

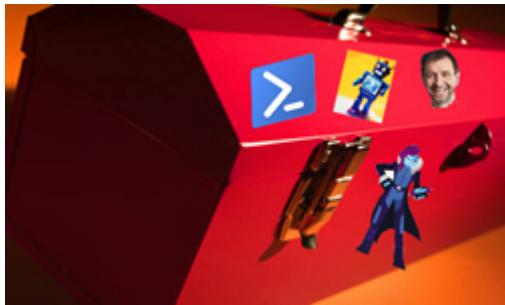
PSScriptTools Help Manual (v2.26.1)

Table of Contents

Installation	2
Uninstall the Module	2
General Tools	3
Get-ModuleCommand	3
Get-PSScriptTools	3
Convert-EventLogRecord	4
Get-WhoIs	5
Compare-Module	6
Get-WindowsVersion	6
New-PSDriveHere	8
Get-MyVariable	8
ConvertFrom-Text	8
Get-PSWho	9
Find-CimClass	10
Out-VerboseTee	11
Remove-Runspace	11
Get-PSLocation	11
Get-PowerShellEngine	12
Get-PathVariable	13
File Tools	14
Test-EmptyFolder	14
Get-FolderSizeInfo	14
Optimize-Text	15
Get-FileItem	15
New-CustomFileName	16
New-RandomFileName	16
ConvertTo-Markdown	17
ToDo	18
Graphical Tools	19
Invoke-InputBox	19
New-WPFMessageBox	20
ConvertTo-WPFGGrid	21
Hashtable Tools	24
Convert-CommandtoHashtable	24
Convert-HashtableString	24
Convert-HashTableToCode	24
ConvertTo-HashTable	25
Join-HashTable	25

Rename-Hashtable	26
Select Functions	27
Time Functions	28
ConvertTo-UTCTime	28
ConvertFrom-UTCTime	28
Get-MyTimeInfo	28
ConvertTo-LocalTime	29
Get-TZList	29
Get-TZData	30
ConvertTo-LexicalTime	31
ConvertFrom-LexicalTime	31
Console Utilities	32
Out-More	32
Out-ConditionalColor	32
Set-ConsoleTitle	33
Set-ConsoleColor	34
Add-Border	34
Show-Tree	34
New-ANSIBar	36
New-RedGreenGradient	37
Write-ANSIProgress	37
Format Functions	39
Format-Percent	39
Format-String	39
Format-Value	39
Scripting Tools	40
Test-Expression	40
Copy-HelpExample	41
Get-GitSize	43
Remove-MergedBranch	43
Test-WithCulture	44
Copy-Command	44
Get-ParameterInfo	44
New-PSFormatXML	45
Test-IsPSWindows	46
Write-Detail	46
Save-GitSetup	47
Other	48
PSAnsiMap	48
PSSpecialChar	51
Sample Scripts	53

Open-PSScriptToolsHelp.....	53
Related Modules	54
Compatibility	55



This module contains a collection of functions, variables and format files that you can use to enhance your PowerShell scripting work. Or get more done from a PowerShell prompt with less typing. Most of the commands are designed to work cross-platform. Please post any questions, problems or feedback in [Issues](#). Any feedback is greatly appreciated.

Installation

You can get the current release from this repository or install this the PowerShell Gallery:

```
Install-Module PSScriptTools
```

or in PowerShell 7:

```
Install-Module PSScriptTools [-scope CurrentUser]
```

Starting in v2.2.0, the module was restructured to better support [Desktop](#) and [Core](#) editions. But starting with version 2.13.0, the module design has reverted. All commands will be exported. Anything that is platform specific should be handled on a per command basis. It is assumed you will be running this module in Windows PowerShell 5.1 or PowerShell 7.

Uninstall the Module

To remove the module from your system you can uninstall it.

```
Get-Module PSScriptTools | Remove-Module  
Uninstall-Module PSScriptTools -allversions
```

General Tools

Get-ModuleCommand

This is an alternative to [Get-Command](#) to make it easier to see at a glance what commands are contained within a module and what they can do. By default, [Get-ModuleCommand](#) looks for loaded modules. Use [-ListAvailable](#) to see commands in module not currently loaded. Note that if the help file is malformed or missing, you might get oddly formatted results.

```
PS C:\> Get-ModuleCommand PSCalendar
```

Verb: Get

Name	Alias	Type	Synopsis
---	---	---	-----
Get-Calendar	cal	Function	Displays a visual representation of a ...

Verb: Show

Name	Alias	Type	Synopsis
---	---	---	-----
Show-Calendar	scal	Function	Display a colorized calendar month in ...
Show-GuiCalendar	gcal	Function	Display a WPF-based calendar

Get module commands using the default formatted view. There is also a default view for [Format-List](#).

Get-PSScriptTools

You can use this command to get a summary list of functions in this module.

```
PS C:\> Get-PSScriptTools

Verb: Add

Name           Alias      Synopsis
----          -----      -----
Add-Border

Verb: Compare

Name           Alias      Synopsis
----          -----      -----
Compare-Module    cmo      Compare PowerShell module versions.

Verb: Convert

Name           Alias      Synopsis
----          -----      -----
Convert-CommandtoHashtable
Convert-EventLogRecord    clr      Convert EventLogRecords to structured objects
Convert-HashtableString
Convert-HashTableToCode
...

```

Here's another way you could use this command to list functions with defined aliases in the PSScriptTools module.

```
PS C:\> Get-PSScriptTools | Where-object alias | Select-Object Name,alias,Synopsis

Name           Alias      Synopsis
----          -----      -----
Compare-Module    cmo      Compare PowerShell module versions.
Convert-EventLogRecord    clr      Convert EventLogRecords to structured objects
ConvertFrom-Text       cft      Convert structured text to objects.
ConvertFrom-UTCTime     frut     Convert a datetime value from universal
ConvertTo-LocalTime     clt      Convert a foreign time to local
...

```

Convert-EventLogRecord

When you use [Get-WinEvent](#), the results are objects you can work with in PowerShell. However, often times there is additional information that is part of the eventlog record, such as replacement strings, that are used to construct a message. This additional information is not readily exposed. You can use this command to convert results of a Get-WinEvent command into a PowerShell custom object with additional information.

```
PS C:\> get-winevent -FilterHashtable @{Logname='System';ID=7045} -MaxEvents 1 | Convert-EventLogRecord
```

```
LogName      : System
RecordType   : Information
TimeCreated  : 1/21/2020 3:49:46 PM
ID          : 7045
ServiceName  : Netwrix Account Lockout Examiner
ImagePath    : "C:\Program Files (x86)\Netwrix\Account Lockout Examiner\ALEService.exe"
ServiceType  : user mode service
StartType    : auto start
AccountName  : bovine320\jeff
Message      : A service was installed in the system.

        Service Name: Netwrix Account Lockout Examiner
        Service File Name: "C:\Program Files (x86)\Netwrix\Account Lockout Examiner\ALEService.exe"
        Service Type: user mode service
        Service Start Type: auto start
        Service Account: bovine320\jeff
Keywords     : {Classic}
Source       : Service Control Manager
Computername : Bovine320
```

Get-WhoIs

This command will retrieve WhoIs information from the ARIN database for a given IPv4 address.

```
PS C:\> get-whois 208.67.222.222 | select-object -Property *
```

```
IP          : 208.67.222.222
Name        : OPENDNS-NET-1
RegisteredOrganization : Cisco OpenDNS, LLC
City         : San Francisco
StartAddress : 208.67.216.0
EndAddress   : 208.67.223.255
NetBlocks    : 208.67.216.0/21
Updated      : 3/2/2012 8:03:18 AM
```

```
PS C:\> '1.1.1.1','8.8.8.8','208.67.222.222' | get-whois
```

Name	IP	RegisteredOrganization	NetBlocks	Updated
----	--	-----	-----	-----
APNIC-1	1.1.1.1	Asia Pacific Network Information Centre	1.0.0.0/8	7/30/2010 8:23:43 AM
LVLT-GOGL-8-8-8	8.8.8.8	Google LLC	8.8.8.0/24	3/14/2014 3:52:05 PM
OPENDNS-NET-1	208.67.222.222	Cisco OpenDNS, LLC	208.67.216.0/21	3/2/2012 8:03:18 AM

This module includes a custom format file for these results.

Compare-Module

Use this command to compare module versions between what is installed against an online repository like the PSGallery

```
PS C:\> Compare-Module Platyps
```

```
Name          : platyPS
OnlineVersion : 0.14.0
InstalledVersion : 0.14.0,0.12.0,0.11.1,0.10.2,0.9.0
PublishedDate  : 4/3/2019 12:46:30 AM
UpdateNeeded   : False
```

Or you can compare and manage multiple modules.

```
Compare-Module | Where UpdateNeeded | Out-GridView -title "Select modules to update" -outputMode multiple | Foreach {
    Update-Module $_.name }
```

This example compares modules and send results to `Out-GridView`. Use `Out-GridView` as an object picker to decide what modules to update.

Get-WindowsVersion

This is a PowerShell version of the `winver.exe` utility. This command uses PowerShell remoting to query the registry on a remote machine to retrieve Windows version information.

```
Computername: WIN10
```

ProductName	EditionID	ReleaseID	Build	InstalledUTC
Windows 10 Enterprise Evaluation	EnterpriseEval	1903	18362	2/6/2020 5:28:34 PM

```
Computername: SRV1
```

ProductName	EditionID	ReleaseID	Build	InstalledUTC
Windows Server 2016 Standard Evaluation	ServerStandardEval	1607	14393	2/6/2020 5:27:42 PM

```
Computername: SRV2
```

ProductName	EditionID	ReleaseID	Build	InstalledUTC
Windows Server 2016 Standard Evaluation	ServerStandardEval	1607	14393	2/6/2020 5:28:13 PM

The output has a default table view but there are other properties you might want to use.

```
PS C:\> get-windowsversion | Select-object *
```

```
ProductName : Windows 10 Pro
EditionID   : Professional
ReleaseID   : 1909
Build       : 18363.657
Branch      : 19h1_release
InstalledUTC : 7/5/2019 10:54:49 PM
Computername : BOVINE320
```

Get-WindowsVersionString

This command is a variation of [Get-WindowsVersion](#) that returns a formatted string with version information.

```
PS C:\> Get-WindowsVersionString
BOVINE320 Windows 10 Pro Version Professional (OS Build 18363.657)
```

New-PSDriveHere

This function will create a new PSDrive at the specified location. The default is the current location, but you can specify any PSPath. The function will take the last word of the path and use it as the name of the new PSDrive.

PS C:\users\jeff\documents\Enterprise Mgmt Webinar> new-psdrivehere					
Name	Used (GB)	Free (GB)	Provider	Root	CurrentLocation
Webinar		146.57	FileSystem	C:\users\jeff\Documents\Enter...	

Get-MyVariable

This function will return all variables not defined by PowerShell or by this function itself. The default is to return all user-created variables from the global scope but you can also specify a scope such as script, local or a number 0 through 5.

NNName	Value	Type
a	bits	ServiceController
dt	10/22/2018 10:49:38 AM	DateTime
foo	123	Int32
r	{1, 2, 3, 4...}	Object[]
...		

Depending on the value and how PowerShell chooses to display it, you may not see the type.

