10 TIPS

TO RUN SCRIPTS FAST & SECURE IN POWERSHELL

WHOAMI ?

- Bruno Buyck (37) (Belgium)
  - Owner @Trouble Shooter BV
    - Using PowerShell since 2007
    - Microsoft Certified Trainer since 2018
    - Azure Solution Architect
    - Script*Runner* partner since 2020

Bruno@powershell.wtf

https://www.linkedin.com/in/brunobuyck
https://www.powershell.wtf

Microsoft CERTIFIED AZURE FUNDAMENTALS

Microsoft CERTIFIED AZURE ADMINISTRATOR ASSOCIATE

Microsoft CERTIFIED AZURE SOLUTIONS ARCHITECT EXPERT

Microsoft 365 CERTIFIED TEAMS ADMINISTRATOR ASSOCIATE

Microsoft 365 CERTIFIED MESSAGING ADMINISTRATOR ASSOCIATE

Script*Runner*® The Campus Powershell .WTF

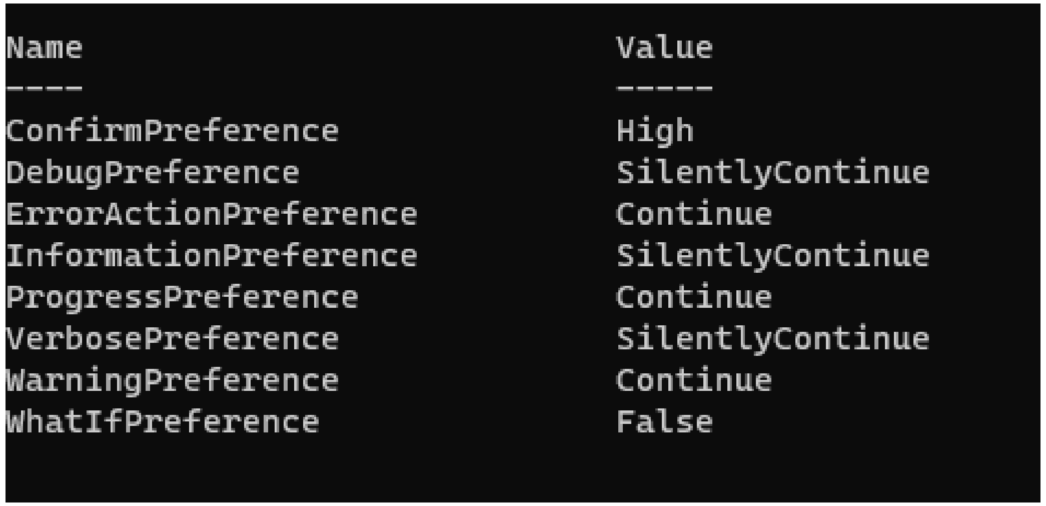
TOPICS

- General
  - Preference variables
  - Debugging
  - Parameter Validation
- Performance
  - Pipelining
  - Hash tables
  - Loops
- Security
  - Working with secrets
  - Protecting scripts
  - Logging

PREFERENCE VARIABLES

PREFERENCE VARIABLES

- Customize the behavior of PowerShell
- Changing them may have severe impact !

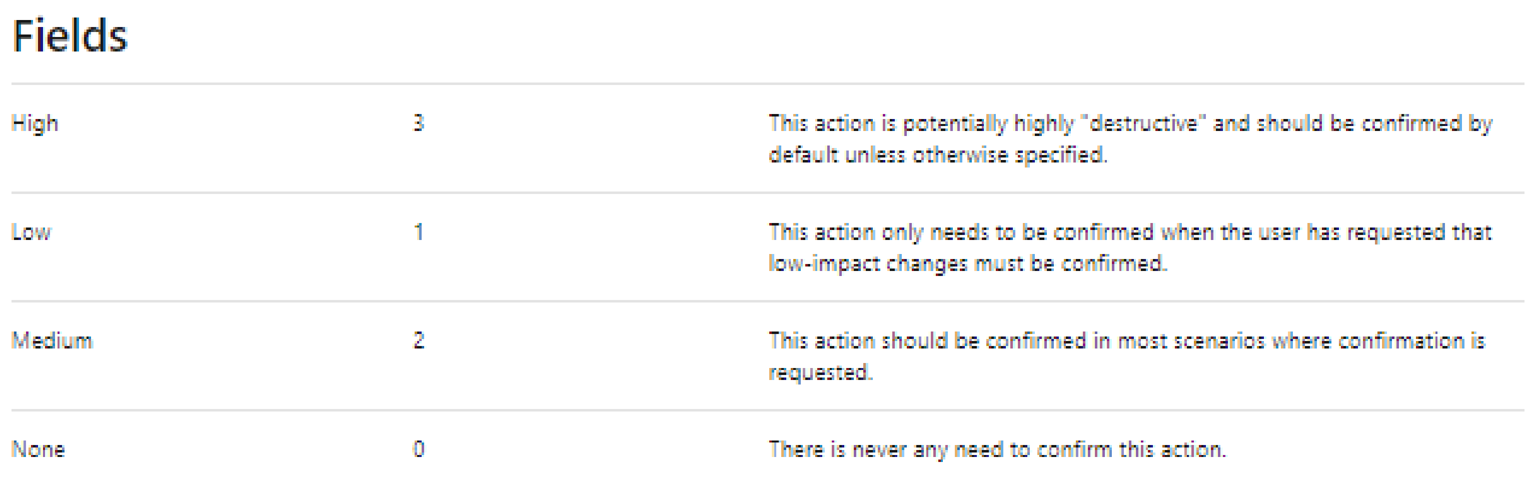


```
Name                           Value
----                           -----
ConfirmPreference              High
DebugPreference                SilentlyContinue
ErrorActionPreference          Continue
InformationPreference          SilentlyContinue
ProgressPreference             Continue
VerbosePreference              SilentlyContinue
WarningPreference              Continue
WhatIfPreference               False
```

INDY/TECH
MAN ACCIDENTALLY 'DELETES HIS ENTIRE COMPANY' WITH ONE LINE OF BAD CODE

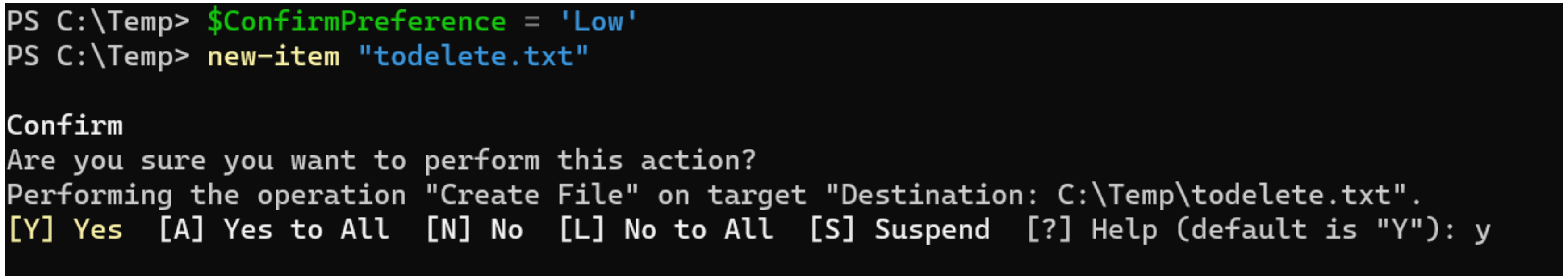
$ConfirmPreference

- Level of "Impact" :



Fields

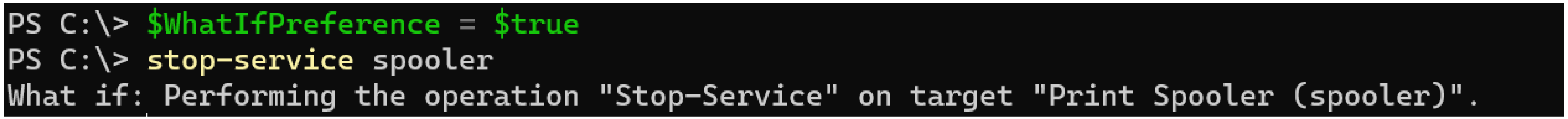
High	3	This action is potentially highly "destructive" and should be confirmed by default unless otherwise specified.
Low	1	This action only needs to be confirmed when the user has requested that low-impact changes must be confirmed.
Medium	2	This action should be confirmed in most scenarios where confirmation is requested.
None	0	There is never any need to confirm this action.



```
PS C:\Temp> $ConfirmPreference = 'Low'
PS C:\Temp> new-item "todelete.txt"

Confirm
Are you sure you want to perform this action?
Performing the operation "Create File" on target "Destination: C:\Temp\todelete.txt".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
```

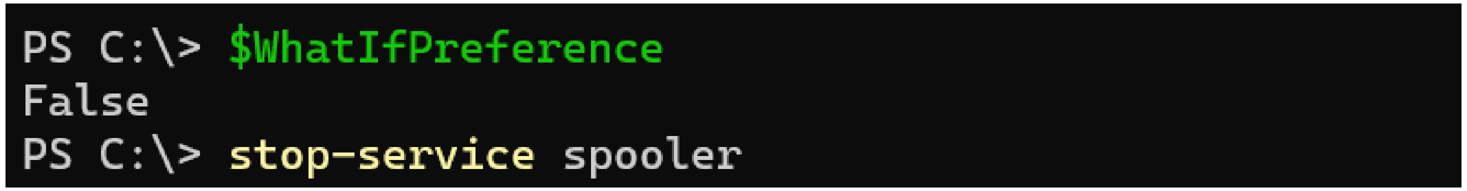
$Whatifpreference

- -Whatif switch on cmdlet simulates the action but doesn’t apply any changes !!!!
- $Whatifpreference = $true



```
PS C:\> $WhatIfPreference = $true
PS C:\> stop-service spooler
What if: Performing the operation "Stop-Service" on target "Print Spooler (spooler)".
```

- $Whatifpreference = $false



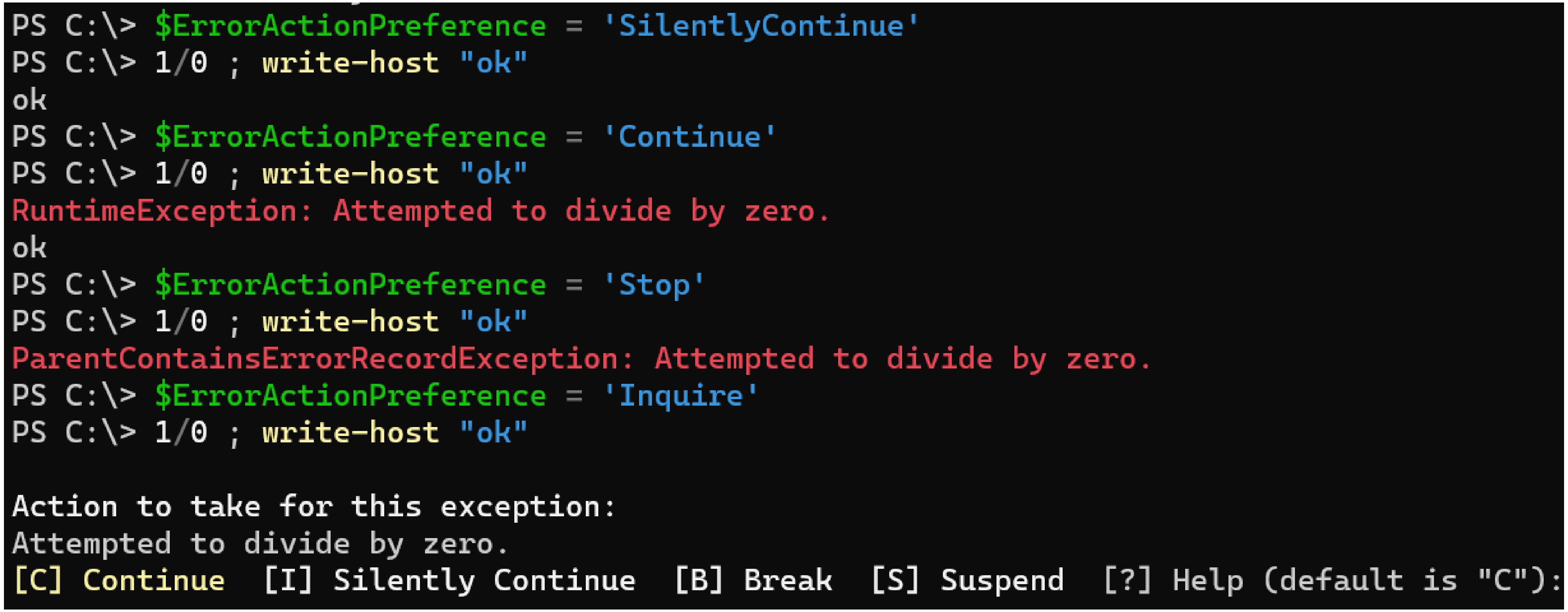
```
PS C:\> $WhatIfPreference
False
PS C:\> stop-service spooler
```

