

PSScriptTools Manual v2.34.0

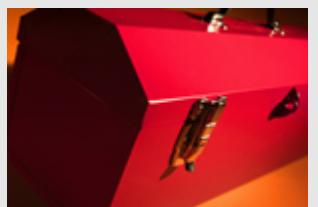


Table of Contents

Introduction	1
PSScriptTools Overview	2
Abstract	3
Installation	4
General Tools	5
Get-MyCounter	5
Get-DirectoryInfo	6
Get-FormatView	8
Copy-PSFunction	8
Get-PSProfile	8
Get-MyAlias	9
Get-ModuleCommand	11
Get-PSScriptTools	12
Convert-EventLogRecord	13
Get-WhoIs	13
Compare-Module	14
Get-WindowsVersion	15
New-PSDriveHere	16
Get-MyVariable	16
ConvertFrom-Text	16
Get-PSWho	17
Find-CimClass	18
Out-VerboseTee	19
Remove-Runspace	19
Get-PSLocation	19
Get-PowerShellEngine	20
Get-PathVariable	21
File Tools	22
Test-EmptyFolder	22
Get-FolderSizeInfo	22
Optimize-Text	23
Get-FileItem	23
New-CustomFileName	24
New-RandomFilename	24
ConvertTo-Markdown	25
ToDo	26
Graphical Tools	27
Invoke-InputBox	27
New-WPFMessageBox	28
ConvertTo-WPFGGrid	29
Hashtable Tools	32

Convert-CommandToHashtable	32
Convert-HashtableString	32
Convert-HashtableToCode	32
ConvertTo-Hashtable	33
Join-Hashtable	33
Rename-Hashtable	33
Select Functions	35
Select-First	35
Select-Last	35
Select-After	35
Select-Before	36
Select-Newest	37
Select-Oldest	38
Time Functions	39
ConvertTo-UTCTime	39
ConvertFrom-UTCTime	39
Get-MyTimeInfo	39
ConvertTo-LocalTime	40
Get-TZList	40
Get-TZData	40
ConvertTo-LexicalTime	41
ConvertFrom-LexicalTime	41
Console Utilities	43
ConvertTo-ASCIIArt	43
Out-Copy	44
Out-More	44
Out-ConditionalColor	45
Set-ConsoleTitle	46
Set-ConsoleColor	47
Add-Border	47
Show-Tree	48
New-ANSIBar	50
New-RedGreenGradient	50
Write-ANSIProgress	51
Format Functions	53
Format-Percent	53
Format-String	53
Format-Value	53
Scripting Tools	55
Trace-Message	55
Get-CommandSyntax	56
Test-Expression	57
Copy-HelpExample	59

Get-GitSize	61
Remove-MergedBranch	61
Test-WithCulture	61
Copy-Command	62
Get-ParameterInfo	62
New-PSFormatXML	63
Test-IsPSWindows	65
Write-Detail	65
Save-GitSetup	65
ANSI Tools	66
PSAnsiMap	67
Get-PSAnsiFileMap	69
Set-PSAnsiFileMap	70
Remove-PSAnsiFileEntry	70
Export-PSAnsiFileMap	70
Other Module Features	71
PSSpecialChar	71
Sample Scripts	73
Open-PSScriptToolsHelp	73
Related Modules	75
Compatibility	76
Module Commands	77
Add-Border	79
Synopsis	79
Syntax	79
Description	79
Examples	79
Parameters	81
Inputs	83
Outputs	83
Notes	83
Related Links	83
Compare-Module	84
Synopsis	84
Syntax	84
Description	84
Examples	84
Parameters	85
Inputs	86
Outputs	86
Notes	86
Related Links	86
Compare-Script	87

Synopsis	87
Syntax	87
Description.....	87
Examples	87
Parameters.....	88
Inputs	88
Outputs.....	88
Notes.....	89
Related Links	89
Convert-CommandToHashtable.....	90
Synopsis	90
Syntax	90
Description.....	90
Examples	90
Parameters.....	90
Inputs	91
Outputs.....	91
Notes.....	91
Related Links	91
Convert-EventLogRecord.....	92
Synopsis	92
Syntax	92
Description.....	92
Examples	92
Parameters.....	95
Inputs	96
Outputs.....	96
Related Links	96
Convert-HashtableString.....	97
Synopsis	97
Syntax	97
Description.....	97
Examples	97
Parameters.....	97
Inputs	98
Outputs.....	98
Notes.....	98
Related Links	98
Convert-HashtableToCode	99
Synopsis	99
Syntax	99
Description.....	99
Examples	99

Parameters.....	100
Inputs	101
Outputs.....	101
Notes.....	101
Related Links	101
ConvertFrom-LexicalTimespan.....	102
Synopsis	102
Syntax	102
Description.....	102
Examples	102
Parameters.....	103
Inputs	103
Outputs.....	103
Notes.....	104
Related Links	104
ConvertFrom-Text.....	105
Synopsis	105
Syntax	105
Description.....	105
Examples	105
Parameters.....	107
Inputs	109
Outputs.....	109
Notes.....	109
Related Links	109
ConvertFrom-UTCTime	110
Synopsis	110
Syntax	110
Description.....	110
Examples	110
Parameters.....	110
Inputs	111
Outputs.....	111
Notes.....	111
Related Links	111
ConvertTo-ASCIIArt	112
Synopsis	112
Syntax	112
Description.....	112
Examples	112
Parameters.....	112
Inputs	113
Outputs.....	113

Notes.....	113
Related Links	113
ConvertTo-Hashtable.....	114
Synopsis	114
Syntax.....	114
Description.....	114
Examples	114
Parameters.....	115
Inputs	116
Outputs.....	116
Notes.....	116
Related Links	117
ConvertTo-LexicalTimespan	118
Synopsis	118
Syntax.....	118
Description.....	118
Examples	118
Parameters.....	118
Inputs	119
Outputs.....	119
Notes.....	119
Related Links	119
ConvertTo-LocalTime.....	120
Synopsis	120
Syntax.....	120
Description.....	120
Examples	120
Parameters.....	121
Inputs	122
Outputs.....	122
Notes.....	122
Related Links	122
ConvertTo-Markdown	123
Synopsis	123
Syntax.....	123
Description.....	123
Examples	123
Parameters.....	124
Inputs	126
Outputs.....	126
Notes.....	126
Related Links	126
ConvertTo-UTCTime.....	127

Synopsis	127
Syntax	127
Description.....	127
Examples	127
Parameters.....	127
Inputs	128
Outputs.....	128
Notes.....	128
Related Links	128
ConvertTo-WPFGGrid	129
Synopsis	129
Syntax	129
Description.....	129
Examples	130
Parameters.....	131
Inputs	133
Outputs.....	134
Notes.....	134
Related Links	134
Copy-Command	135
Synopsis	135
Syntax	135
Description.....	135
Examples	135
Parameters.....	136
Inputs	137
Outputs.....	137
Notes.....	137
Related Links	137
Copy-HelpExample	138
Synopsis	138
Syntax	138
Description.....	138
Examples	138
Parameters.....	139
Inputs	140
Outputs.....	140
Notes.....	140
Related Links	141
Copy-PSFunction.....	142
Synopsis	142
Syntax	142
Description.....	142

Examples	142
Parameters.....	142
Inputs	143
Outputs.....	143
Notes.....	143
Related Links	144
Export-PSAnsFileMap	145
Synopsis	145
Syntax	145
Description.....	145
Examples	145
Parameters.....	145
Inputs	146
Outputs.....	146
Notes.....	147
Related Links	147
Find-CimClass	148
Synopsis	148
Syntax	148
Description.....	148
Examples	148
Parameters.....	149
Inputs	150
Outputs.....	150
Notes.....	150
Related Links	150
Format-Percent.....	151
Synopsis	151
Syntax	151
Description.....	151
Examples	151
Parameters.....	152
Inputs	153
Outputs.....	153
Notes.....	153
Related Links	154
Format-String	155
Synopsis	155
Syntax	155
Description.....	155
Examples	155
Parameters.....	156
Inputs	158

Outputs.....	158
Notes.....	158
Related Links	158
Format-Value.....	159
Synopsis	159
Syntax	159
Description.....	159
Examples	159
Parameters.....	160
Inputs	162
Outputs.....	163
Notes.....	163
Related Links	163
Get-CommandSyntax.....	164
Synopsis	164
Syntax	164
Description.....	164
Examples	164
Parameters.....	165
Inputs	165
Outputs.....	165
Notes.....	165
Related Links	165
Get-DirectoryInfo	166
Synopsis	166
Syntax	166
Description.....	166
Examples	166
Parameters.....	168
Inputs	168
Outputs.....	168
Notes.....	169
Related Links	169
Get-FileItem.....	170
Synopsis	170
Syntax	170
Description.....	170
Examples	170
Parameters.....	171
Inputs	173
Outputs.....	173
Notes.....	173
Related Links	174

Get-FolderSizeInfo	175
Synopsis	175
Syntax	175
Description.....	175
Examples	175
Parameters.....	176
Inputs	177
Outputs.....	177
Notes.....	177
Related Links	177
Get-FormatView	179
Synopsis	179
Syntax	179
Description.....	179
Examples	179
Parameters.....	180
Inputs	181
Outputs.....	181
Notes.....	181
Related Links	182
Get-GitSize	183
Synopsis	183
Syntax	183
Description.....	183
Examples	183
Parameters.....	184
Inputs	184
Outputs.....	184
Notes.....	184
Related Links	185
Get-ModuleCommand	186
Synopsis	186
Syntax	186
Description.....	186
Examples	186
Parameters.....	187
Inputs	189
Outputs.....	189
Notes.....	189
Related Links	189
Get-MyAlias	190
Synopsis	190
Syntax	190

Description	190
Examples	190
Parameters	191
Inputs	191
Outputs	192
Notes	192
Related Links	192
Get-MyCounter	193
Synopsis	193
Syntax	193
Description	193
Examples	193
Parameters	196
Inputs	197
Outputs	198
Notes	198
Related Links	198
Get-MyTimeInfo	199
Synopsis	199
Syntax	199
Description	199
Examples	199
Parameters	201
Inputs	203
Outputs	203
Notes	203
Related Links	203
Get-MyVariable	204
Synopsis	204
Syntax	204
Description	204
Examples	204
Parameters	206
Inputs	206
Outputs	206
Notes	207
Related Links	207
Get-ParameterInfo	208
Synopsis	208
Syntax	208
Description	208
Examples	208
Parameters	210

Inputs	211
Outputs.....	211
Notes.....	211
Related Links	211
Get-PathVariable.....	212
Synopsis	212
Syntax.....	212
Description.....	212
Examples	212
Parameters.....	213
Inputs	213
Outputs.....	213
Notes.....	213
Related Links	213
Get-PowerShellEngine	214
Synopsis	214
Syntax.....	214
Description.....	214
Examples	214
Parameters.....	215
Inputs	215
Outputs.....	215
Notes.....	215
Related Links	215
Get-PSAnsFileMap	217
Synopsis	217
Syntax.....	217
Description.....	217
Examples	217
Parameters.....	217
Inputs	217
Outputs.....	218
Notes.....	218
Related Links	218
Get-PSLocation	219
Synopsis	219
Syntax.....	219
Description.....	219
Examples	219
Parameters.....	219
Inputs	220
Outputs.....	220
Notes.....	220

Related Links	220
Get-PSProfile	221
Synopsis	221
Syntax	221
Description	221
Examples	221
Parameters	222
Inputs	222
Outputs	222
Notes	222
Related Links	223
Get-PSScriptTools	224
Synopsis	224
Syntax	224
Description	224
Examples	224
Parameters	225
Inputs	226
Outputs	226
Notes	226
Related Links	226
Get-PSWho	227
Synopsis	227
Syntax	227
Description	227
Examples	227
Parameters	228
Inputs	229
Outputs	229
Notes	229
Related Links	229
Get-TZData	230
Synopsis	230
Syntax	230
Description	230
Examples	230
Parameters	231
Inputs	232
Outputs	232
Notes	232
Related Links	233
Get-TZList	234
Synopsis	234

