

# Power Shell

Mohamed Amr Mohy

Linkedin: <https://www.linkedin.com/in/mohamed-amr-mohy/>

# Contents

1	Introduction .....	1
1.1	CMD .....	1
2	CMD Browsing .....	5
2.1	CD (Change Directory).....	5
2.2	Get-ChildItem or Dir or Gci or Ls.....	6
2.3	Get-help .....	7
2.4	Get-Command.....	9
2.5	Start-Process .....	10
2.6	Get-Process .....	12
2.7	Pipeline Select .....	13
3	System Environment Variables and Paths .....	15
3.1	Variables in PowerShell: .....	20
3.2	Arithmetic Operators .....	22
3.3	Assignment Operators .....	24
3.4	Logical Operators .....	25
3.5	Logical Operators .....	27
3.6	Redirection Operators .....	28
3.7	Split and join.....	29
4	Conditional Statements .....	30
4.1	If statement .....	30
4.2	Switch Statement .....	32
5	Loops.....	35
5.1	For loops .....	35
5.2	While Loop .....	37
6	Example For Some Scripting .....	39
6.1	Example: Remove Files In Folder .....	39
6.2	Example: Check the status of services and start stopped ones.....	40
6.3	Example: Ping in All of Network.....	41
6.4	Example: To solve the file transfer problem .....	43
6.5	Example: Send an email alert .....	44

6.6	Example of DeepWhite.....	45
6.7	Get-WinEvent for Example .....	45
6.8	Remote Registry Service .....	45

# 1 Introduction

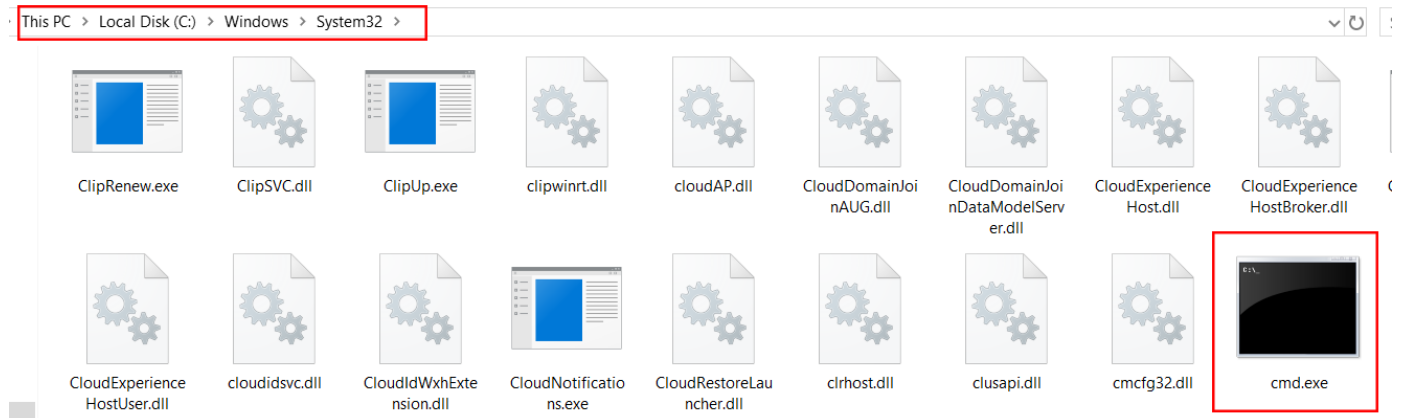
هنبتي نتعلم مع بعض ال PowerShell بالطرق المختلفة

ال PowerShell هي عبارة عن طريقة تقدر توصل بها لل Operating System عشان انت عايز تنفذ حاجه معينه عليه ولكن لضعف CMD بتاعه Windows اللي هي الطبيعيه واللي بتستخدم مثلا لبعض الاوامر البسيطة في Windows ده ادى الى ان السيرفر و Linux عموما بيبقى فيه Automation افضل بكثير جدا من Windows فبالتالي انت ممكن جدا او تنفذ لك Tasks كثيره جدا وتريح بالك منها ولكن الكلام ده ما كانش موجود في Windows لفترة طويله بعد كده جت طورت مايكروسوفت نفسها وانها تقدر تعمل حاجات كثيره جدا لا يمكن ان انت تتصورها زي انك تتحكم في ال Machine بتاعتك و Tasks عليها و كمان ال Machine الثانيه اللي موجوده في Network تعمل كده بالاضافه ان هي اضافت لك كمان موضوع Scripting فاحنا كان عندنا في Linux نقدر نعمل الموضوع Scripting ده ان احنا ننزل فايلات ونعدلها وننظمها و تنفذ واحد وغيره تقدر تحرك Scripting بتاعك زي ما انت عايز وينفذ Command دي بناء على ارشادات معينه انت هتديهاله الموضوع ده كان موجود بسيط جدا في Windows باسم batch scripting وده اللي كان بيبتي يعمل Command دي على CMD (Command Prompt) بتاعه Windows العاديه ولكن ما كانش Efficiency قويه قد Bash Scripting اللي موجود في Linux وبالتالي هنا مايكروسوفت راحت اضافته في PowerShell فكره Scripting وخلتها اقوى وتكون سلسه اكثر في التعامل بالاضافه ان ادبت لها كمان Access على حاجات كثيره جدا موجوده في System زي NET. ودي هتخلي ان ال PowerShell اشبه انه بيتعامل مع Code متبرمج جاهز داخل Windows ويقدر يتعامل معاه كثير جدا مع Libraries اللي موجوده الخاصه بال NET. داخل Windows ويقدر يستخدمها كمان وليه access عليها كل Feature ضافتك حاجات قويه جدا في Bash من خلال PowerShell تقدر تعمل Web time تنزل لك ملفات او تقدر تعمل حاجه تتعامل مع الانترنت مع بعض Programs او تتعامل مع Files او حتى مع Network فبالتالي احنا في الكورس ده باذن الله هنبتي نتكلم في البدايه على PowerShell زي ما قلنا وبعد كده استخداماتها اللي بتستخدم بشكل Automatic مع PowerShell وبعد كده ازاى تبتي تكتب Scripting

## 1.1 CMD

ال CMD، اللي هو اختصار Command Prompt، هو برنامج في نظام الويندوز بنستخدمه عشان نكتب أوامر مباشرة للجهاز. زمان لما مكش فيه واجهات رسومية، الناس كانت بتستخدم الأوامر دي عشان تتحكم في الكمبيوتر. دلوقتي بنستخدمه برضه بس أكثر للمحترفين أو اللي بيشتغلوا في ال IT والبرمجة.

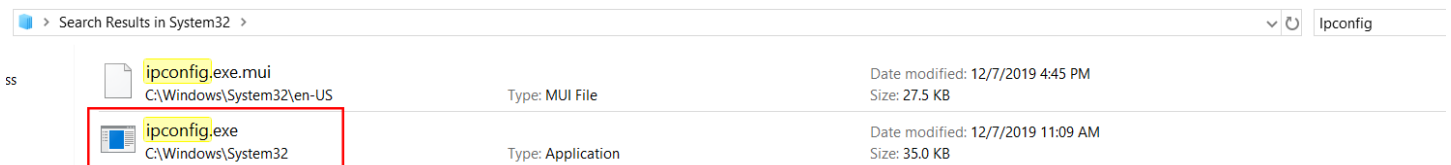
ببساطة، لما تفتح ال CMD، بيطلعك شاشة سودة تقدر تكتب فيها أوامر معينه، زي إنك تفتح فولدر، تمشح ملفات، تنزل برامج، أو حتى تشوف معلومات عن النظام بتاعك. هو زي ما يكون الأداة اللي بتديك القدرة إنك تتحكم في الكمبيوتر بتاعك بشكل أعمق ده غير انها بتقدر تتحكم ف البرامج اللي عندك وتعمل Connect معاهم وبطريقة مختلفة عن الأزارر والماوس.



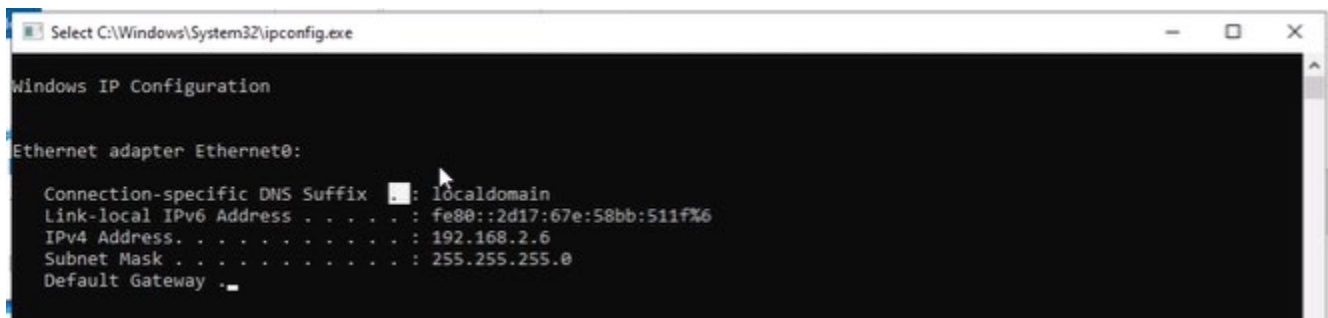
في البدايه اللي عندنا هنلاقي هنا ip Program لازم تبقى عارف ان اي بيكون في cmd ما هو الا Program عندك و CMD بتبتي تنقل Program دا و Output بتاعه بتبتي تعرضه لك عندها فال cmd هي في حد ذاتها هي مش Shell على Windows هي بكل بساطه عباره عن Program بيتيح لك انك تعمل Connect مع Shell بتاعه Windows او تتعامل مع Shell بتاعه Windows بالاضافه على كده ان هي كمان تقدر تنده برامج ثانيه مكتوبه بال Console او مكتوبه بالواجهه بتاعتها

لازم تعرف ان ip هنلاقي عندك هنا هنلاقيه تفتح بسرعه زي ما انتم شايفين بالشكل ده ادي الاي بيتبتي يفتح لي الاي بي بتاعتي تمام ففي الاول وفي الاخر هي بتنادي على البروجرام ده وبتطلع بتاعه جميل فال سي ان دي هي وسيله للتعامل مع الشيف Windows كذلك الامر موجوده فين لو جئنا فتحنا هنا PowerShell بالشكل ده موجود عندنا هنا وهي برده الباور شيلد بتاعه Windows وبيتبتي ننفذ عليها الاوامر تمام بمعنى ان انا عايزك تعرف ان انت حتى في Linux لو تلاحظ ان انت ممكن تسطب اكثر من نوع تيرمنال لو انت عملت قبل كده فعندنا في اللينس كثير جدا من انواع الترمس الى اخره ولكن كلهم بينادوا على Program

تعالى ناخذ بصه على Ipconfig مثلا لو جينا فتحنا دا جوا C:\Windows\System32



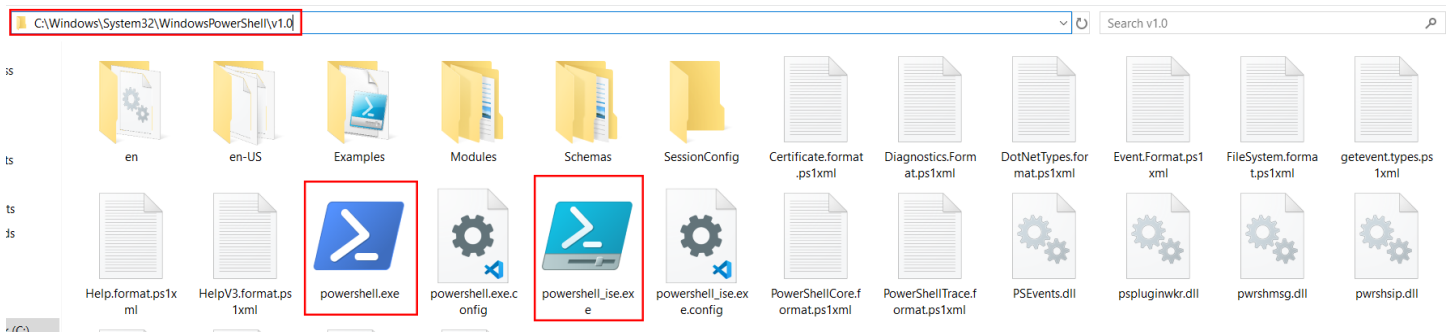
هنلاقي انه عباره عن برنامج بيفتحه تعالي نفتحه كدا