$ErrorActionPreference

- Determines how PowerShell responds to a non-terminating error (an error that does not stop the cmdlet processing)

- $ErrorActionPreference
    - Stop: Displays the error message and stops executing.
    - Inquire: Displays the error message and asks you whether you want to continue.
    - Continue: Displays the error message and continues (Default) executing.
    - SilentlyContinue: No effect (=On Error Resume Next)

$ErrorActionPreference



```
PS C:\> $ErrorActionPreference = 'SilentlyContinue'
PS C:\> 1/0 ; write-host "ok"
ok
PS C:\> $ErrorActionPreference = 'Continue'
PS C:\> 1/0 ; write-host "ok"
RuntimeException: Attempted to divide by zero.
ok
PS C:\> $ErrorActionPreference = 'Stop'
PS C:\> 1/0 ; write-host "ok"
ParentContainsErrorRecordException: Attempted to divide by zero.
PS C:\> $ErrorActionPreference = 'Inquire'
PS C:\> 1/0 ; write-host "ok"

Action to take for this exception:
Attempted to divide by zero.
[C] Continue  [I] Silently Continue  [B] Break  [S] Suspend  [?] Help (default is "C"):
```

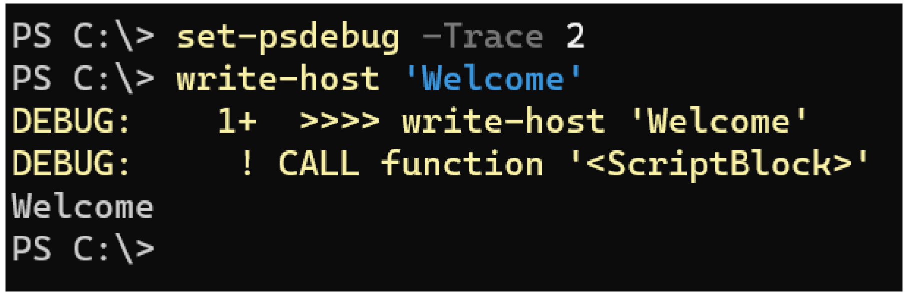
DEBUGGING SCRIPTS

in the console

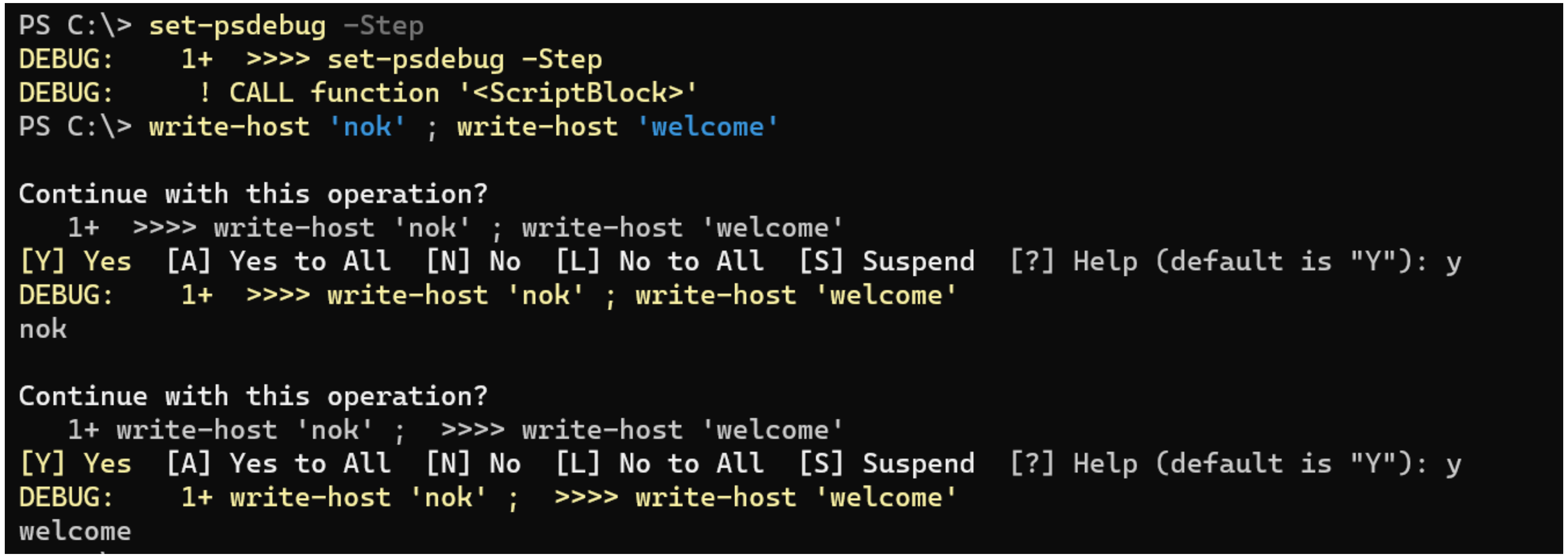
SET-PSDEBUG

- Debugging :
  - Set-PSDebug
    - **-Trace**
      - 0 – tracing off
      - 1 – Trace script lines as they are executed
      - 2 – Trace script lines, variable assignments, function calls, and scripts.
    - **-Step**
      - Turns on script stepping.
      - Before each line is run, the user is prompted to stop, continue, or enter a new interpreter level to inspect the state of the script.

SET-PSDEBUG



```
PS C:\> set-psdebug -Trace 2
PS C:\> write-host 'Welcome'
DEBUG:    1+  >>>> write-host 'Welcome'
DEBUG:     ! CALL function '<ScriptBlock>'
Welcome
PS C:\>
```



```
PS C:\> set-psdebug -Step
DEBUG:    1+  >>>> set-psdebug -Step
DEBUG:     ! CALL function '<ScriptBlock>'
PS C:\> write-host 'nok' ; write-host 'welcome'

Continue with this operation?
   1+  >>>> write-host 'nok' ; write-host 'welcome'
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
DEBUG:    1+  >>>> write-host 'nok' ; write-host 'welcome'
nok

Continue with this operation?
   1+ write-host 'nok' ;  >>>> write-host 'welcome'
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
DEBUG:    1+ write-host 'nok' ;  >>>> write-host 'welcome'
welcome
```

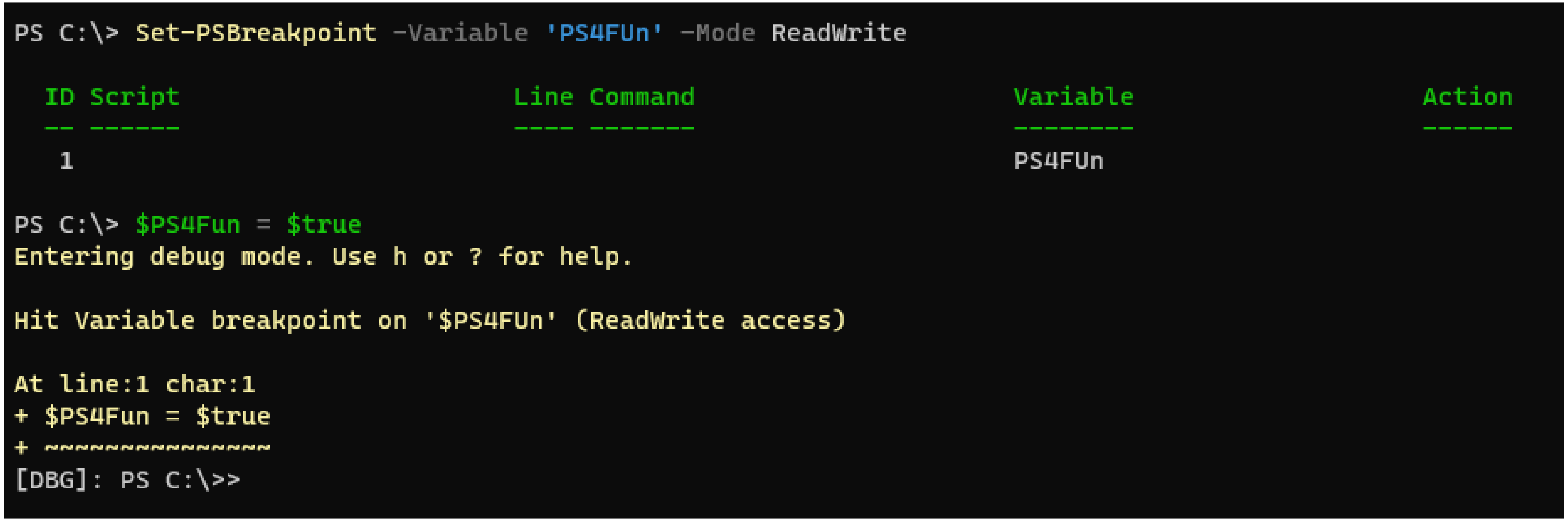
SET-PSBREAKPOINT

- Set-PsBreakPoint

- The Set-PSBreakpoint cmdlet sets a breakpoint in a script or in any command run in the current session.
  You can use Set-PSBreakpoint to set a breakpoint before executing a script or running a command, or during debugging, when stopped at another breakpoint.

