

#POWERCON2023

PowerShell Deep Dive: Best Practices, Security and AI scripting assistant

Ermanno Goletto

Microsoft MVP Alumni
e.goletto@outlook.it



[/devadmin.it](https://www.facebook.com/devadmin.it)



[@ermannog](https://twitter.com/ermannog)



[/ermannogoletto](https://www.linkedin.com/company/ermannogoletto)

Roberto Massa

Microsoft MVP Alumni
robimassa@outlook.it



[/robi.massa.cn](https://www.facebook.com/robi.massa.cn)



[@robi_massa](https://twitter.com/robi_massa)



[/robimassa](https://www.linkedin.com/company/robimassa)

Speaker

Ermanno Goletto è laureato in Ingegneria Elettronica al Politecnico di Torino e nel 1997 inizia a lavorare nel campo dell'Information Technology presso aziende informatiche occupandosi di progettazione e sviluppo di applicazioni, amministrazione di database, architettura di sistema e sicurezza su piattaforma Microsoft

Attualmente in qualità di **Funzionario Informatico presso una Pubblica Amministrazione locale amministra l'infrastruttura informatica e si occupa della gestione della sicurezza informatica**

Ermanno è **certificato sulle tecnologie Microsoft dal 2004** (MCP, MCSA, MCTS su .NET, Exchange, SQL Server e SharePoint, MCITP su Windows Server e Virtualization, MCSE Server Infrastructure), fa parte dello **staff della community IT Pro ICTPower.it** e **collabora con la community Torino Technologies Group**

Microsoft MVP dal 2008 al 2018 nelle categorie Directory Services, Cloud and Datacenter Management e Enterprise Mobility, attualmente è **MVP Alumni** e **dal 2010 interviene come speaker in conferenze ed eventi dedicati al mondo ICT**



Speaker

Roberto Massa si occupa di Informatica dal 1989 nella **gestione di sistemi distribuiti in ambito networking in ambienti Windows Server, Linux, e Netware**, ha approfondito la conoscenza di **sistemi di monitoraggio Open-Source in particolare Nagios e Zenoss**

Attualmente è **Sistemista presso l'Azienda Ospedaliera S. Croce e Carle di Cuneo** dove si occupa anche delle analisi e delle implementazioni di soluzioni legate alla sicurezza dell'infrastruttura

Roberto è **certificato sulle tecnologie Microsoft Hyper-V** e da alcuni anni **si occupa di sicurezza** implementando soluzioni opensource basate su sistemi OpenBSD e OpenVPN ed in generale dell'**analisi di soluzioni tecniche legate alla fruizioni di applicazioni all'esterno del perimetro aziendale**

Microsoft MVP dal 2016 al 2019 per la categoria Cloud and Datacenter Management, attualmente è **MVP Alumni** e **dal 2007 partecipa come speaker ad eventi dedicati a Linux e alle tecnologie Microsoft**

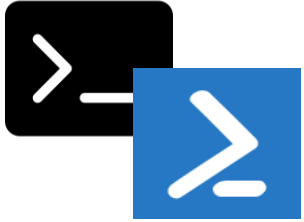


Agenda

- Evoluzione di PowerShell
- PowerShell Scripting Best Practices
- Sicurezza e distribuzione degli script Powershell
- Utilizzo dell'AI per lo sviluppo di script

Evoluzione di PowerShell

Evoluzione delle shell a riga di comando di Windows



Windows ha due shell della riga di comando: il **Prompt dei comandi** e **PowerShell**, entrambe permettono una comunicazione diretta tra l'utente e il sistema operativo o l'applicazione, fornendo un ambiente per automatizzare le operazioni IT



Il Prompt dei comandi è stata la prima shell integrata in Windows per automatizzare attività di routine, come la gestione degli account utente o i backup notturni, tramite file batch (.bat), successivamente con **Windows Script Host** è stato possibile eseguire script più sofisticati



PowerShell è stato progettato per estendere le funzionalità del Prompt dei comandi tramite l'esecuzione di cmdlet ed ad un linguaggio di scripting estensibile

*For the most robust, up-to-date Windows automation, **we recommend using PowerShell instead of Windows Commands or Windows Script Host for Windows automation***

<https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands>

***The Windows Cmd / Command-Line shell is NOT being removed from Windows in the near or distant future!
Also, Cmd != MS-DOS!***

January 4th, 2017 - Rich Turner Sr. Program Manager, Windows Console & Command-Line
<https://devblogs.microsoft.com/commandline/rumors-of-cmds-death-have-been-greatly-exaggerated/>

Scripting deprecated features



October 2023 VBScript is deprecated.

In future releases of Windows, VBScript will be available as a feature on demand before its removal from the operating system. VBScript will be available as a feature on demand before being retired in future Windows releases. Initially, the VBScript feature on demand will be preinstalled to allow for uninterrupted use while you prepare for the retirement of VBScript.



May 2021 The WMIC tool is deprecated in Windows 10, version 21H1 and the 21H1 General Availability Channel release of Windows Server.

This tool is superseded by Windows PowerShell for WMI. Note: This deprecation only applies to the command-line management tool. WMI itself isn't affected.



October 2017 (Windows 10 1709) Windows PowerShell 2.0

Applications and components should be migrated to PowerShell 5.0+.



November 2016 (Windows 10 Insider Preview Build 14971) PowerShell is now the defacto command shell from File Explorer.

It replaces Command Prompt (aka, "cmd.exe") in the WIN + X menu, in File Explorer's File menu, and in the context menu that appears when you shift-right-click the whitespace in File Explorer. Typing "cmd" (or "powershell") in File Explorer's address bar will remain a quick way to launch the command shell at that location.