ConvertFrom-Text

This command can be used to convert text from a file or a command line tool into objects. It uses a regular expression pattern with named captures and turns the result into a custom object. You have the option of specifying a typename in case you are using custom format files.

```

PS C:\> $arp = '(?<IPAddress>(\d{1,3}\.){3}\d{1,3})\s+(?<MAC>(\w{2}-){5}\w{2})\s+(?<Type>\w+$)'
PS C:\> arp -g -N 172.16.10.22 | select -skip 3 | foreach {$_.Trim()} | ConvertFrom-Text $arp -TypeName arpData
-NoProgress

 IPAddress      MAC          Type
-----        ---          ---
 172.16.10.1    b6-fb-e4-16-41-be   dynamic
 172.16.10.100  00-11-32-58-7b-10   dynamic
 172.16.10.115  5c-aa-fd-0c-bf-fa   dynamic
 172.16.10.120  5c-1d-d9-58-81-51   dynamic
 172.16.10.159  3c-e1-a1-17-6d-0a   dynamic
 172.16.10.162  00-0e-58-ce-8b-b6   dynamic
 172.16.10.178  00-0e-58-8c-13-ac   dynamic
 172.16.10.185  d0-04-01-26-b5-61   dynamic
 172.16.10.186  e8-b2-ac-95-92-98   dynamic
 172.16.10.197  fc-77-74-9f-f4-2f   dynamic
 172.16.10.211  14-20-5e-93-42-fb   dynamic
 172.16.10.222  28-39-5e-3b-04-33   dynamic
 172.16.10.226  00-0e-58-e9-49-c0   dynamic
 172.16.10.227  48-88-ca-e1-a6-00   dynamic
 172.16.10.239  5c-aa-fd-83-f1-a4   dynamic
 172.16.255.255 ff-ff-ff-ff-ff-ff   static
 224.0.0.2       01-00-5e-00-00-02   static
 224.0.0.7       01-00-5e-00-00-07   static
 224.0.0.22      01-00-5e-00-00-16   static
 224.0.0.251     01-00-5e-00-00-fb   static
 224.0.0.252     01-00-5e-00-00-fc   static
 239.255.255.250 01-00-5e-7f-ff-fa   static

```

This example uses a previously created and import format.ps1xml file for the custom type name.

Get-PSWho

This command will provide a summary of relevant information for the current user in a PowerShell Session. You might use this to troubleshoot an end-user problem running a script or command.

```

PS C:\> Get-PSWho

User          : BOVINE320\Jeff
Elevated      : True
Computername  : BOVINE320
OperatingSystem : Microsoft Windows 10 Pro [64-bit]
OSVersion     : 10.0.18363
PSVersion     : 5.1.18362.145
Edition       : Desktop
PSHost        : ConsoleHost
WSMan         : 3.0
ExecutionPolicy : RemoteSigned
Culture       : English (United States)

```

You can also turn this into a text block using the **AsString** parameter. This is helpful when you want to include the output in some type of report.

```

Admin: PowerShell 7.0
PS C:\> add-border -textblock (get-pswho -asstring) -ANSIBorder "`e[92m" -border $PSSpecialChar.Lozenge

oooooooooooooooooooooooooooooooooooooooooooo
>User : BOVINE320\Jeff
>Elevated : True
>Computername : BOVINE320
>OperatingSystem : Microsoft Windows 10 Pro [64-bit]
>OSVersion : 10.0.18363
>PSVersion : 7.0.1
>Edition : Core
>PSHost : ConsoleHost
>WSMan : 3.0
>ExecutionPolicy : RemoteSigned
>Culture : English (United States)
oooooooooooooooooooooooooooooooooooooooooooo
PS C:\>

```

Find-CimClass

This function is designed to search an entire CIM repository for a class name. Sometimes, you may have a guess about a class name but not know the full name or even the correct namespace. [Find-CimClass](#) will recursively search for a given classname. You can use wildcards and search remote computers.

```

Administrator: Windows PowerShell 5.1.17134
PS C:\>
PS C:\>

Find-CimClass
Searching for class *protection* in 150 namespaces
[oooooo]                                         ]
processing \\BOVINE320\Root\CIMV2\ms_409

NameSpace: Root/CIMV2/mdm/dmmap

CimClassName          CimClassMethods      CimClassProperties
-----              -----            -----
MDM_AppLocker_EnterpriseDataProt... {}           {InstanceID, ParentID, Policy}
MDM_AppLocker_EnterpriseDataProt... {}           {InstanceID, ParentID, Policy}
MDM_EnterpriseDataProtection {}                  {InstanceID, ParentID, Status}
MDM_EnterpriseDataProtection_Set... {}           {AllowAzureRMSForEDP, AllowUserDecryption, DataRecoveryCert...}
MDM_Policy_Config01_DataProtecti... {}           {AllowDirectMemoryAccess, InstanceID, LegacySelectiveWipeID...}
MDM_Policy_Result01_DataProtecti... {}           {AllowDirectMemoryAccess, InstanceID, LegacySelectiveWipeID...}
MDM_Reportng_EnterpriseDataProt... {}           {InstanceID, LogCount, Logs, ParentID...}
MDM_Reportng_EnterpriseDataProt... {}           {InstanceID, Logs, ParentID, StartTime...}
MDM_WindowsAdvancedThreatProtect... {}           {InstanceID, Offboarding, Onboarding, ParentID}
MDM_WindowsAdvancedThreatProtect... {}           {GroupIds, InstanceID, ParentID, SampleSharing...}
MDM_WindowsAdvancedThreatProtect... {}           {Criticality, Group, IdMethod, InstanceID...}
MDM_WindowsAdvancedThreatProtect... {}           {InstanceID, LastConnected, OnboardingState, OrgId...}

```

Out-VerboseTee

This command is intended to let you see your verbose output and write the verbose messages to a log file. It will only work if the verbose pipeline is enabled, usually when your command is run with -Verbose. This function is designed to be used within your scripts and functions. You either have to hard code a file name or find some other way to define it in your function or control script. You could pass a value as a parameter or set it as a [PSDefaultParameterValue](#).

This command has aliases of [Tee-Verbose](#) and [tv](#).

```
Begin {
    $log = New-RandomFilename -useTemp -extension log
    Write-Detail "Starting $($myinvocation.mycommand)" -Prefix begin | Tee-Verbose $log
    Write-Detail "Logging verbose output to $log" -prefix begin | Tee-Verbose -append
    Write-Detail "Initializing data array" -Prefix begin | Tee-Verbose $log -append
    $data = @()
} #begin
```

When the command is run with -Verbose you will see the verbose output **and** it will be saved to the specified log file.

Remove-Runspace

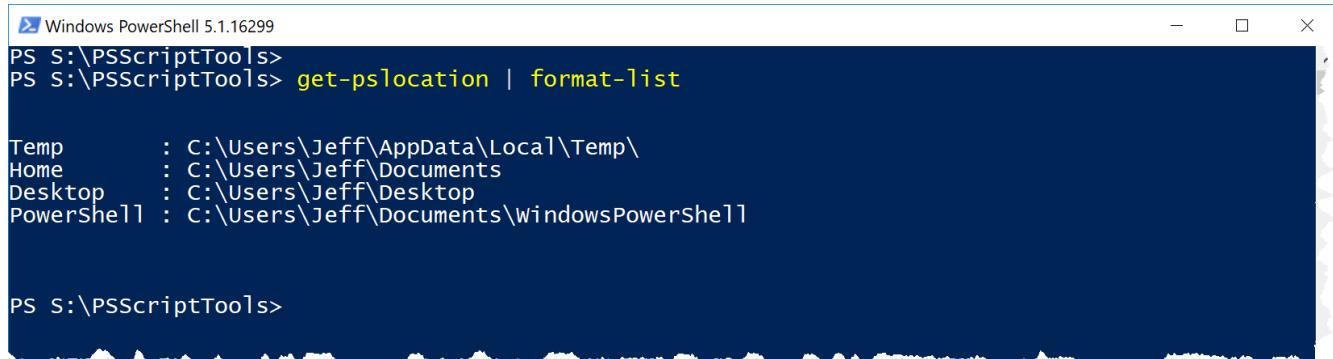
During the course of your PowerShell work, you may discover that some commands and scripts can leave behind runspaces such as [ConvertTo-WPFGrid](#). You may even deliberately be creating additional runspaces. These runspaces will remain until you exit your PowerShell session. Or use this command to cleanly close and dispose of runspaces.

```
PS C:\> Get-RunSpace | where ID -gt 1 | Remove-RunSpace
```

Get all runspaces with an ID greater than 1, which is typically your current session, and remove the runspace.

Get-PSLocation

A simple function to get common locations. This can be useful with cross-platform scripting.



```
Windows PowerShell 5.1.16299
PS S:\PSScriptTools> get-pslocation | format-list

Temp      : C:\Users\Jeff\AppData\Local\Temp\
Home     : C:\Users\Jeff\Documents
Desktop  : C:\Users\Jeff\Desktop
PowerShell : C:\Users\Jeff\Documents\WindowsPowerShell

PS S:\PSScriptTools>
```

```
PS /mnt/c/scripts/PSScriptTools/samples> get-pslocation
Temp   Home      Desktop PowerShell
----  ----  -----
/ttmp/ /home/jhicks    /home/jhicks/.config/powershell

PS /mnt/c/scripts/PSScriptTools/samples>
```

Get-PowerShellEngine

Use this command to quickly get the path to the PowerShell executable. In Windows you should get a result like this:

```
PS C:\> Get-PowerShellEngine
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
```

But PowerShell on non-Windows platforms is a bit different:

```
PS /home/jhicks> Get-PowerShellEngine
/opt/microsoft/powershell/7/pwsh
```

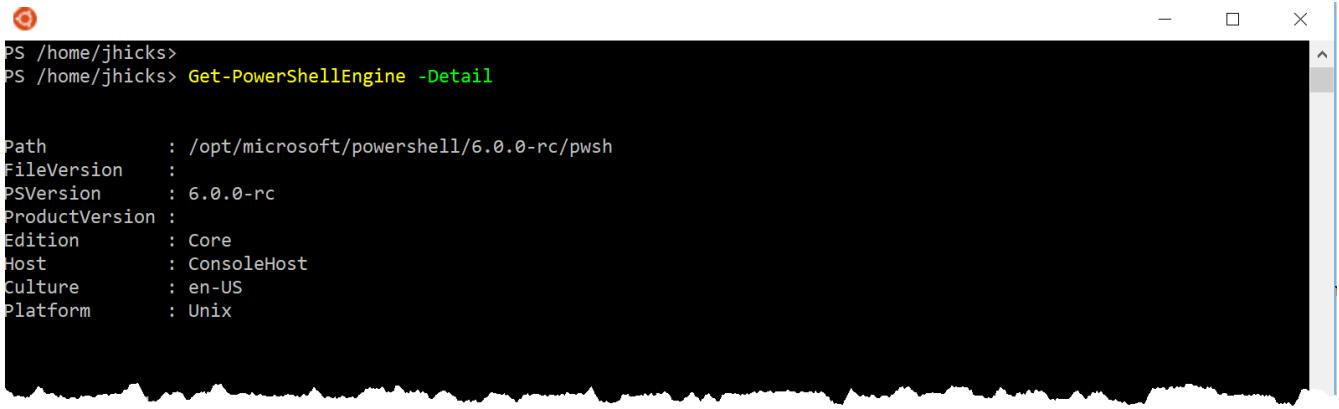
You can also get detailed information.

```
PS S:\> get-powershellengine -Detail

Path          : C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
FileVersion   : 10.0.16299.15 (WinBuild.160101.0800)
PSVersion     : 5.1.16299.64
ProductVersion: 10.0.16299.15
Edition       : Desktop
Host          : ConsoleHost
Culture       : en-US
Platform      :
```

```
PS C:\> get-powershellengine -Detail

Path          : C:\Program Files\PowerShell\6.0.0-rc\pwsh.exe
FileVersion   : 6.0.0
PSVersion     : 6.0.0-rc
ProductVersion: 6.0.0-rc
Edition       : Core
Host          : ConsoleHost
Culture       : en-US
Platform      : Win32NT
```



```
PS /home/jhicks>
PS /home/jhicks> Get-PowerShellEngine -Detail

Path          : /opt/microsoft/powershell/6.0.0-rc/pwsh
FileVersion   :
PSVersion     : 6.0.0-rc
ProductVersion:
Edition       : Core
Host         : ConsoleHost
Culture       : en-US
Platform      : Unix
```

Results will vary depending on whether you are running PowerShell on Windows nor non-Windows systems.

Get-PathVariable

Over time, as you add and remove programs, your %PATH% might change. An application may add a location but not remove it when you uninstall the application. This command makes it easier to identify locations and whether they are still good.

```
PS C:\> Get-PathVariable
```

Scope	UserName	Path	Exists
User	Jeff	C:\Program Files\kdiff3	True
User	Jeff	C:\Program Files (x86)\Bitvise SSH Client	True
User	Jeff	C:\Program Files\OpenSSH	True
User	Jeff	C:\Program Files\Intel\WiFi\bin\	True
User	Jeff	C:\Program Files\Common Files\Intel\WirelessCommon\	True
User	Jeff	C:\Users\Jeff\AppData\Local\Programs\Microsoft VS Code\bin	True
User	Jeff	C:\Program Files (x86)\Vale\	True
...			