بيتبتي يقولي المعلومات بتاعته

جرب nslookup , notepad , calc دول برامج هو بينادي عليها

طب بالنسبه لل PowerShell مكانها ف ال Url اللي موجود ف الاسكرين دا



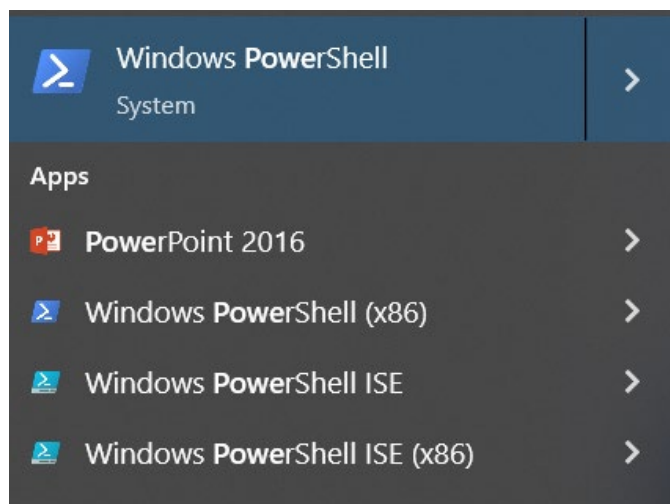
في عندنا PowerShell\_ise دي بتاعت Integrated Scripting Environment دي مخصصه لل Scripting بس

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

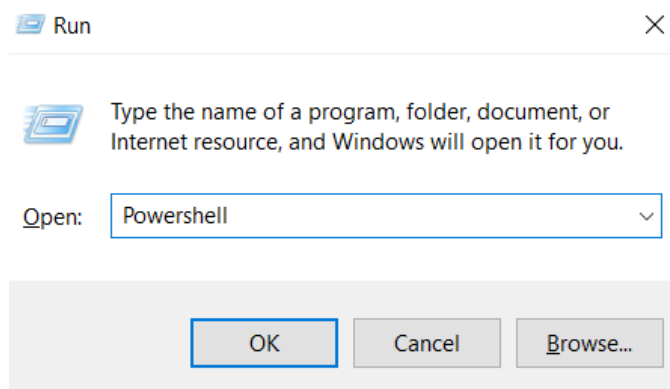
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\windows\system32\windowspowershell\v1.0>
```

دا شكلها طبعا احنا عندنا اكثر من نوع لل PowerShell زي



بص كل الانواع دي بتتد علي نفس ال Program من نوع ال Exe وغيره بتختلف ف حاجات بسيطة هنفهمها بعدين  
طب ممكن نفتحها اكثر من طريقه اول حاجه ف علامه ال Search تحت زي ما شوفنا او من علامه Windows + R  
هيفتحلنا دي كدا ونكتب PowerShell



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Mohamed Amr>
```

طب هو ليه نستعمل Shell ومش نستعمل GUI بسهولة ونشغل دماغنا بيه

لان ببساطه لو انا حببت اعمل Create لفولدر معين باسم مثلا من 1900 لحد 2000 هتلاقي الحوار متعب ان كل شويه اعمل Right click وبعدين انشئ واحد ورا الثاني ده قصه كبيره وارجع اضيف اسم اسم علي الرغم اني ممكن استعمل Foreach بالPowerShell واعملي كل Folders دي ف ثانيه الا ثانيه حرفيا تمام جدا ده غير حاجات كتير جدا هنشوفها قدام.

## 2 CMD Browsing

في Command عندي بتبقي Built in مش مجرد Program وبنادي عليها ودي اللي هنتكلم عنها

### 2.1 CD (Change Directory)

الأمر `cd` في PowerShell هو اختصار للأمر `Set-Location`. الأمر ده بيستخدم لتغيير المسار الحالي (current directory) اللي انت شغال فيه. يعني تقدر تروح لأي فولدر تاني في نظام الملفات عن طريق كتابة مساره.

هنا شرح مفصل لكيفية استخدام الأمر `cd` (أو `Set-Location`) في PowerShell:

1. تغيير المسار الحالي إلى فولدر آخر:

- لو عايز تروح لفولدر معين، تكتب الأمر وبعده المسار.

```
cd C:\Path\To\Your\Folder
```

2. الرجوع إلى الفولدر السابق:

- تقدر تستخدم `cd ..` عشان ترجع خطوة واحدة فوق في الهيكل الهرمي للفولدرات.

```
cd ..
```

3. الانتقال إلى الـ Home Directory:

- ممكن تروح للفولدر الرئيسي للمستخدم الحالي باستخدام الاختصار `~`.

```
cd ~
```

4. تغيير المسار إلى درايف آخر:

- لو عايز تروح لدرايف آخر، تكتب حرف الدرايف متبوع بنقطتين.

```
cd D:
```

5. استخدام المسارات النسبية:

- تقدر تستخدم المسارات النسبية للتنقل بين الفولدرات اللي قريب منك.

```
cd .\Subfolder\AnotherSubfolder
```

6. استخدام المسارات المطلقة:

- تقدر تستخدم المسارات المطلقة للوصول لأي مكان على الجهاز.

```
cd C:\Users\YourUserName\Documents
```

اختصارات وميزات:

- في PowerShell، تقدر تستخدم زر الـ Tab لإكمال أسماء الفولدرات والملفات بشكل تلقائي. مثلاً، لو كتبت `cd Doc` وضغطت Tab، هيكملها لـ `Documents` لو كانت موجودة.

ملاحظة:

- على الرغم من إن `cd` هو اسم مستعار للأمر `Set-Location`، إلا إن استخدام `Set-Location` بيدي نفس النتيجة. دي المرونة اللي بتوفرها PowerShell عشان تسهل عليك الشغل.



## 2.2 Get-ChildItem or Dir or Gci or Ls

الأمر 'dir' في PowerShell هو أمر مشهور ومستخدم بشكل واسع لعرض محتويات المجلدات. في PowerShell، 'dir' هو اسم مستعار (Alias) للأمر 'Get-ChildItem'. الأمر ده بيسمحك تستعرض الملفات والفولدرات اللي موجودة في مسار معين.

هنا شرح مفصل للأمر 'Get-ChildItem' وكل الخيارات اللي ممكن تستخدمها معاه:

### 1. الأمر الأساسي:

- يعرض جميع الملفات والفولدرات في المسار الحالي.

```
Get-ChildItem
```

### 2. عرض محتويات مسار معين:

- تقدر تحدد مسار معين عشان تشوف محتوياته.

```
Get-ChildItem -Path "C:\Path\To\Your\Folder"
```

### 3. عرض ملفات وفولدرات محددة باستخدام Wildcards:

- ممكن تستخدم الرموز الخاصة لتحديد نوع الملفات اللي عايز تعرضها.

```
Get-ChildItem -Path "C:\Path\To\Your\Folder\*.txt"
```

### 4. عرض جميع العناصر بما في ذلك العناصر المخفية:

- الافتراضي إن الأمر 'Get-ChildItem' بيعرض العناصر المخفية. لو عايز تعرضها، استخدم المفتاح '-Force'.

```
Get-ChildItem -Path "C:\Path\To\Your\Folder" -Force
```

### 5. عرض محتويات الفولدرات الفرعية (Recursive):

- لو عايز تعرض جميع العناصر بما في ذلك اللي موجودة في الفولدرات الفرعية.

```
Get-ChildItem -Path "C:\Path\To\Your\Folder" -Recurse
```

### 6. لو بتدور علي فايل معين جوا فولدرات معينة

- لو بتدور علي فايل معين جوا الملفات.

```
Get-ChildItem -Path "C:\Path\To\Your\Folder" -Recurse -filter *.exe
```

### 7. لو بتدور علي فايل وعايز تحدد خواص معينة ليه جوا فولدرات معينة

```
Get-ChildItem 'C:\Program Files\' -Recurse -Directory | select name,Mode
```

### 8. عرض الملفات فقط:

- لو عايز تعرض الملفات فقط بدون الفولدرات.

```
Get-ChildItem -Path "C:\Path\To\Your\Folder" -File
```

### 9. عرض الفولدرات فقط:

- لو عايز تعرض الفولدرات فقط بدون الملفات.

```
Get-ChildItem -Path "C:\Path\To\Your\Folder" -Directory
```

10. عرض تفاصيل أكثر عن العناصر:

- ممكن تستخدم 'Format-List' أو 'Format-Table' عشان تعرض تفاصيل أكثر عن الملفات والفولدرات.

```
Get-ChildItem -Path "C:\Path\To\Your\Folder" | Format-List
```

دي كانت الأوامر الأساسية والخيارات اللي ممكن تستخدمها مع 'Get-ChildItem' في PowerShell. الأمر ده قوي ومرن جدًا وببساطة تتنقل وتتعامل مع الملفات والفولدرات بسهولة.

## 2.3 Get-help

الأمر 'Get-Help' في PowerShell هو أداة قوية جدًا بتساعدك تفهم وتتعلم كيفية استخدام الأوامر المختلفة في PowerShell. الأمر ده بيعرض لك معلومات تفصيلية عن أي أمر أو دالة أو مفهوم داخل PowerShell.

هنا شرح مفصل لكيفية استخدام الأمر 'Get-Help':

الاستخدام الأساسي:

1. عرض المساعدة الأساسية لأمر معين:

- لو عايز تشوف المساعدة الأساسية لأمر معين، تكتب 'Get-Help' متبوع باسم الأمر.

```
Get-Help Get-ChildItem
```

خيارات إضافية للأمر Get-Help:

1. عرض التفاصيل الكاملة:

- عشان تشوف المساعدة الكاملة والمفصلة لأمر معين.

```
Get-Help Get-ChildItem -Full
```

2. عرض أمثلة الاستخدام:

- عشان تشوف أمثلة عملية لكيفية استخدام الأمر.

```
Get-Help Get-ChildItem -Examples
```

3. عرض المساعدة الفنية (Technical):

- بيعرض تفاصيل تقنية أكثر عن الأمر.

```
Get-Help Get-ChildItem -Detailed
```

4. عرض معلومات حول المعلمات (Parameters):

- لو عايز تشوف معلومات مفصلة عن المعلمات اللي بيقبلها الأمر.

```
Get-Help Get-ChildItem -Parameter*
```

5. تحديث محتويات المساعدة:

- عشان تتأكد إن معلومات المساعدة محدثة، ممكن تستخدم الأمر ده لتحديث المساعدة عبر الإنترنت.

## Update-Help

6. استخدام المساعدة عبر الإنترنت:

- عشان تعرض المساعدة عبر الإنترنت لو المعلومات المحلية مش كافية أو محدثة.

```
Get-Help Get-ChildItem -Online
```

أمثلة إضافية:

1. عرض قائمة بالأوامر المتاحة:

- لو عايز تعرف كل الأوامر المتاحة في PowerShell، تقدر تستخدم الأمر التالي.

```
Get-Command
```

2. البحث عن الأوامر المتعلقة بكلمة معينة:

- لو عايز تبحث عن أوامر تتعلق بكلمة معينة، مثل "service".

```
Get-Help *service*
```

3. معلومات عن الموضوعات العامة:

- تقدر تطلب معلومات عن موضوعات معينة مش مرتبطة بأمر محدد.

```
Get-Help about*_
```

ملاحظات:

- الأمر 'Get-Help' هو زي الدليل الكامل لكل حاجة ممكن تعملها في PowerShell. لو عندك أي سؤال عن أمر معين أو معلومة معينة، 'Get-Help' هو أفضل مكان تبدأ منه.

- تقدر تعدل إعدادات الأمان في PowerShell للسماح بتنزيل وتحديث المساعدة عبر الإنترنت باستخدام 'Update-Help'.

باختصار، 'Get-Help' هو أداة لا غنى عنها لأي حد يستخدم PowerShell، سواء كنت مبتدئ أو محترف، لأنه يوفر لك كل المعلومات اللي ممكن تحتاجها لفهم واستخدام الأوامر بشكل صحيح.

```
Windows PowerShell
PS C:\Users\Mohamed Amr> Get-Help Get-ChildItem

NAME
----
Get-ChildItem

SYNTAX
-----
Get-ChildItem [[-Path] <string[]>] [[-Filter] <string>] [-Include <string[]>] [-Exclude <string[]>] [-Recurse] [-Depth <uint32>] [-Force] [-Name] [-UseTransaction] [-Attributes {ReadOnly | Hidden | System | Directory | Archive | Device | Normal | Temporary | SparseFile | ReparsePoint | Compressed | Offline | NotContentIndexed | Encrypted | IntegrityStream | NoScrubData}] [-FollowSymlink] [-Directory] [-File] [-Hidden] [-ReadOnly] [-System] [CommonParameters]

Get-ChildItem [[-Filter] <string>] -LiteralPath <string[]> [-Include <string[]>] [-Exclude <string[]>] [-Recurse] [-Depth <uint32>] [-Force] [-Name] [-UseTransaction] [-Attributes {ReadOnly | Hidden | System | Directory | Archive | Device | Normal | Temporary | SparseFile | ReparsePoint | Compressed | Offline | NotContentIndexed | Encrypted | IntegrityStream | NoScrubData}] [-FollowSymlink] [-Directory] [-File] [-Hidden] [-ReadOnly] [-System] [CommonParameters]

ALIASES
-----
gci
ls
dir

REMARKS
-----
Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
-- To download and install Help files for the module that includes this cmdlet, use Update-Help.
-- To view the Help topic for this cmdlet online, type: "Get-Help Get-ChildItem -Online" or go to https://go.microsoft.com/fwlink/?LinkID=113308.
```

ال Aliase دي المقصود بيها ان الامر دا ليه اكثر من شكل يعني ممكن تكتبه ls او gci ايا يكن يعتبر اسم دلع لل-Get-ChildItem مش بيفرق عن الباقي حاجه

ترجم الامر دا مع نفسك عشان تفهم وظيفه PS

```
gci 'C:\Program Files\' -Recurse -Filter *.exe -ErrorAction SilentlyContinue -Name
```

## 2.4 Get-Command

الأمر 'Get-Command' في PowerShell هو أداة قوية بتساعدك تستعرض وتبحث عن الأوامر المتاحة، بما في ذلك الأوامر الخاصة بـ PowerShell، والـ Cmdlets، والوظائف (Functions)، والبرامج النصية (Scripts)، والتطبيقات القابلة للتنفيذ (Executables)، وغيرها.