Syntax	234
Description	234
Examples	234
Parameters	235
Inputs	236
Outputs	236
Notes	236
Related Links	236
Get-WhoIs	237
Synopsis	237
Syntax	237
Description	237
Examples	237
Parameters	237
Inputs	238
Outputs	238
Notes	238
Related Links	238
Get-WindowsVersion	239
Synopsis	239
Syntax	239
Description	239
Examples	239
Parameters	240
Inputs	243
Outputs	243
Notes	243
Related Links	243
Get-WindowsVersionString	244
Synopsis	244
Syntax	244
Description	244
Examples	244
Parameters	244
Inputs	247
Outputs	247
Notes	247
Related Links	247
Invoke-InputBox	248
Synopsis	248
Syntax	248
Description	248
Examples	248

Parameters.....	248
Inputs	250
Outputs.....	250
Notes.....	250
Related Links	250
Join-Hashtable.....	251
Synopsis	251
Syntax	251
Description.....	251
Examples	251
Parameters.....	252
Inputs	253
Outputs.....	253
Notes.....	253
Related Links	253
New-ANSIBar.....	254
Synopsis	254
Syntax	254
Description.....	254
Examples	254
Parameters.....	255
Inputs	256
Outputs.....	256
Notes.....	256
Related Links	256
New-CustomFileName.....	257
Synopsis	257
Syntax	257
Description.....	257
Examples	257
Parameters.....	259
Inputs	260
Outputs.....	260
Notes.....	260
Related Links	261
New-PSDriveHere.....	262
Synopsis	262
Syntax	262
Description.....	262
Examples	262
Parameters.....	263
Inputs	265
Outputs.....	265

Notes.....	265
Related Links	266
New-PSFormatXML.....	267
Synopsis	267
Syntax.....	267
Description.....	267
Examples	267
Parameters.....	270
Inputs	274
Outputs.....	274
Notes.....	274
Related Links	274
New-RandomFileName	275
Synopsis	275
Syntax.....	275
Description.....	275
Examples	275
Parameters.....	276
Inputs	277
Outputs.....	277
Notes.....	277
Related Links	277
New-RedGreenGradient	278
Synopsis	278
Syntax.....	278
Description.....	278
Examples	278
Parameters.....	279
Inputs	280
Outputs.....	280
Notes.....	280
Related Links	280
New-WPFMessageBox	281
Synopsis	281
Syntax.....	281
Description.....	281
Examples	281
Parameters.....	282
Inputs	285
Outputs.....	285
Notes.....	285
Related Links	285
Open-PSScriptToolsHelp	286

Synopsis	286
Syntax	286
Description.....	286
Examples	286
Parameters.....	286
Inputs	286
Outputs.....	286
Notes.....	286
Related Links	287
Optimize-Text	288
Synopsis	288
Syntax	288
Description.....	288
Examples	288
Parameters.....	290
Inputs	292
Outputs.....	292
Notes.....	292
Related Links	293
Out-ConditionalColor.....	294
Synopsis	294
Syntax	294
Description.....	294
Examples	295
Parameters.....	296
Inputs	297
Outputs.....	297
Notes.....	297
Related Links	298
Out-Copy	299
Synopsis	299
Syntax	299
Description.....	299
Examples	299
Parameters.....	300
Inputs	301
Outputs.....	301
Notes.....	301
Related Links	301
Out-More	302
Synopsis	302
Syntax	302
Description.....	302

Examples	302
Parameters.....	303
Inputs	303
Outputs.....	304
Notes.....	304
Related Links	304
Out-VerboseTee	305
Synopsis	305
Syntax	305
Description.....	305
Examples	305
Parameters.....	306
Inputs	307
Outputs.....	307
Notes.....	307
Related Links	307
Remove-MergedBranch	309
Synopsis	309
Syntax	309
Description.....	309
Examples	309
Parameters.....	310
Inputs	311
Outputs.....	311
Notes.....	311
Related Links	311
Remove-PSAnsFileEntry	312
Synopsis	312
Syntax	312
Description.....	312
Examples	312
Parameters.....	312
Inputs	313
Outputs.....	314
Notes.....	314
Related Links	314
Remove-Runspace	315
Synopsis	315
Syntax	315
Description.....	315
Examples	315
Parameters.....	316
Inputs	317

Outputs.....	317
Notes.....	317
Related Links	317
Rename-Hashtable.....	318
Synopsis	318
Syntax	318
Description.....	318
Examples	318
Parameters.....	319
Inputs	321
Outputs.....	321
Notes.....	321
Related Links	322
Save-GitSetup	323
Synopsis	323
Syntax	323
Description.....	323
Examples	323
Parameters.....	323
Inputs	324
Outputs.....	324
Notes.....	324
Related Links	324
Select-After	325
Synopsis	325
Syntax	325
Description.....	325
Examples	325
Parameters.....	326
Inputs	327
Outputs.....	327
Notes.....	327
Related Links	327
Select-Before	328
Synopsis	328
Syntax	328
Description.....	328
Examples	328
Parameters.....	329
Inputs	330
Outputs.....	330
Notes.....	330
Related Links	330

Select-First	331
Synopsis	331
Syntax	331
Description.....	331
Examples	331
Parameters.....	331
Inputs	333
Outputs.....	333
Notes.....	333
Related Links	333
Select-Last	334
Synopsis	334
Syntax	334
Description.....	334
Examples	334
Parameters.....	335
Inputs	336
Outputs.....	336
Notes.....	336
Related Links	336
Select-Newest	337
Synopsis	337
Syntax	337
Description.....	337
Examples	337
Parameters.....	338
Inputs	339
Outputs.....	339
Notes.....	339
Related Links	339
Select-Oldest	340
Synopsis	340
Syntax	340
Description.....	340
Examples	340
Parameters.....	341
Inputs	342
Outputs.....	342
Notes.....	342
Related Links	342
Set-ConsoleColor	343
Synopsis	343
Syntax	343

Description	343
Examples	343
Parameters	343
Inputs	345
Outputs	345
Notes	345
Related Links	345
Set-ConsoleTitle	346
Synopsis	346
Syntax	346
Description	346
Examples	346
Parameters	346
Inputs	347
Outputs	348
Notes	348
Related Links	348
Set-PSAnsFileMap	349
Synopsis	349
Syntax	349
Description	349
Examples	349
Parameters	349
Inputs	351
Outputs	351
Notes	351
Related Links	352
Show-ANSISequence	353
Synopsis	353
Syntax	353
Description	353
Examples	353
Parameters	354
Inputs	356
Outputs	356
Related Links	356
Show-Tree	357
Synopsis	357
Syntax	357
Description	357
Examples	357
Parameters	359
Inputs	362

Outputs.....	362
Notes.....	362
Related Links	362
Test-EmptyFolder	363
Synopsis	363
Syntax	363
Description.....	363
Examples	363
Parameters.....	364
Inputs	365
Outputs.....	365
Notes.....	365
Related Links	365
Test-Expression.....	366
Synopsis	366
Syntax	366
Description.....	366
Examples	366
Parameters.....	369
Inputs	371
Outputs.....	371
Notes.....	371
Related Links	371
Test-ExpressionForm	372
Synopsis	372
Syntax	372
Description.....	372
Examples	372
Parameters.....	372
Inputs	372
Outputs.....	373
Notes.....	373
Related Links	373
Test-IsPSWindows	374
Synopsis	374
Syntax	374
Description.....	374
Examples	374
Parameters.....	374
Inputs	374
Outputs.....	374
Notes.....	374
Related Links	375

Test-WithCulture.....	376
Synopsis	376
Syntax	376
Description.....	376
Examples	376
Parameters.....	377
Inputs	378
Outputs.....	378
Notes.....	378
Related Links	378
Trace-Message	379
Synopsis	379
Syntax	379
Description.....	379
Examples	379
Parameters.....	380
Inputs	381
Outputs.....	381
Notes.....	381
Related Links	382
Write-ANSIProgress	383
Synopsis	383
Syntax	383
Description.....	383
Examples	383
Parameters.....	384
Inputs	386
Outputs.....	386
Notes.....	386
Related Links	386
Write-Detail	387
Synopsis	387
Syntax	387
Description.....	387
Examples	387
Parameters.....	388
Inputs	389
Outputs.....	389
Notes.....	389
Related Links	389
Change Log for PSScriptTools	390
Archive	392

Introduction

This manual is a PDF version of several module-related reference files as well as all of the command help. The goal is to provide a single source for all module documentation. Be aware that many of the source files contain internal cross-references. Best efforts have been made to port those links to this document, but a few links may fail to open. External links should work as expected.

If you need to ask a question or report a problem, please visit the module's [Github repository](#).

PSScriptTools Overview

Abstract

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 at [Issues](#). Any feedback is greatly appreciated.

The contents of this file and other documentation can be viewed using the `Open-PSScriptToolsHelp` command. You can also use `Get-PSScriptTools` to see a summary of module commands.

Please note that code samples have been formatted to fit an 80 character width. Some example code breaks lines without using line continuation characters. I'm trusting that you can figure out how to run the example.

Installation

You can get the current release from this repository or install this from the [PowerShell Gallery](#):

```
Install-Module PSScriptTools
```

or in PowerShell 7:

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

Starting in v2.2.0, the module was restructured to better support Desktop and Core editions. However, starting with v2.13.0, the module design has reverted. All module 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.

It is recommended to install this module from the PowerShell Gallery and not GitHub.

To remove the module from your system you can easily uninstall it with common PowerShell commands.

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

General Tools

Get-MyCounter

Get-MyCounter is an enhanced version of Get-Counter which is available on Windows platforms to retrieve performance counter data. One of the challenges with Get-Counter is how it formats results. Get-MyCounter takes the same information and writes a custom object to the pipeline that is easier to work with. You can pipe counters from Get-Counter to this command.

```
PS C:\> Get-Counter -List IPV4 | Get-MyCounter
```

Computername: PROSPERO			
Timestamp	Category	Counter	Value
11/4/2020 10:59:43 AM	ipv4	datagrams/sec	42.3661
11/4/2020 10:59:43 AM	ipv4	datagrams received/sec	29.5577
11/4/2020 10:59:43 AM	ipv4	datagrams received header errors	0
11/4/2020 10:59:43 AM	ipv4	datagrams received address errors	11815
11/4/2020 10:59:43 AM	ipv4	datagrams forwarded/sec	0
11/4/2020 10:59:43 AM	ipv4	datagrams received unknown protocol	0
11/4/2020 10:59:43 AM	ipv4	datagrams received discarded	10283
11/4/2020 10:59:43 AM	ipv4	datagrams received delivered/sec	14.7789
11/4/2020 10:59:43 AM	ipv4	datagrams sent/sec	12.8083
11/4/2020 10:59:43 AM	ipv4	datagrams outbound discarded	41
11/4/2020 10:59:43 AM	ipv4	datagrams outbound no route	26
11/4/2020 10:59:43 AM	ipv4	fragments received/sec	0
11/4/2020 10:59:43 AM	ipv4	fragments re-assembled/sec	0
11/4/2020 10:59:43 AM	ipv4	fragment re-assembly failures	0
11/4/2020 10:59:43 AM	ipv4	fragmented datagrams/sec	0
11/4/2020 10:59:43 AM	ipv4	fragmentation failures	0
11/4/2020 10:59:43 AM	ipv4	fragments created/sec	0

```
PS C:\>
```

```
PS C:\> Get-MyCounter -ComputerName thinkp1,prospero | Sort-Object -Property Computername
```

Computername: PROSPERO			
Timestamp	Category	Counter	Value
11/4/2020 11:17:42 AM	network interface(intel[r] ethernet connection [11] i219-lm)	bytes total/sec	34005.154
11/4/2020 11:17:42 AM	network interface(intel[r] wi-fi 6 ax201 160mhz)	bytes total/sec	5
11/4/2020 11:17:42 AM	processor(_total)	% processor time	1.2989
11/4/2020 11:17:42 AM	memory	% committed bytes in use	40.3811
11/4/2020 11:17:42 AM	memory	cache faults/sec	0
11/4/2020 11:17:42 AM	physicaldisk(_total)	% disk time	0.1627
11/4/2020 11:17:42 AM	physicaldisk(_total)	current disk queue length	0