File Tools

Test-EmptyFolder

This command will test if a given folder path is empty of all files anywhere in the path. This includes hidden files. The command will return True even if there are empty sub-folders. The default output is True or False but you can use -Passthru to get more information.

```
PS C:\> Get-Childitem c:\work -Directory | test-EmptyFolder -passthru | Where-object {$_.IsEmpty} | Foreach-Object {  
    Remove-Item -LiteralPath $_.path -Recurse -force -whatif}  
What if: Performing the operation "Remove Directory" on target "C:\work\demo3".  
What if: Performing the operation "Remove Directory" on target "C:\work\installers".  
What if: Performing the operation "Remove Directory" on target "C:\work\new".  
What if: Performing the operation "Remove Directory" on target "C:\work\sqlback".  
What if: Performing the operation "Remove Directory" on target "C:\work\todd".  
What if: Performing the operation "Remove Directory" on target "C:\work\[data]".
```

Find all empty sub-folders under C:\Work and pipe them to Remove-Item. This is one way to remove empty folders. The example is piping objects to ForEach-Object so that Remove-Item can use the -LiteralPath parameter, because C:\work[data] is a non-standard path.

Get-FolderSizeInfo

Use this command to quickly get the size of a folder. You also have an option to include hidden files. The command will measure all files in all subdirectories.

```
PS C:\> get-foldersizeinfo c:\work  


| Computername | Path    | TotalFiles | TotalSize |
|--------------|---------|------------|-----------|
| BOVINE320    | C:\work | 931        | 137311146 |

  
PS C:\> get-foldersizeinfo c:\work -Hidden  


| Computername | Path    | TotalFiles | TotalSize |
|--------------|---------|------------|-----------|
| BOVINE320    | C:\work | 1375       | 137516856 |


```

The command includes a format file with additional view to display the total size in KB, MB, GB or TB.

```
PS C:\> Get-ChildItem D:\ -Directory | Get-FolderSizeInfo -Hidden | Where-Object TotalSize -gt 1gb | Sort-Object TotalSize -Descending | Format-Table -View gb
```

Computername	Path	TotalFiles	TotalSizeGB
BOVINE320	D:\Autolab	159	137.7192
BOVINE320	D:\VMDisks	18	112.1814
BOVINE320	D:\ISO	17	41.5301
BOVINE320	D:\FileHistory	104541	36.9938
BOVINE320	D:\Vagrant	13	19.5664
BOVINE320	D:\Vms	83	5.1007
BOVINE320	D:\2016	1130	4.9531
BOVINE320	D:\video	125	2.592
BOVINE320	D:\blog	21804	1.1347
BOVINE320	D:\pstranscripts	122092	1.0914

Or you can use the `name` view.

```
PS C:\> Get-ChildItem c:\work -Directory | Get-FolderSizeInfo -Hidden | Where-Object {$_.totalsize -ge 2mb} | Format-Table -view name
```

Path: C:\work

Name	TotalFiles	TotalKB
A	20	5843.9951
keepass	15	5839.084
PowerShellBooks	26	4240.3779
sunday	47	24540.6523

Optimize-Text

Use this command to clean and optimize content from text files. Sometimes text files have blank lines or the content has trailing spaces. These sorts of issues can cause problems when passing the content to other commands.

This command will strip out any lines that are blank or have nothing by white space, and trim leading and trailing spaces. The optimized text is then written back to the pipeline. Optionally, you can specify a property name. This can be useful when your text file is a list of computer names and you want to take advantage of pipeline binding.

Get-FileItem

A PowerShell version of the CLI `where.exe` command. You can search with a simple or regex pattern.

```
PS C:\> pswhere winword.exe -Path c:\ -Recurse -first  
C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE
```

Note that you might see errors for directories where you don't have access permission. This is normal.

New-CustomFileName

This command will generate a custom file name based on a template string that you provide.

```
PS C:\> New-CustomFileName %computername_%day%monthname%yr-%time.log  
COWPC_28Nov19-142138.log
```

```
PS C:\> New-CustomFileName %dayofweek-#####.dat  
Tuesday-3128.dat
```

You can create a template string using any of these variables. Most of these should be self-explanatory.

- %username
- %computername
- %year - 4 digit year
- %yr - 2 digit year
- %monthname - The abbreviated month name
- %month - The month number
- %dayofweek - The full name of the week day
- %day
- %hour
- %minute
- %time
- %string - A random string
- %guid

You can also insert a random number using **%** followed by a **#** character for each digit you want.

```
22 = %##  
654321 = %#####
```

New-RandomFileName

Create a new random file name. The default is a completely random name including the extension.

```
PS C:\> new-randomfilename  
fykxecvh.ipw
```

But you can specify an extension.

```
PS C:\> new-randomfilename -extension dat  
emevgq3r.dat
```

Optionally you can create a random file name using the TEMP folder or your HOME folder. On Windows platforms this will default to your Documents folder.

```
PS C:\> new-randomfilename -extension log -UseHomeFolder  
C:\Users\Jeff\Documents\kbyw4fda.log
```

On Linux machines it will be the home folder.

```
PS /mnt/c/scripts> new-randomfilename -home -Extension tmp  
/home/jhicks/oces0epq.tmp
```

ConvertTo-Markdown

This command is designed to accept pipelined output and create a markdown document. The pipeline output will be formatted as a text block or a table. You can optionally define a title, content to appear before the output and content to appear after the output. You can run a command like this:

```
Get-Service Bits,Winrm | Convertto-Markdown -title "Service Check" -precontent "## $($env:computername)" -postcontent  
"_report $(Get-Date)_"
```

which generates this markdown:

```
# Service Check  
  
## BOVINE320  
  
```text  

Status Name DisplayName
----- ----+
Running Bits Background Intelligent Transfer Ser...
Running Winrm Windows Remote Management (WS-Manag...
```\br/>  
_report 09/25/2019 09:57:12_
```

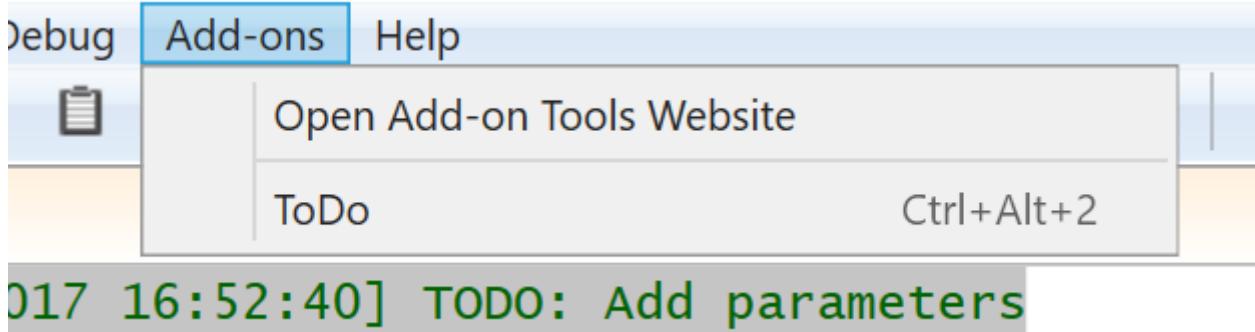
Because the function writes markdown to the pipeline you will need to pipe it to a command **Out-File** to create a file.

ToDo

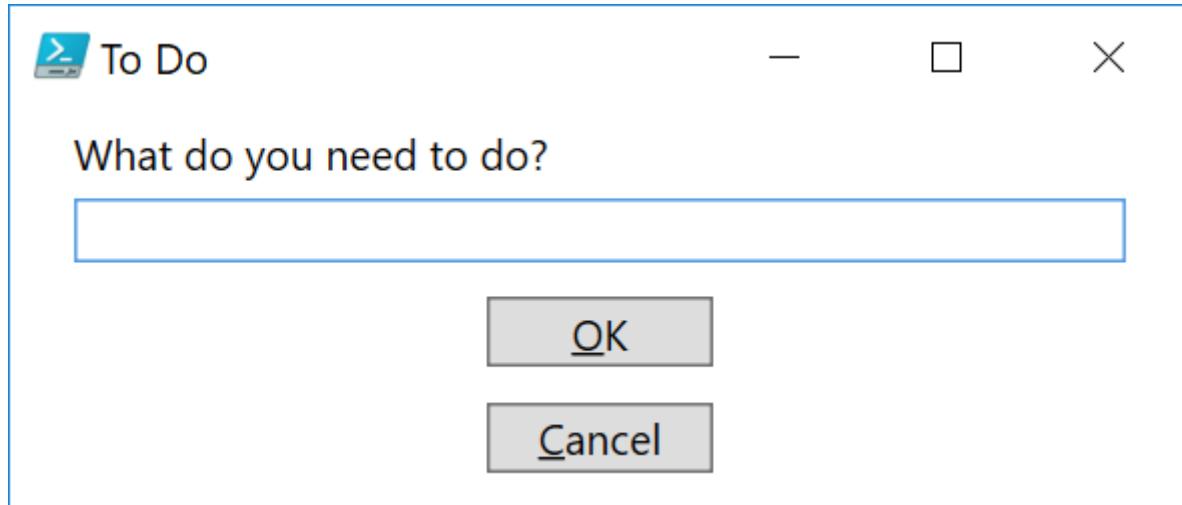
Because this module is intended to make scripting easier for you, it adds options to insert ToDo statements into PowerShell files. If you are using the PowerShell ISE or VS Code and import this module, it will add the capability to insert a line like this:

```
# [12/13/2018 16:52:40] TODO: Add parameters
```

In the PowerShell ISE, you will get a new menu under Add-Ons.



You can use the menu or keyboard shortcut which will launch an input box.



The comment will be inserted at the current cursor location.

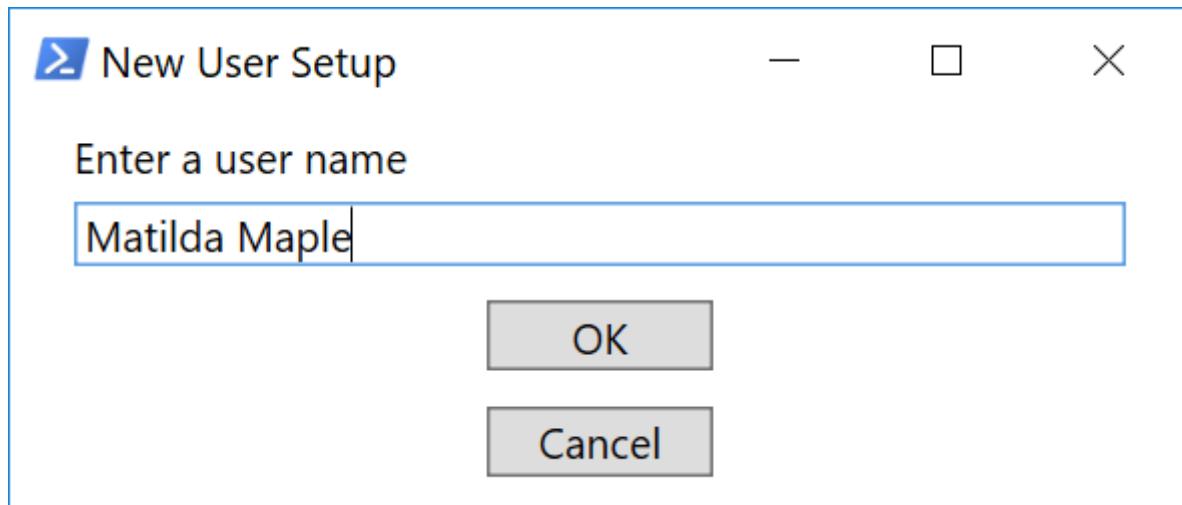
In VS Code, access the command palette (Ctrl+Shift+P) and then [PowerShell: Show Additional Commands from PowerShell Modules](#). Select [Insert ToDo](#) from the list and you'll get the same input box. Note that this will only work for PowerShell files.

Graphical Tools

Invoke-InputBox

This function is a graphical replacement for `Read-Host`. It creates a simple WPF form that you can use to get user input. The value of the text box will be written to the pipeline.

```
$name = Invoke-InputBox -Prompt "Enter a user name" -Title "New User Setup"
```



You can also capture a secure string.

```
Invoke-Inputbox -Prompt "Enter a password for $Name" -AsSecureString -BackgroundColor red
```



This example also demonstrates that you can change form's background color. This function will **not** work in PowerShell Core.