الاستخدام الأساسي:

1. عرض جميع الأوامر المتاحة:

- بدون أي معلمات، الأمر ده بيعرض قائمة بجميع الأوامر المتاحة في جلسة PowerShell الحالية.

```
Get-Command
```

خيارات إضافية للأمر Get-Command:

1. البحث باستخدام اسم أو جزء من اسم الأمر:

- لو عايز تبحث عن أمر معين أو أوامر تحتوي على نص معين في أسمائها.

```
Get-Command *Service*
```

2. تحديد نوع الأمر:

- تقدر تحدد نوع الأوامر اللي عايز تعرضها. الأنواع تشمل Cmdlet، Function، Alias، Application، Script.

```
Get-Command -CommandType Cmdlet
```

- البحث عن الوظائف فقط.

```
Get-Command -CommandType Function
```

- البحث عن التطبيقات القابلة للتنفيذ فقط.

```
Get-Command -CommandType Application
```

3. عرض الأوامر الخاصة بموديل معين:

- لو عايز تعرض الأوامر اللي تخص موديل معين.

```
Get-Command -Module Microsoft.PowerShell.Management
```

4. عرض الأوامر الخاصة بمجموعة معينة:

- يمكن تحديد مجموعة معينة للبحث داخلها.

```
Get-Command -Noun Process
```

5. عرض خصائص معينة للأوامر:

- ممكن تستخدم خصائص معينة للأوامر عشان تحددتها بشكل أكثر دقة.

```
Get-Command -Syntax
```

6. عرض اسم لبدايه ملف:

- هيجيبلك اي ملف بدايته start.

```
Get-Command -verb start  
Get-Command -name start*
```

أمثلة إضافية:

1. البحث عن الأمر بواسطة الاسم الدقيق:

- لو عايز تعرف تفاصيل أمر معين باستخدام اسمه الدقيق.

```
Get-Command Get-Process
```

2. عرض جميع الأوامر المتاحة في النظام بما فيها الأوامر الغير افتراضية:

- لو عايز تشوف كل الأوامر اللي النظام بيستخدمها، بما فيها الأوامر اللي مش موجودة بشكل افتراضي في PowerShell.

```
Get-Command -All
```

3. البحث عن الأوامر التي لها اسم مستعار معين:

- لو عايز تعرف الأوامر اللي ليها اسم مستعار.

```
Get-Command -type Alias
```

4. البحث باستخدام الأنماط المتقدمة:

- تقدر تستخدم الأنماط المتقدمة للبحث عن الأوامر اللي تشتمل على خصائص معينة.

```
Get-Command -ParameterName Name
```

5. البحث عن حاجه معينه:

- تقدر تستخدم الطريقه دي عشان لو بتدور علي cmdlet معينه.

```
Get-Command -type Cmdlet -name *proc*
```

ملاحظات:

- الأمر 'Get-Command' بيوفر وسيلة فعالة جدًا لاكتشاف وفهم الأوامر المتاحة في بيئة PowerShell. هو أداة لا غنى عنها لأي حد بيشتغل على PowerShell سواء كان مبتدئ أو محترف.

- يمكن استخدام الأمر بالتكامل مع أوامر أخرى مثل 'Get-Help' للحصول على معلومات أكثر تفصيلية عن أي أمر تجدها بواسطة 'Get-Command'.

باختصار، 'Get-Command' هو الأمر اللي بيساعدك تكتشف وتتعرف على كل الأوامر المتاحة ليك في PowerShell، وده ببسهل عليك بشكل كبير إنك تلاقي الأوامر اللي تحتاجها وتفهم كيفية استخدامها.

## 2.5 Start-Process

الاستخدام الأساسي:

## 1. تشغيل برنامج معين:

- لو عايز تشغل برنامج زي "النوت باد"، هتكتب كده.

```
Start-Process notepad
```

خيارات إضافية لأمر Start-Process:

### 1. تشغيل برنامج باستخدام المسار الكامل:

- لو عندك برنامج في مكان معين على الجهاز وعايز تشغله.

```
Start-Process "C:\Path\To\Your\Application.exe"
```

### 2. فتح ملف باستخدام البرنامج الافتراضي:

- لو عايز تفتح ملف زي مستند وورد باستخدام البرنامج اللي النظام بتاعك بيستخدمه تلقائيًا لفتح النوع ده من الملفات.

```
Start-Process "C:\Path\To\Your\Document.docx"
```

### 3. تشغيل برنامج كمسؤول:

- لو عايز تشغل برنامج بامتيازات المسؤول (يعني بصلاحيات أعلى).

```
Start-Process "powershell.exe" -Verb RunAs
```

### 4. تحديد شكل النافذة:

- ممكن تحدد شكل نافذة البرنامج لما يفتح (مكبرة، مصغرة، مخفية، أو عادية).

```
Start-Process "notepad.exe" -WindowState Minimized
```

### 5. تشغيل برنامج في نفس النافذة:

- لو عايز تشغل برنامج من غير ما يفتح نافذة جديدة.

```
Start-Process "notepad.exe" -NoNewWindow
```

### 6. تحديد مجلد التشغيل:

- ممكن تحدد المجلد اللي البرنامج هيشغل منه.

```
Start-Process "notepad.exe" -WorkingDirectory  
"C:\Path\To\Your\Folder"
```

### 7. الحصول على معلومات عن العملية:

- لو عايز تعرف تفاصيل أكثر عن العملية اللي شغلتها.

```
$process = Start-Process "notepad.exe" -PassThru  
$process.Id
```

أمثلة إضافية:

### 1. فتح موقع على الإنترنت:

- ممكن تستخدم الأمر ده لفتح موقع على المتصفح الافتراضي.

```
Start-Process "https://www.google.com"
```

2. انتظار انتهاء البرنامج:

- ممكن تستخدم 'Wait' لو عايز تستنى البرنامج يخلص تشغيل قبل ما تكمل باقي الأوامر.

```
Start-Process "notepad.exe" -Wait
```

3. إعادة توجيه إخراج العملية:

- ممكن تخلي إخراج البرنامج يتسجل في ملف.

```
Start-Process "ping.exe" -ArgumentList "google.com" -  
RedirectStandardOutput "C:\Path\To\Output.txt"
```

ملاحظات:

- الأمر 'Start-Process' مفيد لما تحتاج تشغل برامج أو سكريبتات من جوه PowerShell، سواء لفتح ملفات أو تشغيل أدوات النظام أو حتى فتح مواقع ويب.

- تأكد من استخدام المعاملات الصح عشان تحقق اللي انت محتاجه، زي تشغيل البرنامج وكأنك Admin

## 2.6 Get-Process

الأمر 'Get-Process' في PowerShell هو أداة قوية بتسمحك تعرض معلومات عن العمليات (Processes) اللي شغالة على النظام بتاعك. ممكن تستخدمه عشان تشوف العمليات الحالية، تعرف استهلاك الموارد زي الـ CPU والذاكرة، وكمان تقدر تصفي وتحدد العمليات اللي انت مهتم بيها.

1. عرض كل العمليات:

- ببساطة، لو عايز تشوف كل العمليات اللي شغالة على النظام.

```
Get-Process
```

خيارات إضافية للأمر Get-Process:

1. تصفية العمليات باسم معين:

- لو عايز تشوف عملية معينة أو عمليات باسم معين.

```
Get-Process -Name "notepad"
```

2. تصفية العمليات بمعرف العملية (ID):

- لو عايز تشوف تفاصيل عملية بمعرفها.

```
Get-Process -Id 1234
```

3. عرض العمليات الخاصة بمستخدم معين:

- لو عايز تشوف العمليات اللي بيشغلها مستخدم معين.

```
Get-Process -IncludeUserName
```

4. عرض تفاصيل أكثر عن العمليات:

- لو عايز تفاصيل أكثر عن كل عملية.

```
Get-Process | Format-List*
```

5. عرض العمليات وترتيبها بناءً على استخدام الـ CPU:

- تقدر تشوف العمليات مرتبة بناءً على استهلاكها للـ CPU.

```
Get-Process | Sort-Object CPU -Descending
```

6. عرض العمليات واستهلاك الذاكرة:

- تقدر تشوف استهلاك العمليات للذاكرة.

```
Get-Process | Sort-Object WorkingSet -Descending
```

استخدامات متقدمة:

1. إيقاف عملية معينة:

- لو عايز توقف عملية معينة باستخدام 'Stop-Process'.

```
Stop-Process -Name "notepad"
```

2. الحصول على العمليات من جهاز آخر:

- تقدر تستخدم 'ComputerName-' للحصول على العمليات من جهاز آخر في الشبكة.

```
Get-Process -ComputerName "RemotePC"
```

3. استخدام معرف العمليات (PID) لإجراءات أخرى:

- لو عايز تاخذ معرف العمليات وتستخدمه لإجراءات ثانية.

```
$ process = Get-Process -Name "notepad"
```

```
$ process.Id
```

ملاحظات:

- الأمر 'Get-Process' بيدي معلومات مفيدة جداً لإدارة النظام ورصد الأداء.

- ممكن تستخدمه في سكريبتات PowerShell لأتمتة مهام معينة أو لمراقبة النظام بشكل دوري.

باختصار، 'Get-Process' يساعدك تراقب وتدير العمليات اللي شغالة على النظام بتاعك، وبيوفرلك طرق كتير لتصفية وترتيب البيانات عشان تحصل على المعلومات اللي انت محتاجها بسرعة وكفاءة.

## 2.7 Pipeline Select

في PowerShell، الـ Pipeline هو طريقة لنقل البيانات من أمر لأمر تاني. ببساطة، بتاخذ نتيجة أمر معين وتخليها تكون مدخل لأمر تاني. ده بيخلي كتابة الأوامر وتنفيذ المهام أكثر كفاءة وأسهل في القراءة.

مثال بسيط:

```
Get-Process | Sort-Object CPU -Descending
```

هنا بنستخدم الـ Pipeline (اللي هو الرمز '|') عشان ناخذ العمليات اللي رجعتها الأمر 'Get-Process' وندخلها للأمر 'Sort-Object' عشان نرتبها حسب استخدام الـ CPU.



الأمر 'Select-Object' يستخدم عشان تختار أو تعمل Filter خصائص معينة من الناتج بتاع أمر ثاني. ممكن تفكر فيه إنه زي الفلتر اللي بيخليك تختار الأعمدة اللي انت عايزها من جدول كبير.

استخدامات 'Select-Object':

1. اختيار خصائص معينة:

- لو عايز تختار خصائص معينة من الناتج.

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

هنا بنعرض بس اسم العملية، واستخدام الـ CPU، ومعرف العملية (ID).

2. عرض أول كام نتيجة:

- لو عايز تشوف أول عدد معين من النتائج.

```
Get-Process | Select-Object -First 5
```

3. عرض آخر كام نتيجة:

- لو عايز تشوف آخر عدد معين من النتائج.

```
Get-Process | Select-Object -Last 5
```

4. عرض العمليات وترتيبها حسب الذاكرة واختيار اسم العملية ومعرفها:

```
Get-Process | Sort-Object WorkingSet -Descending | Select-Object  
Name, Id
```

ملاحظات:

- الأمر 'Select-Object' بيساعدك تصفي البيانات اللي محتاجها بالظبط، وده بيخلي النتائج أكثر وضوح وسهولة في التحليل.

- استخدام الـ Pipeline بيخليك تكتب الأوامر بشكل أكثر تنظيم وكفاءة، وده بيكون مفيد جدا لما تكون بتكتب سكريبتات معقدة.

باختصار، الـ Pipeline والأمر 'Select-Object' هما أدوات قوية في PowerShell بتساعدك تنقل البيانات بين الأوامر وتصفية النتائج عشان تحصل على المعلومات اللي محتاجها بشكل دقيق وسريع.

### 3 System Environment Variables and Paths

Environment Variables هي زي صناديق صغيرة بتخزن معلومات معينة عن النظام أو عن المستخدم اللي شغال عليه. المعلومات دي بتكون مفيدة للبرامج والتطبيقات اللي بتشتغل على الكمبيوتر عشان تعرف تستخدمها بسهولة من غير ما تدخلها كل مرة.