Computername: THINKP1			
Timestamp	Category	Counter	Value
11/4/2020 11:17:44 AM	network interface(intel[r] ethernet connection [7] i219-v)	bytes total/sec	0
11/4/2020 11:17:44 AM	network interface(np cap loopback adapter)	bytes total/sec	0
11/4/2020 11:17:44 AM	network interface(intel[r] wireless-ac 9560 160mhz)	bytes total/sec	32650.3095
11/4/2020 11:17:44 AM	processor(_total)	% processor time	0.3377
11/4/2020 11:17:44 AM	memory	% committed bytes in use	13.2772
11/4/2020 11:17:44 AM	memory	cache faults/sec	0
11/4/2020 11:17:44 AM	physicaldisk(_total)	% disk time	0
11/4/2020 11:17:44 AM	physicaldisk(_total)	current disk queue length	0

One advantage of Get-MyCounter over Get-Counter is that the performance data is easier to work with.

```
Get-MyCounter '\IPV4\datagrams/sec' -MaxSamples 60 -SampleInterval 5 -Computer SRV1 | Export-Csv c:\work\srv1_ipperf.csv -NoTypeInformation
```

In this example, the performance counter is sampled 60 times every 5 seconds and the data is exported to a CSV file which could easily be opened in Microsoft Excel. Here's a sample of the output object.

```
Computername : SRV1
Category     : ipv4
Counter      : datagrams/sec
Instance     :
Value        : 66.0818918347238
Timestamp    : 11/4/2020 11:31:29 AM
```

`Get-MyCounter` writes a custom object to the pipeline which has an associated formatting file with custom views.

```
PS C:\>
PS C:\> Get-MyCounter -ComputerName ThinkP1 | Format-table -view category

Category: network interface(intel[r] ethernet connection [7] i219-v)
Computername      Timestamp          Counter          Value
-----           -----           -----
THINKP1           11/4/2020 11:21:27 AM bytes total/sec      0

Category: network interface(npcap loopback adapter)
Computername      Timestamp          Counter          Value
-----           -----           -----
THINKP1           11/4/2020 11:21:27 AM bytes total/sec      0

Category: network interface(intel[r] wireless-ac 9560 160mhz)
Computername      Timestamp          Counter          Value
-----           -----           -----
THINKP1           11/4/2020 11:21:27 AM bytes total/sec  26231.2466

Category: processor(_total)
Computername      Timestamp          Counter          Value
-----           -----           -----
THINKP1           11/4/2020 11:21:27 AM % processor time   1.1277

Category: memory
Computername      Timestamp          Counter          Value
-----           -----           -----
THINKP1           11/4/2020 11:21:27 AM % committed bytes in use  13.2964
THINKP1           11/4/2020 11:21:27 AM cache faults/sec       0

Category: physicaldisk(_total)
Computername      Timestamp          Counter          Value
-----           -----           -----
THINKP1           11/4/2020 11:21:27 AM % disk time        0
```

Get-DirectoryInfo

This command, which has an alias of `dw`, is designed to provide quick access to top-level directory information. The default behavior is to show the total number of files in the immediate directory. Although the command will also capture the total file size in the immediate directory. You can use the `Depth` parameter to recurse through a specified number of levels. The default displays use ANSI escape sequences.

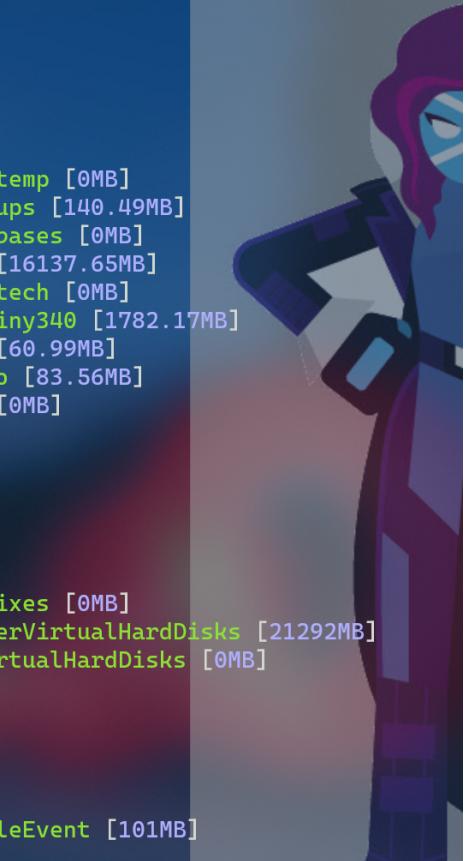


```
PS C:\> Get-DirectoryInfo D:\

Path: D:\

autolab [0]
Databases [0]
MailRecovery [0]
video [1]           backtemp [0]
Exports [0]          iso [6]
NewTiny340 [7]       OneDrive [1]
VMDisks [5]          VMs [0]           Backups [1]
JDHIT [23]
rip [5]
Windows10Upgrade [2] bovine320 [0]
Logitech [0]
temp [17]
```

The command output will use a wide format by default. However, other wide views are available.



```
PS C:\> Get-DirectoryInfo D:\ | Format-Wide -View sizemb

Path: D:\

autolab [0MB]
Backup [0.01MB]
bovine320 [0MB]
Exports [0MB]
JDHIT [35.58MB]
MailRecovery [0MB]
OneDrive [0MB]
temp [10.67MB]
VMDisks [68053MB]
Windows10Upgrade [6807.64MB]           backtemp [0MB]
                                         Backups [140.49MB]
                                         Databases [0MB]
                                         iso [16137.65MB]
                                         Logitech [0MB]
                                         NewTiny340 [1782.17MB]
                                         rip [60.99MB]
                                         video [83.56MB]
                                         VMs [0MB]

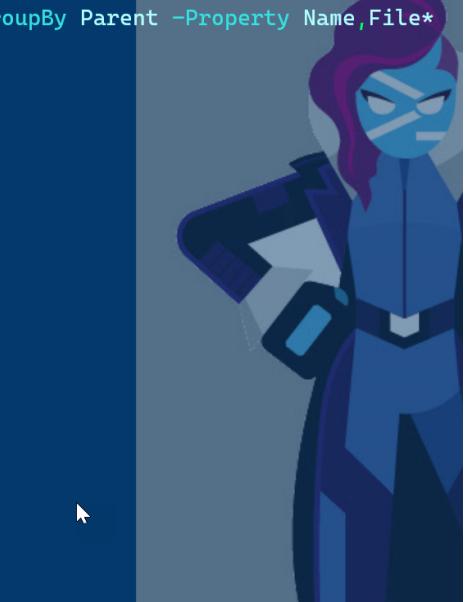
Path: D:\autolab

Configurations [0MB]
ISOs [16058.7MB]
Resources [0MB]           Hotfixes [0MB]
                           MasterVirtualHardDisks [21292MB]
                           VMVirtualHardDisks [0MB]

Path: D:\bovine320

Scripting [5.19MB]           TickleEvent [101MB]
```

You can use the object in other ways.



```
PS C:\> Get-DirectoryInfo D:\onedrive\Presentations\ | Format-Table -GroupBy Parent -Property Name,File*
```

Parent: D:\onedrive\Presentations		
Name	FileCount	FileSize
Azure-PowerShell-DeepDive	1	25664889
Creating CIM Commands	11	2909760
Creating REST Tools	6	307122
DataDecisions	5	3280395
Intro to DSC	11	14924113
JEA	1	4052530
JEA-Fundamentals	3	6207972
PowerShell Scheduled Jobs	3	1056828
PowerShell Scripting Secrets	1	1588874
PowerShell Traps	5	6340269
PowerShell Workshop	1	8229621
PSChatt	3	1309
PSChatt19	3	1378512
PSChicago2020	3	3917202
PSConfAsia18	1	14800282

Get-FormatView

PowerShell's formatting system includes a number of custom views that display objects in different ways. Unfortunately, this information is not readily available to a typical PowerShell user. This command displays the available views for a given object type.

```
PS C:\> Get-FormatView system.serviceprocess.servicecontroller
```

Type: System.ServiceProcess.ServiceController

Format	Name
-----	-----
Table	service
List	System.ServiceProcess.ServiceController
Table	service
Table	Ansi

This command has an alias of gfv.

Copy-PSFunction

This command is designed to solve the problem when you want to run a function loaded locally on a remote computer. Copy-PSFunction will copy a PowerShell function that is loaded in your current PowerShell session to a remote PowerShell session. The remote session must already be created. The copied function only exists remotely for the duration of the remote PowerShell session.

```
$s = new-psession -ComputerName win10 -cred $art
Copy-PSFunction Get-Status -Session $s
```

Once copied, you might use Invoke-Command to run it.

```
Invoke-Command { get-status -asstring } -session $s
```

If the function relies on external or additional files, you will have to copy them to the remote session separately.

Get-PSProfile

This command is designed for Windows systems and make it easy to identify all possible PowerShell profile scripts. Including those for hosts such as VSCode or the PowerShell ISE. The command writes a custom object to the pipeline which has defined formatting. The default view is a table.

```
PS C:\> get-psprofile
```

Name: PowerShell		
Scope	Path	Exists
AllUsersCurrentHost	C:\Program Files\PowerShell\7\Microsoft.PowerShell_profile.ps1	False
AllUsersAllHosts	C:\Program Files\PowerShell\7\profile.ps1	False
CurrentUserAllHosts	C:\Users\Jeff\Documents\PowerShell\profile.ps1	True
CurrentUserCurrentHost	C:\Users\Jeff\Documents\PowerShell\Microsoft.PowerShell_profile.ps1	True

Name: Windows PowerShell		
Scope	Path	Exists
AllUsersCurrentHost	C:\WINDOWS\System32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1	True
AllUsersAllHosts	C:\WINDOWS\System32\WindowsPowerShell\v1.0\profile.ps1	True
CurrentUserAllHosts	C:\Users\Jeff\Documents\WindowsPowerShell\profile.ps1	True
CurrentUserCurrentHost	C:\Users\Jeff\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1	True

There is also a list view.

```
PS C:\> get-psprofile | Where-Object {$_ .name -eq 'powershell'} | Format-List
```

```
Name: PowerShell

Scope      : AllUsersCurrentHost
Path       : C:\Program Files\PowerShell\7\Microsoft.PowerShell_profile.ps1
Exists     : False
LastModified :

Scope      : AllUsersAllHosts
Path       : C:\Program Files\PowerShell\7\profile.ps1
Exists     : False
LastModified :

Scope      : CurrentUserAllHosts
Path       : C:\Users\Jeff\Documents\PowerShell\profile.ps1
Exists     : True
LastModified : 9/9/2020 2:35:45 PM

Scope      : CurrentUserCurrentHost
Path       : C:\Users\Jeff\Documents\PowerShell\Microsoft.PowerShell_profile.ps1
Exists     : True
LastModified : 9/9/2020 2:03:44 PM
```

Get-MyAlias

Often you might define aliases for functions and scripts you use all of the time. It may difficult sometimes to remember them all or to find them in the default `Get-Alias` output. This command will list all currently defined aliases that are not part of the initial PowerShell state.

CommandType	Name	Version	Source
Alias	awk -> awk.exe	2.27.0	PSScriptTools
Alias	cart -> ConvertTo-ASCIIArt	2.27.0	PSScriptTools
Alias	cb -> Set-Clipboard	2.27.0	PSScriptTools
Alias	cc -> Copy-Command	2.27.0	PSScriptTools
Alias	cfn -> New-CustomFileName	2.27.0	PSScriptTools
Alias	cft -> ConvertFrom-Text	2.27.0	PSScriptTools
Alias	chc -> Convert-HashTableToCode	2.27.0	PSScriptTools
Alias	che -> Copy-HelpExample	2.27.0	PSScriptTools
Alias	clr -> Convert-EventLogRecord	2.27.0	PSScriptTools
Alias	clt -> ConvertTo-LocalTime	2.27.0	PSScriptTools
Alias	cmo -> Compare-Module	2.27.0	PSScriptTools
Alias	ctm -> ConvertTo-Markdown	2.27.0	PSScriptTools
Alias	cwg -> ConvertTo-WPFGrid	2.27.0	PSScriptTools
Alias	daily -> dailysummary.ps1	2.27.0	PSScriptTools
Alias	df -> Get-DiskFree		
Alias	dirdate -> Get-Dirdate		
Alias	fcc -> Find-CimClass	2.27.0	PSScriptTools
Alias	ff -> firefox.exe		
Alias	fhx -> Format-Hex	7.0.0.0	Microsoft.PowerShell.utility
Alias	fimo -> Find-Module	2.2.4.1	PowerShellGet
Alias	First -> Select-First	2.27.0	PSScriptTools
Alias	fp -> Format-Percent	2.27.0	PSScriptTools
Alias	frut -> ConvertFrom-UTCTime	2.27.0	PSScriptTools
Alias	fs -> Format-String	2.27.0	PSScriptTools
Alias	fv -> Format-Value	2.27.0	PSScriptTools
Alias	gcb -> Get-Clipboard	7.0.0.0	Microsoft.PowerShell.Management
Alias	gcm2 -> Get-Command2		
Alias	gin -> Get-ComputerInfo	7.0.0.0	Microsoft.PowerShell.Management
Alias	gma -> Get-MyAlias	2.27.0	PSScriptTools

These are all aliases defined in the current session that aren't part of the initial session state. You can filter aliases to make it easier to find those that aren't defined in a module. These aliases should be ones created in your stand-alone scripts or PowerShell profile.

