede che nel file di testo ci finisce System.Security.SecureString e non la password.crittografata:

```
PS C:\ > (get-credential).password > pwd.txt
Type pwd.txt
System.Security.SecureString
```

```
PS C:\ > "miapass" | ConvertTo-SecureString -AsPlainText -Force | ConvertFrom-SecureString | Out-File
"C:\Temp\Password.txt"
Oppure:
(Get-Credential).Password | ConvertFrom-SecureString | Out-File "C:\Temp\Password.txt"

Read-Host "Enter Password" -AsSecureString |  ConvertFrom-SecureString | Out-File
"C:\Temp\Password.txt"
```

Ora invece ho una stringa criptata

```
PS C:\ > Get-Content "C:\Temp\Password.txt"
01000000d08c9ddf0115d1118c7a00c04fc297eb01000000fe7f2db340e9d341bfcd044ab848ca92000000000
2000000000010660000000100002000000397c2e2acef0a8fbda429348c17a50d9a01611905038213fadda1
ac71090007a000000000e8000000002000020000000c63401de3f02afc8086557248b3765e1dd7274f31ce71
74014df2ed69a729a571000000041ea161f3ffd9b7cce3ca18e268c18f940000000428c478f620fcd233d3cbd6
92c1f3288436e279d60456cfe9c475672dc131c4e73144b21198cf1d2c5e04bbe8d85d2c5c82222c78974115c
3fe98775a1249c08
```

Altro Esempio:

```
$cred = Get-Credential -message "inserire credenziali" -User $env:USERDOMAIN\$env:USERNAME
$cred | get-member
convertfrom-securestring $cred.password
$cred.GetNetworkCredential() | Select username, password
```

Occorre sottolineare che la stringa cosi' criptata puo' essere utilizzata solo dallo stesso utente che l'ha creata in quanto gestita dal sistema operativo locale, oppure solo nella stessa macchina, cioe' non si puo' accedere se il file password fosse in un'altra macchina in rete

```
PS C:\ > Get-Content c:\temp\password.txt | ConvertTo-SecureString
ConvertTo-SecureString : Chiave non utilizzabile nello stato specificato.
In riga:1 car:36
```

```
+ Get-Content c:\temp\password.txt | ConvertTo-SecureString
+                                   ~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (:) [ConvertTo-SecureString], CryptographicException
    + FullyQualifiedErrorId :
ImportSecureString_InvalidArgument_CryptographicError,Microsoft.PowerShell.Commands.Conv
  ertToSecureStringCommand
```

Per poter renderla utilizzabile da altri occorre ricorrere il parametro -Key o -SecureKey.
Creiamo una chiave

```
$Key = New-Object Byte[] 32
[Security.Cryptography.RNGCryptoServiceProvider]::Create().GetBytes($Key)
$Key | out-file "\\Machine1\SharedPath\AES.key

$PasswordFile = "\\Machine1\SharedPath\Password.txt"
$KeyFile = "\\Machine1\SharedPath\AES.key"
$Key = Get-Content $KeyFile
$Password = "P@ssword1" | ConvertTo-SecureString -AsPlainText -Force
$Password | ConvertFrom-SecureString -key $Key | Out-File $PasswordFile
#oppure:
(get-credential).Password | ConvertFrom-SecureString -key (get-content $KeyFile) | set-content
$PasswordFile
$password = Get-Content $PasswordFile | ConvertTo-SecureString -Key (Get-Content $KeyFile)
$credential = New-Object System.Management.Automation.PsCredential("utente",$password)

$User = "MyUserName"
$PasswordFile = "\\Machine1\SharedPath\Password.txt"
$KeyFile = "\\Machine1\SharedPath\AES.key"
$key = Get-Content $KeyFile
$MyCredential = New-Object -TypeName System.Management.Automation.PSCredential `
 -ArgumentList $User, (Get-Content $PasswordFile | ConvertTo-SecureString -Key $key)
```

Riferimenti

[1] https://docs.microsoft.com/en-us/powershell/
[2] https://xainey.github.io/2016/powershell-classes-and-concepts/
[3] https://docs.microsoft.com/it-it/troubleshoot/windows-server/identity/security-identifiers-in-windows