أمثلة على Variables البيئية:

#### 1. PATH:

- ده أهم متغير بيئي تقريبًا. بيحتوي على قائمة من المسارات (Paths) اللي بتقول للكمبيوتر فين يدور على البرامج التنفيذية لما تكتب اسم برنامج في الـ Command Prompt أو PowerShell. يعني لو كتبت اسم برنامج من غير ما تكتب المسار الكامل، الكمبيوتر بيشفوف في المسارات اللي في متغير 'PATH' عشان يلاقي البرنامج ده.

```
C:\Users\Mohamed Amr>echo %path%
C:\Program Files (x86)\VMware\VMware Workstation\bin;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH;C:\Program Files (x86)\HP\IHDR_15.2.10.1114\;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR;C:\Program Files\Cloudflare\Cloudflare WARP;C:\Program Files\dotnet\;C:\Program Files\PowerShell\7\;C:\Program Files\PowerShell\7-preview\preview;C:\Users\Mohamed Amr\AppData\Local\Programs\Python\Python312\Scripts\;C:\Users\Mohamed Amr\AppData\Local\Programs\Python\Python312\;C:\Users\Mohamed Amr\AppData\Local\Microsoft\WindowsApps;C:\Users\Mohamed Amr\AppData\Local\Programs\Microsoft VS Code\bin
```

#### 2. TEMP و 'TMP':

- دول بيحتويوا على المسار اللي بيخزن فيه النظام الملفات المؤقتة.

```
C:\Users\Mohamed Amr>echo %tmp%
C:\Users\MOHAME~1\AppData\Local\Temp
```

#### 3. 'USERPROFILE':

- ده بيحتوي على المسار لمجلد المستخدم الحالي (اللي هو مجلد المستخدم بتاعك).

```
C:\Users\Mohamed Amr>echo %userprofile%
C:\Users\Mohamed Amr
```

#### 4. SystemRoot:

- ده بيشير لمجلد النظام الرئيسي، غالبًا بيكون 'C:\Windows'.

```
C:\Users\Mohamed Amr>echo %systemroot%
C:\Windows
```

دي أشهر حاجات موجود ف النظام ولو حابب اشوف معظمها هعمل الامر دا علي Powershell خد بالك الاوامر اللي فوق بتشتغل بالشكل دا بس علي CMD اما ف الـ Power shell ف ليها شكل مختلف شويه

```
PS C:\Users\Mohamed Amr> ls env:/
```

Name	Value
----	----
__PSLockDownPolicy	0
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\Mohamed Amr\AppData\Roaming
CommonProgramFiles	C:\Program Files\Common Files
CommonProgramFiles(x86)	C:\Program Files (x86)\Common Files
CommonProgramW6432	C:\Program Files\Common Files
COMPUTERNAME	DESKTOP-V6SJVQ9
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
FPS_BROWSER_APP_PROFILE_STRING	Internet Explorer
FPS_BROWSER_USER_PROFILE_ST...	Default
HOMEDRIVE	C:
HOMEPath	\Users\Mohamed Amr
LOCALAPPDATA	C:\Users\Mohamed Amr\AppData\Local
LOGONSERVER	\\DESKTOP-V6SJVQ9
NUMBER_OF_PROCESSORS	8
OneDrive	C:\Users\Mohamed Amr\OneDrive
OS	Windows_NT
Path	C:\Program Files (x86)\VMware\VMware Workstation\bin\;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;...
POWERSHELL_DISTRIBUTION_CHA...	MSI:Windows 10 Pro
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	8e0c
ProgramData	C:\ProgramData
ProgramFiles	C:\Program Files
ProgramFiles(x86)	C:\Program Files (x86)
ProgramW6432	C:\Program Files
PSModulePath	C:\Users\Mohamed Amr\Documents\WindowsPowerShell\Modul...
PUBLIC	C:\Users\Public
SESSIONNAME	Console
SystemDrive	C:
SystemRoot	C:\Windows
TEMP	C:\Users\MOHAME~1\AppData\Local\Temp

Environment Variables زي منت شايف دي معظم الحاجات اللي استعملتها فوق باستخدام ال CMD لما ابي استعملها ف ال Powershell مش هتختلف كثير عموما هتكتب بالشكل دا تعالي نشوف امثله علي حاجات تانيه

```
PS C:\Users\Mohamed Amr> echo $env:TEMP
C:\Users\MOHAME~1\AppData\Local\Temp
PS C:\Users\Mohamed Amr> echo $env:windir
C:\Windows
PS C:\Users\Mohamed Amr> echo $env:OS
Windows_NT
PS C:\Users\Mohamed Amr> echo $env:PUBIC
C:\Users\Public
PS C:\Users\Mohamed Amr>
```

System Paths هي زي عناوين بتحدد فين الملفات والمجلدات موجودة على الكمبيوتر. لما بنقول "Paths"، بنقصد بيه الطريق الكامل اللي لازم تمشيته عشان توصل للملف أو المجلد المطلوب.

مثال على Paths:

1. مسار لملف نصي:

```
'C:\Users\YourUsername\Documents\example.txt'
```

- ده بيقولك إن الملف "example.txt" موجود في مجلد "Documents" اللي جوه مجلد المستخدم بتاعك "YourUsername" على الدرايف 'C:'.

ليه بنستخدم System Environment Variables and Paths ؟

1. تسهيل الوصول للبرامج:

- متغير 'PATH' ببسهل عليك تشغيل البرامج من غير ما تكتب المسار الكامل ليها كل مرة.

2. توحيد الأماكن المستخدمة:

- متغيرات زي 'TEMP' و 'USERPROFILE' بتساعد البرامج تعرف فين تخزن الملفات المؤقتة أو فين تلاقى ملفات المستخدم.

3. أتمتة العمليات:

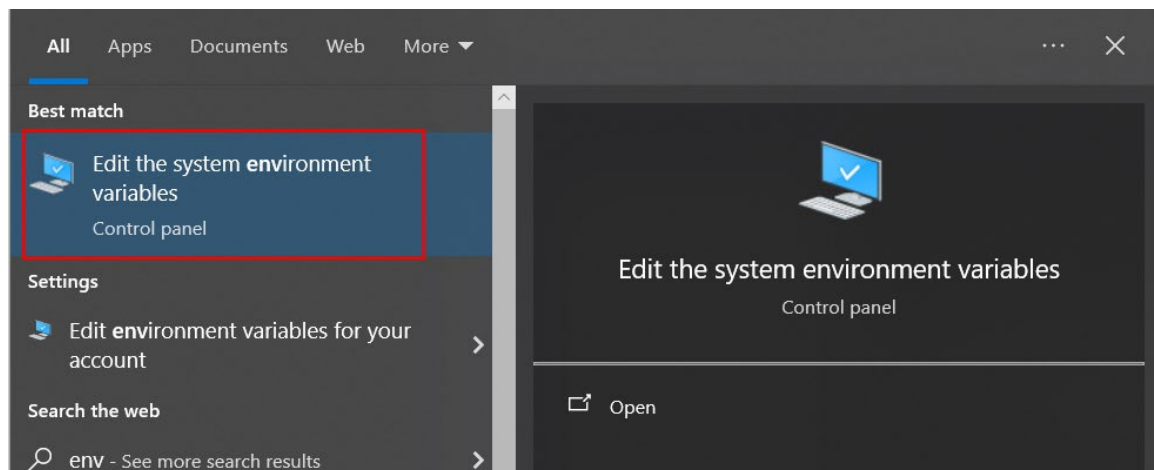
- لما تكتب سكريبتات أو برامج، ممكن تستخدم Variables البيئية عشان تكون مرنة وتشغل على أي جهاز بنفس الكفاءة من غير ما تغير فيها.

إزاي تشوف أو تعدل Variables البيئية في Windows:

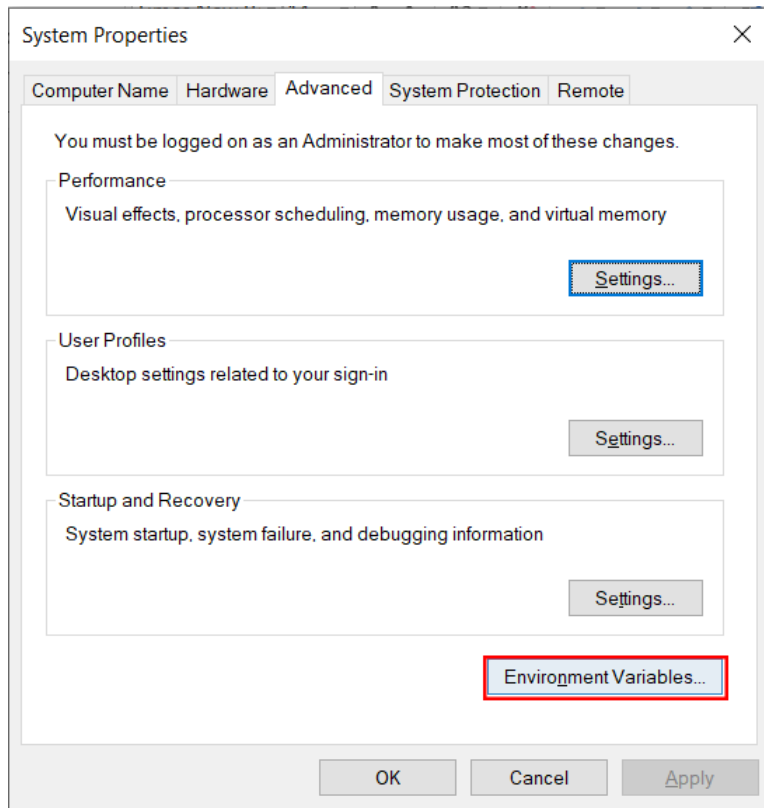
عن طريق واجهة المستخدم (GUI):

1. عرض Variables البيئية:

- اعمل Search علي the system environment variables .

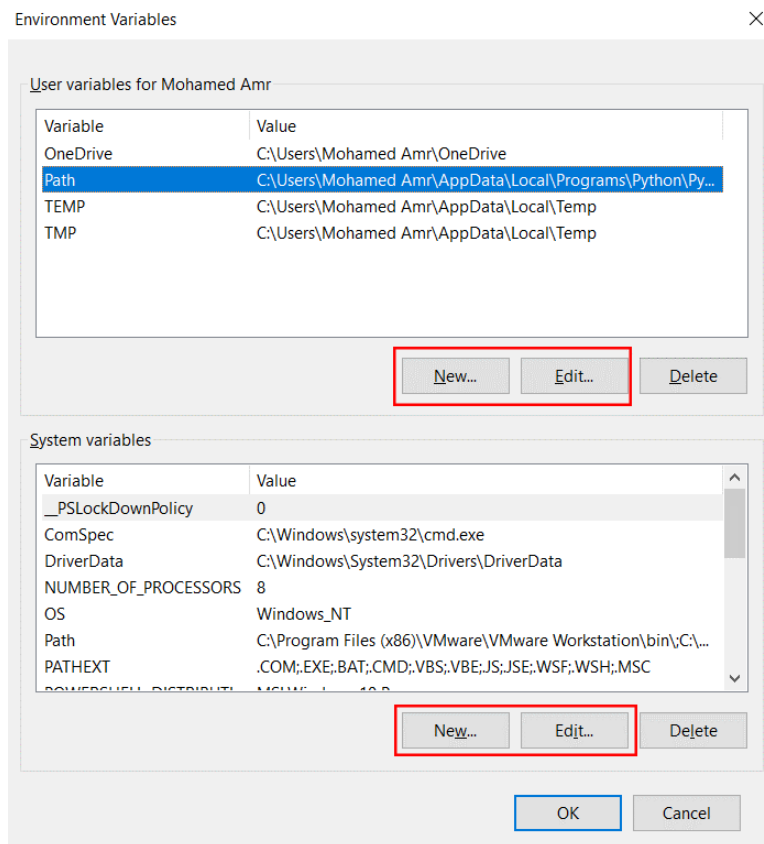


- في التبويب "Advanced"، اضغط على الزر "Environment Variables".

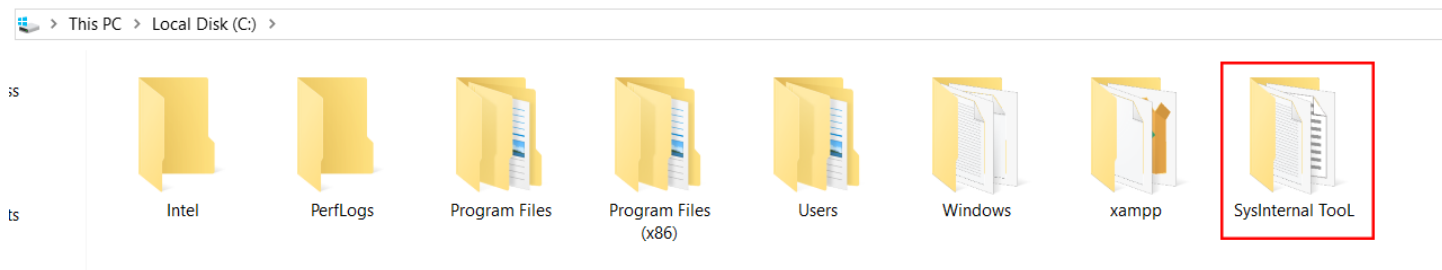


## 2. تعديل Variables البيئية:

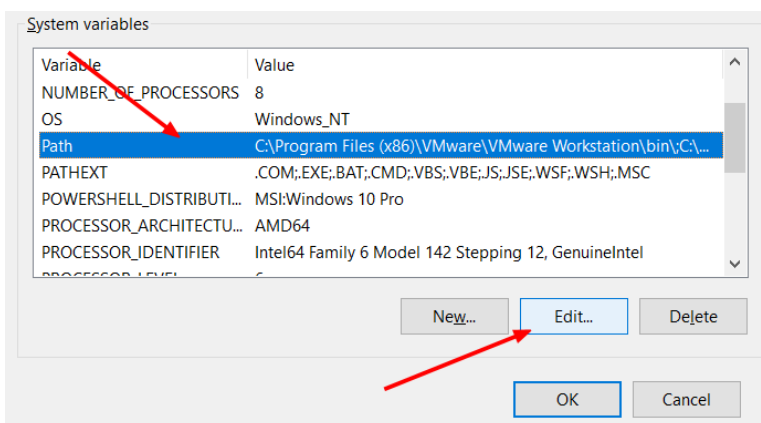
- في نافذة "Environment Variables"، هتلاقى قسمين: "User variables" و "System variables".
- عشان تعدل أو تضيف متغير بيئي، اختاره واضغط "Edit" أو "New".