CommandType	Name	Version	Source
Alias	awk -> awk.exe		
Alias	cart -> ConvertTo-ASCIIArt		
Alias	cb -> Set-Clipboard		
Alias	daily -> dailysummary.ps1		
Alias	df -> Get-DiskFree		
Alias	dirdate -> Get-Dirdate		
Alias	ff -> firefox.exe		
Alias	gcm2 -> Get-Command2		
Alias	gmf -> Get-MyFunctions		
Alias	grep -> grep.exe		
Alias	grok -> Get-Help		
Alias	gst -> Get-Status		

The PSScriptTools module also includes a custom formatting file for alias objects which you can use with `Get-Alias` or `Get-MyAlias`.

```
Get-Alias | Sort-Object Source | Format-Table -View source
```

gcb	Get-Clipboard
gtz	Get-TimeZone
gin	Get-ComputerInfo
scb	Set-Clipboard

↳ Source: Microsoft.PowerShell.Utility 7.0.0.0

Name	Definition
---	-----
fhx	Format-Hex

Source: PowerShellGet 2.2.4.1

Name	Definition
---	-----
inmo	Install-Module
pumo	Publish-Module
upmo	Update-Module
fimo	Find-Module

Source: PSScriptTools 2.27.0

Name	Definition
---	-----
Tee-Verbose	Out-VerboseTee
tex	Test-Expression
pswho	Get-PSWho
che	Copy-HelpExample

This command has an alias of gma.

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 the 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...

Verb: Show

Name	Alias	Type	Synopsis
----	-----	-----	-----
Show-Calendar	scal	Function	Display a color...
Show-GuiCalendar	gcal	Function	Display a WPF-b...

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		Create a text border around a string.

Verb: Compare

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

...

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 the 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 | Format-List
```

```
IP : 1.1.1.1
Name : APNIC-1
RegisteredOrganization : Asia Pacific Network Information Centre
City : South Brisbane
StartAddress : 1.0.0.0
EndAddress : 1.255.255.255
NetBlocks : 1.0.0.0/8
Updated : 7/30/2010 9:23:43 AM
```

```
IP : 8.8.8.8
Name : LVLT-GOGL-8-8-8
RegisteredOrganization : Google LLC
City : Mountain View
StartAddress : 8.8.8.0
EndAddress : 8.8.8.255
NetBlocks : 8.8.8.0/24
Updated : 3/14/2014 4:52:05 PM
```

```
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
```

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.

```
Get-WindowsVersion -Computername win10,srv1,srv2 -Credential company\artd
```

Computername: WIN10					
ProductName	EditionID	ReleaseID	Build	InstalledUTC	
Windows 10 Enterprise Evaluation	EnterpriseEval	1909	18363	5/30/2020 2:49:55 PM	
Computername: SRV1					
ProductName	EditionID	ReleaseID	Build	InstalledUTC	
Windows Server 2016 Standard Evaluation	ServerStandardEval	1607	14393	5/30/2020 2:49:15 PM	
Computername: SRV2					
ProductName	EditionID	ReleaseID	Build	InstalledUTC	
Windows Server 2016 Standard Evaluation	ServerStandardEval	1607	14393	5/30/2020 2:50:00 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. by default, 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 -setlocation  
PS Webinar:>
```

You can use the first word in the leaf location or specify something completely different.

```
New-PSDrivehere \\ds416\backup\ Backup
```

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.

```
PS C:\> Get-MyVariable  
  
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-Object -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.

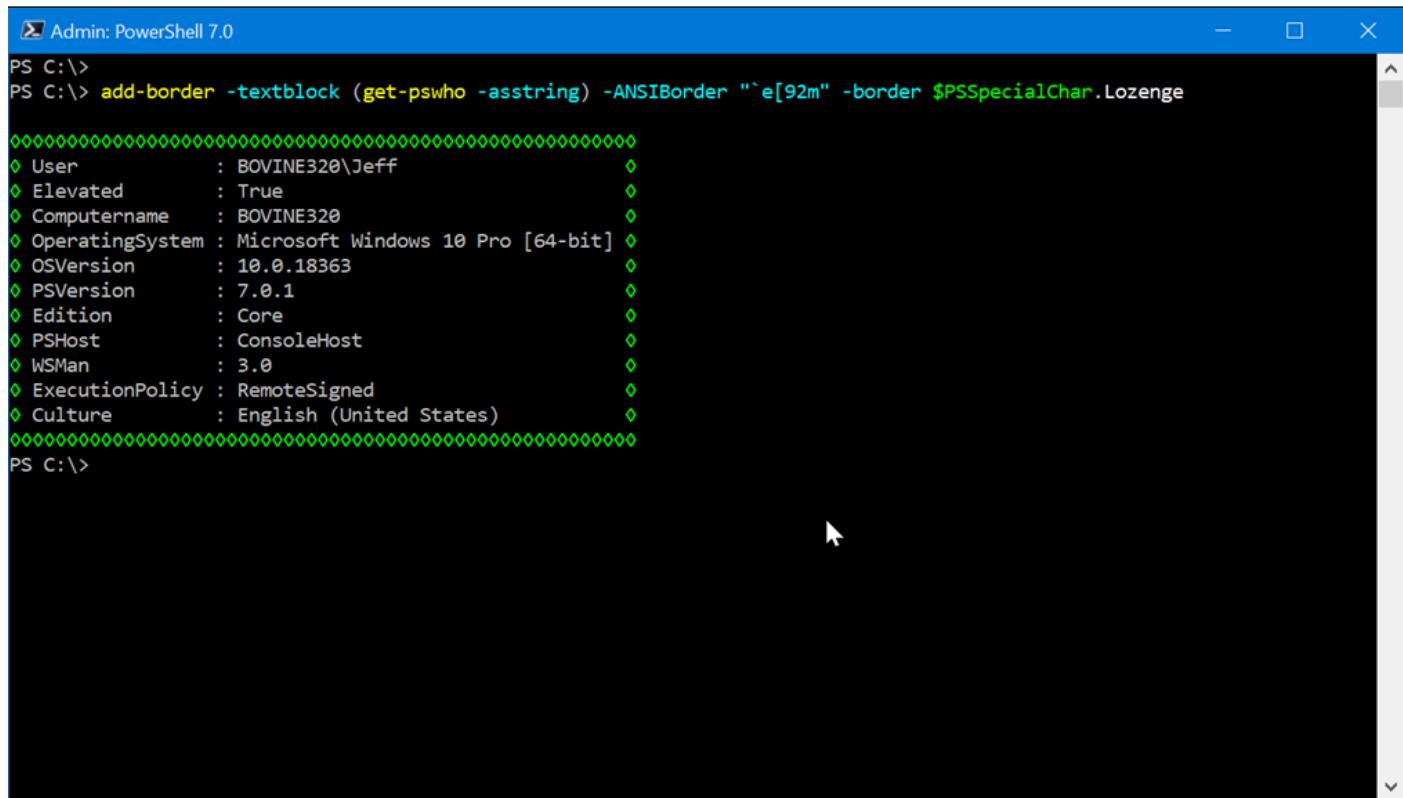
Get-PSWho

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)
```

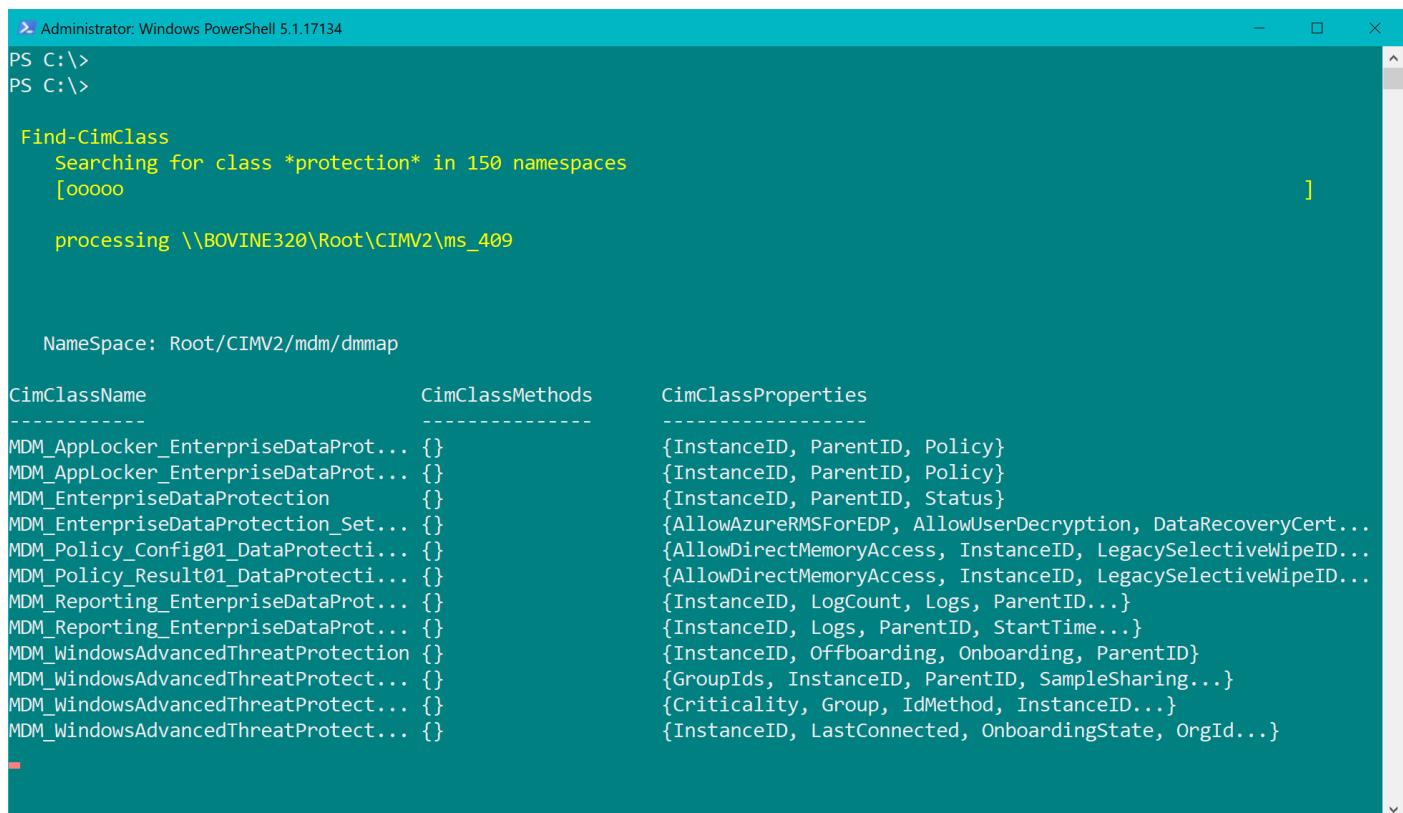
You can also turn this into a text block using the `AsString` parameter. This is helpful when you want to include the output in some type of report.