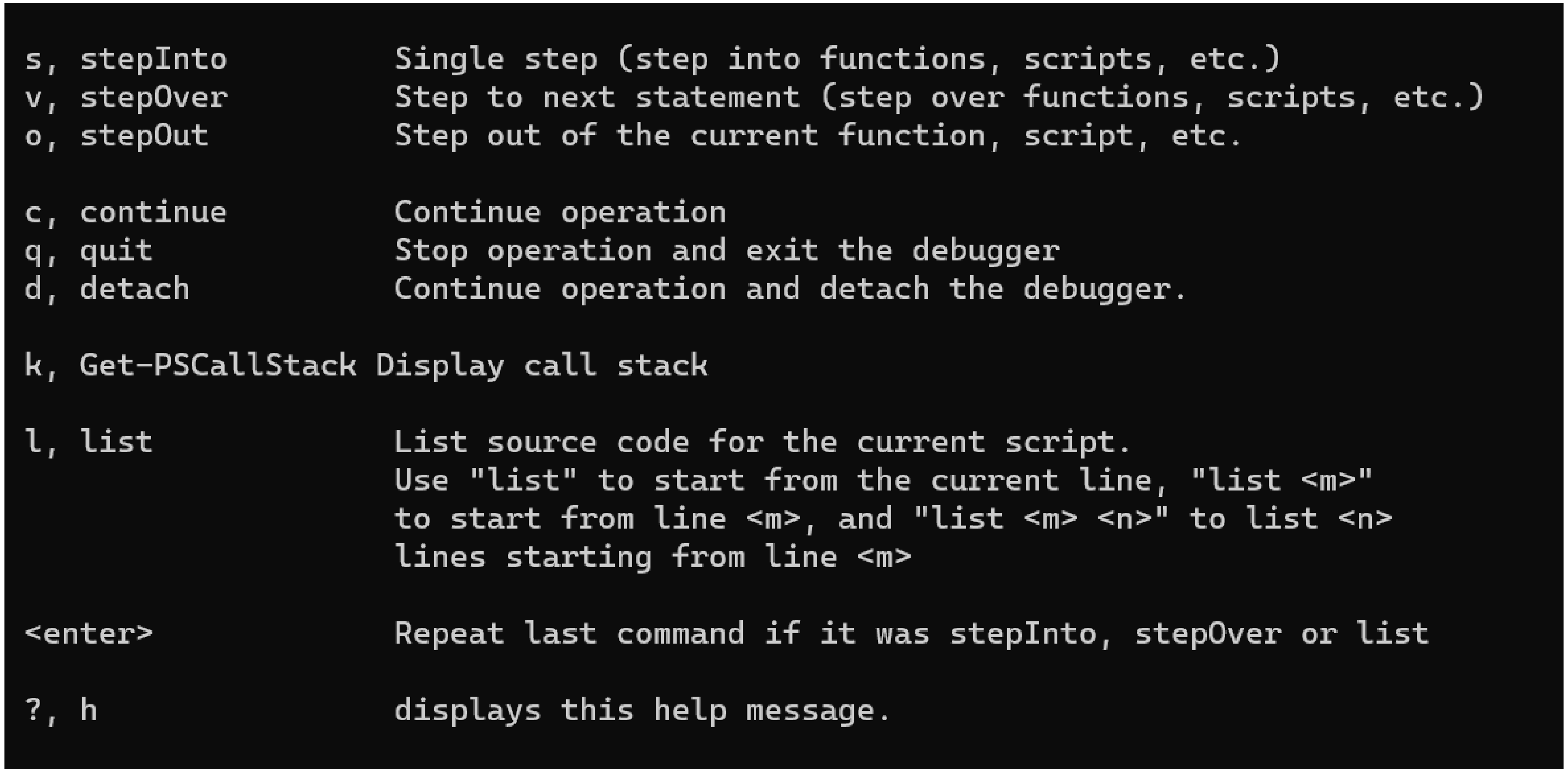
```
Set-PSBreakpoint -Command “remove-*"

Set-PSBreakpoint  -Variable “PS4Fun" -Mode ReadWrite
```

SET-PSBREAKPOINT



```
PS C:\> Set-PSBreakpoint -Variable 'PS4FUn' -Mode ReadWrite

  ID Script                       Line Command                    Variable                 Action
  -- ------                       ---- -------                    --------                 ------
   1                                                              PS4FUn

PS C:\> $PS4Fun = $true
Entering debug mode. Use h or ? for help.

Hit Variable breakpoint on '$PS4FUn' (ReadWrite access)

At line:1 char:1
+ $PS4Fun = $true
+ ~~~~~~~~~~~~~~~
[DBG]: PS C:\>>
```

SET-PSBREAKPOINT



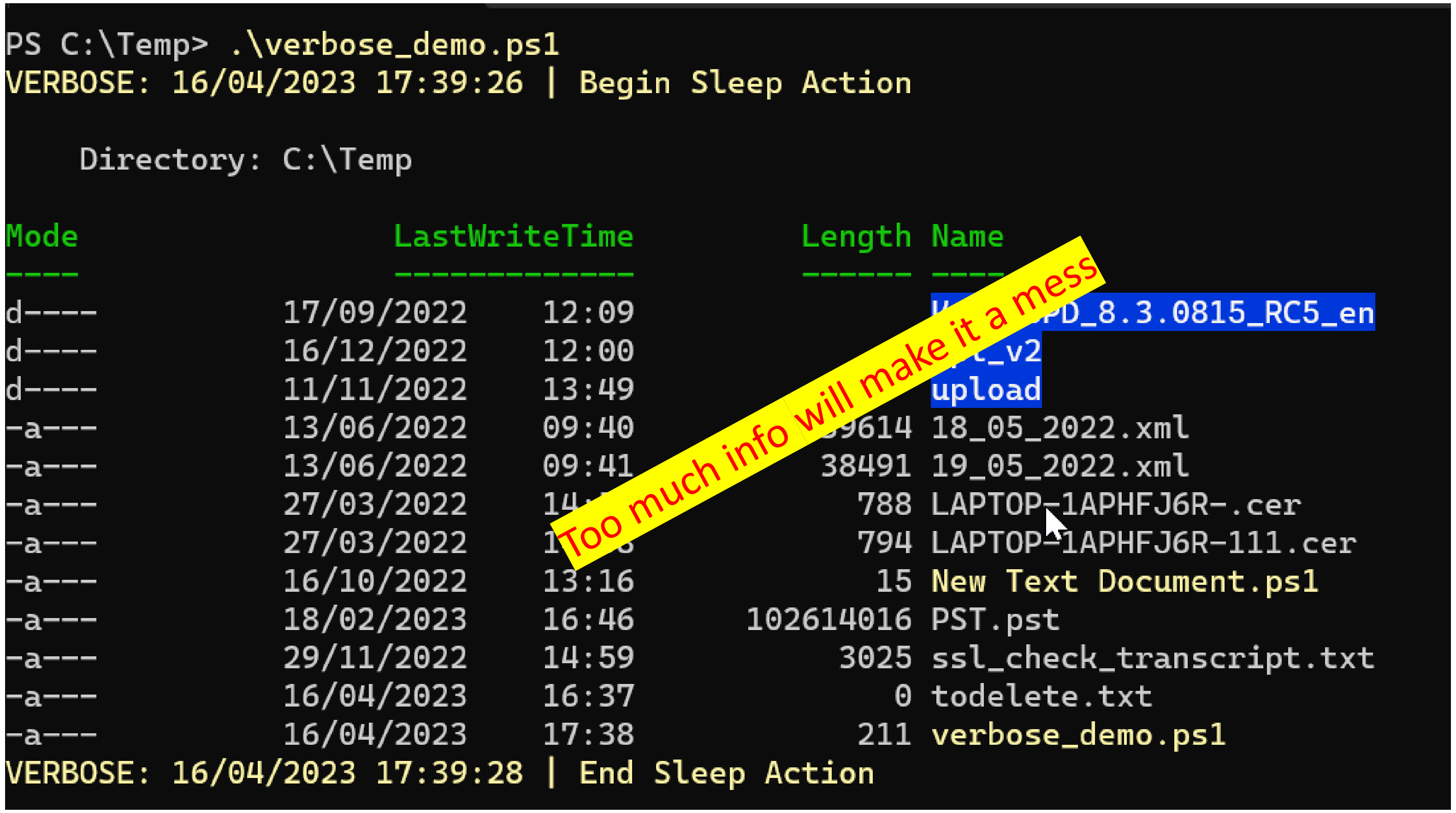
```
s, stepInto         Single step (step into functions, scripts, etc.)
v, stepOver         Step to next statement (step over functions, scripts, etc.)
o, stepOut          Step out of the current function, script, etc.

c, continue         Continue operation
q, quit             Stop operation and exit the debugger
d, detach           Continue operation and detach the debugger.

k, Get-PSCallStack Display call stack

l, list             List source code for the current script.
                    Use "list" to start from the current line, "list <m>"
                    to start from line <m>, and "list <m> <n>" to list <n>
                    lines starting from line <m>

<enter>             Repeat last command if it was stepInto, stepOver or list

?, h                displays this help message.
```

WRITE-VERBOSE

- Write-Verbose cmdlet writes text to PowerShell.
- It is used to deliver information about command processing.
- $Verbosepreference :
    - Stop: Displays the verbose message and stops executing.
    - Inquire: Displays the verbose message and asks whether you want to continue.
    - Continue: Displays the verbose message and continues executing.
    - SilentlyContinue (default): No effect.
- Provide diagnostic output with write-verbose !

WRITE-VERBOSE

```
write-verbose ("{0} | Begin Sleep Action" -f (get-date -Format "dd/MM/yyyy HH:mm:ss"))
start-sleep -seconds 2
ls c:\temp\
write-verbose ("{0} | End Sleep Action"   -f (get-date -Format "dd/MM/yyyy HH:mm:ss"))
```

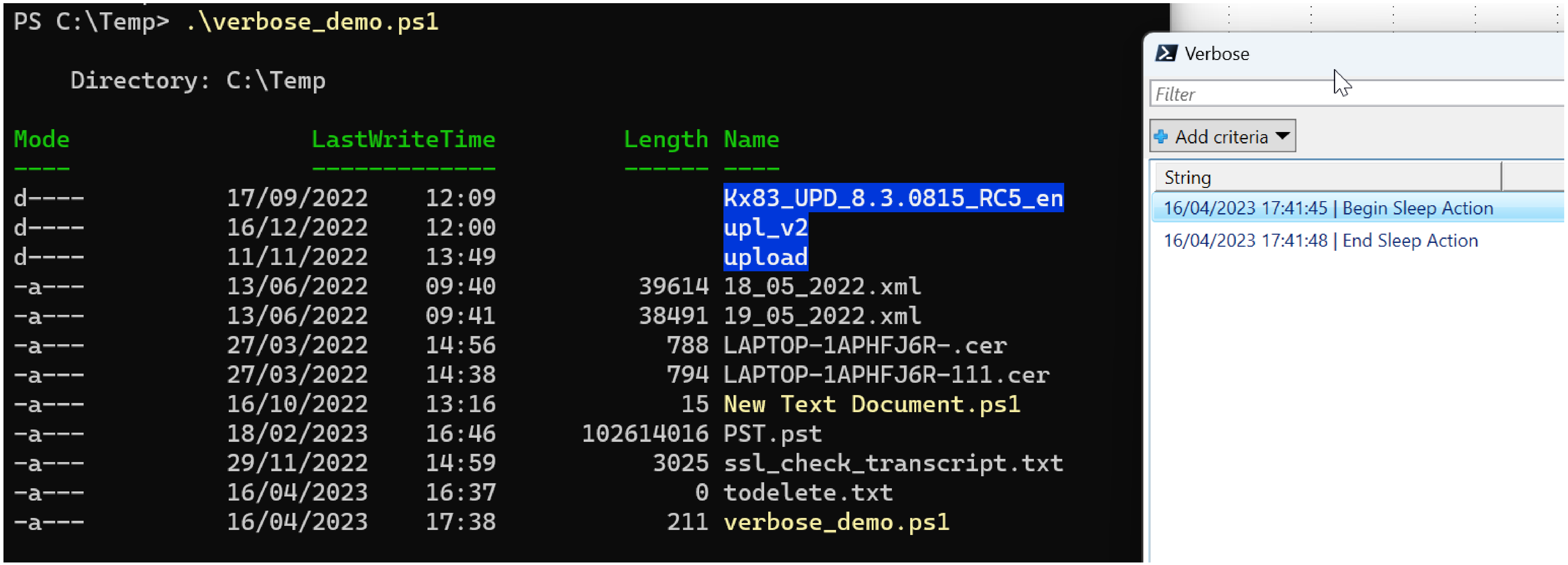


```
PS C:\Temp> .\verbose_demo.ps1
VERBOSE: 16/04/2023 17:39:26 | Begin Sleep Action