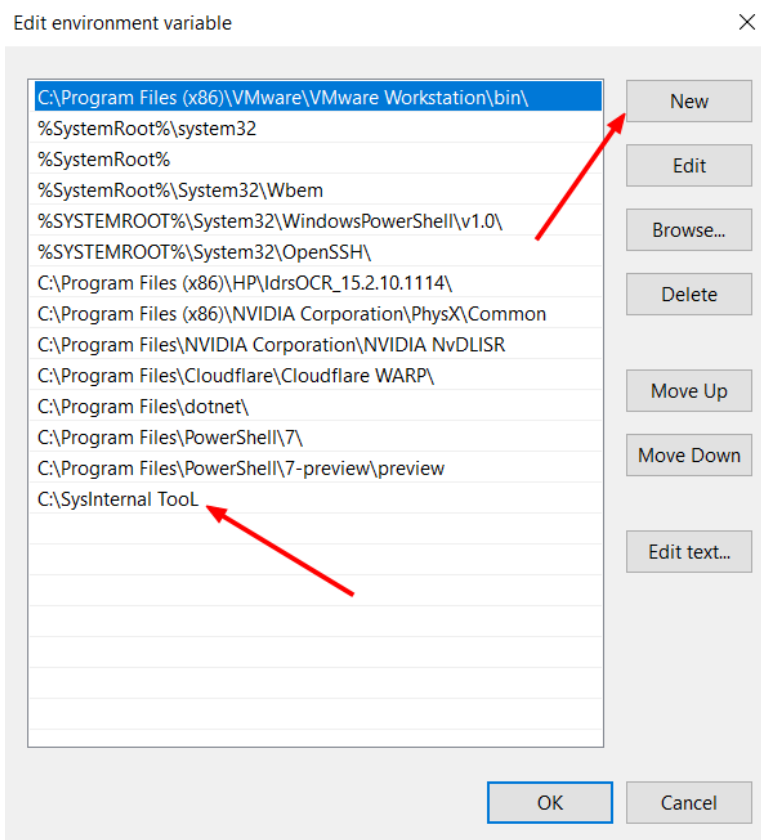
## خلينا ناخذ مثال اعمق شويه نفهم بيه اكثر عندنا فولدر بتاع Sysinternal Tool



لو انا حبيت اضيفه بحيث ان معظم البرامج اللي جوا اقدر اشغلها من اي مكان علي cmd بدل ما اروح استدعيها من مكانها كل مره ف بعمل ايه بقي هضيفها مثلا ف C ف مكان معين وافتح "Environment Variables"، خش جوا ال "System variables" زي ما ف الصورة



هتعمل New ونضيف المسار



مجرد ما تفتح اي Tool جوا ال Sysinternal وانت ف اي مكان علي Powershell او CMD هتلاقي ال Tool دي بتشتغل  
مجبرب ما تكتب اسمها زي Tool اسمها Strings مثلاً

```
C:\Users\Mohamed Amr>strings

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

usage: strings [-a] [-f offset] [-b bytes] [-n length] [-o] [-s] [-u] <file or directory>
-a      Ascii-only search (Unicode and Ascii is default)
-b      Bytes of file to scan
-f      File offset at which to start scanning.
-o      Print offset in file string was located
-n      Minimum string length (default is 3)
-s      Recurse subdirectories
-u      Unicode-only search (Unicode and Ascii is default)
-nobanner
      Do not display the startup banner and copyright message.
```

ملاحظات:

- التغييرات اللي بتعملها عن طريق PowerShell بتكون مؤقتة للجلسة الحالية بس. لو قفلت ال PowerShell وفتحته تاني، التغييرات دي مش هتكون موجودة.

- لو عايز تغييرات دائمة، استخدم واجهة المستخدم (GUI) زي ما شرحنا فوق.

باختصار، Variables البيئية والمسارات هما أدوات مهمة جداً في نظام التشغيل. ببساعدوا في تسهيل الوصول للبرامج والملفات، وبيخلو الحياة أسهل لما تتعامل مع الكمبيوتر.

تمام، خلينا نشرح Variables وأنواع البيانات في PowerShell بطريقة بسيطة وبالعامية المصرية.

### 3.1 Variables in PowerShell:

Variables في PowerShell زي ما قلنا، هي زي صناديق أو حافظات بنخزن فيها بيانات. بنستخدم علامة الدولار '\$' قبل اسم Variable عشان نعرف PowerShell إن ده متغير.

مثال على Variables:

إنشاء متغير اسمه "age" وتخزين فيه قيمة 25

```
$age = 25
```

إنشاء متغير اسمه "name" وتخزين فيه نص "Ahmed"

```
$name = "Ahmed"
```

أنواع البيانات في PowerShell:

زي لغات البرمجة الثانية، PowerShell بيدعم أنواع مختلفة من البيانات اللي نقدر نخزنها في Variables.

الأنواع الأساسية للبيانات:

1. الأعداد الصحيحة (Integers):

- دي الأرقام الصحيحة من غير كسور. زي 1، 10، -5.

```
$age = 25
```

2. الأعداد العشرية (Doubles):

- دي الأرقام اللي فيها كسور عشرية. زي 3.14، 0.99، -2.5.

```
$price = 19.99
```

3. النصوص (Strings):

- دي النصوص أو السلاسل النصية اللي بتتكون من حروف وأرقام ورموز. زي "Hello"، "Ahmed"، "123ABC".

```
$name = "Ahmed"
```

4. القيم المنطقية (Booleans):

- دي القيم اللي بتكون إما \$True أو \$False، وغالبًا بتستخدم في الشروط.

```
$isStudent = $True
```

5. المصفوفات (Arrays):

- دي مجموعة من العناصر اللي ممكن تكون من أي نوع بيانات، وممكن كمان تكون أنواع مختلفة مع بعضها. زي @1، 2، 3 أو @("apple", "banana", "cherry").

```
$fruits = @("apple", "banana", "cherry")
```

6. القواميس (Hashtables):

- دي بنسبها في البرمجة ماب أو دكت، وهي مجموعة من الأزواج (مفتاح: قيمة). زي @{"name"="Ahmed"; "age"=25}.

```
$person = @{"name"="Ahmed"; "age"=25}
```

استخدام Variables وأنواع البيانات معًا:

خليني أوريك إزاي بتستخدم Variables مع أنواع البيانات المختلفة في سكريبت PowerShell بسيط.

مثال عملي:

إنشاء متغيرات بأنواع بيانات مختلفة

```
$name = "Ahmed"
$age = 25
$height = 1.75
$isStudent = $True
$fruits = @("apple", "banana", "cherry")
$person = @{"name"="Ahmed"; "age"=25}
```

طباعة القيم المخزنة في Variables

```
Write-Output "Name: $name"
Write-Output "Age: $age"
```



```
Write-Output "Height: $height"
Write-Output "Is student: $isStudent"
Write-Output "Fruits: $($fruits -join ', ')"
Write-Output "Person info: $($person["name"]),
$($person["age"]) "
```

استخدام Variables في العمليات الحسابية

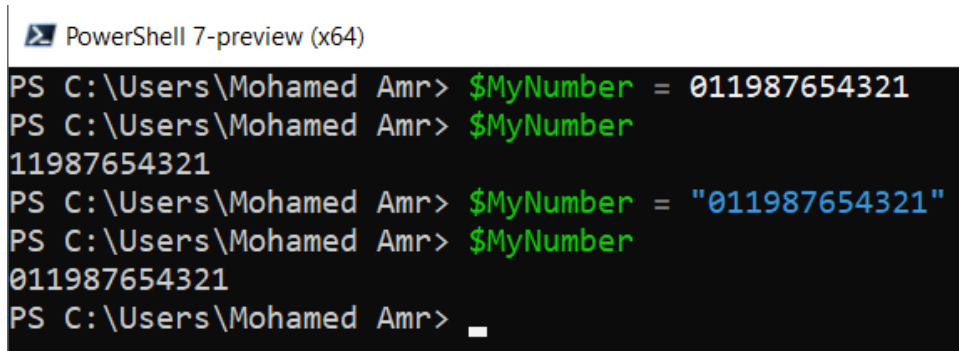
```
$newAge = $age + 1
Write-Output "New Age: $newAge"
```

شرح المثال:

- بننشئ متغيرات بأنواع بيانات مختلفة (نصوص، أعداد صحيحة، أعداد عشرية، قيم منطقية، مصفوفات، وقواميس).
- بنطبع القيم المخزنة في Variables باستخدام الأمر 'Write-Output'.
- بنستخدم Variables في عملية حسابية (زيادة العمر بسنة واحدة).

ملاحظات:

- PowerShell يدعم أنواع كثير من البيانات، وده ببسهل التعامل مع البيانات بطرق مختلفة.
- Variables بتساعدنا نخزن البيانات ونعمل عليها عمليات بسهولة.
- من المهم نعرف نوع البيانات اللي بنشتغل بيها عشان نقدر نستخدمها بشكل صحيح في السكريبتات بتاعتنا.
- لو لاحظت هنا ف Variables الاول نا هنا اتعرف عليه ف الاول علي انه رقم عشان كذا مطبعش الصفر اللي علي الشمال لحد ما حطيته بين علامتين تنصيص "" وسعتها اتعرف عليه انه String



```
PowerShell 7-preview (x64)
PS C:\Users\Mohamed Amr> $MyNumber = 011987654321
PS C:\Users\Mohamed Amr> $MyNumber
11987654321
PS C:\Users\Mohamed Amr> $MyNumber = "011987654321"
PS C:\Users\Mohamed Amr> $MyNumber
011987654321
PS C:\Users\Mohamed Amr> _
```

## 3.2 Arithmetic Operators

بالتأكيد! الـ Arithmetic Operators (العوامل الحسابية) في PowerShell تُستخدم لتنفيذ العمليات الحسابية الأساسية مثل الجمع والطرح والضرب والقسمة. هذه العوامل تشمل:

1. عامل الجمع (+):

- يُستخدم لإضافة قيمتين معًا.

- مثال:  $5 + 3$  ينتج  $8$ .

2. عامل الطرح (-):

- يُستخدم لطرح قيمة من أخرى.

- مثال: `10 - 4` ينتج `6`.

3. عامل الضرب (\*):

- يُستخدم لضرب قيمتين معًا.

- مثال: `7 * 3` ينتج `21`.

4. عامل القسمة (/):

- يُستخدم لقسمة قيمة على أخرى.

- مثال: `20 / 4` ينتج `5`.

5. عامل باقي القسمة (%):

- يُستخدم للحصول على باقي القسمة بين قيمتين.

- مثال: `10 % 3` ينتج `1`.

6. عامل الأس ():

- يُستخدم لرفع قيمة إلى قوة معينة.

- مثال: `2 ^ 3` ينتج `8`.

أمثلة توضيحية في PowerShell

لنرى كيف يمكن استخدام هذه العوامل في PowerShell:

1. الجمع (+):

```
$a = 5
$b = 3
$result = $a + $b
Write-Output $result
```

2. الطرح (-):

```
$a = 10
$b = 4
$result = $a - $b
Write-Output $result # الناتج سيكون 6
```

3. الضرب (\*):

```
$a = 7
$b = 3
$result = $a * $b
```

```
Write-Output $result # الناتج سيكون 21
```

4. القسمة (/):

```
$a = 20
$b = 4
$result = $a / $b
Write-Output $result # الناتج سيكون 5
```

5. باقي القسمة (%):

```
$a = 10
$b = 3
$result = $a % $b
Write-Output $result # الناتج سيكون 1
```

6. الأس ():

```
$a = 2
$b = 3
$result = $a $b
Write-Output $result # الناتج سيكون 8
```

ملاحظات:

- يمكن استخدام هذه العوامل مع الأعداد الصحيحة (integers) والأعداد العشرية (floating-point numbers).
- تأكد من استخدام القوسين `()` لتحديد أولويات العمليات الحسابية عندما تكون هناك عمليات متعددة.
- إذا كنت بحاجة إلى المزيد من الأمثلة أو لديك أسئلة معينة، فلا تتردد في طرحها!

### 3.3 Assignment Operators

في PowerShell، تُستخدم الـ Assignment Operators (عوامل الإسناد) لتعيين قيمة لمتغير. هذه العوامل تتضمن العوامل الحسابية بالإضافة إلى عوامل أخرى تسهل عملية الإسناد والتعديل على القيم الموجودة في Variables. إليك شرحًا لأهم الـ Assignment Operators في PowerShell:

1. عامل الإسناد (=):

- يُستخدم لتعيين قيمة لمتغير.

```
$a = 10
```

2. عوامل الإسناد القصيرة (=%, /=, \*=, -=, +=):

- تُستخدم لإضافة قيمة إلى Variable الحالي أو طرح قيمة منه أو ضربه أو قسمته أو استخراج باقي القسمة.