<https://learn.microsoft.com/en-us/windows/whats-new/deprecated-features>

<https://learn.microsoft.com/en-us/windows/whats-new/deprecated-features-resources>

<https://blogs.windows.com/windows-insider/2016/11/17/announcing-windows-10-insider-preview-build-14971-for-pc/>

Versioni di PowerShell

PowerShell 5.1 è denominato "**Windows PowerShell**"
PowerShell 6.x è denominato "**PowerShell Core**"
PowerShell 7.x è denominato "**PowerShell**"

Windows PowerShell è la versione di PowerShell che usa il .NET Framework e viene eseguito solo in Windows, l'ultima versione è Windows PowerShell 5.1 basato su .NET Framework v4.5 che non verrà più aggiornata con nuove funzionalità e il cui supporto è associato alla versione di Windows in uso

Con il rilascio di PowerShell 6.0, PowerShell è diventato un progetto open source basato su .NET Core 2.0, il passaggio da .NET Framework a .NET Core ha consentito a PowerShell di diventare una soluzione multiplatforma eseguibile in Windows, macOS e Linux

PowerShell 7 è progettato per ambienti cloud, locali e ibridi, viene eseguito side-by-side con Windows PowerShell

\$PSVersionTable

Version	Release Date	Note
PowerShell 7.4 (preview)		Built on .NET 8.0.0-preview
PowerShell 7.3	Nov-2022	Built on .NET 7.0
PowerShell 7.2 (LTS-current)	Nov-2021	Built on .NET 6.0 (LTS-current)
PowerShell 7.1	Nov-2020	Built on .NET 5.0
PowerShell 7.0 (LTS)	Mar-2020	Built on .NET Core 3.1 (LTS)
PowerShell 6.2	Mar-2019	Built on .NET Core 2.1
PowerShell 6.1	Sep-2018	Built on .NET Core 2.1
PowerShell 6.0	Jan-2018	First release, built on .NET Core 2.0. Installable on Windows, Linux, and macOS
Windows PowerShell 5.1	Aug-2016	Released in Windows 10 Anniversary Update and Windows Server 2016, WMF 5.1
Windows PowerShell 5.0	Feb-2016	Released in Windows Management Framework (WMF) 5.0
Windows PowerShell 4.0	Oct-2013	Integrated in Windows 8.1 and with Windows Server 2012 R2, WMF 4.0
Windows PowerShell 3.0	Oct-2012	Integrated in Windows 8 and with Windows Server 2012 WMF 3.0
Windows PowerShell 2.0	Jul-2009	Integrated in Windows 7 and Windows Server 2008 R2, WMF 2.0
Windows PowerShell 1.0	Nov-2006	Optional component of Windows Server 2008

Windows PowerShell vs PowerShell

Windows PowerShell

Windows PowerShell è la versione di PowerShell che usa il **.NET Framework** e viene **eseguito solo in Windows**, l'ultima versione è **Windows PowerShell 5.1** basato su **.NET Framework v4.5** che **non verrà più aggiornata** con nuove funzionalità e il cui supporto è associato alla versione di Windows in uso

Windows PowerShell è compatibile con le versioni precedenti di Windows PowerShell, è possibile avviare una specifica versione tramite il parametro `-Version` di `powershell.exe`

Profile path: "\$HOME\Documents\WindowsPowerShell"

Administrator: C:\Program Files\PowerShell\7\pwsh.exe





PowerShell è basato sulle nuove versioni di .NET anziché su **.NET Framework** e il supporto è basato sulla versione di .NET su cui è stato compilato

PowerShell 7 non è completamente compatibile con le versioni precedenti di Windows PowerShell, ma è possibile utilizzare lo switch `UseWindowsPowerShell` cmdlet `Import-Module`

A partire da PowerShell 6, il file binario `powershell.exe` di PowerShell è stato rinominato `pwsh.exe` per Windows e `pwsh` per macOS e Linux per consentire la coesistenza con `powershell.exe`

Non è possibile utilizzare PowerShell ISE con PowerShell 6 o 7, per avere un ambiente di sviluppo grafico occorre utilizzare Visual Studio Code con installata la Microsoft's PowerShell extension

Windows PowerShell vs PowerShell FAQ

-  **Gli script Windows PowerShell funzionano in PowerShell?**
E' necessario testare ogni singolo script di Windows PowerShell per assicurarsi che funzioni anche in PowerShell
-  **Tutti i moduli per Windows PowerShell vengono eseguiti in PowerShell ?**
No, i moduli complessi devono essere adattati per PowerShell, in alcuni casi, esistono versioni diverse del modulo per entrambe le edizioni di PowerShell
-  **Gli script per PowerShell verranno eseguiti su tutte le piattaforme supportate?**
E' necessario testare ogni script su tutte le piattaforme su cui dovrebbe essere eseguito
-  **Dove è possibile scaricare i pacchetti d'installazione di PowerShell Core (v6.x) e Powershell (v7.x)?**
<https://github.com/PowerShell/PowerShell>

Il modulo di Azure Active Directory per PowerShell funziona solo con PowerShell dalla versione 3 alla versione 5.1
<https://learn.microsoft.com/en-us/troubleshoot/azure/active-directory/cannot-run-scripts-powershell>

PowerShell Scripting Best Practices

Style Rules



Utilizzare la notazione **Pascal Case** per i nomi di tutti gli **identificatori pubblici**: **Moduli, Funzioni, Cmdlet, Classi, Enum, Attributi, Public Fields, Proprietà, Variabili Globali, Costanti e Parametri**

Nome / NomeComposto



Utilizzare la notazione **Camel Case** per i nomi delle **Variabili** private

nome / nomeComposto



Utilizzare la notazione **Camel Case** per i nomi dei **Cmdlet** e il **minuscolo** per gli **operatori**