    Directory: C:\Temp

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          17/09/2022     12:09                [illegible]PD_8.3.0815_RC5_en
d----          16/12/2022     12:00                [illegible]t_v2
d----          11/11/2022     13:49                upload
-a---          13/06/2022     09:40          [illegible]9614 18_05_2022.xml
-a---          13/06/2022     09:41          38491 19_05_2022.xml
-a---          27/03/2022     14:[illegible]            788 LAPTOP-1APHFJ6R-.cer
-a---          27/03/2022     1[illegible]            794 LAPTOP-1APHFJ6R-111.cer
-a---          16/10/2022     13:16             15 New Text Document.ps1
-a---          18/02/2023     16:46      102614016 PST.pst
-a---          29/11/2022     14:59           3025 ssl_check_transcript.txt
-a---          16/04/2023     16:37              0 todelete.txt
-a---          16/04/2023     17:38            211 verbose_demo.ps1
VERBOSE: 16/04/2023 17:39:28 | End Sleep Action
```

WRITE-VERBOSE

```
function write-verbose($message)
{
    Begin
    {
        if(!($global:pipeline))
            {
            $global:pipeline = { Out-GridView -Title “Verbose"}.GetSteppablePipeline()
            $global:pipeline.Begin($true)
            }
    }
    Process
    {
        $global:pipeline.Process($message)
        }
   end
   {

   }
}
```

WRITE-VERBOSE



```
PS C:\Temp> .\verbose_demo.ps1

    Directory: C:\Temp

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          17/09/2022     12:09                Kx83_UPD_8.3.0815_RC5_en
d----          16/12/2022     12:00                upl_v2
d----          11/11/2022     13:49                upload
-a---          13/06/2022     09:40          39614 18_05_2022.xml
-a---          13/06/2022     09:41          38491 19_05_2022.xml
-a---          27/03/2022     14:56            788 LAPTOP-1APHFJ6R-.cer
-a---          27/03/2022     14:38            794 LAPTOP-1APHFJ6R-111.cer
-a---          16/10/2022     13:16             15 New Text Document.ps1
-a---          18/02/2023     16:46      102614016 PST.pst
-a---          29/11/2022     14:59           3025 ssl_check_transcript.txt
-a---          16/04/2023     16:37              0 todelete.txt
-a---          16/04/2023     17:38            211 verbose_demo.ps1
```

Verbose

String
16/04/2023 17:41:45 | Begin Sleep Action
16/04/2023 17:41:48 | End Sleep Action

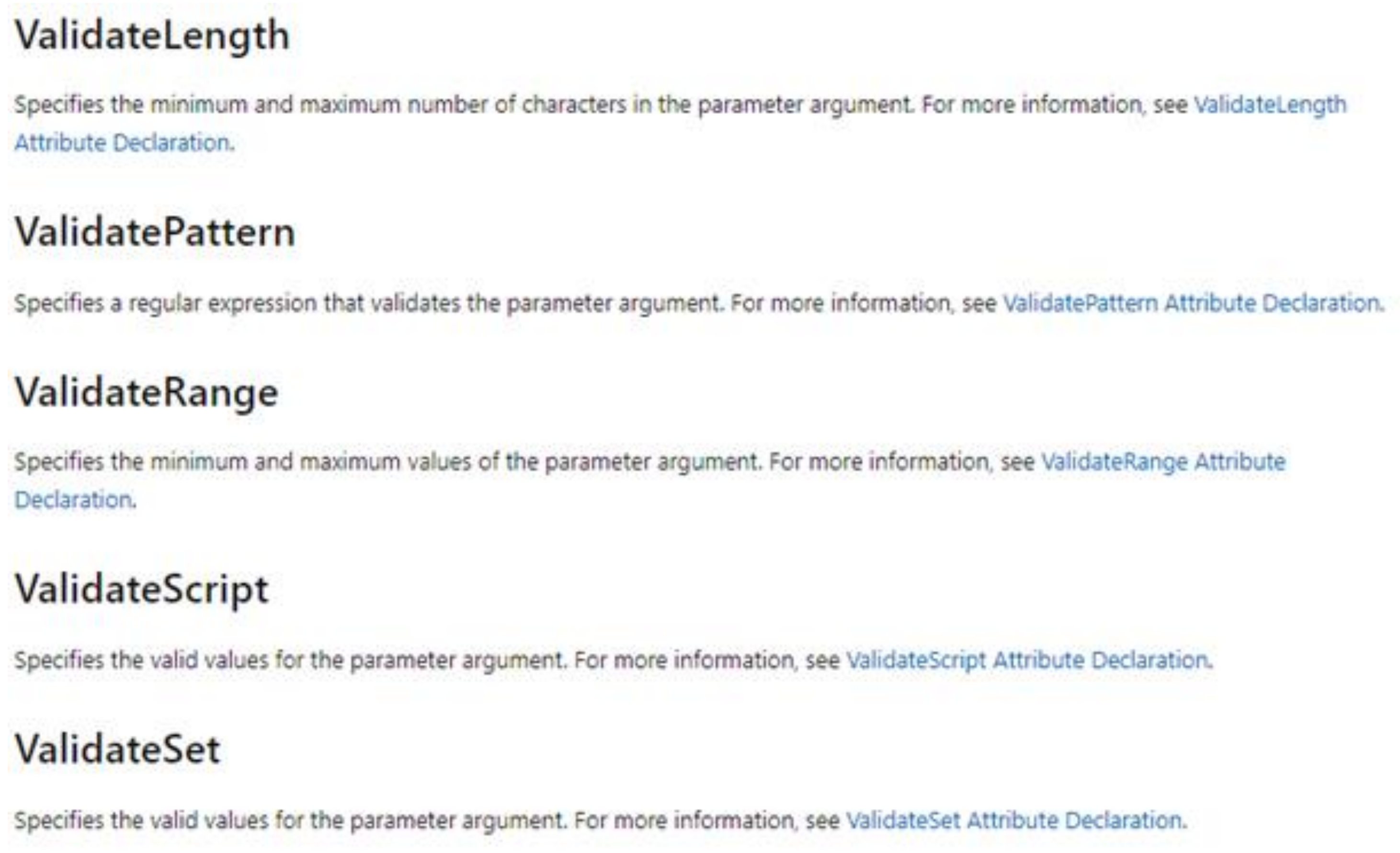
PARAMETER VALIDATION

PARAMETER VALIDATION

- Customize cmdlet behavior or actions
- Start with – and use consistent names across cmdlets
- Often misunderstood / abused by the executing user …. ☹
- Time to protect your function parameters from user-stupidity

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

PARAMETER VALIDATION



ValidateLength

Specifies the minimum and maximum number of characters in the parameter argument. For more information, see ValidateLength Attribute Declaration.

ValidatePattern

Specifies a regular expression that validates the parameter argument. For more information, see ValidatePattern Attribute Declaration.

ValidateRange

Specifies the minimum and maximum values of the parameter argument. For more information, see ValidateRange Attribute Declaration.

ValidateScript

Specifies the valid values for the parameter argument. For more information, see ValidateScript Attribute Declaration.

ValidateSet

Specifies the valid values for the parameter argument. For more information, see ValidateSet Attribute Declaration.

https://www.scriptrunner.com/en/blog/parameter-validation-concepts-powershell/

PARAMETER VALIDATION EXAMPLES

```
[Parameter(Mandatory)]
[ValidatePattern('^\+[1-9]{1}[0-9]{3,14}$')]
[String]
$MobilePhone

[Parameter(Mandatory)]
[ValidateScript({test-Path $_})]
[String]
$FolderPath

[Parameter(Mandatory)]
[ValidateSet('8GB', '16GB', '32GB')]
[String]
$Memory
```

```
[Parameter(Mandatory)]
[ValidateLength(5,15)]
[String]
$FileName
```

```
[Parameter(Mandatory)]
[ValidateRange(-1,10)]
[Int]
$Volume
```

(AVOID) PIPELINING

PIPELINING

- What ?
  - Passing results preceding command to the next command

- Practical
  - Playing with memory allocation
  - Can make scripts slow especially in loops

- Tip :

  - avoid pipelining in loops !
  - always look at properties and methods of your objects

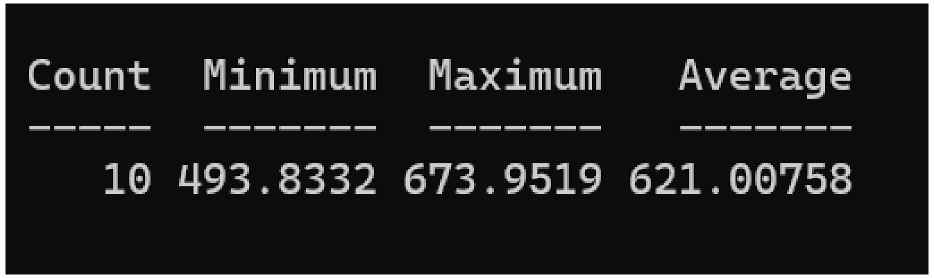
https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_pipelines

Out-Null vs $null (aka hiding output)

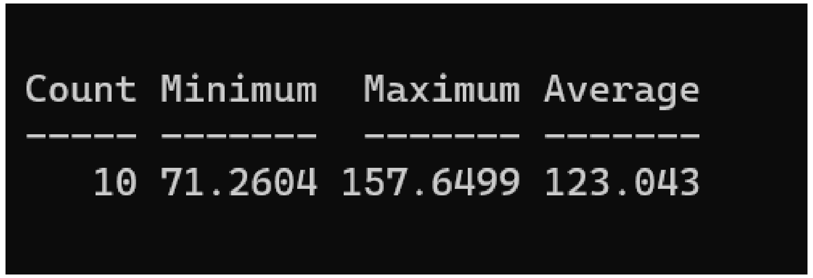
```
$guid = new-guid
```

```
1..10000 |%{ $guid | out-null }
```

```
1..10000 |%{ $null = $guid    }
```



```
Count  Minimum   Maximum    Average
-----  -------   -------    -------
   10 493.8332 673.9519 621.00758
```



```
Count Minimum  Maximum  Average
----- -------  -------  -------
   10 71.2604 157.6499 123.043
```

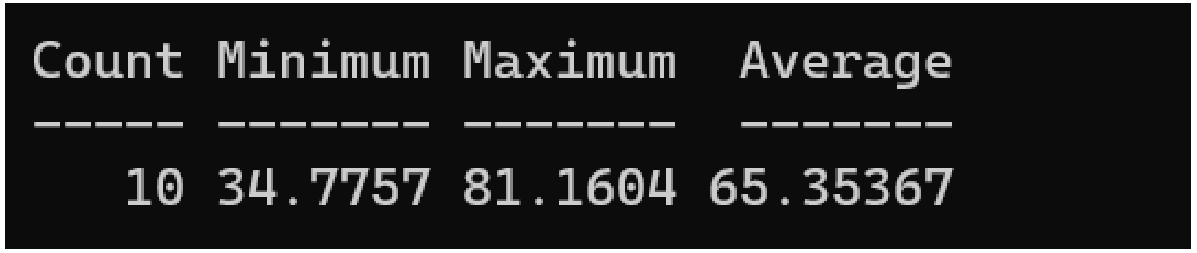
Times in Milliseconds / measured with measure-command

Measure-Object vs .Count

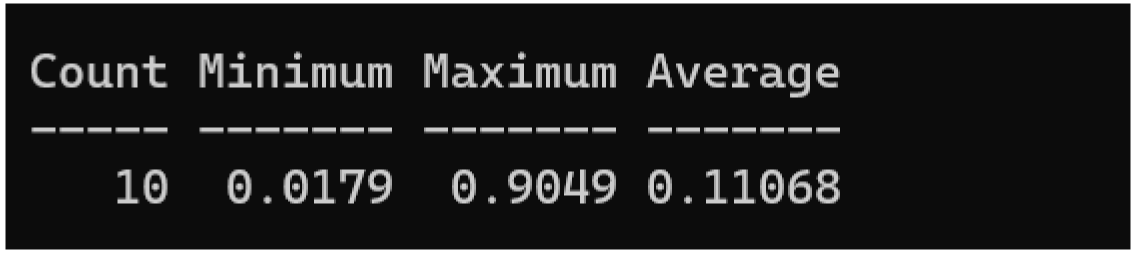
```
$datafield = 1..9999|%{New-Guid}
```