- مثال:

```
$a = 5
```

```
$a += 3 # إلى قيمة 3 ، النتيجة ستكون $a
```

3. عامل الإسناد باستخدام الأس (=):

- يُستخدم لرفع قيمة Variable إلى قوة معينة.

- مثال:

```
$a = 2
```

```
$a = 8 إلى القوة 3 ، النتيجة ستكون $a تُرفع قيمة 3 #
```

أمثلة توضيحية في PowerShell

لنرى كيفية استخدام هذه العوامل في PowerShell:

عوامل الإسناد القصيرة (=+ , =- , \*= , /= , %=):

```
$b = 5
```

```
$b += 3
```

```
$b -= 2
```

```
$b *= 4
```

```
$b /= 3
```

```
$b %= 5
```

```
Write-Output $b
```

ملاحظات:

- يمكن استخدام هذه العوامل لتعديل قيم Variables بطريقة مباشرة وسهلة.
- يجب أن يكون Variable معرفًا بالفعل قبل استخدام عوامل الإسناد.
- العوامل القصيرة تساعد في تقليل كمية الكود المكتوب وجعله أكثر وضوحًا وسهولة في الفهم.
- إذا كان لديك أي استفسارات أخرى أو تحتاج إلى مزيد من الشرح، فأنا هنا للمساعدة!

### 3.4 Logical Operators

في PowerShell، تُستخدم الـ Logical Operators (العوامل المنطقية) للتحقق من شروط معقدة تشمل تحليل البيانات واتخاذ قرارات استنادًا إلى النتائج المنطقية لهذه الشروط. هذه العوامل تشمل:

1. عامل الواقع ('-and') (AND):

- يُستخدم للتحقق من صحة شرطين في نفس الوقت.

- يُعادل العملية اللوجيكية "و".

- مثال:

```
$a = $true
```

```
$b = $false
```

```
$result = $a -and $b # النتيجة ستكون $false
```

```
Write-Output $result
```

2. عامل الأول ('-or') (OR):

- يُستخدم للتحقق من صحة أحد الشروط على الأقل.

- يُعادل العملية اللوجيكية "أو".

- مثال:

```
$a = $true
$b = $false
$result = $a -or $b # النتيجة ستكون $true
Write-Output $result
```

3. عامل الليس ('-not') (NOT):

- يُستخدم لعكس قيمة الشرط، أي يجعل الصحيح خطأ والخطأ صحيح.

- يُعادل العملية اللوجيكية "لا".

- مثال:

```
$a = $true
$result = -not $a # النتيجة ستكون $false
Write-Output $result
```

أمثلة توضيحية في PowerShell

لنرى كيفية استخدام هذه العوامل في PowerShell:

1. عامل الواقع ('-and') (AND):

```
$age = 25
$isAdult = $true
$result = ($age -ge 18) -and $isAdult # يتحقق من أن العمر أكبر
# صحيح $isAdult من أو يساوي 18 وأن
Write-Output $result # النتيجة ستكون $true
```

2. عامل الأول ('-or') (OR):

```
$isHoliday = $true
$isWeekend = $false
$result = $isHoliday -or $isWeekend # يتحقق من أن اليوم عطلة
# أو عطلة نهاية الأسبوع
Write-Output $result # النتيجة ستكون $true
```

- يمكن استخدام هذه العوامل لإنشاء تعبيرات منطقية متعددة وتحقق الشروط المعقدة.

- يمكن دمج هذه العوامل معًا للحصول على تعبيرات منطقية أكثر تعقيدًا باستخدام قوسين للتأكد من تقييم الشروط بالترتيب المطلوب.

- يمكن استخدام هذه العوامل في بناء شروط الـ If statements وأوامر التحكم الأخرى في PowerShell.

### 3.5 Logical Operators

في PowerShell ، تُستخدم الـ Comparison Operators (عوامل المقارنة) لمقارنة قيم مختلفة وتحديد علاقتها ببعضها البعض. هذه العوامل تُستخدم بشكل شائع في بناء شروط مثل في عبارات الـ If وفي التعبيرات الشرطية الأخرى. إليك شرح لأهم الـ Comparison Operators في PowerShell:

مثال لعامل المساوي (Equal to)

```
$a = 10
$b = 10
$result = $a -eq $b # النتيجة ستكون $true
Write-Output $result
```

مثال لعامل غير مساوي (Not equal to)

```
$c = 5
$d = 10
$result = $c -ne $d # النتيجة ستكون $true
Write-Output $result
```

مثال لعامل الأكبر من (Greater than)

```
$e = 15
$f = 10
$result = $e -gt $f # النتيجة ستكون $true
Write-Output $result
```

مثال لعامل الأصغر من (Less than)

```
$g = 5
$h = 10
$result = $g -lt $h # النتيجة ستكون $true
Write-Output $result
```

مثال لعامل الأكبر من أو يساوي (Greater than or equal to)

```
$i = 10
$j = 10
$result = $i -ge $j # النتيجة ستكون $true
Write-Output $result
```

مثال لعامل الأصغر من أو يساوي (Less than or equal to)

```
$k = 5
```

```
$l = 10
$result = $k -le $l # النتيجة ستكون $true
Write-Output $result
```

في PowerShell، تُستخدم عمليات إعادة التوجيه (Redirection)، والفصل (Split)، والانضمام (Join) للتحكم في تدفق البيانات ومعالجتها بشكل فعال. إليك شرحًا لكل منها:

### 3.6 Redirection Operators

تُستخدم عمليات Redirection الـ Outputs التي خارجة من اوامر معينة عشان تحطها جوا ملف او ملفات.

1. Redirection المخرجات القياسية ('>'):

- يُستخدم عشان تحط الـ Outputs في ملف.

- مثال:

```
Get-Process > processes.txt
```

2. إعادة توجيه المخرجات القياسية وإلحاق ('>>'):

- يُستخدم عشان تحط الـ Outputs في ملف ده غير انه بيضيف علي القديم.

- مثال:

```
Get-Service >> services.txt
```

3. إعادة توجيه الأخطاء القياسية ('2>'):

- في الجزء الذي فات لو كان حصل ايرو مكانش بيحفظه جوا الملف هنا بقي لو ف اي ايروور يحطه جوا الملف.

- مثال:

```
Get-Process -Name "nonexistent" 2> errors.txt
```

4. إعادة توجيه الأخطاء القياسية وإلحاق ('2>>'):

- زي الذي فاتت بس بيضيف علي الملف القديم.

- مثال:

```
Get-Process -Name "nonexistent" 2>> errors.txt
```

### 3.7 Split and join

تُستخدم عملية الفصل لتقسيم النص إلى أجزاء باستخدام محدد معين.

1. عامل الفصل ('split-'):

- يُستخدم لتقسيم سلسلة نصية إلى مجموعة من الأجزاء استنادًا إلى محدد معين.
- مثال:

```
$text = "apple,banana,cherry"
$result = $text -split ","
$result # النتيجة ستكون مجموعة تحتوي على ["apple", "banana", "cherry"]
```

Join Operator (عامل الانضمام)

تُستخدم عملية join لدمج مجموعة من strings في سلسلة واحدة باستخدام محدد معين.

1. عامل الانضمام ('join-'):

- يُستخدم لدمج مجموعة من السلاسل النصية في سلسلة واحدة باستخدام محدد معين.
- مثال:

```
$array = @("apple", "banana", "cherry")
$result = $array -join ","
$result # النتيجة ستكون "apple,banana,cherry"
```

ملاحظات:

- عمليات إعادة التوجيه تُستخدم بشكل شائع لتوجيه المخرجات أو الأخطاء إلى ملفات للسجلات أو التحليل.
- عمليات الفصل والانضمام تُستخدم لمعالجة النصوص بسهولة، سواء كنت تحتاج إلى تقسيم النصوص إلى أجزاء أصغر أو دمج مجموعة من النصوص في نص واحد.



## 4 Conditional Statements

### 4.1 If statement

عبارات الـ `If` في PowerShell تُستخدم لتنفيذ الكود بناءً على شرط معين. إذا كان الشرط صحيحاً (True)، يتم تنفيذ الكود الموجود داخل كتلة الـ `If`. إذا لم يكن الشرط صحيحاً، يمكن استخدام عبارات `ElseIf` و `Else` لتنفيذ كود بديل. إليك شرحاً مفصلاً لكيفية استخدام هذه العبارات:

بنية If Statement

3. عبارة If مع ElseIf و Else

- يمكن استخدام عبارات `ElseIf` لتحديد شروط متعددة.

- الصيغة:

```
if (الشرط الاول) {  
    # كود ينفذ إذا كان الشرط الاول صحيحاً  
} elseif (الشرط الثاني) {  
    # كود ينفذ إذا كان الشرط الثاني صحيحاً  
} else {  
    # كود ينفذ إذا كانت جميع الشروط غير صحيحة  
}
```

أمثلة توضيحية

مثال 1: عبارة If الأساسية

```
$a = 10  
if ($a -eq 10) {  
    Write-Output "القيمة تساوي 10"  
}
```

مثال 2: عبارة If مع Else

```
$a = 5  
if ($a -eq 10) {  
    Write-Output "القيمة تساوي 10"  
} else {  
    Write-Output "القيمة لا تساوي 10"  
}
```

مثال 3: عبارة If مع ElseIf و Else

```
$a = 15  
if ($a -eq 10) {  
    Write-Output "القيمة تساوي 10"  
} elseif ($a -eq 15) {
```

```

Write-Output "القيمة تساوي 15"
} else {
    Write-Output "القيمة ليست 10 ولا 15"
}

```

استخدام تعبيرات منطقية مع If

يمكن استخدام تعبيرات منطقية مع عبارات الـ `If` للتحقق من شروط متعددة باستخدام عوامل المقارنة المنطقية مثل `and` و `or`.

مثال: استخدام تعبيرات منطقية

```

$a = 20
$b = 30
if ($a -eq 20 -and $b -eq 30) {
    Write-Output "القيم مطابقة"
} else {
    Write-Output "القيم غير مطابقة"
}

```

ملاحظات:

- يجب وضع الشرط بين أقواس `()` عند كتابة عبارة الـ `If`.

- يجب وضع الكود الذي سيتم تنفيذه بين أقواس `{}`.

- يمكن استخدام عبارات `ElseIf` متعددة للتحقق من شروط متعددة.

- يمكن أن تكون الشروط عبارة عن تعبيرات منطقية معقدة.

شويه امثله تعيشهم مع حالك عشان تبتدي تعمل Script

```

PS C:\Users\Mohamed Amr> $x = (Get-Process).count
PS C:\Users\Mohamed Amr> if ($x -gt 200) {"This is Bigger than for process : $x"} else {"Cool"}
This is Bigger than for process : 221

```

ف ال Script اللي تحت خليته يعمل Check علي notepad اذا كانت شغاله يقفلها ولو مش شغاله يقول لي انها مش شغاله

```

PowerShell 7-preview (x64)
PS C:\Users\Mohamed Amr> notepad.exe .\file.txt
PS C:\Users\Mohamed Amr> if ("notepad" -in ((Get-Process).Name)) {kill ((Get-Process notepad).id)} else {"Notpad Not Running"}
PS C:\Users\Mohamed Amr> if ("notepad" -in ((Get-Process).Name)) {kill ((Get-Process notepad).id)} else {"Notpad Not Running"}
Notpad Not Running
PS C:\Users\Mohamed Amr>

```

## 4.2 Switch Statement

جملة التبديل (Switch) في PowerShell بتسمح لك تقييم تعبير معين ومقارنته بمجموعة من القيم المحددة. لكل قيمة مطابقة، يمكن تنفيذ مجموعة من الأوامر.

مثال توضيحي:

```
$result = switch ( $day )
{
    0 { 'Sunday'      }
    1 { 'Monday'      }
    2 { 'Tuesday'     }
    3 { 'Wednesday'   }
    4 { 'Thursday'    }
    5 { 'Friday'      }
    6 { 'Saturday'    }
}
```

خصائص مهمة:

1. المطابقة التامة يتم التحقق من المطابقة التامة بين التعبير والقيمة.
2. القيم المتعددة يمكن أن تتعامل مع قيم متعددة في جملة Switch.
3. Default تُستخدم لتنفيذ الأوامر إذا لم تتطابق أي قيمة.