Utilizzare la regola di stile **One True Brace Style**, ogni istruzione con parentesi graffa deve avere la parentesi graffa di apertura alla fine di una riga e la parentesi graffa di chiusura all'inizio di una riga

```
if (10 -gt $ParameterOne) {  
    "Greater"  
} else {  
    "Lesser"  
}
```



Utilizzare **quattro spazi per ogni livelli d'indentazione** e **limitare ogni riga ad un massimo di 115 caratteri**



Commentare il codice con **frasi che descrivano il ragionamento** omettendo il punto finale



E' preferibile l'uso di **nomi espliciti completi per comandi e parametri piuttosto che alias o forme brevi**

<https://learn.microsoft.com/en-us/powershell/scripting/developer/cmdlet/strongly-encouraged-development-guidelines>

<https://poshcode.gitbook.io/powershell-practice-and-style/>

Definizione e tipizzazione dei parametri

```
Param(  
  [uint32]$AlertOnEventsFromDays=2,  
  [Parameter(Mandatory=$True)]  
  [string]$From,  
  [Parameter(Mandatory=$True)]  
  [string]$SmtpServer,  
  [Parameter(Mandatory=$True)]  
  [string]$To  
)
```



https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_advanced_parameters

Definizione di un parametro booleano

```
[switch]$SendMailAsAnonymous=$False
```

Esempio di validazione di un parametro stringa

```
[ValidateSet("Center","Fill","Fit","Stretch","Tile", IgnoreCase = true)]  
[String] $Location = 'Center'
```



<https://learn.microsoft.com/en-us/powershell/scripting/lang-spec/chapter-04>

<https://learn.microsoft.com/en-us/powershell/scripting/developer/cmdlet/how-to-validate-an-argument-set>

Impostazione dello Strict Mode

Set-StrictMode -Version **Latest**



- **1.0**
 - Impedisce i riferimenti a variabili non inizializzate, ad eccezione delle variabili non inizializzate nelle stringhe
- **2.0**
 - Impedisce riferimenti a variabili non inizializzate, incluse le variabili non inizializzate nelle stringhe
 - Impedisce riferimenti a proprietà inesistenti di un oggetto
 - Impedisce le chiamate di funzione che usano la sintassi per chiamare i metodi
- **3.0**
 - Impedisce riferimenti a variabili non inizializzate, incluse variabili non inizializzate nelle stringhe
 - Impedisce riferimenti a proprietà inesistenti di un oggetto
 - Impedisce le chiamate di funzione che usano la sintassi per chiamare i metodi
 - Non consentire limiti o indici di matrice non risolvibili
- **Latest**
 - **Seleziona la versione più recente disponibile che è la più rigorosa**, utilizzando questo valore ci si assicura che lo script usi la versione più rigorosa disponibile, anche quando vengono aggiunte nuove versioni a PowerShell

Gestione dell'errore

```
try {  
    Send-MailMessage -To $To -Subject $mailSubject -From $From -Body $mailBody -BodyAsHtml -SmtpServer $SmtpServer -Encoding $encodingUTF8  
    Write-Host ("`n" + "Report mailing to " + $MailTo + " sent successfully.")  
}  
catch {  
    Write-Host ("`n" + "Report mailing to " + $MailTo + " sending failed.")  
    Write-Output $_  
}
```

Una corretta gestione delle eccezioni rende gli script più stabili e semplifica la risoluzione dei problemi relativi a tali eccezioni



```
$command = [System.Data.SqlClient.SqlCommand]::New(queryString, connection)  
try  
{  
    $command.Connection.Open()  
    $command.ExecuteNonQuery()  
}  
catch {  
    Write-Host "An error occurred:"  
    Write-Host $_.ScriptStackTrace  
}  
finally  
{  
    $command.Connection.Close()  
}
```

- **Per liberare le risorse utilizzate aggiungere un blocco final dopo i blocchi try e catch**
- **Le istruzioni nel blocco finally vengono eseguite indipendentemente dal fatto che il blocco try incontri un errore**
- **Il blocco final viene eseguito prima che lo script termini o prima che il blocco corrente esca dall'ambito**
- **Il blocco final viene eseguito anche se si utilizza CTRL+C per interrompere lo script o se la keyword Exit interrompe lo script all'interno di un blocco catch**



https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_try_catch_finally

<https://learn.microsoft.com/en-us/powershell/scripting/learn/deep-dives/everything-about-exceptions>

<https://poshcode.gitbook.io/powershell-practice-and-style/best-practices/error-handling>

Performance

Nel caso lo script **debba gestire grandi set di dati** e vi siano più tecniche per realizzarlo **le performance assumono un aspetto importante**

#Più prestazionale del piping perché alloca tutto in memoria (rischio di esaurimento della memoria)

```
foreach ($result in Do-Something) {  
    $result.PropertyOne + $result.PropertyTwo  
}
```

#Meno prestazionale del costrutto foreach, ma potrebbe essere più leggibile

```
Do-Something | ForEach-Object { $_.PropertyOne + $_.PropertyTwo }
```

Le performance vanno misurate sull'hardware e sulla versione di Powershell utilizzati

Nel caso lo script **non debba gestire grandi set di dati** è meglio **privilegiare la leggibilità e la semplicità** del codice, in quanto un minimo miglioramento prestazionale non supera i vantaggi di un codice lineare e semplicemente manutenibile

*In linea generale valgono le seguenti, ma è possibile misurare le performance tramite il Cmdlet **Measure-Command**:*

- **Le funzionalità native sono più veloci delle funzionalità del .NET Framework**
- **I metodi compilati di oggetti e classi .NET sono più veloci degli script**
- **Uno script è più veloce che chiamare funzioni o Cmdlets a causa dell'overhead del passaggio di dati**