```
($datafield|measure-object).count
```

```
$datafield.count
```



```
Count Minimum Maximum  Average
----- ------- -------  -------
   10 34.7757 81.1604 65.35367
```



```
Count Minimum Maximum Average
----- ------- ------- -------
   10  0.0179  0.9049 0.11068
```

Times in Milliseconds / measured with measure-command

Powershell .WTF

HASHTABLES

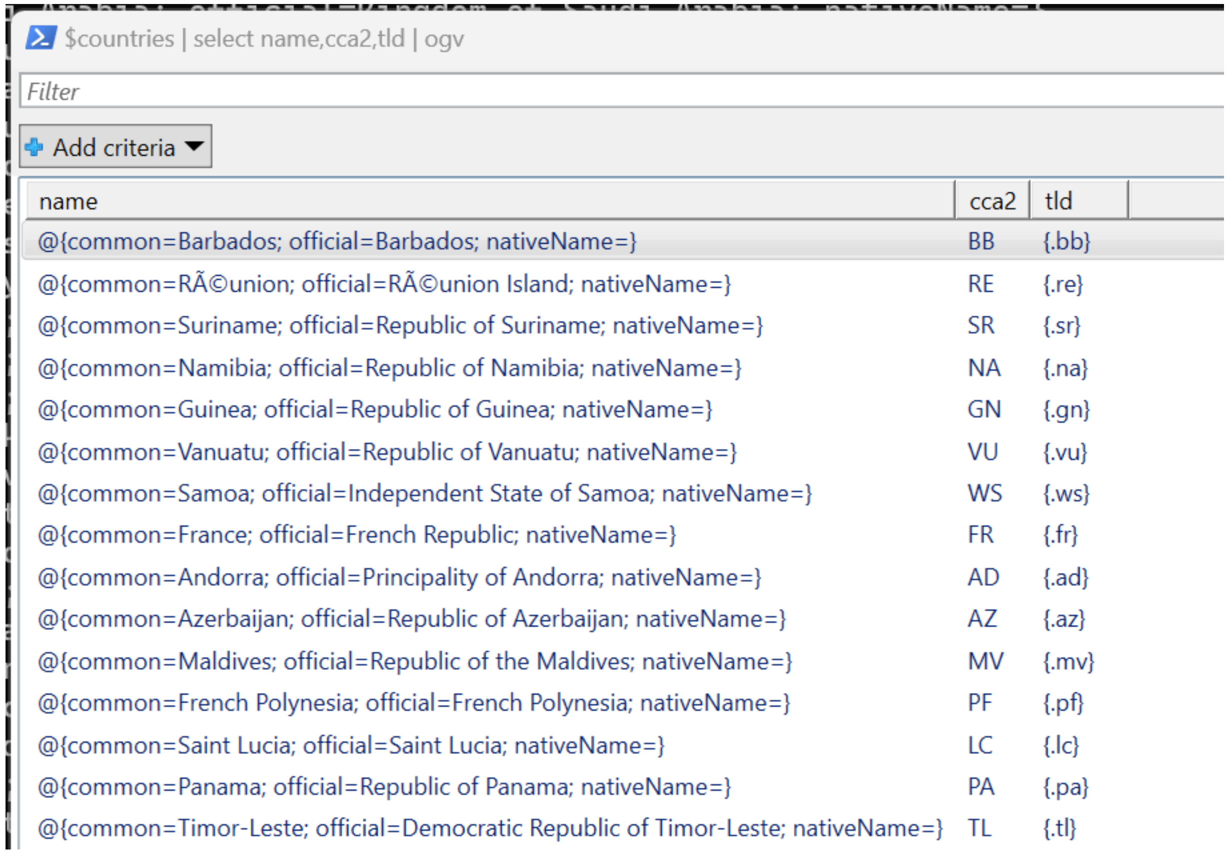
HASHTABLES

- A hash table, also known as a dictionary or associative array, is a data structure that stores one or more key/value pairs
- Begin the hash table with an at sign (@).
- Enclose the hash table in braces ({}).
- Enter one or more key/value pairs for the content of the hash table.
- Use an equal sign (=) to separate each key from its value.
- Use a colon(,) or semicolon (;) or a line break to separate the key\/value pairs.

```
$hashtable  = @{'Bruno'=37 ; 'Ella'=35}

$hashtable.'Bruno'
$hashtable['Bruno']
```

HASHTABLES EXAMPLE

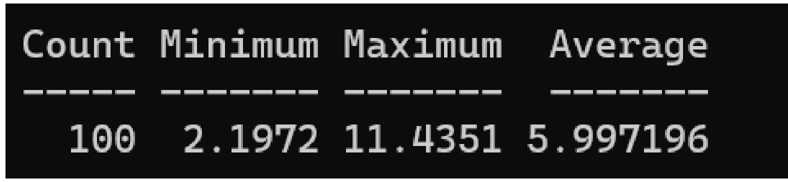
```
$countries = invoke-RestMethod 'https://restcountries.com/v3.1/all'
```



```
$countries | select name,cca2,tld | ogv
Filter
Add criteria
name                                                                          cca2  tld
@{common=Barbados; official=Barbados; nativeName=}                            BB    {.bb}
@{common=RÃ©union; official=RÃ©union Island; nativeName=}                     RE    {.re}
@{common=Suriname; official=Republic of Suriname; nativeName=}                SR    {.sr}
@{common=Namibia; official=Republic of Namibia; nativeName=}                  NA    {.na}
@{common=Guinea; official=Republic of Guinea; nativeName=}                    GN    {.gn}
@{common=Vanuatu; official=Republic of Vanuatu; nativeName=}                  VU    {.vu}
@{common=Samoa; official=Independent State of Samoa; nativeName=}             WS    {.ws}
@{common=France; official=French Republic; nativeName=}                       FR    {.fr}
@{common=Andorra; official=Principality of Andorra; nativeName=}              AD    {.ad}
@{common=Azerbaijan; official=Republic of Azerbaijan; nativeName=}            AZ    {.az}
@{common=Maldives; official=Republic of the Maldives; nativeName=}            MV    {.mv}
@{common=French Polynesia; official=French Polynesia; nativeName=}            PF    {.pf}
@{common=Saint Lucia; official=Saint Lucia; nativeName=}                      LC    {.lc}
@{common=Panama; official=Republic of Panama; nativeName=}                    PA    {.pa}
@{common=Timor-Leste; official=Democratic Republic of Timor-Leste; nativeName=} TL  {.tl}
```

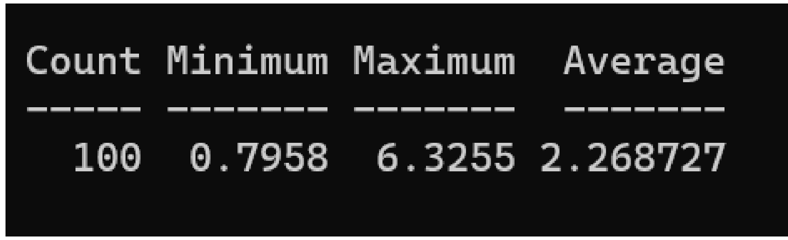
Hashtable vs Where-Object

```powershell
$countries | Where-Object {$_.cca2 -eq "BE"}
```



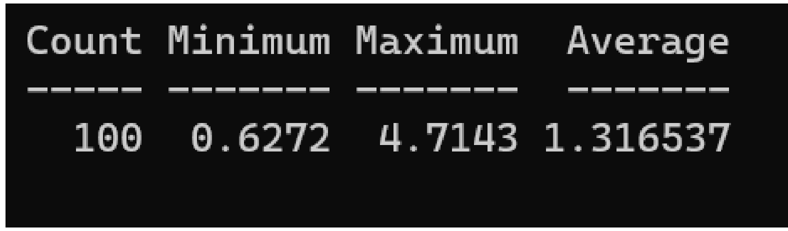
```
Count Minimum Maximum  Average
----- ------- -------  -------
  100  2.1972 11.4351 5.997196
```

```powershell
$countries.Where({$_.cca2 -eq "BE"})
```



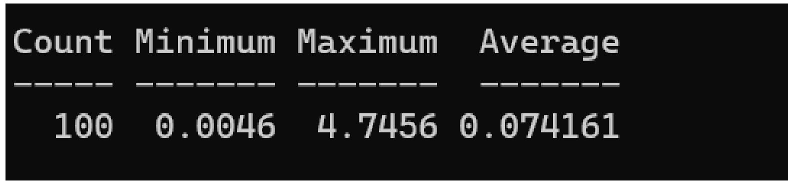
```
Count Minimum Maximum  Average
----- ------- -------  -------
  100  0.7958  6.3255 2.268727
```

```powershell
$hashtable_countries =  @{}
foreach ($country in $countries)
{
  $hashtable_countries.Add($country.cca2, $country)
}
```



```
Count Minimum Maximum  Average
----- ------- -------  -------
  100  0.6272  4.7143 1.316537
```

```powershell
$hashtable_countries.'BE'
```

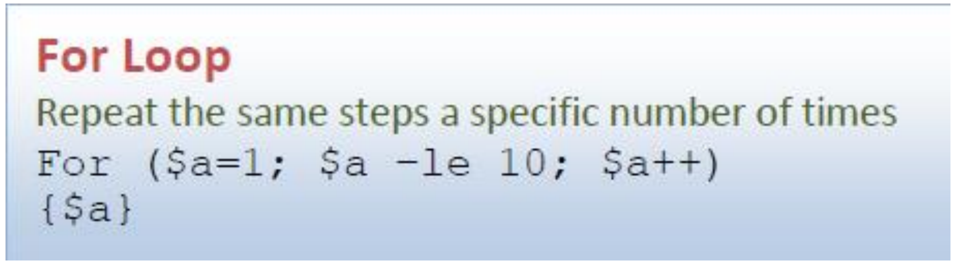


```
Count Minimum Maximum  Average
----- ------- -------  -------
  100  0.0046  4.7456 0.074161
```

(FOR)(EACH)LOOPS

About loops

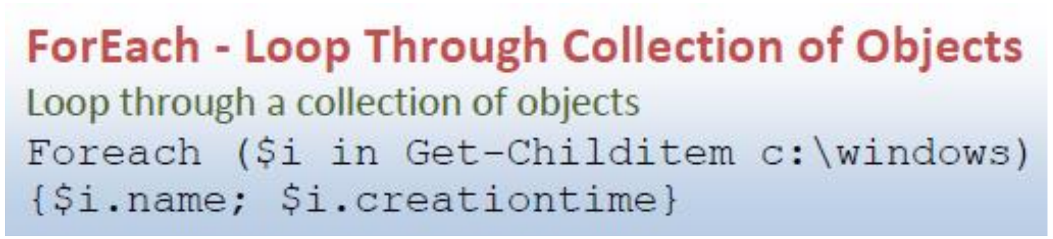
- The For loop is a loop that runs commands in a command block while a specified condition evaluates to $true.
- The foreach statement is a language construct for iterating a series of values in a collection of items.



For Loop

Repeat the same steps a specific number of times

```
For ($a=1; $a -le 10; $a++)
{$a}
```



ForEach - Loop Through Collection of Objects

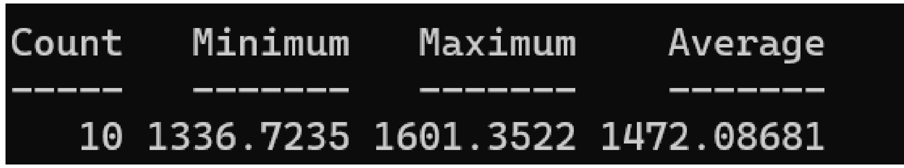
Loop through a collection of objects