مثال ف حاله ان ال Inputs يكون Array

```
$roles = @('WEB','Database')
switch ( $roles ) {
    'Database'    { 'Configure SQL' }
    'WEB'         { 'Configure IIS' }
    'FileServer'  { 'Configure Share' }
}
```

1. استخدام Wildcards:

يمكنك استخدام Wildcards لمطابقة أنماط معينة من النصوص.

```
$number = "abc"
Switch -Wildcard ($number) {
    "a*" { "starts with 'a'" }
    "*b*" { " Secound with 'b'" }
    "*c" { " End with 'c'" }
    Default { "Nothing" }
}
```

```
$message = 'my ssn is 123-23-3456 and credit card: 1234-5678-1234-5678'
switch -regex ($message)
{
    '(?<SSN>\d\d\d-\d\d-\d\d\d\d)'
    {
        Write-Warning "message contains a SSN: $($matches.SSN)"
    }
    '(?<CC>\d\d\d\d-\d\d\d\d-\d\d\d\d-\d\d\d\d)'
    {
        Write-Warning "message contains a credit card number: $($matches.CC)"
    }
    '(?<Phone>\d\d\d-\d\d\d-\d\d\d\d)'
    {
        Write-Warning "message contains a phone number: $($matches.Phone)"
    }
}
```

**Regex of IP :** `\b(?:\d{1,3}\.){3}\d{1,3}\b`

```
$string = "The server IP is 192.168.1.1 and the gateway is 192.168.1.254."
$pattern = "\b(?:\d{1,3}\.){3}\d{1,3}\b"

if ($string -match $pattern) {
    $match = $matches[0]
    Write-Output "Valid IP found: $match"
} else {
    Write-Output "No valid IP found."
}
```

### 3. التعامل مع ملفات:

يمكنك استخدام Switch للتعامل مع محتويات الملفات.

```
$fileContent = Get-Content -Path "example.txt"
Switch ($fileContent) {
    "Error" { "Error found in file" }
    "Warning" { "Warning found in file" }
    Default { "No issues found" }
}
```

الخصائص دي بتسهل التعامل مع Strings في PowerShell باستخدام Switch.

## 5 Loops

### 5.1 For loops

بالأكيد! حلقات التكرار 'for' في PowerShell تُستخدم لتكرار تنفيذ مجموعة من التعليمات لعدد محدد من المرات. وهي مشابهة لحلقات التكرار 'for' في لغات البرمجة الأخرى مثل #C وJavaScript. سأشرح لك بنية حلقة 'for' وكيفية استخدامها مع بعض الأمثلة التوضيحية.

بنية حلقة for في PowerShell

بنية حلقة 'for' في PowerShell تتكون من ثلاثة أجزاء رئيسية: نقطه البدايه، الشرط، والتكرار. يتم تنفيذ هذه الأجزاء على النحو التالي:

```
for (initialization; condition; iteration) {  
    الكود الذي سيتم تنفيذه في كل تكرار  
}
```

- initialization (نقطه البدايه): يتم تنفيذ هذا الجزء مرة واحدة فقط قبل بدء Loop. عادة ما يستخدم لتعريف متغير التحكم في Loop.

- condition (الشرط): يتم تقييم هذا الجزء قبل كل تكرار. إذا كان الشرط صحيحًا ('true\$')، يتم تنفيذ الكود داخل Loop. إذا كان الشرط غير صحيح ('false\$')، تتوقف Loop.

- iteration (التكرار): يتم تنفيذ هذا الجزء بعد كل تكرار. عادة ما يستخدم لتحديث متغير التحكم في Loop.

مثال بسيط على حلقة for

لنبدأ بمثال بسيط يقوم بطباعة الأرقام من 0 إلى 4.

```
for ($i = 0; $i -lt 5; $i++) {  
    Write-Output $i  
}
```

شرح المثال:

1. نقطه البدايه: '\$i = 0'

- يبدأ Variable '\$i' بقيمة 0.

2. الشرط: '\$i -lt 5'

- تستمر Loop طالما أن قيمة '\$i' أقل من 5.

3. التكرار: '\$i++'

- يتم زيادة قيمة '\$i' بمقدار 1 بعد كل تكرار.

مثال آخر: استخدام حلقة for لجمع الأرقام

في هذا المثال، سنقوم باستخدام حلقة 'for' لجمع الأرقام من 1 إلى 10.

```
$total = 0
for ($i = 1; $i -le 10; $i++) {
    $total += $i
}
Write-Output "The total sum is: $total"
```

شرح المثال:

1. نقطه البدايه: `i = 1`
  - يبدأ Variable `i` بقيمة 1.
  2. الشرط: `i -le 10`
  - تستمر Loop طالما أن قيمة `i` أقل من أو تساوي 10.
  3. التكرار: `++i`
  - يتم زيادة قيمة `i` بمقدار 1 بعد كل تكرار.
  4. داخل Loop: `total += \$i`
  - يتم إضافة قيمة `i` إلى Variable `total` في كل تكرار.
  5. خارج Loop: `Write-Output "The total sum is: \$total"`
  - يتم طباعة المجموع الكلي بعد انتهاء Loop.
- مثال معقد: الطباعة الشرطية

في هذا المثال، سنقوم بطباعة الأرقام من 1 إلى 20 ولكننا سنطبع كلمة "Even" للأرقام الزوجية وكلمة "Odd" للأرقام الفردية.

```
for ($i = 1; $i -le 20; $i++) {
    if ($i % 2 -eq 0) {
        Write-Output "$i is Even"
    } else {
        Write-Output "$i is Odd"
    }
}
```

شرح المثال:

1. نقطه البدايه: `i = 1`
- يبدأ Variable `i` بقيمة 1.
2. الشرط: `i -le 20`
- تستمر Loop طالما أن قيمة `i` أقل من أو تساوي 20.

3. التكرار: `++i\$`

- يتم زيادة قيمة `i\$` بمقدار 1 بعد كل تكرار.

4. داخل Loop: استخدام `if` للتحقق إذا كان الرقم زوجياً أم فردياً باستخدام عامل باقي القسمة `%`.

- إذا كانت نتيجة `i % 2` تساوي 0، فإن الرقم زوجي.

- خلاف ذلك، فإن الرقم فردي.

ملاحظات:

- تأكد من وضع الكود الذي سيتم تنفيذه داخل أقواس `{}`.

- يمكن استخدام أي تعبير منطقي كشرط للحلقة.

- تأكد من أن التكرار يتم بشكل صحيح لتجنب Loops اللانهائية.

## 5.2 While Loop

Loop `while` تستمر في تنفيذ مجموعة من التعليمات طالما أن الشرط المحدد صحيح (`true\$`). يتم تقييم الشرط قبل كل تكرار، وإذا كان الشرط غير صحيح (`false\$`)، تتوقف Loop.

مثال بسيط

لنأخذ مثلاً بسيطاً لطباعة الأرقام من 1 إلى 5 باستخدام حلقة `while`:

```
# تعريف المتغير
$i = 1
# حلقة while
while ($i -le 5) {
    Write-Output $i
    $i++
}
```

شرح المثال:

1. تعريف المتغير:

- يبدأ المتغير `i\$` بقيمة 1.

2. الشرط: `i -le 5`

- تستمر Loop طالما أن قيمة `i\$` أقل من أو تساوي 5.

3. داخل Loop:

- في كل تكرار، يتم طباعة قيمة `i\$`.

- يتم زيادة قيمة `i\$` بمقدار 1 بعد كل تكرار (`++i\$`).



مثال آخر: قراءة المستخدم

في هذا المثال، سنستخدم حلقة `while` لقراءة إدخال المستخدم حتى يكتب "exit":

```
# تعريف متغير الإدخال
$userInput = ""
# حلقة while
while ($userInput -ne "exit") {
    $userInput = Read-Host "Enter a command (type 'exit' to quit)"
    Write-Output "You entered: $userInput"
}
Write-Output "Goodbye!"
```

شرح المثال:

1. تعريف متغير الإدخال:

- يبدأ المتغير `\$userInput` بسلسلة فارغة.

2. الشرط: `\$userInput -ne "exit"`

- تستمر Loop طالما أن المستخدم لم يكتب "exit".

3. داخل Loop:

- يتم قراءة إدخال المستخدم باستخدام `Read-Host` وتخزينه في `\$userInput`.

- يتم طباعة الإدخال الذي أدخله المستخدم.

4. خارج Loop:

- عند انتهاء Loop (عندما يكتب المستخدم "exit")، يتم طباعة رسالة وداع.

ملاحظات:

- Loops اللانهائية: تأكد من أن الشرط سيتم تحقيقه في مرحلة ما، وإلا قد تقع في حلقة لانهائية. في المثال الأول، يتم زيادة `i` في كل تكرار، لذا سينتهي الشرط في النهاية. في المثال الثاني، يمكن للمستخدم إنهاء Loop بكتابة "exit".

- تعديل الشرط: يمكنك استخدام أي تعبير منطقي كشرط للحلقة. على سبيل المثال، يمكنك استخدام مقارنة القيم، التحقق من الحالة، أو أي تعبير آخر يعتمد على متغيرات وظروف معينة.

## 6 Example For Some Scripting

### 6.1 Example: Remove Files In Folder

سنقوم بكتابة سكريبت يتحقق من وجود ملفات في مجلد معين وحذف الملفات التي تكون أقدم من عدد معين من الأيام.

```
# تحديد مسار المجلد
$folderPath = "C:\path\to\your\folder"
# تحديد عدد الأيام
$days = 30
# الحصول على تاريخ اليوم مطروحًا منه عدد الأيام المحددة
$cutoffDate = (Get-Date).AddDays(-$days)
# الحصول على جميع الملفات في المجلد
$files = Get-ChildItem -Path $folderPath -File
# التحقق من كل ملف وحذفه إذا كان أقدم من تاريخ القطع
foreach ($file in $files) {
    if ($file.LastWriteTime -lt $cutoffDate) {
        Write-Output "Deleting file: $($file.FullName)"
        Remove-Item -Path $file.FullName -Force
    }
}
```

شرح السكريبت:

1. تحديد مسار المجلد:

- قم بتحديد مسار المجلد الذي تريد فحصه وتحديثه في Variable '\$folderPath'.

2. تحديد عدد الأيام:

- حدد عدد الأيام الذي يعتبر الحد الأقصى لعمر الملفات في Variable '\$days'.

3. حساب تاريخ القطع:

- استخدم 'Get-Date' للحصول على تاريخ اليوم، ثم استخدم 'AddDays(-\$days)' لحساب تاريخ القطع (أي الملفات التي تكون أقدم من هذا التاريخ سيتم حذفها).

4. الحصول على جميع الملفات في المجلد:

- استخدم 'Get-ChildItem' للحصول على جميع الملفات في المجلد المحدد.

5. التحقق من كل ملف وحذفه إذا كان أقدم من تاريخ القطع:

- استخدم حلقة 'foreach' للمرور على كل ملف.

- استخدم شرط 'if' للتحقق من أن تاريخ التعديل الأخير للملف ('LastWriteTime') أقدم من تاريخ القطع ('\$cutoffDate').

- إذا كان الشرط صحيحًا، قم بحذف الملف باستخدام 'Remove-Item'.

بهذه الطريقة، يمكنك بسهولة حذف الملفات القديمة في مجلد معين باستخدام PowerShell. إذا كان لديك أي أسئلة إضافية أو تحتاج إلى تعديل السكريبت ليتناسب مع احتياجاتك، فلا تتردد في طرحها!

## 6.2 Example: Check the status of services and start stopped ones

المشكلة: التحقق من حالة الخدمات وتشغيل المتوقفة منها. المطلوب هو كتابة سكريبت يتحقق من حالة مجموعة من الخدمات المحددة ويقوم بتشغيلها إذا كانت متوقفة.

السكربت:

#تحديد قائمة الخدمات التي نريد التحقق منها

```
$servicesToCheck = @(
    "wuauserv",    # Windows Update
    "BITS",        # Background Intelligent Transfer Service
    "Spooler"      # Print Spooler
)
# المرور على كل خدمة في القائمة
foreach ($serviceName in $servicesToCheck) {
    # الحصول على حالة الخدمة
    $service = Get-Service -Name $serviceName
    # التحقق من حالة الخدمة
    if ($service.Status -eq 'Stopped') {
        Write-Output "Starting service: $serviceName"
        Start-Service -Name $serviceName
    } else {
        Write-Output "Service $serviceName is already running."
    }
}
```

شرح السكريبت:

1. تحديد قائمة الخدمات:

- قم بتحديد قائمة بالخدمات التي تريد التحقق من حالتها في Variable '\$servicesToCheck'.

2. المرور على كل خدمة في القائمة:

- استخدم حلقة 'foreach' للمرور على كل خدمة في القائمة.

3. الحصول على حالة الخدمة:

- استخدم 'Get-Service' للحصول على حالة الخدمة الحالية.

4. التحقق من حالة الخدمة:

- استخدم شرط `if` للتحقق مما إذا كانت حالة الخدمة `Stopped`.

5. تشغيل الخدمة إذا كانت متوقفة:

- إذا كانت الخدمة متوقفة، قم بتشغيلها باستخدام `Start-Service`.

- إذا كانت الخدمة تعمل بالفعل، قم بطباعة رسالة توضح أن الخدمة قيد التشغيل.

تشغيل السكريبت:

1. افتح PowerShell.

2. انسخ السكريبت والصقه في PowerShell.

3. اضغط Enter لتنفيذ السكريبت.

ملاحظات:

- تأكد من تشغيل PowerShell بامتيازات المسؤول (Run as Administrator) إذا كانت الخدمات تتطلب ذلك.

- يمكنك تعديل قائمة الخدمات في Variable `servicesToCheck\$` لتشمل الخدمات التي تريد التحقق منها وتشغيلها إذا كانت متوقفة.