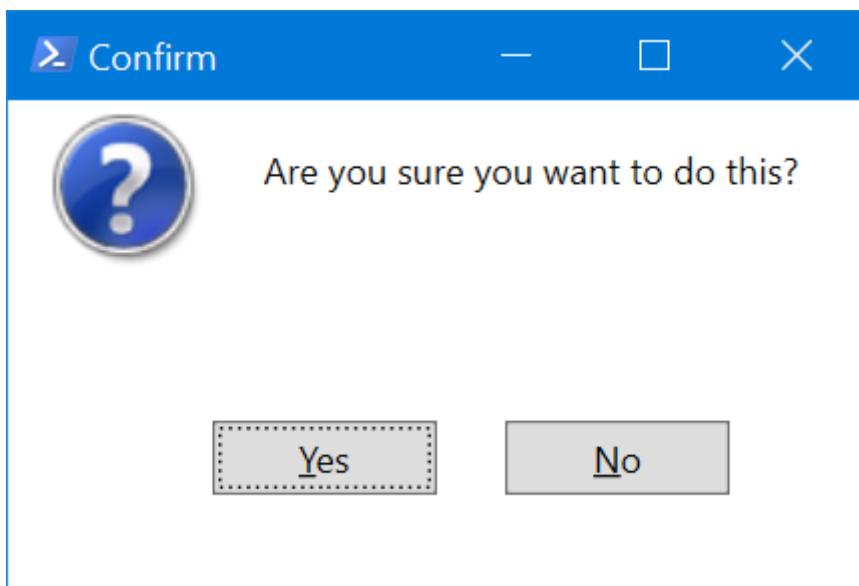
New-WPFMessageBox

This function creates a Windows Presentation Foundation (WPF) based message box. This is intended to replace the legacy MsgBox function from VBScript and the Windows Forms library. The command uses a set of predefined button sets, each of which will close the form and write a value to the pipeline.

- OK = 1
- Cancel = 0
- Yes = \$True
- No = \$False

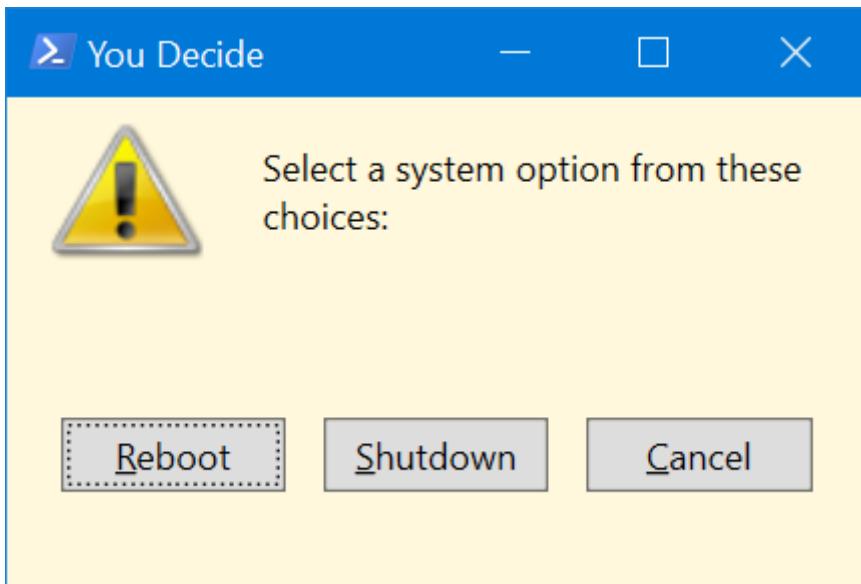
You can also create an ordered hashtable of your own buttons and values. It is assumed you will typically use this function in a script where you can capture the output and take some action based on the value.

```
PS C:\> New-WPFMessageBox -Message "Are you sure you want to do this?" -Title Confirm -Icon Question -ButtonSet YesNo
```



You can also create your own custom button set as well as modify the background color.

```
PS C:\> New-WPFMessageBox -Message "Select a system option from these choices:" -Title "You Decide" -Background cornsilk -Icon Warning -CustomButtonSet ([ordered]@{"Reboot"=1;"Shutdown"=2;"Cancel"=3})
```



ConvertTo-WPFGrid

This command is an alternative to `Out-GridView`. It works much the same way. Run a PowerShell command and pipe it to this command. The output will be displayed in an auto-sized data grid. You can click on column headings to sort. You can resize columns and you can re-order columns.

```
get-eventlog -list -ComputerName DOM1,SRV1,SRV2 |  
Select MachineName, Log, MaximumKilobytes, Overflowaction,  
@{Name="RetentionDays";Expression={$_.'MinimumRetentionDays'}},  
@{Name="Entries";Expression = {$_.'Entries.Count}} |  
ConvertTo-WPFGrid -Title "Event Log Report"
```

Event Log Report

MachineName	Log	MaximumKilobytes	OverflowAction	RetentionDays	Entries
DOM1	Active Directory Web Services	512	OverwriteOlder	7	38
DOM1	Application	20480	OverwriteAsNeeded	0	16722
DOM1	DFS Replication	15168	OverwriteAsNeeded	0	39
DOM1	Directory Service	512	OverwriteAsNeeded	0	291
DOM1	DNS Server	102400	OverwriteAsNeeded	0	2762
DOM1	HardwareEvents	20480	OverwriteAsNeeded	0	0
DOM1	Internet Explorer	512	OverwriteOlder	7	0
DOM1	Key Management Service	20480	OverwriteAsNeeded	0	0
DOM1	Security	131072	OverwriteAsNeeded	0	194724
DOM1	System	20480	OverwriteAsNeeded	0	8105
DOM1	Windows PowerShell	15360	OverwriteAsNeeded	0	459
SRV1	Application	20480	OverwriteAsNeeded	0	17377
SRV1	HardwareEvents	20480	OverwriteAsNeeded	0	0
SRV1	Internet Explorer	512	OverwriteOlder	7	0
SRV1	Key Management Service	20480	OverwriteAsNeeded	0	0
SRV1	Security	20480	OverwriteAsNeeded	0	30287
SRV1	System	20480	OverwriteAsNeeded	0	7463
SRV1	Windows PowerShell	15360	OverwriteAsNeeded	0	1209
SRV2	Application	20480	OverwriteAsNeeded	0	15885
SRV2	HardwareEvents	20480	OverwriteAsNeeded	0	0
SRV2	Internet Explorer	512	OverwriteOlder	7	0
SRV2	Key Management Service	20480	OverwriteAsNeeded	0	0
SRV2	Security	20480	OverwriteAsNeeded	0	17536
SRV2	System	20480	OverwriteAsNeeded	0	6529
SRV2	Windows PowerShell	15360	OverwriteAsNeeded	0	274

last updated 02/12/2019 20:27:44

You can also have automatically refresh the data.

```
get-process | sort-object WS -Descending | Select -first 20 ID,Name,WS,VM,PM,Handles,StartTime | Convertto-WPFGrid -Refresh -timeout 20 -Title "Top Processes"
```

Top Processes

	Id	Name	WS	VM	PM	Handles	StartTime
3092	Memory Compression	1747447808	1814691840	4161536	0		1/30/2019 9:46:01 AM
16728	Code	745938944	2204812578816	775516160	638		2/12/2019 4:35:06 PM
1472	dwm	373415936	2205525884928	1036087296	1292		1/30/2019 9:46:01 AM
11976	slack	361508864	2204708483072	307527680	491		2/12/2019 1:02:50 PM
6984	Code	248057856	2205164032000	244703232	530		2/12/2019 4:35:06 PM
12612	thunderbird	232312832	1015586816	324341760	1028		2/12/2019 9:11:11 AM
2544	brave	202432512	2204164759552	217972736	423		2/12/2019 10:39:54 AM
13968	slack	187772928	2204290772992	237174784	714		1/30/2019 9:47:14 AM
14780	SugarSync	183324672	464482304	241012736	1208		1/30/2019 9:46:38 AM
13972	brave	168230912	2238674284544	302723072	2777		2/7/2019 9:32:50 AM
3016	powershell	164237312	2204586930176	322105344	1620		2/12/2019 12:32:00 PM
3844	Code	153067520	2204334030848	159526912	451		2/12/2019 4:35:08 PM
22104	powershell	140689408	2204716986368	924684288	874		2/12/2019 4:35:10 PM
2088	ekrn	138780672	2203913326592	122093568	956		1/30/2019 9:46:01 AM
17264	slack	122294272	2205174013952	131186688	2617		1/30/2019 9:47:04 AM
7016	sqlservr	120057856	46579154944	637329408	715		1/30/2019 9:46:02 AM
24040	brave	114626560	2204050501632	137789440	418		2/12/2019 9:37:56 AM
9996	explorer	109133824	2204844474368	195948544	3396		1/30/2019 9:46:08 AM
22984	WmiPrvSE	99557376	2204058812416	81039360	1095		2/12/2019 8:18:23 PM
17964	ONENOTE	98279424	637419520	60428288	1149		2/12/2019 5:49:47 PM

Last updated 02/12/2019 20:18:47 - refresh in 6 seconds

Note that in v2.4.0 the form layout was modified and may not be reflected in these screen shots.

Hashtable Tools

Convert-CommandtoHashtable

This command is intended to convert a long PowerShell expression with named parameters into a splatting alternative.

```
PS C:\> Convert-CommandtoHashtable -Text "get-eventlog -listlog -computername a,b,c,d -erroraction stop"

$paramHash = @{
    listlog = $True
    computername = "a","b","c","d"
    erroraction = "stop"
}

Get-EventLog @paramHash
```

Convert-HashtableString

This function is similar to [Import-PowerShellDataFile](#). But where that command can only process a file, this command will take any hashtable-formatted string and convert it into an actual hashtable.

```
PS C:\> get-content c:\work\test.psd1 | unprotect-cmsmessage | Convert-HashtableString
```

Name	Value
CreatedBy	BOVINE320\Jeff
CreatedAt	10/02/2018 21:28:47 UTC
Computername	Think51
Error	
Completed	True
Date	10/02/2018 21:29:35 UTC
Scriptblock	restart-service spooler -force
CreatedOn	BOVINE320

The test.psd1 file is protected as a CMS Message. In this example, the contents are decoded as a string which is then in turn converted into an actual hashtable.

Convert-HashTableToCode

Use this command to convert a hashtable into its text or string equivalent.

```
PS C:\> $h = @{Name="SRV1";Asset=123454;Location="Omaha"}  
PS C:\> convert-hashtabletocode $h  
@[  
    Name = 'SRV1'  
    Asset = 123454  
    Location = 'Omaha'  
]
```

Convert a hashtable object to a string equivalent that you can copy into your script.

ConvertTo-HashTable

This command will take an object and create a hashtable based on its properties. You can have the hashtable exclude some properties as well as properties that have no value.

```
PS C:\> Get-Process -id $pid | select name,id,handles,workingset | ConvertTo-HashTable  


| Name       | Value          |
|------------|----------------|
| WorkingSet | 418377728      |
| Name       | powershell_ise |
| Id         | 3456           |
| Handles    | 958            |


```

Join-HashTable

This command will combine two hashtables into a single hashtable. Join-Hashtable will test for duplicate keys. If any of the keys from the first, or primary hashtable are found in the secondary hashtable, you will be prompted for which to keep. Or you can use -Force which will always keep the conflicting key from the first hashtable.

```

PS C:\> $a=@{Name="Jeff";Count=3;Color="Green"}
PS C:\> $b=@{Computer="HAL";Enabled=$True;Year=2020;Color="Red"}
PS C:\> join hashtable $a $b
Duplicate key Color
A Green
B Red
Which key do you want to KEEP \[AB\]?: A

```

Name	Value
---	-----
Year	2020
Name	Jeff
Enabled	True
Color	Green
Computer	HAL
Count	3

Rename-Hashtable

This command allows you to rename a key in an existing hashtable or ordered dictionary object.

```
PS C:\> $h = Get-Service Spooler | ConvertTo-HashTable
```

The hashtable in \$h has Machinename property which can be renamed.