```
Foreach ($i in Get-Childitem c:\windows)
{$i.name; $i.creationtime}
```

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

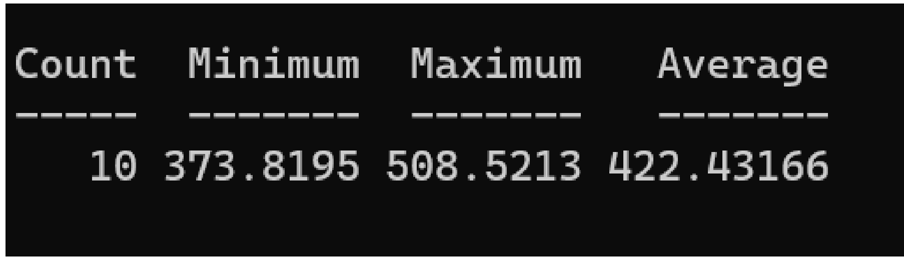
LOOPS

```
1..100000 | ForEach-Object {$a = $_/2 }
```



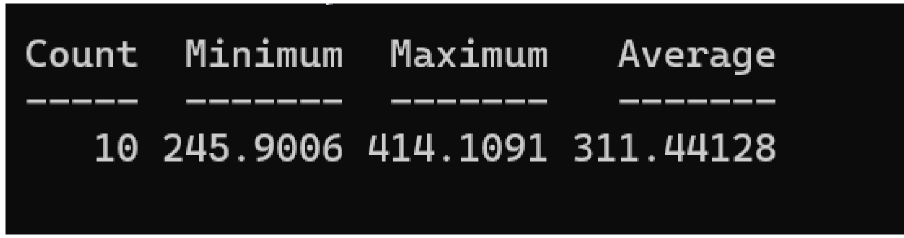
```
Count    Minimum    Maximum     Average
-----    -------    -------     -------
   10  1336.7235  1601.3522  1472.08681
```

```
for ($x = 1; $x -lt 100001; $x++)
{
    $a = $x/2
}
```



```
Count   Minimum   Maximum    Average
-----   -------   -------    -------
   10 373.8195 508.5213 422.43166
```

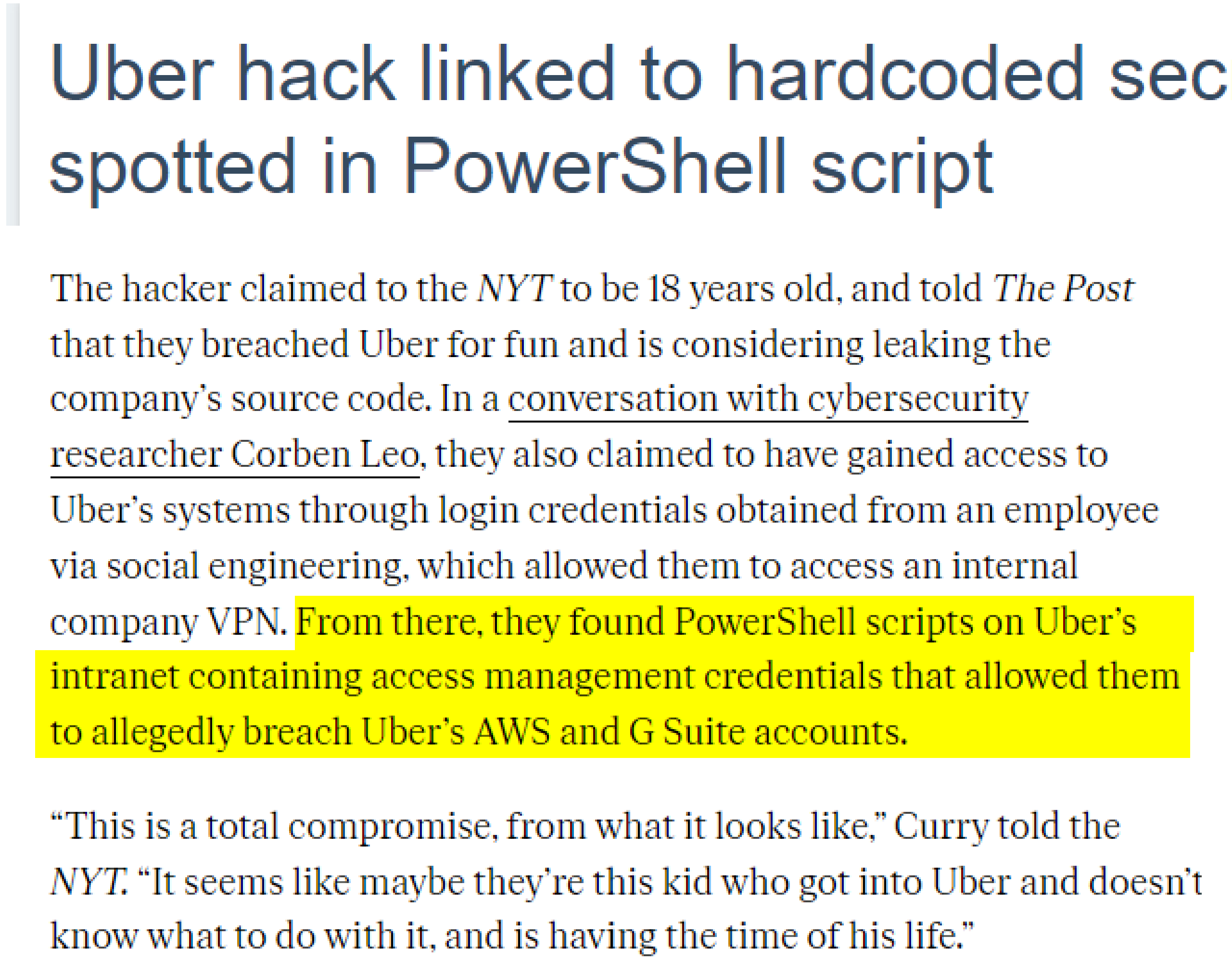
```
foreach($g in (1..100000))
{
        $a = $g/2
}
```