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



```
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_Reportin... EnterpriseDataProt... {}         {InstanceID, LogCount, Logs, ParentID...}
MDM_Reportin... 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

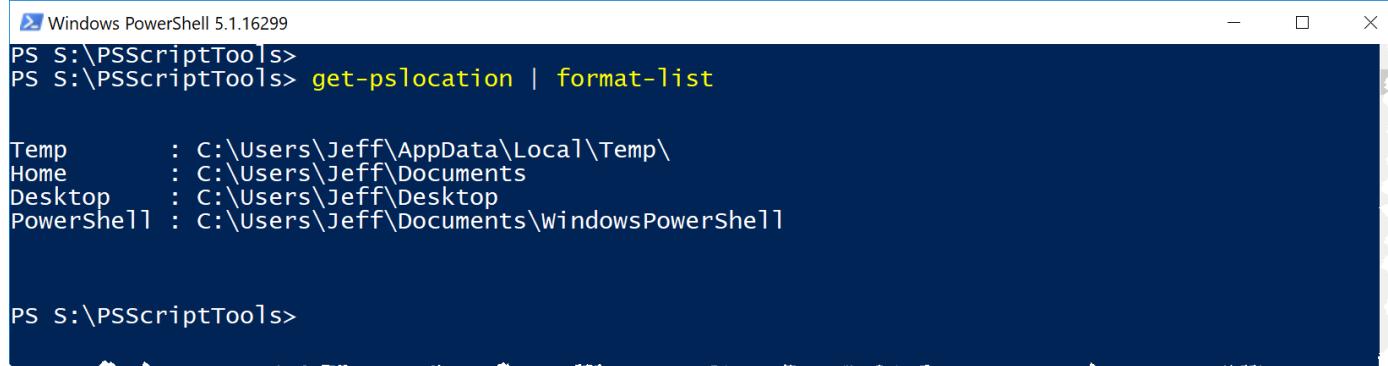
During the course of your PowerShell work, you may discover that some commands and scripts can leave behind runspaces such as ConvertTo-WPFGrid. You may even deliberately be creating additional runspaces. These runspaces will remain until you exit your PowerShell session. Or use this command to cleanly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

```
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>
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
---   ---      -----
/tmp/ /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
```

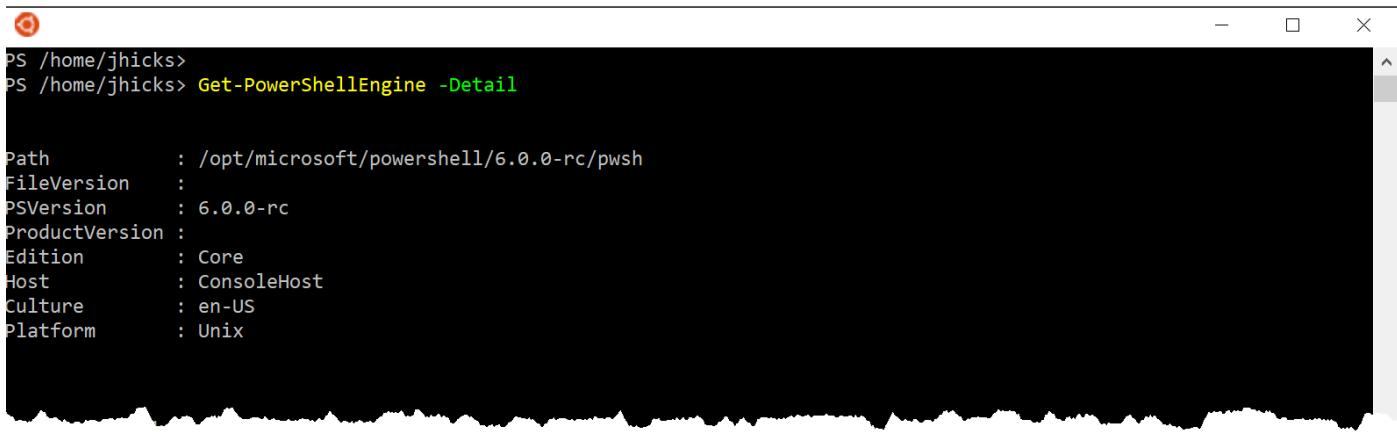
You can also get detailed information.

```
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 Co...  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 an 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...
```

_report 09/25/2019 09:57:12_
```

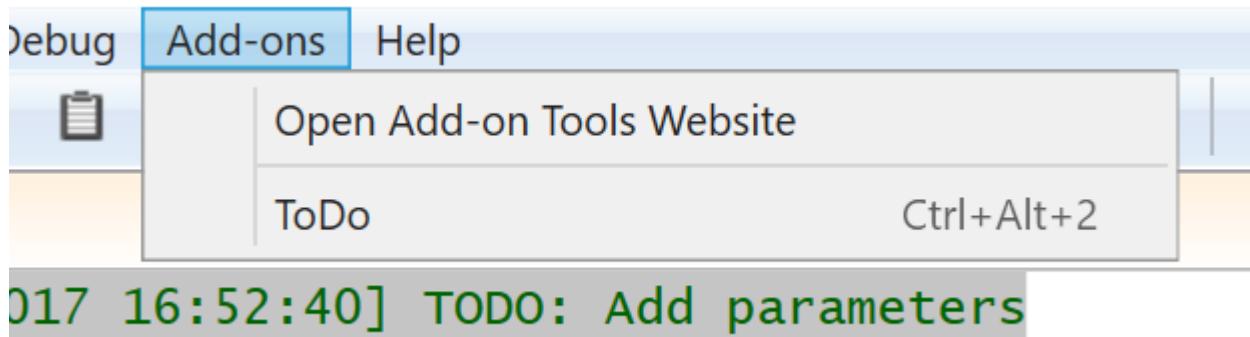
Because the function writes markdown to the pipeline you will need to pipe it to a command `Out-File` to create a file.

To Do

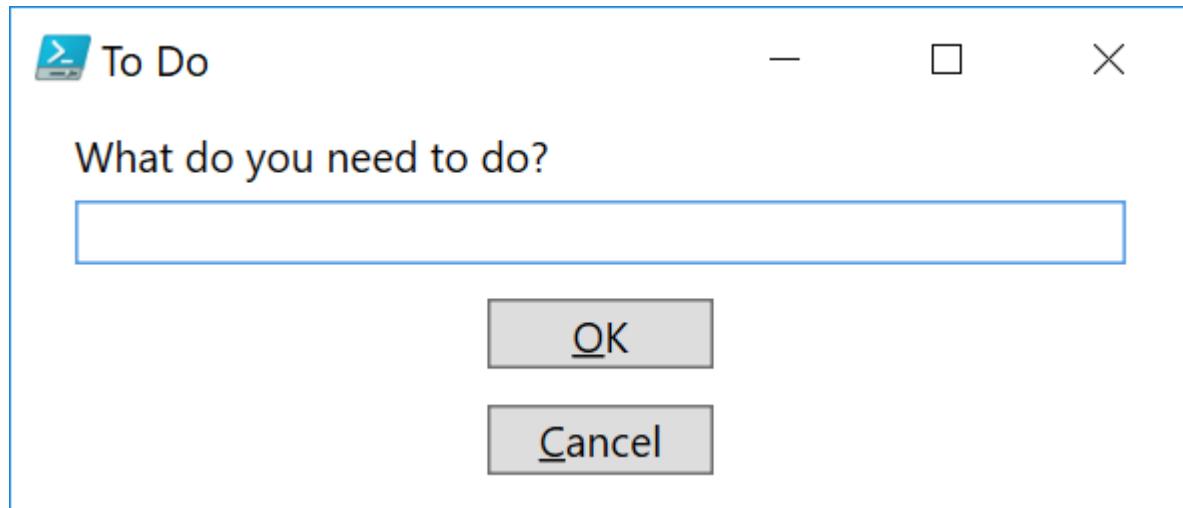
Because this module is intended to make scripting easier for you, it adds options to insert ToDo statements into PowerShell files. If you are using the PowerShell ISE or VS Code and import this module, it will add the capability to insert a line like this:

```
# [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.

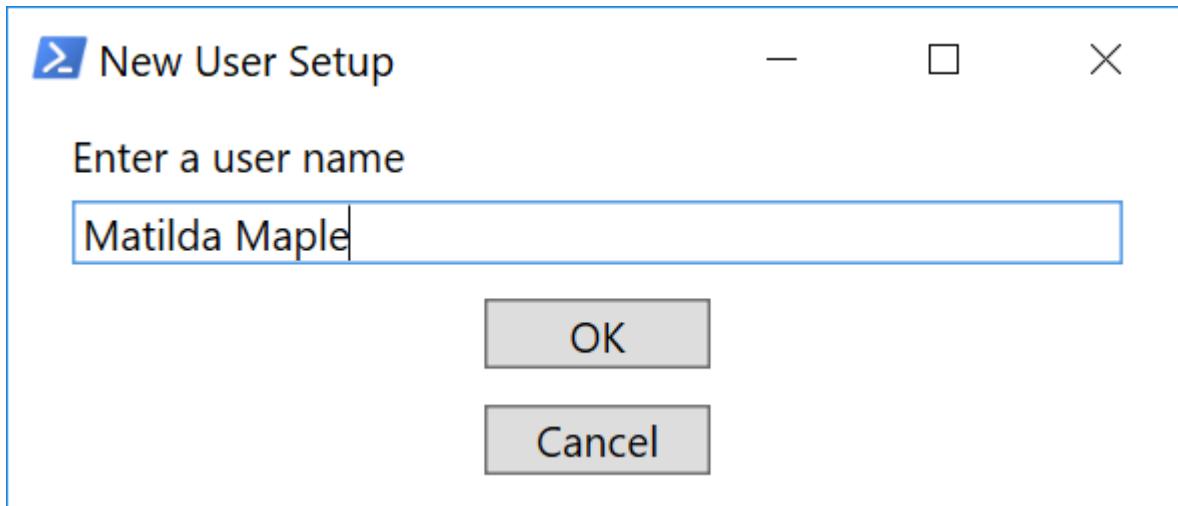
In VS Code, access the command palette (Ctrl+Shift+P) and then `PowerShell: Show Additional Commands from PowerShell Modules`. Select `Insert ToDo` from the list and you'll get the same input box. Note that this will only work for PowerShell files.

Graphical Tools

Invoke-InputBox

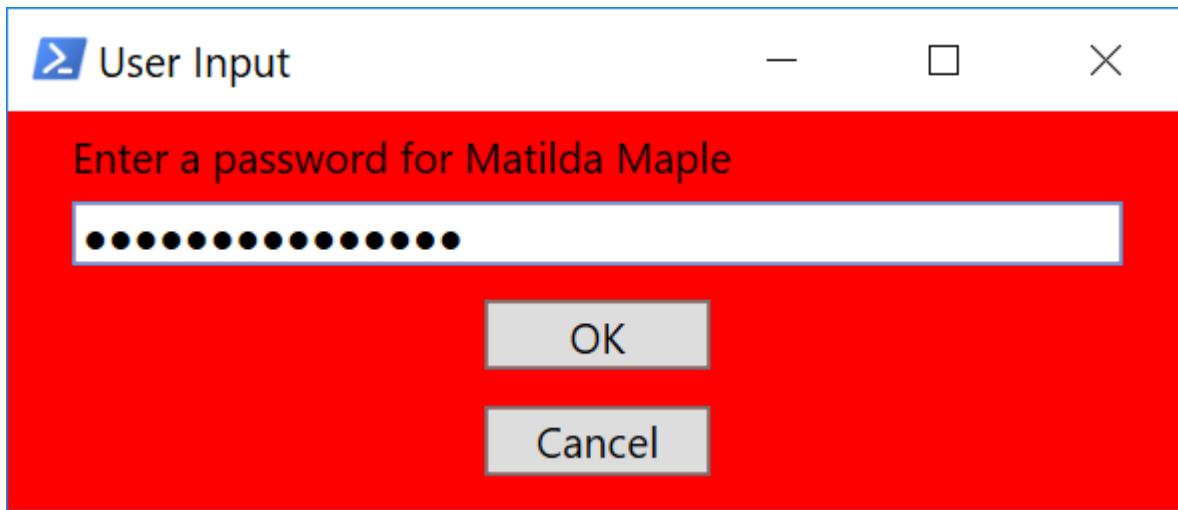
This function is a graphical replacement for `Read-Host`. It creates a simple WPF form that you can use to get user input. The value of the text box will be written to the pipeline.

```
$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 the form's background color. This function will **not** work in PowerShell Core.