PS C:\> Rename-HashTable -Name h -Key Machinename -NewKey Computername -Passthru	
Name	Value
---	-----
ServiceType	Win32OwnProcess, InteractiveProcess
ServiceName	Spooler
Container	
CanPauseAndContinue	False
RequiredServices	{RPCSS, http}
ServicesDependedOn	{RPCSS, http}
Computername	.
CanStop	True
StartType	Automatic
Site	
ServiceHandle	SafeServiceHandle
DisplayName	Print Spooler
CanShutdown	False
Status	Running
Name	Spooler
DependentServices	{Fax}

Select Functions

The module contains 2 functions which simplify the use of [Select-Object](#). The commands are intended to make it easier to select the first or last X number of objects. The commands include features so that you can sort the incoming objects on a given property first.

```
PS C:\> get-process | select-first 5 -Property WS -Descending
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
696	89	615944	426852	391.97	7352	0	sqlservr
541	78	262532	274576	278.41	6208	8	Code
1015	70	227824	269504	137.39	16484	8	powershell_ise
1578	111	204852	254640	98.58	21332	8	firefox
884	44	221872	245712	249.23	12456	8	googledrivesync

Time Functions

The module has a few date and time related commands.

ConvertTo-UTCTime

Convert a local datetime value to universal time. The default is to convert now but you can specify a datetime value.

```
PS C:\> ConvertTo-UTCTime
```

```
Monday, March 4, 2019 5:51:26 PM
```

Convert a datetime that is UTC-5 to universal time.

ConvertFrom-UTCTime

```
PS C:\> ConvertFrom-UTCTime "3/4/2019 6:00PM"
```

```
Monday, March 4, 2019 1:00:00 PM
```

Convert a universal datetime to the local time.

Get-MyTimeInfo

Display a time settings for a collection of locations. This command is a PowerShell equivalent of a world clock. It will display a datetime value against a collection of locations. You can specify an ordered hashtable of locations and time zones. You can run command like:

```
[System.TimeZoneInfo]::GetSystemTimeZones() | out-gridview
```

or

```
Get-TimeZone -listavailable
```

To discover time zone names. Note that the ID is case-sensitive. You can then use the command like this:

```
PS C:\> Get-MyTimeInfo -Locations ([ordered]@{Seattle="Pacific Standard time";"New Zealand" = "New Zealand Standard Time"}) -HomeTimeZone "central standard time" | Select Now,Home,Seattle,'New Zealand'

Now           Home          Seattle        New Zealand
---           ---          -----        -----
3/4/2019 1:18:36 PM 3/4/2019 12:18:36 PM 3/4/2019 10:18:36 AM 3/5/2019 7:18:36 AM
```

This is a handy command when traveling and your laptop is using a locally derived time and you want to see the time in other locations. It is recommended that you set a PSDefaultParameter value for the HomeTimeZone parameter in your PowerShell profile.

ConvertTo-LocalTime

It can be tricky sometimes to see a time in a foreign location and try to figure out what that time is locally. This command attempts to simplify this process. In addition to the remote time, you need the base UTC offset for the remote location.

```
PS C:\> get-timezone -ListAvailable | where id -match hawaii
```

Id	:	Hawaiian Standard Time
DisplayName	:	(UTC-10:00) Hawaii
StandardName	:	Hawaiian Standard Time
DaylightName	:	Hawaiian Daylight Time
BaseUtcOffset	:	-10:00:00
SupportsDaylightSavingTime	:	False

```
PS C:\> ConvertTo-LocalTime "10:00AM" -10:00:00
```

```
Thursday, March 14, 2019 4:00:00 PM
```

In this example, the user first determines the UTC offset for Hawaii. Then 10:00AM in say Honolulu, is converted to local time which in this example is in the Eastern Time zone.

Get-TZList

This command uses a free and publicly available REST API offered by <http://worldtimeapi.org> to get a list of time zone areas. You can get a list of all areas or by geographic location. Use Get-TZData to then retrieve details.

```
PS C:\> get-tzlist Australia
Australia/Adelaide
Australia/Brisbane
Australia/Broken_Hill
Australia/Currie
Australia/Darwin
Australia/Eucla
Australia/Hobart
Australia/Lindeman
Australia/Lord_Howe
Australia/Melbourne
Australia/Perth
Australia/Sydney
```

Get-TZData

This command also uses the API from worldtimeapi.org to retrieve details about a give time zone area.

```
PS C:\> Get-TZData Australia/Hobart
```

Timezone	Label	Offset	DST	Time
-----	-----	-----	---	----
Australia/Hobart	AEDT	11:00:00	True	3/16/2019 3:43:14 AM

The Time value is the current time at the remote location. The command presents a formatted object but you can also get the raw data.

```
PS C:\> Get-TZData Australia/Hobart -Raw
```

```
week_number : 11
utc_offset   : +11:00
unixtime    : 1552668285
timezone    : Australia/Hobart
dst_until    : 2019-04-06T16:00:00+00:00
dst_from     : 2018-10-06T16:00:00+00:00
dst          : True
day_of_year  : 75
day_of_week  : 6
datetime    : 2019-03-16T03:44:45.689655+11:00
abbreviation : AEDT
```

ConvertTo-LexicalTime

When working with timespans or durations in XML files, such as those from scheduled tasks, the format is a little different than what you might expect. The specification is described at <https://www.w3.org/TR/xmlschema-2/#duration>. Use this command to convert a timespan into a lexical format you can use in an XML file where you need to specify a duration.

```
PS C:\> ConvertTo-LexicalTimespan (New-TimeSpan -Days 7 -hours 12)
```

```
P7DT12H
```

ConvertFrom-LexicalTime

Likewise, you might need to convert a lexical value back into a timespan.

```
PS C:\> ConvertFrom-LexicalTimespan P7DT12H
```

```
Days          : 7
Hours         : 12
Minutes       : 0
Seconds       : 0
Milliseconds  : 0
Ticks         : 64800000000000
TotalDays     : 7.5
TotalHours    : 180
TotalMinutes  : 10800
TotalSeconds  : 648000
TotalMilliseconds : 648000000
```

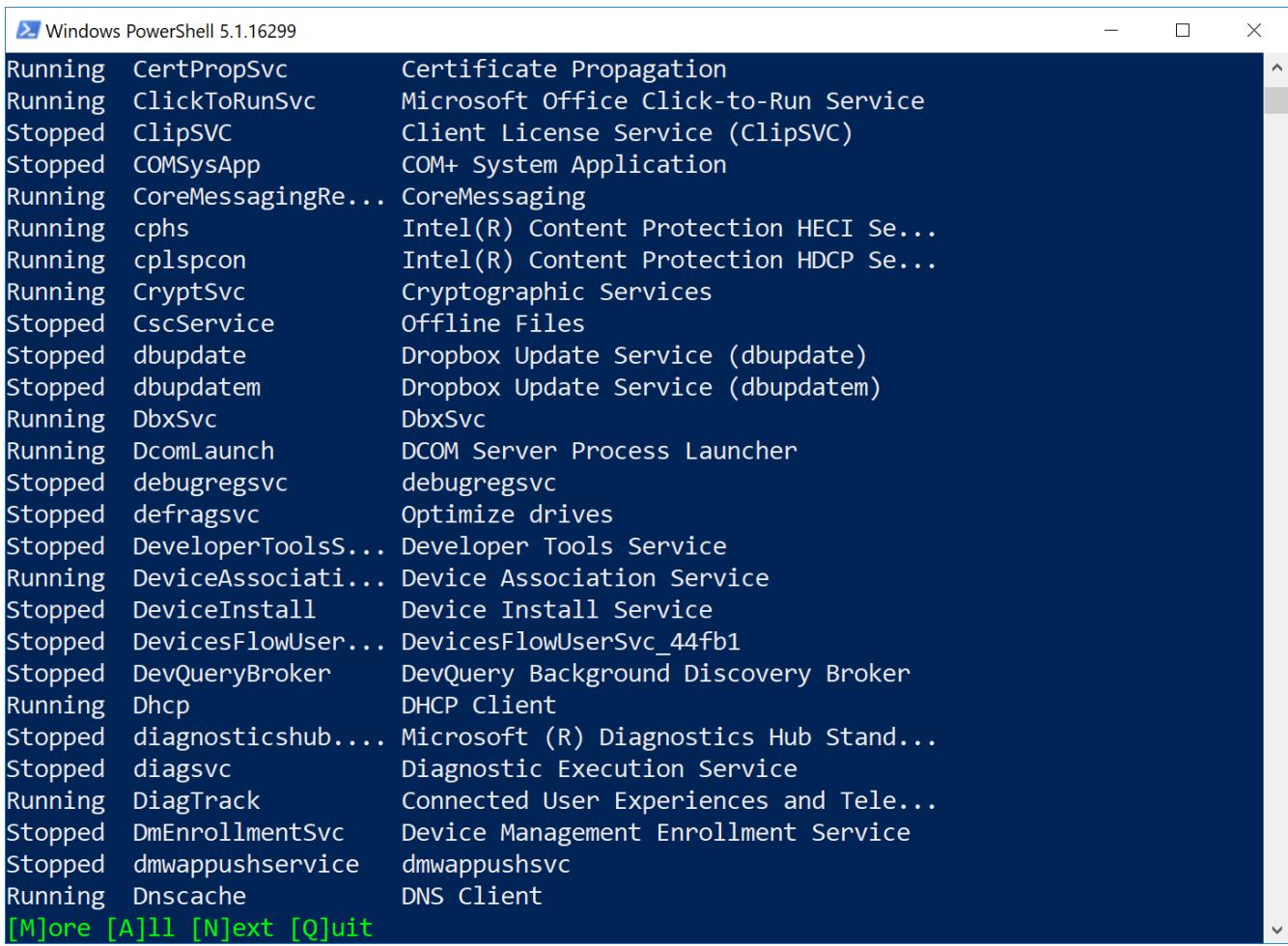
These functions were first described at <https://dhitsolutions.com/blog/powershell/7101/converting-lexical-timespans-with-powershell/>

Console Utilities

Out-More

This command provides a PowerShell alternative to the cmd.exe **MORE** command, which doesn't work in the PowerShell ISE. When you have screens of information, you can page it with this function.

```
get-service | out-more
```



Running	CertPropSvc	Certificate Propagation
Running	ClickToRunSvc	Microsoft Office Click-to-Run Service
Stopped	ClipSVC	Client License Service (ClipSVC)
Stopped	COMSysApp	COM+ System Application
Running	CoreMessagingRe...	CoreMessaging
Running	cphs	Intel(R) Content Protection HECI Se...
Running	cplspccon	Intel(R) Content Protection HDCP Se...
Running	CryptSvc	Cryptographic Services
Stopped	CscService	Offline Files
Stopped	dbupdate	Dropbox Update Service (dbupdate)
Stopped	dbupdatem	Dropbox Update Service (dbupdatem)
Running	DbxSvc	DbxSvc
Running	DcomLaunch	DCOM Server Process Launcher
Stopped	debugregsvc	debugregsvc
Stopped	defragsvc	Optimize drives
Stopped	DeveloperToolsS...	Developer Tools Service
Running	DeviceAssociati...	Device Association Service
Stopped	DeviceInstall	Device Install Service
Stopped	DevicesFlowUser...	DevicesFlowUserService_44fb1
Stopped	DevQueryBroker	DevQuery Background Discovery Broker
Running	Dhcp	DHCP Client
Stopped	diagnosticshub....	Microsoft (R) Diagnostics Hub Stand...
Stopped	diagsvc	Diagnostic Execution Service
Running	DiagTrack	Connected User Experiences and Tele...
Stopped	DmEnrollmentSvc	Device Management Enrollment Service
Stopped	dmwappushservice	dmwappushsvc
Running	Dnscache	DNS Client

This also works in PowerShell Core.

Out-ConditionalColor

This command is designed to take pipeline input and display it in a colorized format, based on a set of conditions. Unlike **Write-Host** which doesn't write to the pipeline, this command will write to the pipeline. You can use a simple hashtable to define a color if the given property matches the hashtable key.