```
Count   Minimum   Maximum    Average
-----   -------   -------    -------
   10 245.9006 414.1091 311.44128
```

WORKING WITH SECRETS

WORKING WITH SECRETS



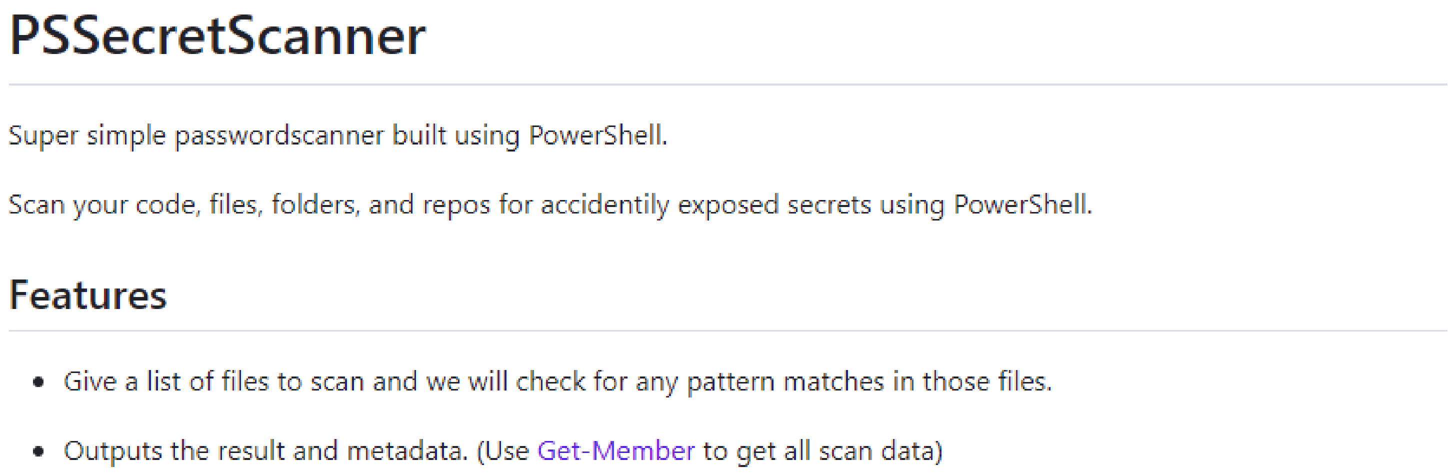
Uber hack linked to hardcoded secrets spotted in PowerShell script

The hacker claimed to the *NYT* to be 18 years old, and told *The Post* that they breached Uber for fun and is considering leaking the company's source code. In a conversation with cybersecurity researcher Corben Leo, they also claimed to have gained access to Uber's systems through login credentials obtained from an employee via social engineering, which allowed them to access an internal company VPN. From there, they found PowerShell scripts on Uber's intranet containing access management credentials that allowed them to allegedly breach Uber's AWS and G Suite accounts.

"This is a total compromise, from what it looks like," Curry told the *NYT*. "It seems like maybe they're this kid who got into Uber and doesn't know what to do with it, and is having the time of his life."

https://www.theverge.com/2022/9/16/23356213/uber-hack-teen-slack-google-cloud-credentials-powershell

INVENTORY PSSecretScanner (Björn Sundling)



PSSecretScanner

Super simple passwordscanner built using PowerShell.

Scan your code, files, folders, and repos for accidentily exposed secrets using PowerShell.

Features

- Give a list of files to scan and we will check for any pattern matches in those files.
- Outputs the result and metadata. (Use Get-Member to get all scan data)

https://github.com/bjompen/PSSecretScanner

SECRETS DON’TS

- Define them in :
  - config File (plain text, encrypted, obfuscated)
  - script (plain text)
  - Registry
- System.Management.Automation.PSCredential
  - Can be reversed

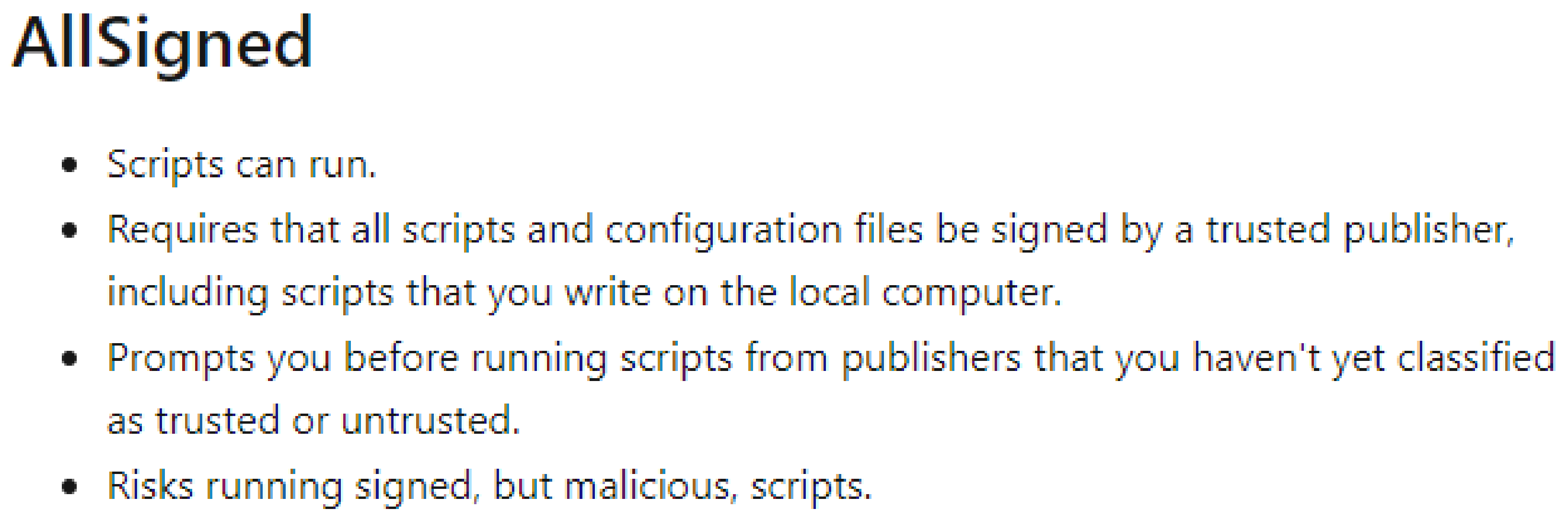
SECRET DO’S

- USE :
  - Windows Key Vault
  - Cloud Key Vault
    - AWS
    - Azure
  - Other Managers
    - Tycotic
    - CyberArk
    - …..

PROTECTING SCRIPTS

CODE-SIGNING : EXECUTION POLICY

- Safety feature to control script execution
- Enfore signed scripts :
  - set-executionpolicy AllSigned



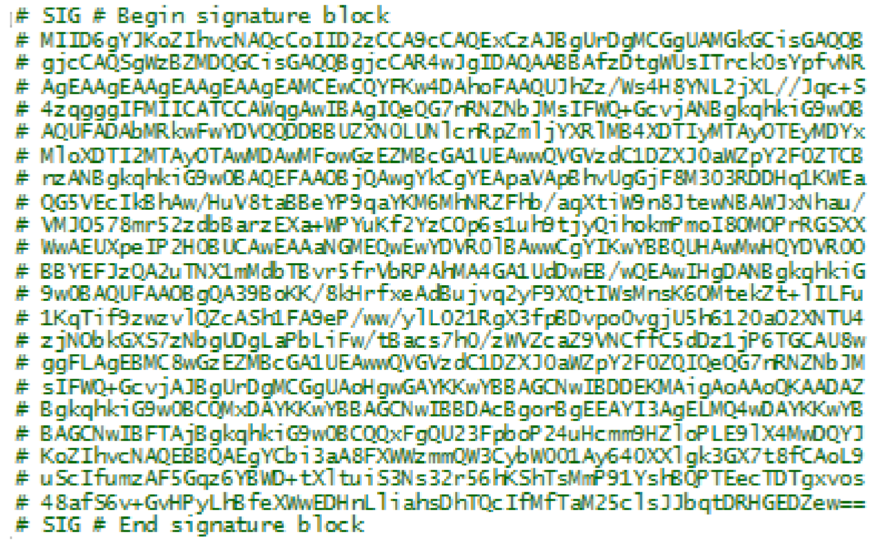
AllSigned

- Scripts can run.
- Requires that all scripts and configuration files be signed by a trusted publisher, including scripts that you write on the local computer.
- Prompts you before running scripts from publishers that you haven't yet classified as trusted or untrusted.
- Risks running signed, but malicious, scripts.

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

CODE SIGNING

- Requirements
  - Scripter : Code signing certificate
  - Client : Trust certificate authority of the code signing certificate
  - Client : executionpolicy allsigned
- Signing
  - Set-AuthenticodeSignature
- Check signing
  - Get-AuthenticodeSignature

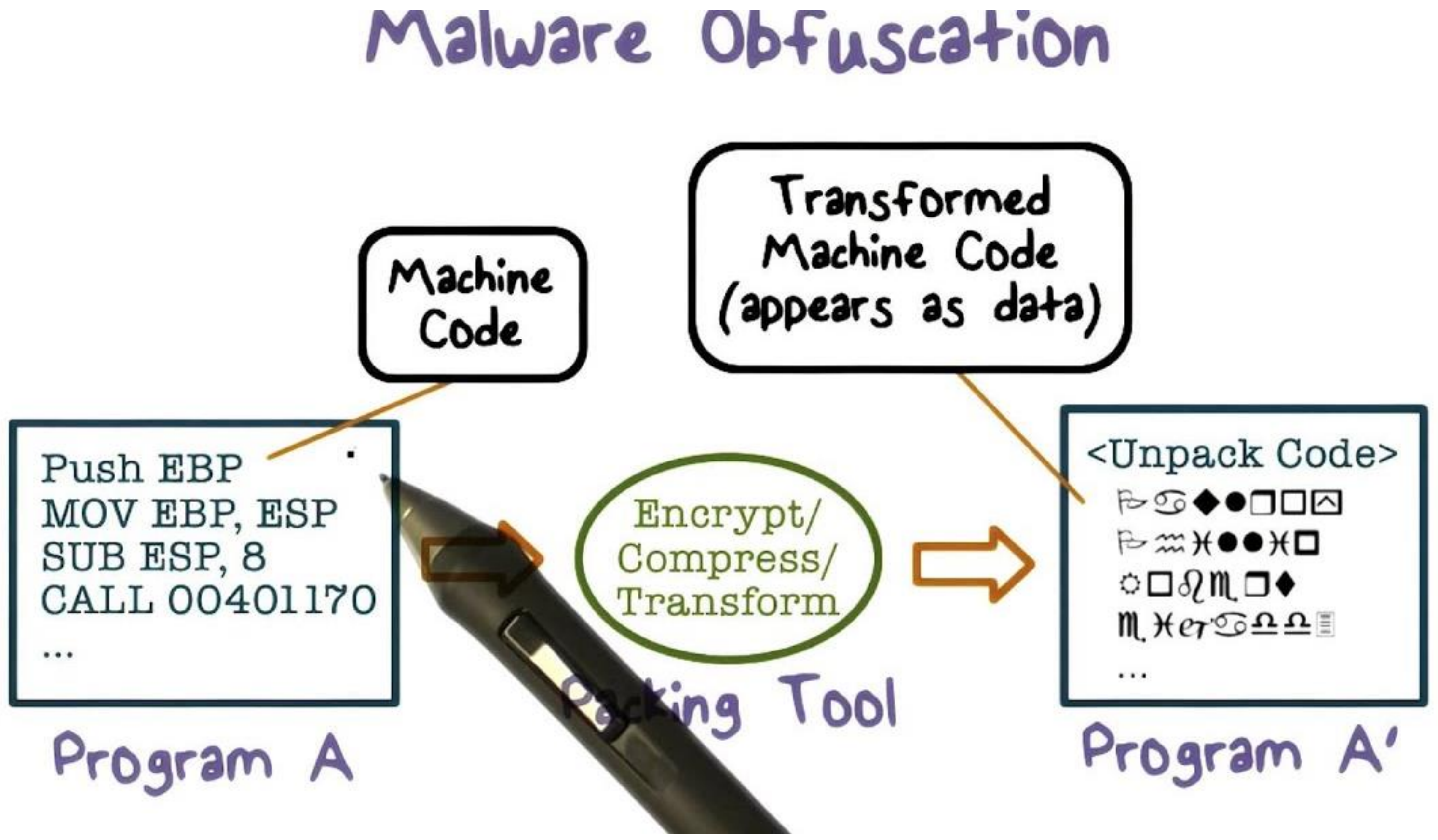


```
# SIG # Begin signature block
# MIID6gYJKoZIhvcNAQcCoIID2zCCA9cCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUJhZz/Ws4H8YNL2jXL//Jqc+S
# 4zqgggIFMIICATCCAWqgAwIBAgIQeQG7nRNZNbJMsIFWQ+GcvjANBgkqhkiG9w0B
# AQUFADAbMRkwFwYDVQQDDBBUZXN0LUNlcnRpZmljYXRlMB4XDTIyMTAyOTEyMDYx
# MloXDTI2MTAyOTAwMDAwMFowGzEZMBcGA1UEAwwQVGVzdC1DZXJ0aWZpY2F0ZTCB
# nzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEApaVApBhvUgGjF8M303RDDHq1KWEa
# QG5VEcIkBhAw/HuV8taBBeYP9qaYKM6MhNRZFHb/agXtiW9n8JtewNBAWJxNhau/
# VMJO578mr52zdbBarzEXa+WPYuKf2YzCOp6s1uH9tjyQihokmPmoI8OMOPrRGSXX
# WwAEUXpeIP2H0BUCAwEAAaNGMEQwEwYDVR0lBAwwCgYIKwYBBQUHAwMwHQYDVR0O
# BBYEFJzQA2uTNX1mMdbTBvr5frVbRPAhMA4GA1UdDwEB/wQEAwIHgDANBgkqhkiG
# 9w0BAQUFAAOBgQA39BoKK/8kHrfxeAdBujvq2yF9XQtIWsMnsK6OMtekZt+lILFu
# 1KqTif9zwzvlQZcASh1FA9eP/ww/ylL021RgX3fpBDvpo0vgjU5h6120aO2XNTU4
# zjNObkGXS7zNbgUDgLaPbLiFw/tBacs7h0/zWVZcaZ9VNCffC5dDz1jP6TGCAU8w
# ggFLAgEBMC8wGzEZMBcGA1UEAwwQVGVzdC1DZXJ0aWZpY2F0ZQIQeQG7nRNZNbJM
# sIFWQ+GcvjAJBgUrDgMCGgUAoHgwGAYKKwYBBAGCNwIBDDEKMAigAoAAoQKAADAZ
# BgkqhkiG9w0BCQMxDAYKKwYBBAGCNwIBBDAcBgorBgEEAYI3AgELMQ4wDAYKKwYB
# BAGCNwIBFTAjBgkqhkiG9w0BCQQxFgQU23FpboP24uHcmm9HZloPLE9lX4MwDQYJ
# KoZIhvcNAQEBBQAEgYCbi3aA8FXWWzmmQW3CybWO01Ay640XXlgk3GX7t8fCAoL9
# uScIfumzAF5Gqz6YBWD+tXltuiS3Ns32r56hKShTsMmP91YshBQPTEecTDTgxvos
# 48afS6v+GvHPyLHBfeXWwEDHnLliahsDhTQcIfMfTaM25clsJJbqtDRHGEDZew==
# SIG # End signature block
```

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

OBFUSCATION

- Can be used to protect your code
- Is used by malware to hide/scramble code



EXAMPLES

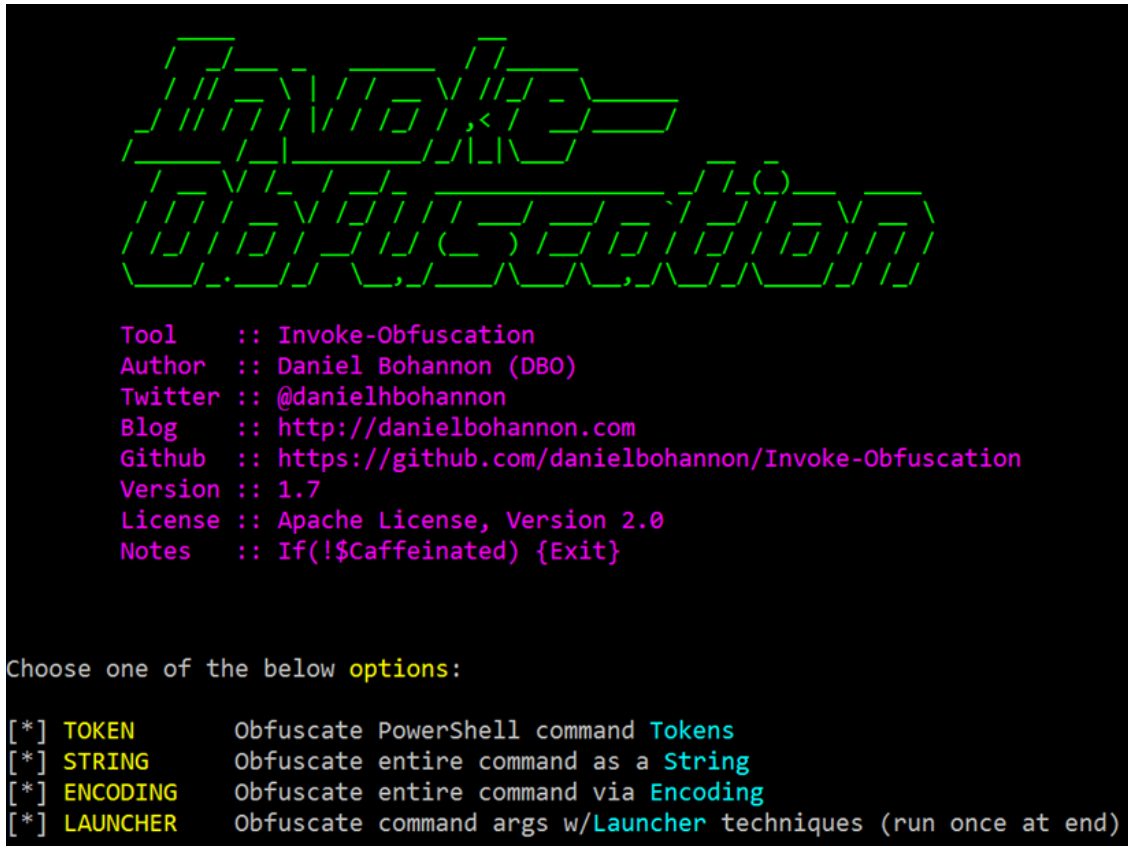
```
Invoke-Expression "& {$(Invoke-RestMethod –Uri ’https://aka.ms/install-powershell.ps1’)} -UseMSI -Quiet“
```