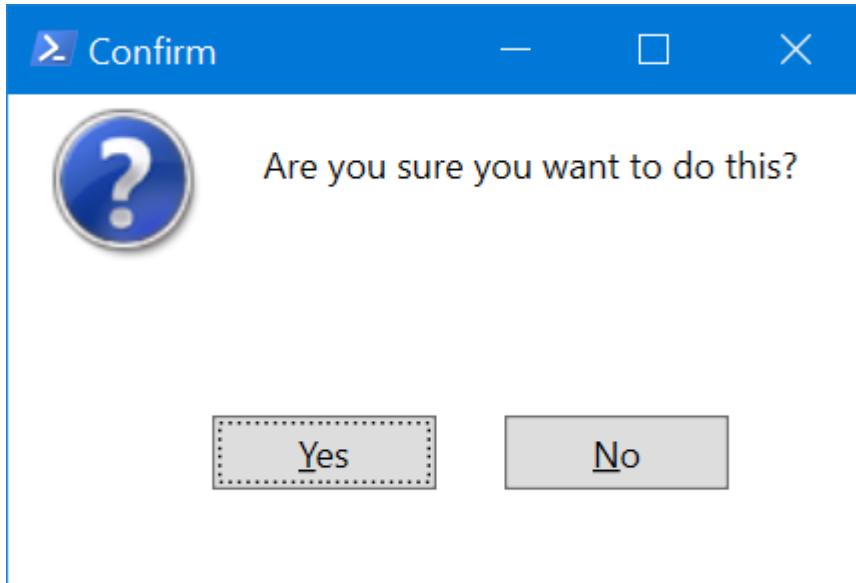
New-WPFMessageBox

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

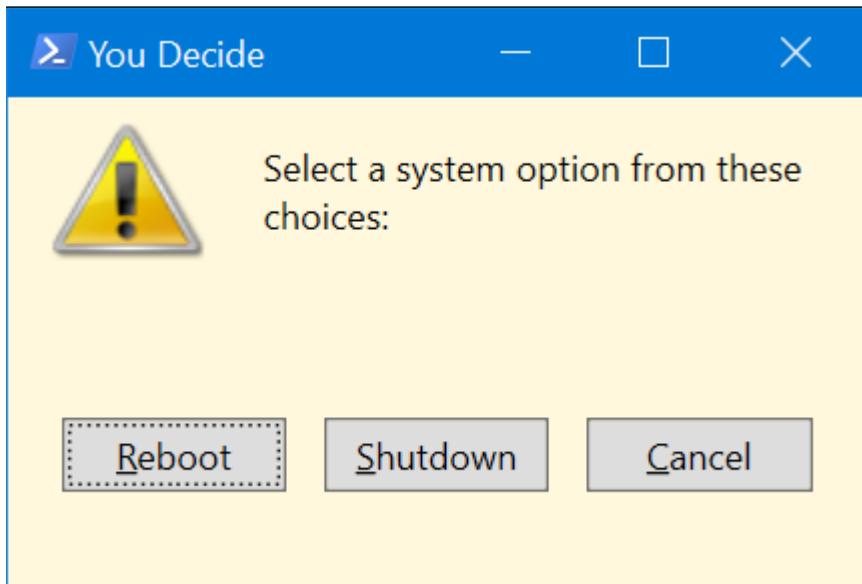
You can also create an ordered hashtable of your own buttons and values. It is assumed you will typically use this function in a script where you can capture the output and take some action based on the value.

```
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.

```
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-Object -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
```

The idea is that you can copy the output of the command into a script file.

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-Object 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 several functions that simplify the use of `Select-Object` or `Select-Object` in conjunction with `Where-Object`. The commands are intended to make it easier to select objects in a pipelined expression. The commands include features so that you can sort the incoming objects on a given property first.

Select-First

Normally, you might run a command with `Select-Object` like this:

```
Get-Process | Select-Object -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 |

To streamline the process a bit, you can use `Select-First`.

```
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 |

Even better, use the command alias `first`.

```
Get-Process | Sort-Object ws -Descending | first 5
```

Select-Last

You can perform a similar operating using `Select-Last` or its alias `last`.

```
Get-Childitem -path c:\scripts\*.ps1 | Sort-Object lastwritetime | last 10
```

Select-After

`Select-After` is a simplified version of `Select-Object`. The premise is that you can pipe a collection of objects to this command and select objects after a given datetime, based on a property, like `LastWriteTime`, which is the default. This command has an alias of `after`.

```
Get-ChildItem -path c:\scripts\ -file | after 11/1/2020
```

Directory: C:\Scripts

| Mode | LastWriteTime | Length | Name |
|-------|--------------------|--------|----------------------------|
| -a--- | 11/2/2020 11:08 AM | 3522 | Get-ServiceWPFRunspace.ps1 |
| -a--- | 11/1/2020 11:05 AM | 5321 | Trace.ps1 |
| -a--- | 11/2/2020 11:39 AM | 2321 | WinFormDemo2.ps1 |

Or you can specify property depending on the object.

```
Get-Process | after (Get-Date).Addminutes(-1) -Property StartTime
```

| NPM(K) | PM(M) | WS(M) | CPU(s) | Id | SI | ProcessName |
|--------|-------|-------|--------|-------|----|-------------|
| 13 | 3.14 | 13.73 | 0.05 | 19156 | 2 | notepad |

This is selecting all processes that started within the last minute.

Select-Before

Select-Before is the opposite of Select-After.

```
Get-ChildItem -path c:\scripts -file | before 1/1/2008
```

Directory: C:\Scripts

| Mode | LastWriteTime | Length | Name |
|-------|-------------------|--------|--------------------|
| -a--- | 12/5/2007 2:19 PM | 29618 | 1000MaleNames.txt |
| -a--- | 4/8/2006 10:27 AM | 3779 | 530215.ps1 |
| -a--- | 8/7/2005 1:00 AM | 4286 | ADUser.wsc |
| -a--- | 9/18/2006 9:27 PM | 1601 | allserviceinfo.ps1 |
| ... | | | |

As with Select-After, you can specify a property to use.

```
Get-Aduser -filter * -Properties WhenCreated |
Before 11/1/2020 -Property WhenCreated | Select-Object Name,WhenCreated
```

| Name | WhenCreated |
|----------------|-----------------------|
| Administrator | 10/26/2020 6:47:39 PM |
| Guest | 10/26/2020 6:47:39 PM |
| DefaultAccount | 10/26/2020 6:47:39 PM |
| krbtgt | 10/26/2020 6:50:47 PM |
| MaryL | 10/26/2020 6:56:24 PM |
| ArtD | 10/26/2020 6:56:24 PM |
| AprilS | 10/26/2020 6:56:25 PM |
| MikeS | 10/26/2020 6:56:25 PM |
| ... | |

Select-Newest

Select-Newest is designed to make it easier to select X number of objects based on a datetime property. The default property value is LastWriteTime.

```
Get-ChildItem -path d:\temp -file | newest 10
```

| Mode | LastWriteTime | Length | Name |
|-------|---------------------|---------|-----------------------------------|
| -a--- | 11/4/2020 5:12 PM | 5149954 | watcherlog.txt |
| -a--- | 11/3/2020 10:00 PM | 3215 | DailyIncremental_202011031000.txt |
| -a--- | 11/2/2020 10:00 PM | 11152 | DailyIncremental_202011021000.txt |
| -a--- | 11/2/2020 3:40 PM | 852 | t.ps1 |
| -a--- | 11/1/2020 10:00 PM | 2376 | DailyIncremental_202011011000.txt |
| -a--- | 10/31/2020 10:00 PM | 3150 | DailyIncremental_202010311000.txt |
| -a--- | 10/30/2020 10:07 PM | 17844 | WeeklyFull_202010301000.txt |
| -a--- | 10/30/2020 1:00 PM | 208699 | datatfile-5.png |
| -a--- | 10/30/2020 12:57 PM | 1264567 | datatfile-4.png |
| -a--- | 10/30/2020 12:27 PM | 421341 | datatfile-3.png |

Or specify a property.

```
Get-ADUser -filter * -Properties WhenCreated |
Select-Newest 5 -Property WhenCreated |
Select-object DistinguishedName,WhenCreated
```

| DistinguishedName | WhenCreated |
|--|-----------------------|
| CN=Marcia Brady,OU=Employees,DC=Company,DC=PRI | 11/4/2020 3:15:27 PM |
| CN=Gladys Kravitz,OU=Employees,DC=Company,DC=PRI | 11/4/2020 3:14:45 PM |
| CN=S.Talone,OU=Employees,DC=Company,DC=PRI | 10/26/2020 3:56:31 PM |
| CN=A.Fieldhouse,OU=Employees,DC=Company,DC=PRI | 10/26/2020 3:56:31 PM |
| CN=K.Moshos,OU=Employees,DC=Company,DC=PRI | 10/26/2020 3:56:31 PM |

Select-Oldest

Select-Oldest is the opposite of Select-Newest and works the same way.

```
Get-Process | newest 5 -Property StartTime
```

| Handles | NPM(K) | PM(K) | WS(K) | CPU(s) | ID | SI | ProcessName |
|---------|--------|-------|-------|--------|-------|----|--------------------|
| 145 | 8 | 1692 | 7396 | 0.02 | 9676 | 0 | SearchFilterHost |
| 344 | 13 | 2604 | 13340 | 0.02 | 33668 | 0 | SearchProtocolHost |
| 114 | 7 | 1340 | 6116 | 0.02 | 35028 | 0 | svchost |
| 140 | 8 | 2684 | 8796 | 0.03 | 32552 | 0 | svchost |
| 118 | 8 | 1580 | 7476 | 0.02 | 35668 | 0 | svchost |

These custom Select commands are not necessarily designed for performance and there may be better ways to achieve the same results from these examples.

Time Functions

The module has a couple of 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 group of 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 a 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-Object 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 : 64800000
```

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

Console Utilities

ConvertTo-ASCIIArt

ConvertTo-ASCIIArt can be used to transform a string of text into ASCII art. It utilizes the web service at <https://artii.herokuapp.com> which allows you to transform text. You might use this to create headers for your scripts or PowerShell profile.

```
PS C:\> ConvertTo-ASCIIArt $env:COMPUTERNAME -Font cybermedium
[ _ ] [ - / \ ] [ | ] [ - ] [ - / \ ] [ | ]
| | R | I | S | E | R | | |
PS C:\>
```

I used this command to convert a string that I'm using with Get-PSScriptTools to display a "splash" header.

```
$h = @"
[ _ ] [ - / \ ] [ | ] [ - ] [ - / \ ] [ | ]
[ _ ] [ \ ] [ / ] [ | ] [ - ] [ \ ] [ / ] [ <
[ _ ] [ - / \ ] [ | ] [ - ] [ - / \ ] [ / ] [ / ]
[ _ ]
"@"
$([char]0x1b)[1;38;5;177m$h$([char]0x1b)[0m" | Write-Host
```

```
PS C:\> Get-PSScriptTools
[ _ ] [ - / \ ] [ | ] [ - ] [ - / \ ] [ | ]
[ _ ] [ \ ] [ / ] [ | ] [ - ] [ \ ] [ / ] [ <
[ _ ] [ - / \ ] [ | ] [ - ] [ - / \ ] [ / ] [ / ]
[ _ ]

Verb: Add
Name          Alias          Synopsis
----          ----          -----
Add-Border    Create a text border around a string.

Verb: Compare
Name          Alias          Synopsis
----          ----          -----
Compare-Module cmo           Compare PowerShell module versions.
Compare-Script csc           Compare PowerShell script versions.
```

Expect a period of trial and error to find a good font that works with your text. The shorter your text, the better.

Out-Copy

This command is intended for writers and those who need to document with PowerShell. You can pipe any command to this function and you will get the regular output in your PowerShell session. Simultaneously a copy of the output will be sent to the Windows clipboard. The copied output will include a prompt constructed from the current location unless you use the `CommandOnly` parameter.

You can run a command like this:

```
Get-Process | Sort WS -Descending | Select -first 5 | Out-Copy
```

And this text will be copied to the clipboard.