```

Windows PowerShell 5.1.16299
PS C:\> Get-Service | Out-ConditionalColor -PropertyConditions @{{Stopped="magenta"} } -property Status
Status      Name          DisplayName
----      --      -----
Stopped    AdobeFlashPlaye... Adobe Flash Player Update Service
Stopped    AJRouter       AllJoyn Router Service
Stopped    ALG           Application Layer Gateway Service
Stopped    AppIDSvc       Application Identity
Running    Appinfo        Application Information
Running    AppMgmt        Application Management
Stopped    AppReadiness   App Readiness
Stopped    AppClient      Microsoft App-V Client
Stopped    AppXSvc        AppX Deployment Service (AppXSVC)
Stopped    AssignedAccessM... AssignedAccessManager Service
Running    AudioEndpointBu... Windows Audio Endpoint Builder
Running    Audiosrv       Windows Audio
Stopped    AxInstSV      ActiveX Installer (AxInstSV)
Stopped    BDESVC         BitLocker Drive Encryption Service
Running    BFE            Base Filtering Engine
Running    BITS           Background Intelligent Transfer Ser...
Stopped    BoxSyncUpdateSe... Box Sync Update Service
Running    BrokerInfrastru... Background Tasks Infrastructure Ser...
Stopped    BthHFSrv       Bluetooth Handsfree Service
Running    bthserv        Bluetooth Support Service
Stopped    camsvc         Capability Access Manager Service
Running    CDPSvc         Connected Devices Platform Service

```

Or you can specify an ordered hashtable for more complex processing.

```

Windows PowerShell 5.1.16299
PS C:\> $h=[ordered]@{
>> {$psitem.ws -gt 500mb}='red'
>> {$psitem.ws -gt 300mb}='yellow'
>> {$psitem.ws -gt 200mb}='cyan'
>> }
PS C:\> get-process | sort ws -descending | Out-ConditionalColor -conditions $h
Handles  NPM(K)    PM(K)      WS(K)      CPU(s)      Id  SI ProcessName
----  --      --      --      --      --  --  --
  907     69  596376    579476    64.70  15576  2 powershell
  775    147  638696    496924   988.75  10892  2 firefox
  732     94  672056    450124   406.59   6548  0 sqlservr
  987    143  482196    391804  1,661.95  14824  2 firefox
1568    145  340752    376272  1,037.64   800  2 firefox
2740    101  370604    336400   44.22  26280  2 Microsoft.Photos
  610     71  278152    304764   44.42  13828  2 firefox
  537    104  283060    297924  142.77  22156  2 firefox
  565     85  251944    282352   418.81  24148  2 Code
3411    156  329504    277424   87.31  14412  2 SnagitEditor
  475     64  189656    224372  164.09  12812  2 slack
  483     66  196240    203720  338.50  9464  2 slack
1075    123  200904    183272   10.33  23168  2 Snagit32
  468     62  178908    180276   215.70  18300  2 slack
1062     57  186472    177460  3,327.45  13316  2 SugarSync
  465     62  174072    171168   232.03  18628  2 slack
  455     61  170240    168932   35.80  20420  2 slack
  438     60  171432    168016   276.92  23188  2 slack
  454     61  168456    167448   133.80   340  2 slack

```

This command doesn't always work depending on the type of object you pipe to it. The problem appears to be related to the formatting system. Development and testing is ongoing.

Set-ConsoleTitle

Set the title bar of the current PowerShell console window.

```
PS C:\> if (Test-IsAdministrator) { Set-ConsoleTitle "Administrator: $($PSVersionTable.PSedition) $($PSVersionTable.PSVersion)" -Verbose }
VERBOSE: [10:33:17.0420820 BEGIN ] Starting Set-ConsoleTitle
VERBOSE: [10:33:17.0440568 PROCESS] Setting console title to Administrator: Desktop 5.1.17763.316
VERBOSE: Performing the operation "Set-ConsoleTitle" on target "Administrator: Desktop 5.1.17763.316".
VERBOSE: [10:33:17.0584056 END   ] Ending Set-ConsoleTitle
```

Set-ConsoleColor

Configure the foreground or background color of the current PowerShell console window. Note that if you are running the PSReadline module, this command won't work. You should use [Set-PSReadlineOption](#) or similar command to configure your session settings.

```
Set-ConsoleColor -background DarkGray -foreground Yellow
```

Add-Border

This command will create a character or text based border around a line of text. You might use this to create a formatted text report or to improve the display of information to the screen.

```
PS C:\> add-border $env:computername
*****
* COWPC *
*****
```

Starting in v2.23.0 you can also use ANSI escape sequences to color the text and/or the border.

```
PS C:\> add-border -Text "Today is a good day for PowerShell" -ANSIBorder ``e[38;5;47m" -ANSIText ``e[93m"
*****
* Today is a good day for PowerShell *
*****
```

Show-Tree

Shows the specified path as a graphical tree in the console. This is intended as PowerShell alternative to the tree DOS command. This function should work for any type of PowerShell provider and can be used to explore providers used for configuration like the WSMAN provider or the registry. By default, the output will only show directory or equivalent structures. But you can opt to include items well as item details.

```
Administrator: C:\Program Files\PowerShell\6\pwsh.exe
PS C:\> show-tree c:\work
C:\work
+--A
| \--B
|   \--C
+--dnssuffix
  +-docs
  +-en-us
  \--images
---gpo
  ---{65D9E940-AAD4-4508-A199-86EAE4E9E535}
    \--DomainSysvol
      \--GPO
        +-Machine
          +-Applications
          +-microsoft
            \--windows nt
              \--SecEdit
        +-Preferences
          +-Folders
          \--NetworkShares
        \--Scripts
          +-Shutdown
          \--Startup
        \--User
  \--{7E7F01CE-6889-44B0-9D03-818F8284EDE0}
    \--DomainSysvol
      \--GPO
        +-Machine
          +-Applications
```

If you are running PowerShell 7 and specifying a file system path, you can display the tree in a colorized format by using the `-InColor` dynamic parameter.

```
PS C:\> pstree c:\work\alpha -ShowItem -InColor
C:\work\alpha
+--bravo
| +-delta
| | +-FunctionDemo.ps1
| | +-function-form.ps1
| | +-function-logstamp.ps1
| | +-FunctionNotes.ps1
| | \--Function-SwitchTest.ps1
| +-gamma
| | \--x.txt
| +-images
| | +-wpfbox-1.png
| | +-wpfbox-2.png
| | +-wpfgrid.png
| | \--wpfgrid2.png
| +-data.txt
| +-sample-1.json
| +-sample-2.json
| +-sample-3.json
| +-sample-4.json
| \--something2.xml
+-documents-log.csv
+-dropbox-log.csv
+-GoogleDrive-log.csv
+-junk.txt
+-Scripts-log.csv
+-stuff.tmp
\--test.data
PS C:\>
```

Beginning with v2.21.0, this command uses ANSI Color schemes from a json file. You can customize the file if you wish. See the [PSAnsiMap](#) section of this README.

This command has an alias of `pstree`.

```
PS C:\> pstree c:\work\alpha -files -properties LastWriteTime,Length

C:\work\Alpha\
+-- LastWriteTime = 02/28/2020 11:19:32
+--bravo
|   +-- LastWriteTime = 02/28/2020 11:20:30
|   +--delta
|   |   +-- LastWriteTime = 02/28/2020 11:17:35
|   |   +--FunctionDemo.ps1
|   |   |   +-- Length = 888
|   |   |   \-- LastWriteTime = 06/01/2009 15:50:47
|   |   +--function-form.ps1
|   |   |   +-- Length = 1117
|   |   |   \-- LastWriteTime = 04/17/2019 17:18:28
|   |   +--function-logstamp.ps1
|   |   |   +-- Length = 598
|   |   |   \-- LastWriteTime = 05/23/2007 11:39:55
|   |   +--FunctionNotes.ps1
|   |   |   +-- Length = 617
|   |   |   \-- LastWriteTime = 02/24/2016 08:59:03
|   |   \--Function-SwitchTest.ps1
|   |       +-- Length = 242
|   |       \-- LastWriteTime = 06/09/2008 15:55:44
|   +--gamma
...
...
```

This example is using parameter and command aliases. You can display a tree listing with files including user specified properties. Use a value of * to show all properties.

New-ANSIBar

You can use this command to create colorful bars using ANSI escape sequences based on a 256 color scheme. The default behavior is to create a gradient bar that goes from first to last values in the range and then back down again. Or you can create a single gradient that runs from the beginning of the range to the end. You can use one of the default characters or specify a custom one.

```
Admin: PowerShell 7.0

PS C:\>
PS C:\> New-ANSIBar -Range (232..255)

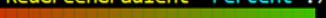
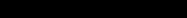
PS C:\>
PS C:\> New-ANSIBar -range (46..51) -Character BlackSquare -Spacing 3
  

PS C:\>
PS C:\> New-ANSIBar -range (214..219) -Gradient -Spacing 5 -Character DarkShade
  

PS C:\>
```

New-RedGreenGradient

A related command is `New-RedGreenGradient` which displays a bar going from red to green. This might be handy when you want to present a visual indicator.

```
PS C:\> Admin: PowerShell 7.0
PS C:\> New-RedGreenGradient -Percent .75
PS C:\>

PS C:\> Get-Volume | Where-Object {$_.FileSystemType -eq 'NTFS' -AND $_.driveletter -match "[C-Zc-z]"} |
>> Sort-Object -property DriveLetter |
>> Select-Object -property DriveLetter, FileSystemLabel,
>> @{Name="FreeGB";Expression={Format-Value -input $_.SizeRemaining -unit GB}},
>> @{Name = "PctFree"; Expression = {$pct = Format-Percent -value $_.sizeRemaining -total $_.size -decimal 2;
>> "{1} {0}" -f $(New-RedGreenGradient -percent ($pct/100) -step 6),$pct}}
DriveLetter FileSystemLabel FreeGB PctFree
-----
    C Windows          87 36.86 
    D Data            102 21.37
PS C:\>
```

Write-ANSIProgress

You could also use `Write-ANSIProgress` to show a custom ANSI bar.

Or you can use it in your code to display a console progress bar.

```
$sb = {
    Clear-Host
    $top = Get-ChildItem c:\scripts -Directory
    $i = 0
    $out=@()
    $pos = $host.ui.RawUICursorPosition
    Foreach ($item in $top) {
        $i++
        $pct = [math]::round($i/$top.count,2)
        Write-ANSIProgress -PercentComplete $pct -position $pos
        Write-Host " Processing $($item.fullname).padright(80)" -ForegroundColor Yellow -NoNewline
        $out+= Get-ChildItem -path $item -Recurse -file | Measure-Object -property length -sum |
        Select-Object @{Name="Path";Expression={$item.fullname}},Count,@{Name="Size";Expression={$_.[Sum]}}
    }
    Write-Host ""
    $out | Sort-object -property Size -Descending
}
```

The screenshot shows a PowerShell window titled 'Admin: PowerShell 7.0'. In the top left corner, there is a small icon of a person with a gear. The title bar also displays the window name. Below the title bar, a progress bar indicates '42%' completion. The main area of the window contains the command 'Processing C:\scripts\ MyClass' in yellow text. The background of the window is blue at the top and black at the bottom.

Format Functions

A set of simple commands to make it easier to format values.

Format-Percent

Treat a value as a percentage. This will write a [double] and not include the % sign.

```
PS C:\> format-percent -Value 123.5646MB -total 1GB -Decimal 4  
12.0669
```

Format-String

Use this command to perform one of several string manipulation "tricks".

```
PS C:\> format-string "powershell" -Reverse -Case Proper  
Llehsrewop  
PS C:\> format-string PowerShell -Randomize  
wSlhoeepLr  
PS C:\> format-string "!MySecretPWord" -Randomize -Replace @{S="$";e=8{Get-Random -min 1 -max 9};o="^"} -Reverse  
yr7!^7WcMtr$Pd
```

Format-Value

This command will format a given numeric value. By default it will treat the number as an integer. Or you can specify a certain number of decimal places. The command will also allow you to format the value in KB, MB, etc.

```
PS C:\> format-value 1235465676 -Unit kb  
1206509  
PS C:\> format-value 123.45 -AsCurrency  
$123.45  
PS C:\> (get-process | measure-object ws -sum).sum | format-value -Unit mb | format-value -AsNumber  
9,437
```

Or pull it all together:

```
PS C:\> Get-CimInstance win32_operatingsystem |  
select-object @{Name = "TotalMemGB";Expression={Format-Value $_.TotalVisibleMemorySize -Unit mb}},  
@{Name="FreeMemGB";Expression={Format-Value $_.FreePhysicalMemory -unit mb -Decimal 2}},  
@{Name="PctFree";Expression={Format-Percent -Value $_.FreePhysicalMemory -Total $_.totalVisibleMemorySize -Decimal 2}}  
  
TotalMemGB FreeMemGB PctFree  
-----  
32      14.05   44.06
```

Scripting Tools

Test-Expression

The primary command can be used to test a PowerShell expression or scriptblock for a specified number of times and calculate the average runtime, in milliseconds, over all the tests.

Why

When you run a single test with `Measure-Command` the result might be affected by any number of factors. Likewise, running multiple tests may also be influenced by things such as caching. The goal in this module is to provide a test framework where you can run a test repeatedly with either a static or random interval between each test. The results are aggregated and analyzed. Hopefully, this will provide a more meaningful or realistic result.

Examples

The output will also show the median and trimmed values as well as some metadata about the current PowerShell session.