<https://devblogs-microsoft-com.translate.google.com/scripting/getting-to-know-foreach-and-foreach-object>

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/measure-command>

<https://poshcode.gitbook.io/powershell-practice-and-style/best-practices/performance>



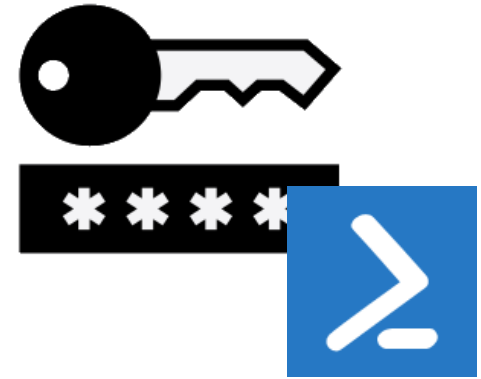
Gestione delle credenziali negli script

Evitare di memorizzare la password in una variabile stringa o di passarla tramite un parametro, ma **gestire sempre le credenziali tramite gli oggetti PSCredential** che memorizzano le password in un oggetto SecureString

Gli script dovrebbero sempre gestire le credenziali tramite un parametro di tipo PSCredentials senza richiamare mai Get-Credential all'interno dello script per consentire di riutilizzare le credenziali archiviate in una variabile

Utilizzare l'attributo Credential, come fanno i comandi integrati, in questo modo se l'utente passa il proprio nome utente, invece di un oggetto PSCredential, gli verrà richiesta la password in una finestra di dialogo sicura di Windows

Se è assolutamente necessario passare una password in una stringa ad una chiamata API .NET o a una libreria di terze parti, **decrittografare la credenziale mentre viene passata anziché salvarla in una variabile**



```
param (  
    [System.Management.Automation.PSCredential]  
    [System.Management.Automation.Credential()]  
    $Credentials  
)
```

```
# Get the cleartext password for a method call  
$Insecure.SetPassword($Credentials.GetNetworkCredential().Password)
```

Sicurezza e distribuzione degli script PowerShell

Distribuzione degli script

E' possibile **distribuire gli script** tramite:

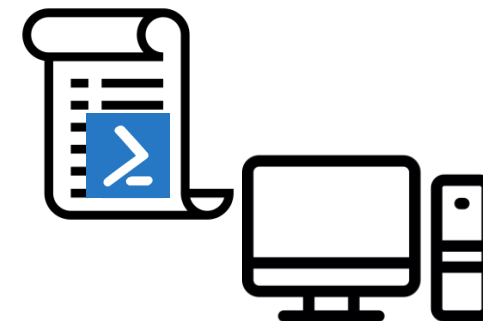
- Una **share di rete**
- **Copiando gli script in locale da una share tramite una GPO Preference** (utile se lo script deve essere disponibile anche quando il computer non è connesso alla rete aziendale)

E' possibile **eseguire gli script** tramite:

- Uno **script di accesso al computer** (privilegi dell'account di sistema locale)
- Uno **script di accesso utente** (privilegi dell'utente corrente)
- Un'**operazione pianificata** (privilegi dell'account di sistema locale o dell'utente corrente o di un account specifico)

Occorre prestare **particolare attenzione alla sicurezza impedendo che utenti non autorizzati possano modificare lo script:**

- **Impostare sullo script e/o sulla cartella che contiene lo script opportuni permessi per consentire solo i privilegi minimi necessari** (nel caso di copia in locale è possibile impostare la sicurezza di file system tramite GPO)
- **Firmare digitalmente gli script**



Overview delle PowerShell Execution Policies

Le Execution Policies sono una **funzionalità di sicurezza** che controlla come PowerShell carica i file di configurazione ed esegue gli script, con lo scopo di **prevenire l'esecuzione di script dannosi**

In un computer Windows è possibile impostare criteri di esecuzione per il **computer locale**, per l'**utente corrente** o per una determinata **sessione** tramite **cmdlet** o tramite **GPO**

Le Execution Policies per il computer locale e l'utente corrente vengono archiviate nel **Registro di sistema**, non è necessario impostare i criteri di esecuzione nel profilo di PowerShell, mentre le Execution Policies per una determinata sessione vengono archiviate solo in **memoria** e perse quando la sessione viene chiusa

Le Execution Policies non sono un sistema di sicurezza che limita le azioni degli utenti, se un'execution policy blocca l'esecuzione di uno script gli utenti possono bypassarla digitandone il contenuto a riga di comando o avviare lo script con un'execution policy differente, **lo scopo delle Execution Policies è consentire di impostare regole di base per impedire agli utenti di violarle involontariamente**

```
REM Avvio di uno script con una specifica Execution Policy  
powershell -ExecutionPolicy RemoteSigned -Command script.ps1
```

```
# Visualizzazione Execution Policies per Scope
```

```
Get-ExecutionPolicy -List
```

- Process
- CurrentUser
- LocalMachine

```
# Impostazione Execution Policy
```

```
Set-ExecutionPolicy <Policy> -Scope <Scope>
```



GPO Turn on Script Execution

Computer o User in:

WindowsComponents\WindowsPowerShell



Registry value ExecutionPolicy (REG_SZ)

HKLM o HKCU in:

SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell



PowerShell Execution Policies



Restricted (Default per Windows Client)

- **Non consente l'esecuzione di script** (.ps1), moduli (.psm1) e file di configurazione e formattazione (.ps1xml), ma **solo l'esecuzione di singoli comandi**

RemoteSigned (Default per Windows Server)

- **Non richiede la firma digitale sugli script locali**
- **Gli script e i file di configurazione scaricati da Internet devono essere firmati in modo attendibile o sbloccati** (Unblock-File)
- Se il sistema non distingue tra path UNC e path Internet gli script UNC non vengono eseguiti

Unrestricted (Default per non-Windows)

- **Non richiede la firma digitale sugli script**
- Warning prima di eseguire script e file di configurazione che non appartengono alla local Intranet zone

Default

- Restricted per Windows clients
- RemoteSigned per Windows servers

AllSigned

- **Tutti gli scripts e i file di configurazione (locali e remoti) devono essere firmati**
- Warning se si tenta di eseguire script con firma non attendibile

Bypass

- **Nulla viene bloccato niente e non vengono visualizzati avvisi**
- Questo criterio di esecuzione è progettato per le configurazioni in cui uno script di PowerShell viene eseguito nel contesto di un'applicazione ha un proprio modello di sicurezza.