```
iex "& {$(irm -Uri
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('aAB0AHQAcABzADoALwAvAGEAawBhAC4AbQ
BzAC8AaQBuAHMAdABhAGwAbAAtAHAAbwB3AGUAcgBzAGgAZQBsAGwALgBwAHMAMQA='))))} -UseMSI -Quiet"
```

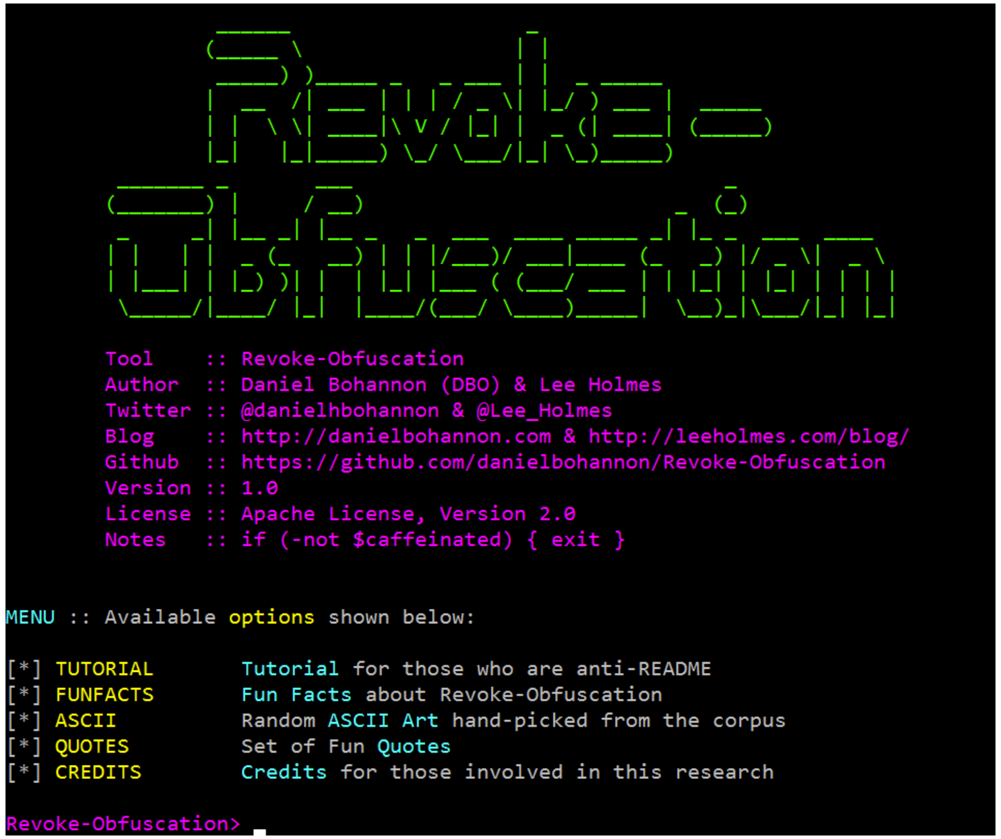
```
$variable_pwsh_is_fun = 'Yes it is'

${___/=\_/====\/\/=} =
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('WQBlAHMAIABpAHQAIABpAHMA')))
```

INVOKE/REVOKE-OBFUSCATION

- https://github.com/danielbohannon/



```
Tool     :: Invoke-Obfuscation
Author   :: Daniel Bohannon (DBO)
Twitter  :: @danielhbohannon
Blog     :: http://danielbohannon.com
Github   :: https://github.com/danielbohannon/Invoke-Obfuscation
Version  :: 1.7
License  :: Apache License, Version 2.0
Notes    :: If(!$Caffeinated) {Exit}


Choose one of the below options:

[*] TOKEN       Obfuscate PowerShell command Tokens
[*] STRING      Obfuscate entire command as a String
[*] ENCODING    Obfuscate entire command via Encoding
[*] LAUNCHER    Obfuscate command args w/Launcher techniques (run once at end)
```



```
Tool     :: Revoke-Obfuscation
Author   :: Daniel Bohannon (DBO) & Lee Holmes
Twitter  :: @danielhbohannon & @Lee_Holmes
Blog     :: http://danielbohannon.com & http://leeholmes.com/blog/
Github   :: https://github.com/danielbohannon/Revoke-Obfuscation
Version  :: 1.0
License  :: Apache License, Version 2.0
Notes    :: if (-not $caffeinated) { exit }


MENU :: Available options shown below:

[*] TUTORIAL      Tutorial for those who are anti-README
[*] FUNFACTS      Fun Facts about Revoke-Obfuscation
[*] ASCII         Random ASCII Art hand-picked from the corpus
[*] QUOTES        Set of Fun Quotes
[*] CREDITS       Credits for those involved in this research

Revoke-Obfuscation> _
```

LOGGING

ENABLE TRANSCRIPTION

- Records console commands within a session (start-transcript)
- Includes the console output
- Starting with Windows PowerShell 5.0:
  - The log file name includes computer name and timestamp
  - Supports remoting
  - Can be enabled on non-console host applications
  - Can be redirected to a network share
- To enable, use:
  - Group Policy
  - Direct registry modification
  - PowerShell script

```
HKLM:\Software\Policies\Microsoft\Windows\PowerShell\Transcription
```

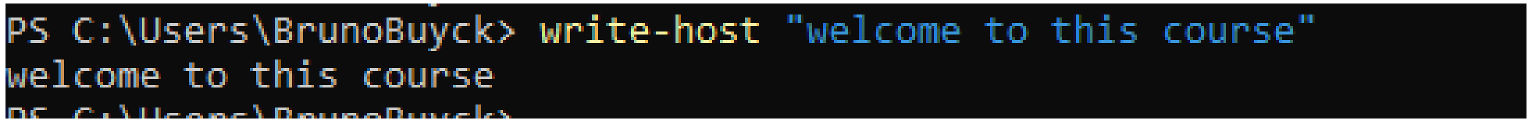
SCRIPTBLOCK LOGGING

- Records content of all script blocks
- Introduced in Windows PowerShell 5.0:
  - Uses ETW Microsoft-Windows-PowerShell\Operational log
  - Identified by the event ID 4104
  - Captures dynamic code generation (e.g. Invoke-Expression)

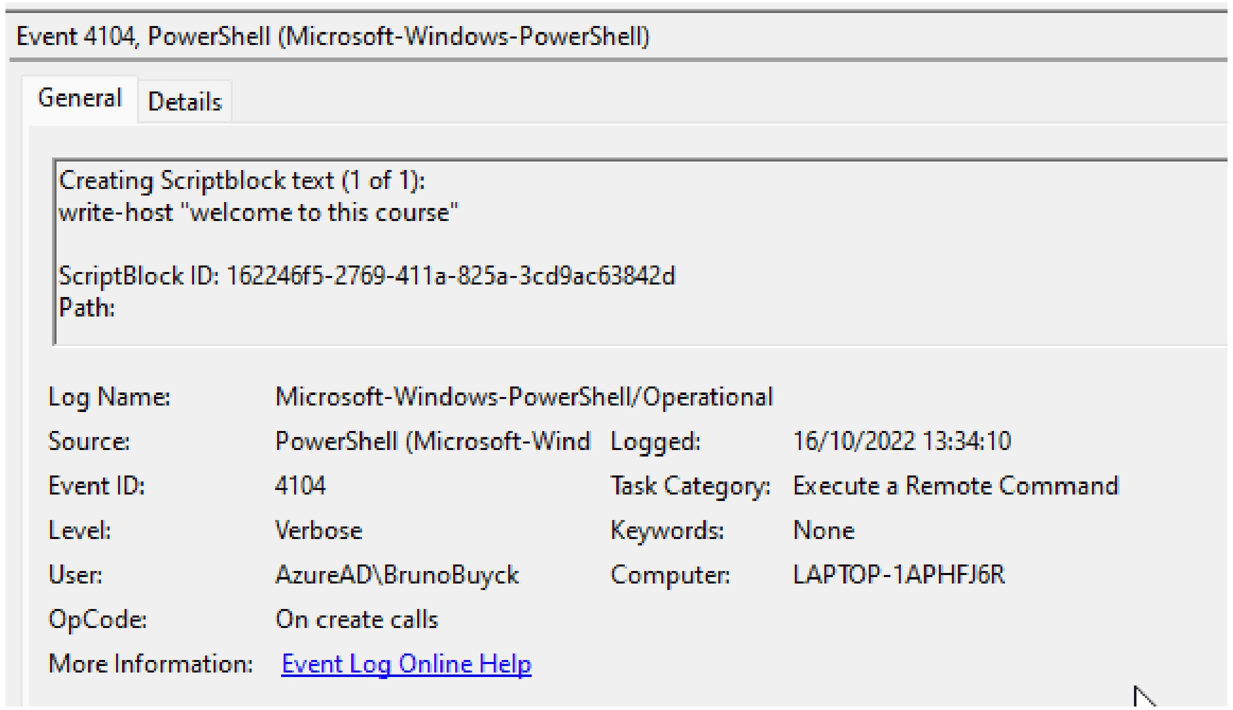
- To enable, use:
  - Group Policy
  - Direct registry modification
  - PowerShell script

`HKLM:\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging`.

SCRIPTBLOCK LOGGING



```
PS C:\Users\BrunoBuyck> write-host "welcome to this course"
welcome to this course
```



Event 4104, PowerShell (Microsoft-Windows-PowerShell)

```
Creating Scriptblock text (1 of 1):
write-host "welcome to this course"

ScriptBlock ID: 162246f5-2769-411a-825a-3cd9ac63842d
Path:
```

Log Name:	Microsoft-Windows-PowerShell/Operational		
Source:	PowerShell (Microsoft-Wind	Logged:	16/10/2022 13:34:10
Event ID:	4104	Task Category:	Execute a Remote Command
Level:	Verbose	Keywords:	None
User:	AzureAD\BrunoBuyck	Computer:	LAPTOP-1APHFJ6R
OpCode:	On create calls		
More Information:	Event Log Online Help		

https://news.sophos.com/en-us/2022/03/29/reconstructing-powershell-scripts-from-multiple-windows-event-logs/

Conclusion

TIPS

- 1. Testing scripts : $whatifpreference / $erroractionpreference
- 2. Debugging : set breakpoints / provide diagnostic info
- 3. Functions : use parameter validation
- 4. Avoid pipelines, use properties and methods of the objects
- 5. Use hash tables for quick lookups
- 6. Loops : evaluate loops case by case
- 7. Identify & store your secrets in a vault
- 8. Sign your scripts , obfuscate when required
- 9. Enable host transcription
- 10. Enable Scriptblock logging

Q & A

THANK YOU FOR YOUR ATTENTION

Bruno@powershell.wtf