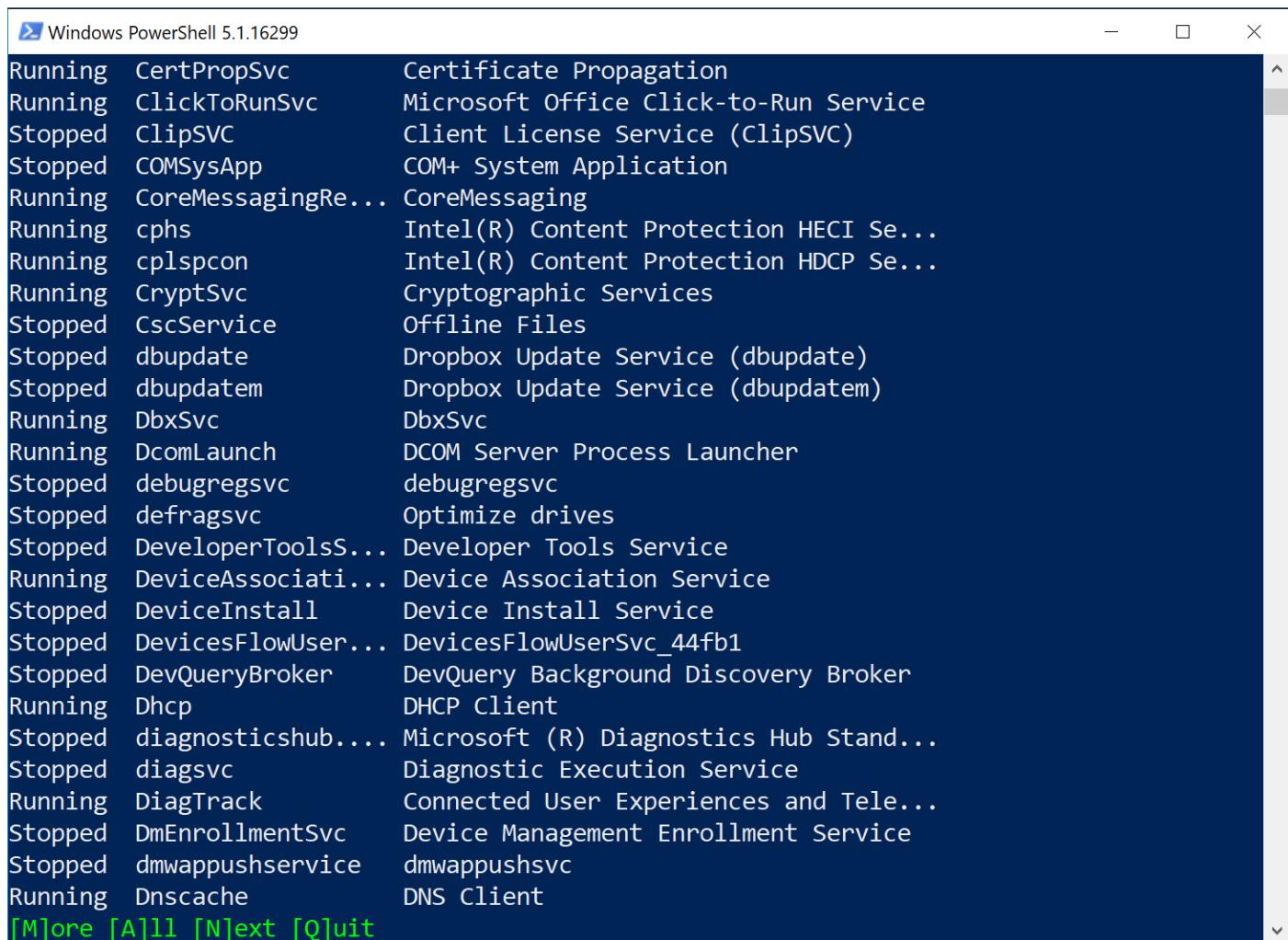
```
PS C:\> Get-Process | Sort WS -Descending | Select -first 5
```

| Handles | NPM(K) | PM(K) | WS(K) | CPU(s) | Id | SI | ProcessName |
|---------|--------|--------|--------|--------|-------|----|-------------|
| 1849 | 253 | 810320 | 820112 | 445.38 | 17860 | 1 | firefox |
| 765 | 61 | 949028 | 758200 | 23.36 | 6052 | 0 | sqlservr |
| 446 | 115 | 441860 | 471032 | 28.59 | 18204 | 1 | Teams |
| 2307 | 192 | 313204 | 459616 | 325.23 | 15748 | 1 | firefox |
| 2050 | 163 | 451744 | 433772 | 94.63 | 19780 | 1 | thunderbird |

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
```



A screenshot of a Windows PowerShell window titled "Windows PowerShell 5.1.16299". The window displays a list of system services with their status, names, and descriptions. The services listed include CertPropSvc, ClickToRunSvc, ClipSVC, COMSysApp, CoreMessagingRe..., cphs, cplspcon, CryptSvc, CscService, dbupdate, dbupdatem, DbxSvc, DcomLaunch, debugregsvc, defragsvc, DeveloperToolsS..., DeviceAssociati..., DeviceInstall, DevicesFlowUser..., DevQueryBroker, Dhcp, diagnosticshub...., diagsvc, DiagTrack, DmEnrollmentSvc, dmwappushservice, Dnscache, and [M]ore [A]ll [N]ext [Q]uit. The descriptions for each service are truncated with ellipses.

| Status | Service Name | Description |
|---------|--------------------|--|
| 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 | cplspcon | 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:\> PS C:\> Get-Service | Out-ConditionalColor -PropertyConditions @{{Stopped="magenta"} } -property Status

Status      Name          DisplayName
----      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    AppVClient    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
----  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.

```
if (Test-IsAdministrator) {  
    Set-ConsoleTitle "Administrator: $($PSVersionTable.PSVersion)"  
}
```

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 *  
*****  
PS C:\> _
```

```
$params =@{  
    textblock = (Get-PSWho |AsString ).trim()  
    ANSIBorder = ``e[38;5;214m"  
    Character = ([char]0x25CA)  
    ANSIText = ``e[38;5;225m"  
}  
Add-Border @params
```

```
PS C:\> Add-Border @params
```

PS C:\>

Show-Tree

Shows the specified path as a graphical tree in the console. This is intended as a 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`.

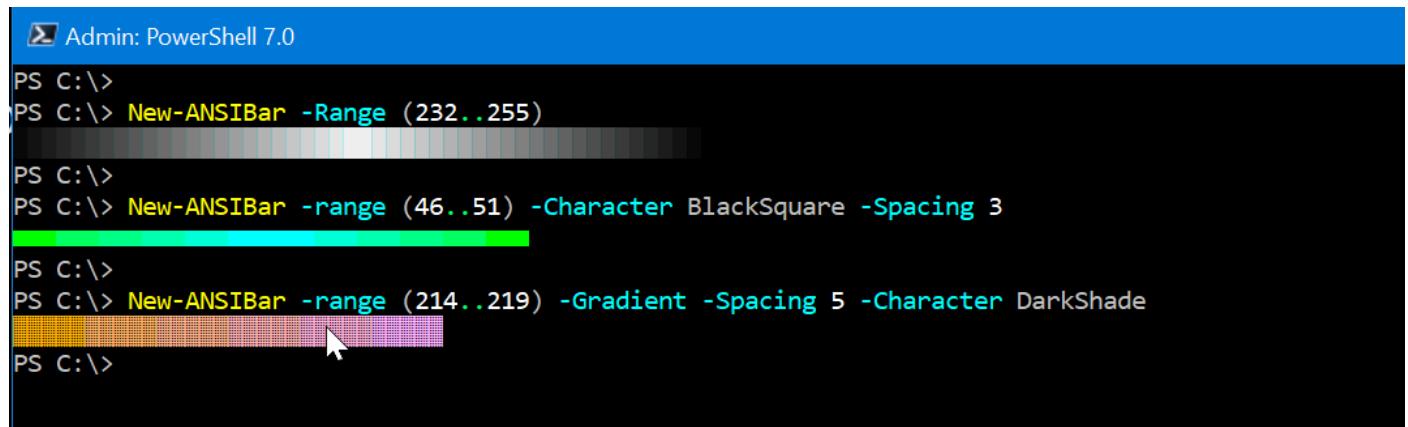
```
PS C:\> pstree c:\work\alpha -files -properties LastWriteTime,Length

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.



```
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:\> 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.

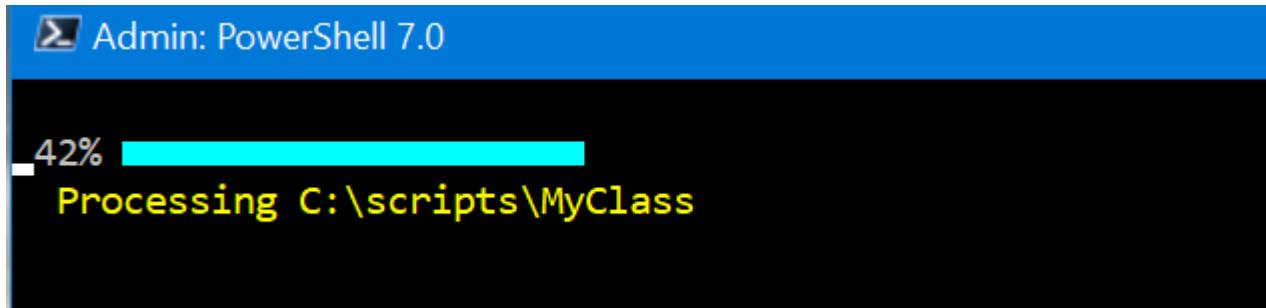
```
PS C:\> Admin: PowerShell 7.0
PS C:\> Write-ANSIProgress -PercentComplete .78 -BarSymbol Circle -ProgressColor "$([char]0x1b)[92m"
78% 
PS C:\>
PS C:\> ■

PS C:\> Get-CimInstance win32_operatingsystem |
>> Select-Object @｛Name = "MemGB"; Expression = {$_._TotalVisibleMemorySize/1MB -as [int]}},
>> FreePhysicalMemory,
>> @｛Name = "PctFree"; Expression = { $pct = $($([math]::round($_.freephysicalmemory/$_.totalvisiblememorysize, 2));
>> Write-ANSIProgress -PercentComplete $pct | Select-Object -last 1}}}

MemGB FreePhysicalMemory PctFree
-----
32      14687996  44% 
PS C:\>
```

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. The title bar contains the text 'Admin: PowerShell 7.0'. The main area of the window displays the following text:

```
42% ██████████
Processing C:\scripts\ MyClass
```

The progress bar is at 42%, and the text 'Processing C:\scripts\ MyClass' is displayed below it.

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  
wSlhoeepIrl  
PS C:\> Format-String "!MySecretPWord" -Randomize  
-Replace @{S="$";e=&{Get-Random -min 1 -max 9};o="^"} -Reverse  
yr7!^7WcMtr$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:

```
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

Trace-Message

Trace-Message is designed to be used with your script or function on a Windows platform. Its purpose is to create a graphical trace window using Windows Presentation Foundation (WPF). Inside the function or script, you can use this command to send messages to the window. When finished, you have an option to save the output to a text file.

There are 3 steps to using this function. First, in your code, you need to create a boolean global variable called TraceEnabled. When the value is \$True, the Trace-Message command will run. When set to false, the command will be ignored. Second, you need to initialize a form, specifying the title and dimensions. Finally, you can send trace messages to the window. All messages are prepended with a timestamp.

Here is a code excerpt from \$PSSamplePath\Get-Status.ps1:

```
Function Get-Status {
    [cmdletbinding(DefaultParameterSetName = 'name')]
    [alias("gst")]
    Param(
        ...
        [Parameter(HelpMessage="Enable with graphical trace window")]
        [switch]$Trace
    )

    Begin {
        Write-Verbose "[{0} BEGIN] Starting $($myinvocation.mycommand)"
        if ($trace) {
            $global:TraceEnabled = $True
            $traceTitle = "{0} Trace Log" -f $($myinvocation.MyCommand)
            Trace-Message -title $traceTitle
            Trace "Starting $($myinvocation.mycommand)"
        }
    } #begin
    Process {
        Write-Verbose "[{0} PROCESS] Using parameter set $($pscmdlet.ParameterSetName)"
        Trace-Message -message "Using parameter set: $($pscmdlet.ParameterSetName)"
        ...
    } #close function
    $data = Get-Status -trace
```

The trace window starts with pre-defined metadata.