Undefined

- Nessuna policy definta nello scope corente
- Se in tutti gli scope la policy è Undefined verrà utilizzata la Restricted per Windows clients e RemoteSigned per Windows servers

Firma di uno script PowerShell



Scelta del certificato in base al deploy dello script

- **Deploy Pubblico:** è necessario un certificato emesso da una **CA Pubblica**
- **Deploy Interno all'azienda:** è possibile utilizzare una **CA interna**
- **Deploy di Test:** è possibile utilizzare un **certificato autofirmato**

E' possibile creare un certificato autofirmato tramite il cmdlet **New-SelfSignedCertificate** introdotto con PowerShell 3.0 e migliorato nelle versioni successive

```
# Lettura certificato da Store Locale
$cert=Get-ChildItem -Path Cert:\CurrentUser\My -CodeSigningCert
# Lettura certificato da file pfx
$cert = Get-PfxCertificate -FilePath C:\Test\Mysign.pfx

# Firma di uno script
Set-AuthenticodeSignature -FilePath C:\Test\script.ps1 -Certificate $cert

# Firma di uno script specificando un timestamp server
Set-AuthenticodeSignature -FilePath C:\Test\script.ps1
-Certificate $cert -TimeStampServer "http://timestamp.digicert.com"

# Check del certificate digitale di uno script
Get-AuthenticodeSignature -FilePath C:\Test\script.ps1 |
Select-Object -Property *
```

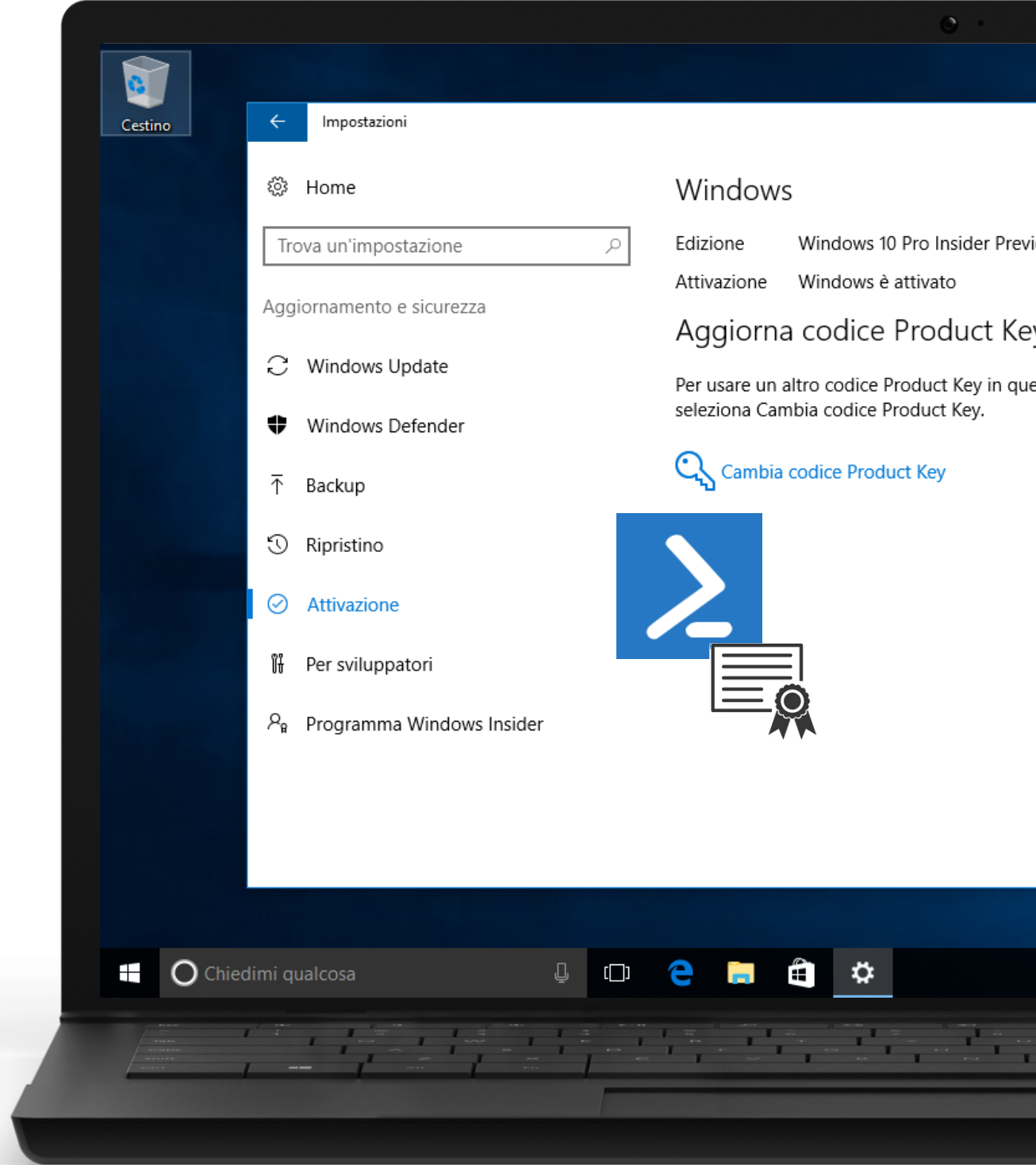


<https://www.devadmin.it/2017/04/23/creazione-certificato-autofirmato-in-powershell/>