```
PS C:\> $cred = Get-Credential globomantics\administrator
PS C:\> Test-Expression {param($cred) get-wmiobject win32_logicaldisk -computer chi-dc01 -credential $cred }
-argumentList $cred

Tests      : 1
TestInterval : 0.5
AverageMS   : 1990.6779
MinimumMS    : 1990.6779
MaximumMS    : 1990.6779
MedianMS     : 1990.6779
TrimmedMS    :
PSVersion   : 5.1.17763.134
OS          : Microsoft Windows 10 Pro
```

You can also run multiple tests with random time intervals.

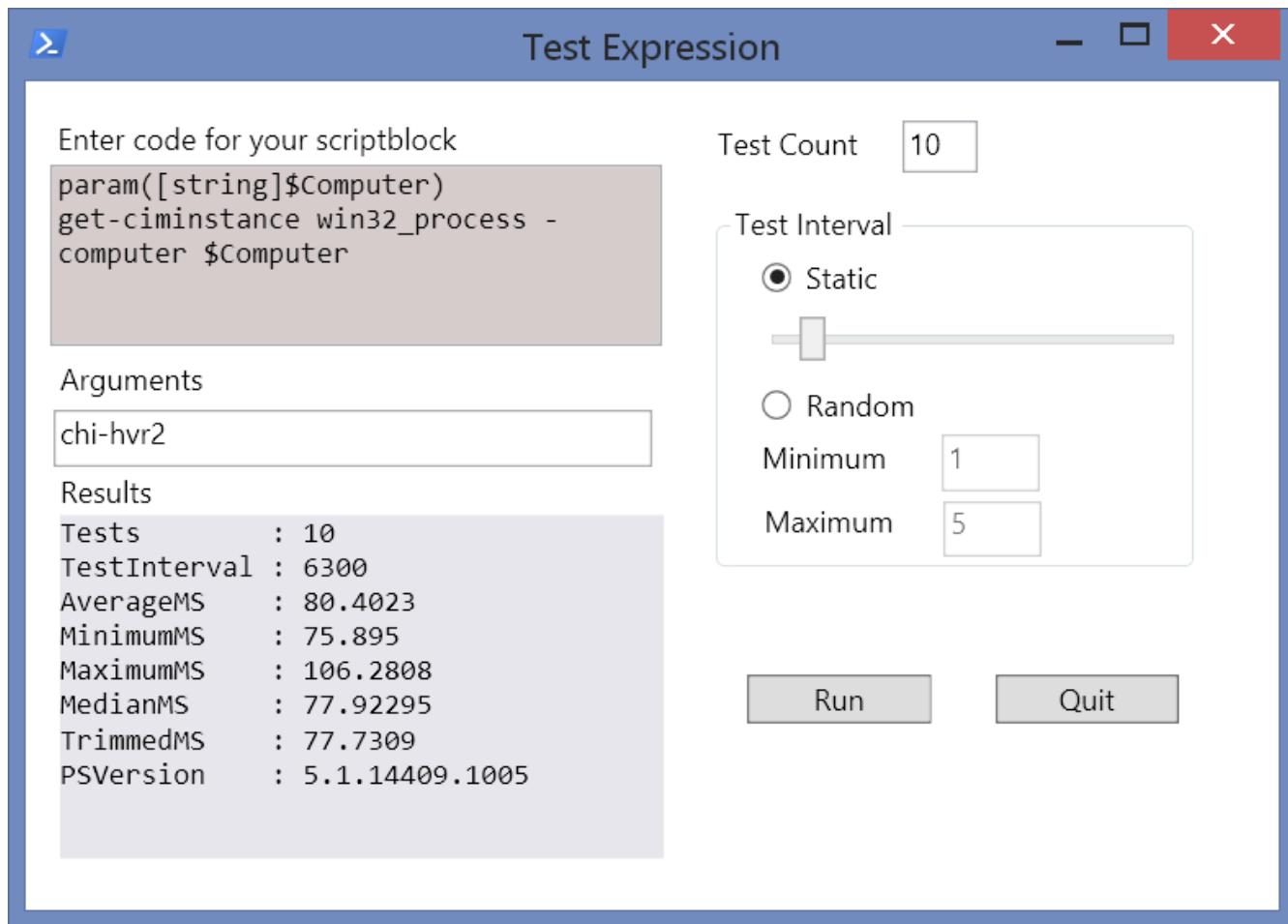
```
PS C:\> Test-Expression {param([string[]]$Names) get-service $names} -count 5 -IncludeExpression -argumentlist
@('bits','wuauserv','winrm') -RandomMinimum .5 -RandomMaximum 5.5

Tests      : 5
TestInterval : Random
AverageMS   : 1.91406
MinimumMS    : 0.4657
MaximumMS    : 7.5746
MedianMS     : 0.4806
TrimmedMS    : 0.51
PSVersion   : 5.1.17763.134
OS          : Microsoft Windows 10 Pro
Expression   : param([string[]]$Names) get-service $names
Arguments    : {bits, wuauserv, winrm}
```

For very long running tests, you can run them as a background job.

Graphical Testing

The module also includes a graphical command called `Test-ExpressionForm`. This is intended to serve as both an entry and results form.



When you quit the form the last result will be written to the pipeline including all metadata, the scriptblock and any arguments.

Copy-HelpExample

This command is designed to make it (slightly) easier to copy code snippets from help examples. Specify the name of a function or cmdlet, presumably one with documented help examples, and you will be offered a selection of code snippets to copy to the clipboard. Code snippets have been trimmed of blank lines, most prompts, and comments. Many examples include output. You will have to manually remove what you don't want after pasting.

The default behavior is to use a console based menu which works cross-platform.

```
Administrator: Windows PowerShell 5.1.19041
PS C:\> Copy-HelpExample Stop-Service

Code Samples

Each help example is numbered to the left. At the prompt below,
select the code samples you want to copy to the clipboard. Separate
multiple values with a comma.

Some example code includes the output.

[1] Example 1: Stop a service on the local computer
    Stop-Service -Name "sysmonlog"

[2] Example 2: Stop a service by using the display name
    Get-Service -DisplayName "telnet" | Stop-Service

[3] Example 3: Stop a service that has dependent services
    Get-Service -Name "iisadmin" | Format-List -Property Name, DependentServices
    Stop-Service -Name "iisadmin" -Force -Confirm

Please select items to copy to the clipboard by number. Separate multiple entries with a comma. Press Enter alone to cancel:
```

Enter the number of the code to copy to the clipboard. Enter multiple numbers separated by commas.

If you are running a Windows platform there is a dynamic help parameter to use Out-GridView.

Copy-HelpExample Stop-Service **-UseGridView**

title	CodeSample
Example 1: Stop a service on the local computer	Stop-Service -Name "sysmonlog"
Example 2: Stop a service by using the display...	Get-Service -DisplayName "telnet" Stop-Service
Example 3: Stop a service that has dependent...	Get-Service -Name "iisadmin" Format-List -Property N... Stop-Service -Name "iisadmin" -Force -Confirm

OK Cancel

If you are running this in the PowerShell ISE, this is the default behavior even if you don't specify the parameter.

Get-GitSize

Use this command to determine how much space the hidden `.git` folder is consuming.

```
PS C:\scripts\PSScriptTools> Get-GitSize
```

Path	Files	SizeKB
C:\scripts\PSScriptTools	751	6859.9834

This is the default, formatted view. The object has other properties you can use.

```
Name      : PSScriptTools
Path     : C:\scripts\PSScriptTools
Files    : 751
Size     : 7024623
Date     : 3/5/2020 2:57:06 PM
Computername : BOVINE320
```

Remove-MergedBranch

When using `git` you may create a number of branches. Presumably you merge these branches into the main or master branch. You can use this command to remove all merged branches other than master and the current branch. You must be in the root of your project to run this command.

```
PS C:\MyProject> Remove-MergedBranch

Remove merged branch from MyProject?
2.1.1
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): n

Remove merged branch from MyProject?
dev1
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
Deleted branch dev1 (was 75f6ab8).

Remove merged branch from MyProject?
dev2
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
Deleted branch dev2 (was 75f6ab8).

Remove merged branch from MyProject?
patch-254
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): n

PS C:\MyProject>
```

By default you will be prompted to remove each branch.

Test-WithCulture

When writing PowerShell commands, sometimes the culture you are running under becomes critical. For example, European countries use a different datetime format than North Americans which might present a problem with your script or command. Unless you have a separate computer running under the foreign culture, it is difficult to test. This command will allow you to test a scriptblock or even a file under a different culture, such as DE-DE for German.

```
PS C:\> Test-WithCulture fr-fr -Scriptblock {  
    Get-winEvent -log system -max 500 |  
    Select-Object -Property TimeCreated, ID, OpCodeDisplayName, Message |  
    Sort-Object -property TimeCreated |  
    Group-Object {$_.TimeCreated.ToShortDateString()} -NoElement}  
  
Count Name  
---- --  
 165 10/07/2019  
 249 11/07/2019  
   17 12/07/2019  
   16 13/07/2019  
   20 14/07/2019  
   26 15/07/2019  
    7 16/07/2019
```

Copy-Command

This command will copy a PowerShell command, including parameters and help to a new user-specified command. You can use this to create a "wrapper" function or to easily create a proxy function. The default behavior is to create a copy of the command complete with the original comment-based help block.

Get-ParameterInfo

Using Get-Command, this function will return information about parameters for any loaded cmdlet or function. The common parameters like Verbose and ErrorAction are omitted. [Get-ParameterInfo](#) returns a custom object with the most useful information an administrator might need to know.

```
PS C:\> Get-ParameterInfo -Command Get-Counter -Parameter computername
```

```
Name          : computername
Aliases       : Cn
Mandatory     : False
Position      : Named
ValueFromPipeline : False
ValueFromPipelineByPropertyName : False
Type          : System.String[]
IsDynamic     : False
ParameterSet   : __AllParameterSets
```

New-PSFormatXML

When defining custom objects with a new typename, PowerShell by default will display all properties. However, you may wish to have a specific default view, be it a table or list. Or you may want to have different views display the object differently. Format directives are stored in format.ps1xml files which can be tedious to create. This command simplifies that process.

Define a custom object:

```
$tname = "myThing"
$obj = [PSCustomObject]@{
    PSTypeName  = $tname
    Name        = "Jeff"
    Date        = (Get-Date)
    Computername = $env:computername
    OS          = (Get-CimInstance win32_operatingsystem -Property Caption).Caption
}
Update-TypeData -TypeName $tname -MemberType "ScriptProperty" -MemberName "Runtime" -Value {(Get-Date) - [datetime]"1/1/2019"} -Force
```

The custom object looks like this by default:

```
PS C:\> $obj

Name      : Jeff
Date      : 2/10/2019 8:49:10 PM
Computername : BOVINE320
OS        : Microsoft Windows 10 Pro
Runtime    : 40.20:49:43.9205882
```

Now you can create new formatting directives.