إذا كان لديك أي أسئلة إضافية أو تحتاج إلى تعديل السكريبت ليتناسب مع احتياجاتك، فلا تتردد في طرحها!

### 6.3 Example: Ping in All of Network

بالتأكيد! سنقوم بكتابة سكريبت PowerShell يقوم بعمل Ping على جميع العناوين في الشبكة 192.168.1.1/24 هذا يعني أننا سنقوم بعمل Ping على العناوين من 192.168.1.1 إلى 192.168.1.254.

السكريبت:

```
# تحديد الشبكة الأساسية
$baseIP = "192.168.1."
# إنشاء قائمة لتخزين نتائج Ping
$pingResults = @()
# التكرار على العناوين من 1 إلى 254
for ($i = 1; $i -le 254; $i++) {
    # الحالي IP تكوين عنوان
    $currentIP = $baseIP + $i
    # الحالي IP على عنوان Ping إجراء عملية
    $pingResult = Test-Connection -ComputerName $currentIP -
Count 1 -Quiet
    # إلى القائمة Ping إضافة نتيجة
    $pingResults += [PSCustomObject]@{
        IPAddress = $currentIP
        Status     = if ($pingResult)
```

```

        { "Reachable" }
    else
        { "Unreachable" }
    }
}

```

# عرض النتائج

```
$pingResults | Format-Table -AutoSize
```

شرح السكريبت:

1. تحديد الشبكة الأساسية:

- قمنا بتحديد الجزء الأساسي من عنوان الشبكة في Variable `baseIP\$`.

2. إنشاء قائمة لتخزين نتائج Ping:

- قمنا بإنشاء قائمة فارغة `pingResults\$` لتخزين نتائج عملية الـ Ping.

3. التكرار على العناوين من 1 إلى 254:

- استخدمنا حلقة `for` للتكرار على جميع العناوين في الشبكة من 1 إلى 254.

4. تكوين عنوان IP الحالي:

- في كل تكرار، نقوم بتكوين عنوان IP الحالي عن طريق دمج الجزء الأساسي مع قيمة Variable `i\$`.

5. إجراء عملية Ping على عنوان IP الحالي:

- استخدمنا `Test-Connection` لإجراء عملية Ping على عنوان IP الحالي. الخيار `Quiet-` يجعل `Test-Connection` يعيد قيمة منطقية (`true\$` إذا كان العنوان يمكن الوصول إليه و `false\$` إذا لم يكن يمكن الوصول إليه).

6. إضافة نتيجة Ping إلى القائمة:

- قمنا بإضافة نتيجة عملية الـ Ping إلى القائمة `pingResults\$` كـ Object مخصص يحتوي على عنوان IP الحالي وحالته (إما "Reachable" أو "Unreachable").

7. عرض النتائج:

- استخدمنا `Format-Table` لعرض النتائج في جدول منظم.

بهذا السكريبت، يمكنك بسهولة عمل Ping على جميع العناوين في الشبكة 192.168.1.1/24 وعرض حالة الوصول لكل عنوان IP. إذا كان لديك أي أسئلة إضافية أو تحتاج إلى مزيد من الشرح، فلا تتردد في طرحها!

## 6.4 Example: To solve the file transfer problem

لحل مشكلة نقل الملفات التي تحتوي على كلمة معينة في اسمها من مجلد إلى آخر:

```
# تحديد مسار المجلد المصدر والمجلد الوجهة
$sourceFolderPath = "C:\SourceFolder"
$destinationFolderPath = "C:\DestinationFolder"
# تحديد الكلمة المراد البحث عنها في أسماء الملفات
$keyword = "example"
# الحصول على قائمة الملفات التي تحتوي على الكلمة المحددة في
أسمائها
$filesToMove = Get-ChildItem -Path $sourceFolderPath -File |
Where-Object {
    $_.Name -like "$keyword*"
}
# التحقق مما إذا كانت هناك ملفات لنقلها
if ($filesToMove.Count -gt 0) {
    # نقل الملفات إلى المجلد الوجهة
    foreach ($file in $filesToMove) {
        $destinationPath = Join-Path -Path
$destinationFolderPath -ChildPath $file.Name
        Move-Item -Path $file.FullName -Destination
$destinationPath -Force
    }
    Write-Output "Files containing the word '$keyword' have been
moved to the destination folder."
} else {
    Write-Output "There are no files with '$keyword' in their
names to be moved."
}
```

هذا السكريبت يقوم بالتالي:

1. تحديد مسار المجلد المصدر والمجلد الوجهة.
2. تحديد الكلمة التي سيتم البحث عنها في أسماء الملفات.
3. الحصول على قائمة الملفات في المجلد المصدر التي تحتوي على الكلمة المحددة في أسمائها.
4. التحقق مما إذا كانت هناك ملفات لنقلها.
5. إذا كانت هناك ملفات تحتوي على الكلمة المحددة، يتم نقلها إلى المجلد الوجهة باستخدام الأمر 'Move-Item'.
6. عرض رسالة توضح ما إذا تم نقل الملفات التي تحتوي على الكلمة المحددة أو إذا لم توجد ملفات ليتم نقلها.

## 6.5 Example: Send an email alert

لحل مشكلة إرسال تنبيه عبر البريد الإلكتروني عند انخفاض المساحة الحرة على القرص الصلب تحت حد معين:

```
# تحديد المسار المنطقي للقرص
$driveLetter = "C:"
# تحديد الحد الأدنى للمساحة الحرة بالميغابايت
$minFreeSpaceMB = 5000
# تحديد معلومات البريد الإلكتروني
$smtpServer = "smtp.example.com"
$smtpFrom = "alert@example.com"
$smtpTo = "admin@example.com"
$smtpSubject = "Lower space in driver : $driveLetter"
# الحصول على معلومات القرص
$drive = Get-PSDrive -Name $driveLetter
# التحقق من المساحة الحرة
if ($drive.Used -ne $null) {
    $freeSpaceMB = $drive.Free / 1MB
    if ($freeSpaceMB -lt $minFreeSpaceMB) {
        # إعداد نص البريد الإلكتروني
        $smtpBody = "Disk $driveLetter less than
$minFreeSpaceMB png. Current free space: $freeSpaceMB png."
        # إرسال البريد الإلكتروني
        Send-MailMessage -SmtpServer $smtpServer -From $smtpFrom
-To $smtpTo -Subject $smtpSubject -Body $smtpBody
        Write-Output "Email alert sent."
    } else {
        Write-Output "$drive Letter Enough free disk space:
$freeSpaceMB MB."
    }
} else {
    Write-Output "Disk $driveLetter not found."
}
```

هذا السكريبت يقوم بالتالي:

1. تحديد المسار المنطقي للقرص (مثل "C:").
2. تحديد الحد الأدنى للمساحة الحرة بالميغابايت الذي سيؤدي إلى إرسال التنبيه.
3. إعداد معلومات البريد الإلكتروني (خادم SMTP، المرسل، المستلم، وموضوع البريد الإلكتروني).

4. الحصول على معلومات القرص باستخدام الأمر 'Get-PSDrive'.
5. التحقق مما إذا كانت المساحة الحرة أقل من الحد الأدنى المحدد.
6. إذا كانت المساحة الحرة أقل من الحد الأدنى، يتم إعداد نص البريد الإلكتروني وإرساله باستخدام الأمر 'Send-MailMessage'.
7. عرض رسالة توضح ما إذا تم إرسال التنبيه أو إذا كانت المساحة الحرة كافية.

## 6.6 Example of DeepWhite

DeepWhite ده إطار عمل (Framework) لـ PowerShell بيستخدم علشان بيعت هاشات SHA256 لـ VirusTotal. علشان تعمل كده، لازم يكون عندك مفتاح API من VirusTotal، المفتاح الشخصي (اللي مجاني) ممكن بيعت أربع استفسارات في الدقيقة.

DeepWhite بيقدر يجمع هاشات SHA256 من أحداث معينة في Sysmon زي إنشاء العمليات (process creation)، تحميل الدرايفرز (Driver loads)، وتحميل الصور (Image/DLL loads). DLL. كمان ممكن بيعت قائمة من هاشات SHA256 من ملف.

وبيجي معاه دعم للقائمة البيضاء (whitelist)، اللي ممكن تتعمل مباشرة من PowerShell عن طريق الأمر ده:

```
Get-ChildItem c:\windows\system32 -Include
'*.*.exe','*.*.dll','*.*.sys','*.*.com' -Recurse | Get-FileHash |
Export-Csv -Path whitelist.csv
```

الجزء ده بيقولك ازاى تقدر تعمل قائمة بيضاء لكل الملفات التنفيذية والديناميكية في مجلد system32 وتحفظها في ملف CSV.

## 6.7 Get-WinEvent for Example

```
Get-WinEvent -LogName security | Group-Object id -NoElement |
sort count
```

تقدر تستخدم PowerShell علشان تجمع معلومات من security log في ويندوز.

أمر "Get-WinEvent -LogName security" بيعني اجلب Events من Security Logs. بعدين "Group-Object id -NoElement" بيقول: افرز Events دي حسب الـ id بتاع كل Event، ومش تعرضلي عددهم.

"sort count" بيعني افرز النتيجة حسب عدد الأحداث من أكثر لأقل.

تقدر تجرب نفس الأمر على جهازك الخاص براحتك، بس لازم تكون مفتوح PowerShell كـ administrator علشان تقدر توصل للسجل الأمني بشكل صحيح.

```
Get-WinEvent @{"Logname="Security"; ID=4688}
```

في PowerShell يقوم بجلب Events من (Security Log) حيث يكون رقم الـ ID للحدث هو 4688. هذا الـ ID يشير إلى حدث "New Creation Process" في النظام.

## 6.8 Remote Registry Service

هذا السكريبت يستخدم PowerShell كغلاف لاستخدام خدمة التسجيل البعيد (remote registry service) لجمع مفاتيح التشغيل في التسجيل (registry run keys). يمكن استخدام هذا السكريبت عبر الشبكة بدون الحاجة إلى PowerShell Remoting. هنا هو السكريبت:



```

$user = "starbuck"
$password = "cyl0n"
$array = @("192.168.1.1", "192.168.1.2")
foreach ($ip in $array) {
    # لإنشاء اتصال مؤقت مع الجهاز البعيد net use لتشغيل أمر
    net use \\$ip $password /u:$user | out-null # تجاهل الإخراج
    # طباعة عنوان IP
    $ip
    # جمع مفاتيح التشغيل في التسجيل
    reg query
    \\$ip\HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
    reg query
    \\$ip\HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce
    reg query
    \\$ip\HKU\DEFAULT\Software\Microsoft\Windows\CurrentVersion\Run
    # تجاهل أي أخطاء لعدم وجود هذه المفاتيح على أنظمة 32 بت
    reg query
    \\$ip\HKLM\SOFTWARE\Wow6432node\Microsoft\Windows\CurrentVersion
    \Run 2> $null
    reg query
    \\$ip\HKLM\SOFTWARE\Wow6432node\Microsoft\Windows\CurrentVersion
    \RunOnce 2> $null
    # إلغاء اتصال الشبكة المؤقت
    net use \\$ip /delete
}

```

1. Variables `user\$` و `password\$` تحتوي على اسم المستخدم وكلمة المرور المستخدمة للاتصال بجهاز الوجهة.
  2. Variable `array\$` يحتوي على قائمة بعناوين IP للأجهزة البعيدة التي يجب جمع مفاتيح التشغيل في التسجيل منها.
  3. في كل دورة من Loop `foreach`، يتم الاتصال بالجهاز البعيد باستخدام أمر `net use` لإنشاء اتصال مؤقت.
  4. يتم طباعة عنوان الـ IP للجهاز الحالي.
  5. يتم استعلام مفاتيح التشغيل في التسجيل باستخدام أوامر `reg query`، مع التركيز على الفروق بين الأنظمة 32 بت و 64 بت باستخدام `Wow6432node`.
  6. أي أخطاء تحدث بسبب عدم وجود المفاتيح على أنظمة 32 بت يتم تجاهلها باستخدام `<2` \$null.
  7. في النهاية، يتم إلغاء الاتصال الشبكي المؤقت بالجهاز البعيد باستخدام `net use /delete`.
- هذا السكريبت يستخدم الـ PowerShell لتسهيل جمع معلومات من التسجيل عبر الشبكة، وهو يتطلب صلاحيات مسؤول على الأقل لتشغيل أوامر الاتصال والاستعلام عن التسجيل على الأجهزة البعيدة.

[TryHackMe | Hacking with PowerShell](#)

[Century – UTW \(underthewire.tech\)](#)

تقدر تطلع علي دول

[samratashok/nishang: Nishang - Offensive PowerShell for red team, penetration testing and offensive security. \(github.com\)](#)

[EmpireProject/Empire: Empire is a PowerShell and Python post-exploitation agent. \(github.com\)](#)

[How to bypass AMSI and execute ANY malicious Powershell code | zc00l blog \(0x00-0x00.github.io\)](#)

[Malandrone/PowerDecode: PowerDecode is a PowerShell-based tool that allows to deobfuscate PowerShell scripts obfuscated across multiple layers. The tool performs code dynamic analysis, extracting malware hosting URLs and checking http response. It can also detect if the malware attempts to inject shellcode into memory. \(github.com\)](#)