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-authenticodesignature>

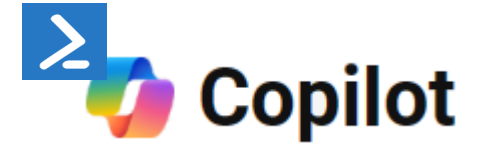
DEMO

Firma di script PowerShell



Utilizzo dell'AI per lo sviluppo di script

Copilot e PowerShell



Fonti utilizzate per generare script PowerShell

1. **Conoscenza interna:** Ho una vasta conoscenza interna di PowerShell e delle sue funzionalità. Questa conoscenza include la sintassi, i comandi (Cmdlet), le pipeline, le variabili, gli operatori, gli script, le funzioni, e molto altro.
2. **Best Practice di PowerShell:** Le mie risposte seguono le best practice di PowerShell, che includono l'uso di nomi di variabili descrittivi, l'uso di commenti per spiegare il codice, l'uso di funzioni per riutilizzare il codice, ecc.
3. **La mia capacità di generare script PowerShell si basa sulla mia conoscenza interna pre-programmata e sulle informazioni che ho appreso durante la mia formazione.** Questa formazione include una vasta gamma di dati, tra cui libri, siti web e altri materiali di riferimento su PowerShell e su una varietà di altri argomenti. **Tuttavia, non ho la capacità di accedere o recuperare informazioni da specifiche fonti esterne o di aggiornare la mia conoscenza oltre il punto in cui sono stato addestrato.**

La mia conoscenza di PowerShell è aggiornata fino alla versione 7.1, che è stata rilasciata nel novembre 2020. Tuttavia, tieni presente che non ho la capacità di aggiornare autonomamente la mia conoscenza, quindi potrei non essere a conoscenza delle funzionalità introdotte nelle versioni di PowerShell rilasciate dopo il 2021.

Non ho la capacità di eseguire o testare direttamente il codice che genero. Tuttavia, è responsabilità dell'utente eseguire e testare lo script in un ambiente sicuro.

Ci sono alcuni tipi di script PowerShell che non posso generare per motivi di sicurezza e conformità. Ad esempio:

1. **Script dannosi:** Non genererò script che possono causare danni, come script che cancellano file o modificano le impostazioni di sistema senza il consenso dell'utente.
2. **Script che violano la privacy:** Non genererò script che raccolgono o divulgano informazioni personali senza il consenso dell'utente.
3. **Script illegali:** Non genererò script che facilitano attività illegali, come la pirateria del software o l'hacking.
4. **Script che richiedono privilegi elevati:** Non genererò script che richiedono privilegi di amministratore o di sistema, a meno che non sia assolutamente necessario e l'utente ne sia pienamente consapevole.

Ricorda, è importante che tu comprenda cosa fa uno script prima di eseguirlo.

Generazione codice PS con Copilot

Crea uno script PowerShell per ricercare gli errori e warning degli ultimi due giorni nel registro eventi di sistema relativi al disco

```
# Ottieni la data di due giorni fa  
$data = (Get-Date).AddDays(-2)
```



```
# Filtra gli eventi di sistema per errori e avvisi relativi al disco  
Get-WinEvent -FilterHashtable @{LogName='System'; StartTime=$data} |  
Where-Object {($_.LevelDisplayName -eq 'Error' -or $_.LevelDisplayName -eq 'Warning') -and $_.ProviderName -eq 'Disk'} |  
Format-Table TimeCreated, LevelDisplayName, Message -AutoSize -Wrap
```

L'utilizzo di Where-Object è meno efficiente rispetto all'uso di Filter perché occorre estrarre tutti gli oggetti

```
# Imposta la data di inizio e fine per il periodo di ricerca  
$StartDate = (Get-Date).AddDays(-$AlertOnEventsFromDays)
```

```
# Ricerca degli errori (Level 2) e warning (Level 3) relativi al disco  
$Events = Get-WinEvent -FilterHashtable @{LogName = "System"; StartTime = $StartDate; ProviderName = "disk"; Level = 2,3}
```

ChatGPT 3.5 e PowerShell

Fonti utilizzate per generare script PowerShell

1. **Documentazione ufficiale di Microsoft:** Questa è una fonte primaria di informazioni su PowerShell. Include guide, documenti tecnici, e reference per cmdlet, moduli, e funzionalità specifiche.
2. **Community e forum tecnici:** Monitoro forum come **Stack Overflow, Reddit** (ad es. r/PowerShell), e altri simili dove gli sviluppatori e gli esperti condividono conoscenze, esperienze e risorse relative a PowerShell.
3. **Guide di scripting e tutorial online:** Esistono numerose risorse online, come **blog, tutorial e guide dedicate a PowerShell**, che forniscono esempi pratici e spiegazioni dettagliate su come scrivere script efficaci.
4. **Esperienza accumulata:** Attraverso l'analisi di una vasta quantità di testi e informazioni disponibili, ho acquisito una comprensione approfondita di PowerShell e delle sue funzionalità.
5. **Versioni e aggiornamenti di PowerShell:** Mantengo aggiornate le mie conoscenze fino alla mia data di taglio, comprendendo le nuove funzionalità, le modifiche, e le **best practice raccomandate da Microsoft e dalla community**.
6. **Risorse educative e corsi online:** Piattaforme di formazione online, come **Microsoft Learn, Pluralsight, Udemy** e altre simili, offrono corsi dedicati a PowerShell che forniscono conoscenze approfondite e pratica per la scrittura di script.
7. **Pubblicazioni e libri specializzati:** **Testi specializzati** su PowerShell scritti da esperti del settore offrono una prospettiva dettagliata su concetti avanzati, trucchi e best practice.