```
$obj | New-PSFormatXML -Properties Name, Date, Computername, OS -FormatType Table -path "C:\work\$tname.format.ps1xml"
$obj | New-PSFormatXML -Properties Name, OS, Runtime -FormatType Table -view runtime -path "C:\work\$tname.format.ps1xml"
-append
$obj | New-PSFormatXML -FormatType List -path "C:\work\$tname.format.ps1xml" -append
Update-FormatData -appendpath "C:\work\$tname.format.ps1xml"
```

And here is what the object looks like now:

```
PS C:\> $obj

Name Date Computername Operating System
---- -- -----
Jeff 2/10/2019 8:49:10 PM BOVINE320 Microsoft Windows 10 Pro

PS C:\> $obj | format-table -View runtime

Name OS Runtime
---- - -----
Jeff 40.20:56:24.5411481

PS C:\> $obj | format-list

Name : Jeff
Date : Sunday, February 10, 2019
Computername : BOVINE320
OperatingSystem : Microsoft Windows 10 Pro
Runtime : 40.21:12:01
```

If you run this command within VS Code and specify `-PassThru`, the resulting file will be opened in your editor.

Test-IsPSWindows

PowerShell Core introduced the `$IsWindows` variable. However it is not available on Windows PowerShell. Use this command to perform a simple test if the computer is either running Windows or using the Desktop PSEdition. The command returns `True` or `False`.

Write-Detail

This command is designed to be used within your functions and scripts to make it easier to write a detailed message that you can use as verbose output. The assumption is that you are using an advanced function with a `Begin`, `Process` and `End` scriptblocks. You can create a detailed message to indicate what part of the code is being executed. The output can be configured to include a datetime stamp or just the time.

```
PS C:\> write-detail "Getting file information" -Prefix Process -Date  
9/15/2018 11:42:43 [PROCESS] Getting file information
```

In a script you might use it like this:

```
Begin {  
    Write-Detail "Starting $($myinvocation.mycommand)" -Prefix begin -time | Write-Verbose  
    $tabs = "`t" * $tab  
    Write-Detail "Using a tab of $tab" -Prefix BEGIN -time | Write-Verbose  
} #begin
```

Save-GitSetup

This command is intended for Windows users to easily download the latest 64bit version of [Git](#).

```
PS C:\> Save-GitSetup -Path c:\work -Passthru  
  
Directory: C:\work  
  
Mode          LastWriteTime        Length Name  
----          -----          -----  
-a---  1/23/2020 4:31 PM      46476880 Git-2.25.0-64-bit.exe
```

You will need to manually install the file.

Other

From time to time I will include additional items that you might find useful. One item that you get when you import this module is a custom format.ps1xml file for services. You can run [Get-Service](#) and pipe it to the table view.

```
PS C:\> Get-Service | Format-Table -view ansi
```

This will display the service status color-coded.

Stopped	WFDSSConMgrSvc	Wi-Fi Direct Services Connection Manager ...
Stopped	WiaRpc	Still Image Acquisition Events
Stopped	WinDefend	Windows Defender Antivirus Service
Running	WinHttpAutoProxyS...	WinHTTP Web Proxy Auto-Discovery Service
Paused	Winmgmt	Windows Management Instrumentation
Running	WinRM	Windows Remote Management (WS-Management)
Stopped	wisvc	Windows Insider Service
Running	WlanSvc	WLAN AutoConfig
Stopped	wlidsvc	Microsoft Account Sign-in Assistant
Stopped	wlpasvc	Local Profile Assistant Service
Stopped	WManSvc	Windows Management Service
Stopped	wmiApSrv	WMI Performance Adapter
Stopped	WMPNetworkSvc	Windows Media Player Network Sharing Serv...
Stopped	workfolderssvc	Work Folders
Stopped	WpcMonSvc	Parental Controls
Stopped	WPDBusEnum	Portable Device Enumerator Service
Running	WpnService	Windows Push Notifications System Service
Running	WpnUserService_d9...	Windows Push Notifications User Service_d...
Running	wscsvc	Security Center
Running	WSearch	Windows Search

This will not work in the PowerShell ISE as it is not ANSI aware.

PSAnsiMap

I have done something similar for output from [Get-ChildItem](#). The module includes json file that is exported as a global variable called [PSAnsiFileMap](#).

```
PS C:\scripts\PSScriptTools> $PSAnsifileMap
```

Description	Pattern	Ansi
PowerShell	\.ps(d m)?1\$	
Text	\.(txt) (md) (log)\$	
DataFile	\.(json) (xml) (csv)\$	
Executable	\.(exe) (bat) (cmd) (sh)\$	
Graphics	\.(jpg) (png) (gif) (bmp) (jpeg)\$	
Media	\.(mp3) (m4v) (wav) (au) (flac) (mp4)\$	
Archive	\.(zip) (rar) (tar) (gzip)\$	
TopContainer		
ChildContainer		

The map includes ANSI settings for different file types. You won't see the ANSI value in the output. The module will add a custom table view called `ansi` which you can use to display file results colorized in PowerShell 7.

```
PowerShell 7
PS C:\> dir c:\work\alpha -Recurse | format-table -view ansi

Directory: C:\work\Alpha

Mode          LastWriteTime      Length Name
----          -----          ---- 
d---          3/5/2020  4:46 PM        0    bravo
-a--          11/8/2019 3:29 PM     12109 documents-log.csv
-a--          11/9/2019 9:00 AM     30335 dropbox-log.csv
-a--          11/9/2019 1:00 AM       671 GoogleDrive-log.csv
-a--          10/31/2019 1:42 PM        45 junk.txt
-a--          11/9/2019 9:03 AM    166435 Scripts-log.csv
-a--          11/10/2019 4:32 PM     2673 stuff.tmp
-a--          11/10/2019 12:49 PM        43 test.data

Directory: C:\work\Alpha\bravo

Mode          LastWriteTime      Length Name
----          -----          ---- 
d---          2/28/2020 11:17 AM        0    delta
d---          11/6/2017 4:21 PM        0    gamma
d---          2/28/2020 11:16 AM        0    images
-a--          11/6/2017 4:47 PM      636 data.txt
-a--          11/7/2019 10:32 AM      131 sample-1.json
-a--          11/7/2019 10:32 AM      131 sample-2.json
-a--          11/7/2019 10:32 AM      131 sample-3.json
-a--          11/7/2019 10:32 AM      131 sample-4.json
-a--          10/31/2019 5:25 PM   5769412 something2.xml
-a--          3/5/2020  4:46 PM        0 zz.foo

Directory: C:\work\Alpha\bravo\delta

Mode          LastWriteTime      Length Name
----          -----          ---- 
-a--          6/1/2009  3:50 PM      888 FunctionDemo.ps1
-a--          4/17/2019 5:18 PM     1117 function-form.ps1
-a--          5/23/2007 11:39 AM      598 function-logstamp.ps1
```

The mapping file is user customizable. Copy the `psansifilemap.json` file from the module's root directory to `$HOME`. When you import this module, if the file is found, it will be imported and used as `psansifilemap`, otherwise the module's file will be used.

The file will look like this:

```
[
  {
    "Description": "PowerShell",
    "Pattern": "\\.ps(d|m)?1$",
    "Ansi": "\u001b[38;2;252;127;12m"
  },
  {
    "Description": "Text",
    "Pattern": "\\.(txt)|(md)|(log)$",
    "Ansi": "\u001b[38;2;58;120;255m"
  },
  {
    "Description": "DataFile",
    "Pattern": "\\.(json)|(xml)|(csv)$",
    "Ansi": "\u001b[38;2;249;241;165m"
  },
  {
    "Description": "Executable",
    "Pattern": "\\.(exe)|(bat)|(cmd)|(sh)$",
    "Ansi": "\u001b[38;2;197;15;31m"
  },
  {
    "Description": "Graphics",
    "Pattern": "\\.(jpg)|(png)|(gif)|(bmp)|(jpeg)$",
    "Ansi": "\u001b[38;2;255;0;255m"
  },
  {
    "Description": "Media",
    "Pattern": "\\.(mp3)|(m4v)|(wav)|(au)|(flac)|(mp4)$",
    "Ansi": "\u001b[38;2;255;199;6m"
  },
  {
    "Description": "Archive",
    "Pattern": "\\.(zip)|(rar)|(tar)|(gzip)$",
    "Ansi": "\u001b[38;2;118;38;113m"
  },
  {
    "Description": "TopContainer",
    "Pattern": "",
    "Ansi": "\u001b[38;2;0;255;255m"
  },
  {
    "Description": "ChildContainer",
    "Pattern": "",
    "Ansi": "\u001b[38;2;255;255;0m"
  }
]
```

You can create or modify file groups. The Pattern value should be a regular expression pattern to match on the filename. Don't forget you will need to escape characters for the json format. The Ansi

value will be an ANSI escape sequence. You can use \u001b for the 'e character.

PSSpecialChar

A number of the commands in this module can use special characters. To make it easier, when you import the module it will create a global variable that is a hash table of common special characters. Because it is a hashtable you can add ones you also use.

```
PS C:\> $PSSpecialChar
```

Name	Value
MediumShade	■
FullBlock	█
WhiteSquare	□
Heart	♥
DarkShade	▨
SixPointStar	*
Spade	♠
WhiteCircle	○
LightShade	▨
BlackSquare	■
DownTriangle	▼
BlackSmallSquare	▪
WhiteSmallSquare	▫
Diamond	◆
WhiteFace	☺
UpTriangle	▲
BlackFace	☻
Lozenge	◇
Club	♣
BlackCircle	●

```
PS C:\> $PSSpecialChar.blackcircle
```

```
●
```

```
PS C:\> $PSSpecialChar.blackcircle -as [int]
```

```
9679
```

```
PS C:\> [char]9679
```

```
●
```

```
PS C:\> █
```

The names are the same as used in CharMap.exe. Don't let the naming confuse you. It may say **BlackSquare** but the color will depend on how you use it.

```
Get-WindowsVersionString | Add-Border -border $PSSpecialChar.BlackSmallSquare -ANSIBorder "$([char]0x1b)[38;5;214m"
```

```
PS C:\> Get-WindowsVersionString | Add-Border -border $PSSpecialChar.BlackSmallSquare -ANSIBorder "$([char]0x1b)[38;5;214m"
PS C:\>
-----  
▪ BOVINE320 Windows 10 Pro Version Professional (OS Build 18363.836) ▪  
-----  
PS C:\>
```

Sample Scripts

This PowerShell module contains a number of functions you might use to enhance your own functions and scripts. The [Samples](#) folder contains demonstration script files. You can access the folder in PowerShell using the `$PSSamplePath`. The samples provide suggestions on how you might use some of the commands in this module. The scripts are offered AS-IS and are for demonstration purposes only.

```
PS C:\> . $PSSamplePath\ProcessPercent.ps1
```

Name	Id	Handles	WS(MB)	PctWS	
TabNine	35188	293	1384	10.73	<div style="width: 100%; background-color: #0070C0;"></div>
Memory Compression	3044	0	1249	09.69	<div style="width: 100%; background-color: #0070C0;"></div>
firefox	18936	1630	798	06.18	<div style="width: 50%; background-color: #0070C0;"></div>
LenovoVantageService	5724	1273	784	06.08	<div style="width: 100%; background-color: #0070C0;"></div>
dwm	1532	2835	382	02.96	<div style="width: 25%; background-color: #0070C0;"></div>
firefox	18368	3187	349	02.71	<div style="width: 20%; background-color: #0070C0;"></div>
firefox	21912	1573	338	02.62	<div style="width: 20%; background-color: #0070C0;"></div>
pwsh	25220	1183	311	02.41	<div style="width: 15%; background-color: #0070C0;"></div>
thunderbird	23268	2032	247	01.91	<div style="width: 15%; background-color: #0070C0;"></div>
powershell_ise	4896	946	244	01.89	<div style="width: 15%; background-color: #0070C0;"></div>
firefox	28208	901	224	01.74	<div style="width: 15%; background-color: #0070C0;"></div>
Code	34948	598	213	01.65	<div style="width: 15%; background-color: #0070C0;"></div>
powershell	24608	917	209	01.62	<div style="width: 15%; background-color: #0070C0;"></div>
pwsh	21864	1219	203	01.57	<div style="width: 15%; background-color: #0070C0;"></div>

Open-PSScriptToolsHelp

I've created a PDF version of this document which I thought you might find useful since it includes screen shots and sample output rendered nicer than what you can get in PowerShell help. Run this to open the PDF using your default associated application.

Open-PSScriptToolsHelp

Related Modules

If you find this module useful, you might also want to look at my tools for [creating and managing custom type extensions](#), [managing scheduled jobs](#) and [running remote commands outside of PowerShell Remoting](#).

Compatibility

Where possible these commands have been tested with PowerShell 7, but not every platform. If you encounter problems, have suggestions or other feedback, please post an [issue](#). It is assumed you will **not** be running this commands on any edition of PowerShell Core or any beta releases of PowerShell 7.

Last Updated 2020-07-09 15:16:38Z UTC