La mia conoscenza è aggiornata fino alla versione di PowerShell disponibile fino alla mia data di taglio (gennaio 2022), che è la versione **PowerShell 7.x**.

Attualmente non ho la capacità di eseguire codice o interagire direttamente con sistemi o ambienti esterni. **Non posso eseguire script o comandi in un ambiente reale o verificare direttamente l'output di uno script PowerShell.**

Ci sono linee guida etiche e restrizioni imposte alla mia programmazione che mi impediscono di generare script per compiti che possono essere considerati illegali, dannosi, dannosi per la privacy o che violano le politiche di sicurezza.



Generazione codice PS con ChatGPT 3.5

Crea uno script PowerShell per ricercare gli errori e warning degli ultimi due giorni nel registro eventi di sistema relativi al disco

```
# Imposta la data di inizio e fine per il periodo di ricerca (ultimi due giorni)
$StartDate = (Get-Date).AddDays(-2)
$EndDate = Get-Date
```



```
# Ricerca degli errori e warning relativi al disco negli ultimi due giorni
```

```
$Events = Get-WinEvent -FilterHashtable @{
```

```
    LogName = "System"
```

```
    StartTime = $StartDate
```

```
    EndTime = $EndDate
```

```
    Level = 2, 3 # Livello 2 (Errore) e 3 (Warning)
```

```
    ID = 7, 52 # ID 7 (Errore disco) e 52 (Warning disco)
```

```
} | Where-Object { $_.Message -match "disk" } # Filtra per messaggi contenenti la parola "disk"
```

L'utilizzo di Where-Object è meno efficiente rispetto all'uso di Filter perché occorre estrarre tutti gli oggetti

<https://techcommunity.microsoft.com/t5/skype-for-business-blog/filter-vs-where-object/ba-p/618442>

```
# Imposta la data di inizio e fine per il periodo di ricerca
```

```
$StartDate = (Get-Date).AddDays(-$AlertOnEventsFromDays)
```



```
# Ricerca degli errori (Level 2) e warning (Level 3) relativi al disco
```

```
$Events = Get-WinEvent -FilterHashtable @{LogName = "System"; StartTime = $StartDate; ProviderName = "disk"; Level = 2,3}
```

Bard 2023.10.30 e PowerShell

Fonti utilizzate per generare script PowerShell

1. **La documentazione ufficiale di PowerShell** è la fonte più importante per la generazione di script PowerShell. Questa documentazione è disponibile sul **sito web di Microsoft** e fornisce informazioni complete sui cmdlet, i moduli e le altre funzionalità disponibili in PowerShell.
2. **Il codice PowerShell disponibile online** è un'altra importante fonte per la generazione di script PowerShell. Ci sono molte risorse online che forniscono esempi di codice PowerShell, che posso utilizzare come base per i miei script. Alcune delle risorse più popolari includono:
 - **Il sito web di Microsoft**, che fornisce esempi di codice PowerShell per una varietà di attività.
 - **Il sito web di PowerShell.org**, che ospita una comunità di sviluppatori PowerShell che condividono i propri codici.
 - **Il sito web di GitHub**, che ospita un repository di codice PowerShell open source.

Ci sono degli script PowerShell che non posso generare:

- Uno script che accede a un database privato.
- Uno script che automatizza l'installazione e la configurazione di un sistema operativo.
- Uno script che cancella tutti i file su un computer.
- Uno script che invia spam o malware.
- Uno script che viola la privacy di un utente.

A partire dal 3 agosto 2023, sono aggiornato alla **versione 7.4 di PowerShell**.

Ho modo di eseguire il codice che genero.

Posso farlo in diversi modi:

- Posso eseguire il codice manualmente, utilizzando un terminale o un editor di testo.
- Posso utilizzare un framework di test, come Pester, per eseguire il codice in modo automatico.
- Posso eseguire il codice su un computer di test, che è una macchina virtuale o un computer fisico dedicato alla fase di test.



Generazione codice PS con Bard 2023.10.30

Crea uno script PowerShell per ricercare gli errori e warning degli ultimi due giorni nel registro eventi di sistema relativi al disco

Imposta la data e l'ora di inizio della ricerca

```
$startdate = (Get-Date).AddDays(-2).ToString("yyyy-MM-dd")  
$starttime = (Get-Date).AddDays(-2).ToString("HH:mm:ss")
```

Imposta la data e l'ora di fine della ricerca

```
$enddate = (Get-Date).ToString("yyyy-MM-dd")  
$endtime = (Get-Date).ToString("HH:mm:ss")
```

Cerca gli eventi di sistema

```
$events = Get-EventLog -LogName System -Source "Microsoft-Windows-DiskDrive" -AfterTime $startdate -AfterTime $starttime -BeforeTime $enddate -BeforeTime $endtime
```

Filtra gli eventi per errore o warning

```
$events = $events | Where-Object { $_.Level -eq "Error" -or $_.Level -eq "Warning" }
```

Get-EventLog uses a Win32 API that is deprecated. The results may not be accurate. Use the Get-WinEvent cmdlet instead.

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-eventlog>



Imposta la data di inizio e fine per il periodo di ricerca

```
$StartDate = (Get-Date).AddDays(-$AlertOnEventsFromDays)
```

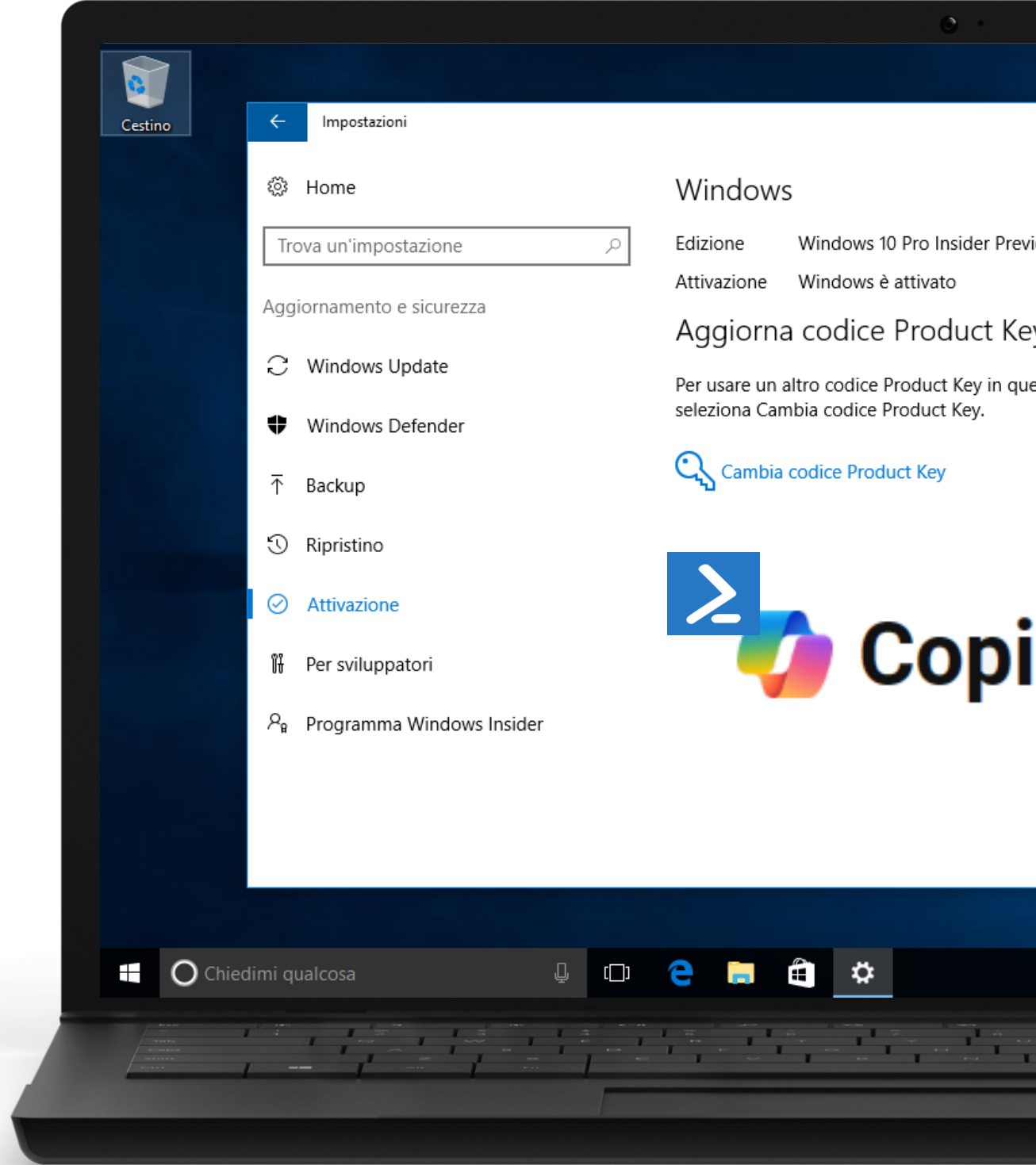
Ricerca degli errori (Level 2) e warning (Level 3) relativi al disco

```
$Events = Get-WinEvent -FilterHashtable @{LogName = "System"; StartTime = $StartDate; ProviderName = "disk"; Level = 2,3}
```



DEMO

Copilot e PowerShell



Conclusioni e link utili



PowerShell richiede una **curva di apprendimento** e la conoscenza della **programmazione** e di **SQL** permette di scrivere script più semplicemente



Ci sono scenari in cui lo sviluppo di uno script PowerShell risulta più oneroso dell'uso di tool nativi o espone a rischi di errori (ad esempio task di automazione per Active Directory o modifica di attributi di oggetti AD)



L'utilizzo della **AI** permette di **velocizzare la stesura di scripts**, ma occorre essere in grado di comprendere quanto viene generato per **utilizzare il risultato in modo consapevole** soprattutto per quanto riguarda l'approccio, i cmdlet utilizzati, le performance e la sicurezza



[Installare e configurare PowerShell for Linux](#)



[PowerShell Core 6: ancora Open Source in trionfo](#)



[Creare rapidamente laboratori di test per W10, WS 2016 e WS 2019 utilizzando PowerShell](#)



[PowerShell scripting performance considerations](#)



[PowerShell Team · GitHub](#)

[GitHub - ermannog/PowerShell: PowerShell functions and scripts](#)

[GitHub - ermannog/PowerTray: PowerTray allows the execution and display of PowerShell script output](#)

Grazie

Ermanno Goletto

*Microsoft MVP Alumni
e.goletto@outlook.it*



[/devadmin.it](#)



[@ermannog](#)



[/ermannogoletto](#)

Roberto Massa

*Microsoft MVP Alumni
robimassa@outlook.it*



[/robi.massa.cn](#)



[@robi_massa](#)



[/robimassa](#)