```
16:50:34.2557095 - User: PROSPERO\Jeff
16:50:34.5196190 - Elevated : True
16:50:34.7823501 - Computer: PROSPERO
16:50:35.0466358 - OS: Microsoft Windows 10 Pro
16:50:35.3115122 - Ver.: 10.0.19042
16:50:35.5770522 - Architecture: 64-bit
16:50:35.8423966 - ****
16:50:36.0942951 - Starting Get-Status
16:50:36.3466886 - Using parameter set: name
16:50:36.6105952 - Create a temporary Cimsession
16:50:36.9161293 - Querying Win32_OperatingSystem
16:50:37.2267868 - Querying Win32_Logicaldisk
16:50:37.4999070 - Creating new object
16:50:37.7541594 -
Name                Value
-----
Computername        PROSPERO
Uptime              09:49:33.2247843
PctFreeMem          63.95
PctFreeC             50.96
PctFreeD             78.16

16:50:38.0186744 - Removing temporary cimsession
16:50:38.2847403 - Ending Get-Status
```

Quit Save

Your output might vary from this screen shot.

You have an option to Save the text. The default location is \$env:temp.

Get-CommandSyntax

Some PowerShell commands are provider aware and may have special syntax or parameters depending on what PSDrive you are using when you run the command. In Windows PowerShell, the help system could show you syntax based on a given path. However, this no longer appears to work. `Get-CommandSyntax` is intended as an alternative and should work in both Windows PowerShell and PowerShell 7.

Specify a cmdlet or function name, and the output will display the syntax detected when using different providers.

```
Get-CommandSyntax -Name Get-Item
```

Dynamic parameters will be highlighted with an ANSI-escape sequence.

FileSystem

```
Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [-Stream <string[]>] [<CommonParameters>]
Get-Item -LiteralPath <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [-Stream <string[]>] [<CommonParameters>]
```

Function

```
Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [<CommonParameters>]
```

```
Get-Item -LiteralPath <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [<CommonParameters>]
```

Variable

```
Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [<CommonParameters>]
```

```
Get-Item -LiteralPath <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [<CommonParameters>]
```

Certificate

```
Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [-CodeSigningCert] [-DocumentEncryptionCert] [-SSLServerAuthentication] [-DnsName <dnsNameRepresentation>] [-Eku <string[]>] [-ExpiringInDays <int>] [<CommonParameters>]
```

This command has an alias of `gsyn`.

Test-Expression

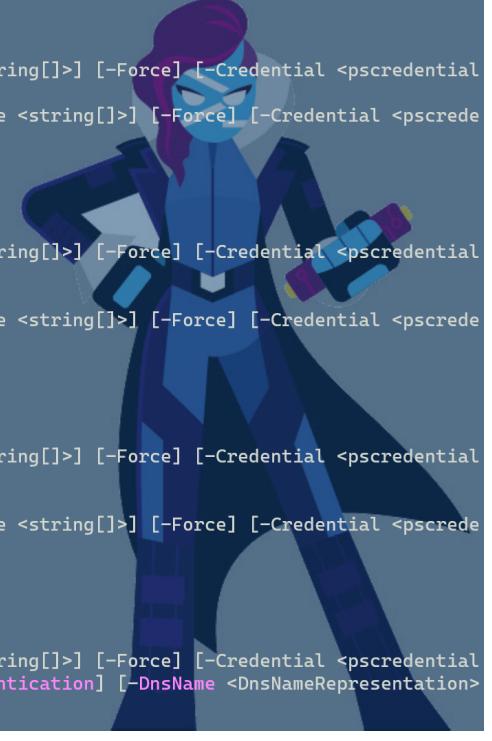
The primary command can be used to test a PowerShell expression or scriptblock for a specified number of times and calculate the average runtime, in milliseconds, over all the tests.

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.

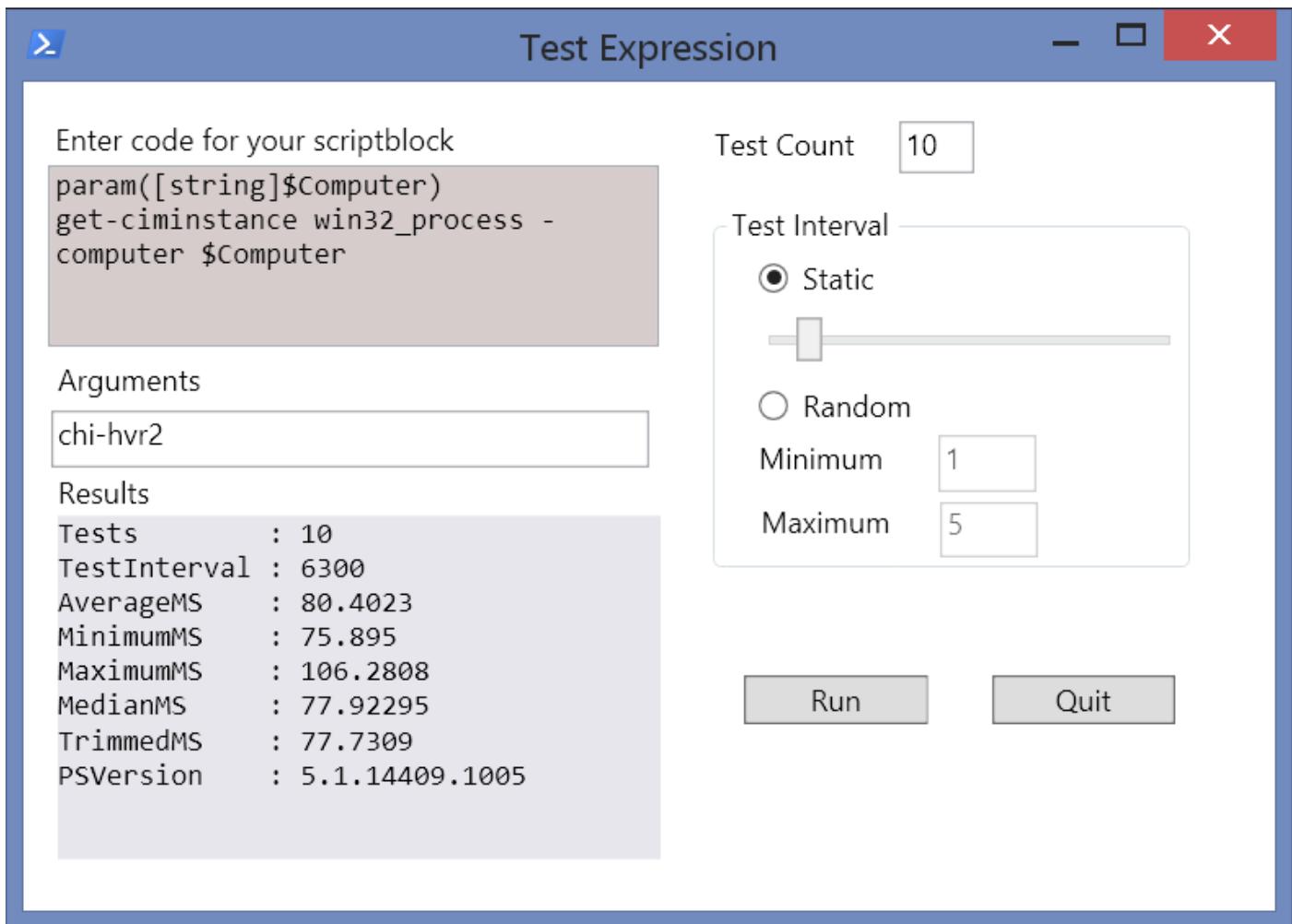
```
PS C:\>Test-Expression {
    param([string[]]$Names)
    Get-Service $names
} -count 5 -IncludeExpression -argumentlist @('bits','wuauserv','winrm') ` 
-RandomMinimum .5 -RandomMaximum 5.5

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 command 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

Select one or more code samples to copy

Filter

Add criteria ▾

| 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 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. The custom object includes default format views for a list and table.

```
PS C:\> Get-ParameterInfo Test-WSMan | Sort ParameterSet | Format-Table

ParameterSet: __AllParameterSets

Name          Aliases Mandatory Position Type
----          ----- ---------
CertificateThumbprint      False     Named   System.String
Credential           cred,c  False     Named   System.Management.Automation.PSCredential
ComputerName          cn       False     0       System.String
Authentication        auth,am  False     Named   Microsoft.WSMan.Management.AuthenticationMechanism

ParameterSet: ComputerName

Name          Aliases Mandatory Position Type
----          ----- ---------
UseSSL          False     Named   System.Management.Automation.SwitchParameter
Port            False     Named   System.Int32
ApplicationName    False     Named   System.String

PS C:\> |
```



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

ParameterSet: __AllParameterSets

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

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).Caption
}
$upParams = @{
    TypeName = $tname
    MemberType = "ScriptProperty"
    MemberName = "Runtime"
    value = {((Get-Date) - [datetime]"1/1/2019")}
    force = $True
}
Update-TypeData @upParams
```

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.

```
$tname = "myThing"
$params = @{
    Properties = "Name", "Date", "Computername", "OS"
    FormatType = "Table"
    Path = "C:\scripts\$tname.format.ps1xml"
}
$obj | New-PSFormatXML @params

$params.Properties= "Name", "OS", "Runtime"
$params.Add("Viewname", "runtime")
$params.Add(Append,$True)
$obj | New-PSFormatXML @params

$params.formatType = "list"
$params.remove("Properties")
$obj | New-PSFormatXML @params

Update-FormatData -appendpath $params.path
```

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
```

Starting with v2.31.0, you can also use a hashtable to define custom properties from scriptblocks.

```
$p = @{
    FormatType = "List"
    ViewName = "run"
    Path = "c:\scripts\run.ps1xml"
    Properties = "ID", "Name", "Path", "StartTime",
    @{Name="Runtime"; Expression={(Get-Date) - $_.starttime}}
}
Get-Process -id $pid | New-PSFormatXML @p
```

If you run this command from Visual Studio 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.

ANSI Tools

This module includes several custom format files for common objects like services. You can run `Get-Service` and pipe it to the custom table view.

```
Get-Service | Format-Table -view ansi
```

This will display the service status color-coded.

| | | |
|---------|----------------------|--|
| Stopped | WFDSConMgrSvc | 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.

Here's another custom view for Aliases.

```
PS C:\> Get-Alias | Sort-Object Source | Format-Table -view Source
```

Source:

| Name | Definition |
|-------|--------------------------------|
| nmo | New-Module |
| ni | New-Item |
| npssc | New-PSSessionConfigurationFile |
| nv | New-Variable |
| nsn | New-PSSession |
| ... | |

Source: Microsoft.PowerShell.Management 3.1.0.0

| Name | Definition |
|------|--------------|
| gtz | Get-TimeZone |
| stz | Set-TimeZone |
| ... | |

Source: Microsoft.PowerShell.Utility 3.1.0.0

| Name | Definition |
|------|--------------------|
| fhx | Format-Hex |
| CFS | ConvertFrom-String |

Source: PSScriptTools 2.31.0

| Name | Definition |
|------|------------------------|
| clr | Convert-EventLogRecord |
| gsi | Get-FolderSizeInfo |
| wver | Get-WindowsVersion |
| gpi | Get-ParameterInfo |
| che | Copy-HelpExample |
| ... | |

Use [Get-FormatView](#) to discover available format views. Or if you'd like to create your own custom views look at [New-PSFormatXML](#)

PSAnsiMap

I have done something similar for output from `Get-ChildItem`. The module includes a JSON file that is exported as a global variable called `PSAnsiFileMap`.

```
PS C:\> $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 colorized file results.

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

Directory: C:\work\Alpha

| Mode | LastWriteTime | Length | Name |
|-------|---------------------|--------|---------------------|
| da--- | 3/5/2020 4:46 PM | | 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 | | delta |
| da--- | 11/6/2017 4:21 PM | | gamma |
| d---- | 2/28/2020 11:16 AM | | 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 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.

If you prefer not to edit JSON files, you can use the PSAnsifileMap commands from the module.

Get-PSAnsifileMap

This command will display the value of the \$PSAnsifileMap variable, but will also show the ANSI sequence using the sequence itself.

| Description | Pattern | ANSI |
|----------------|---|----------------------|
| PowerShell | \.(ps(d m)?1) (ps1xml))\$ | `e[38;2;252;127;12m |
| Text | \.((txt) (log) (htm(l)?))\$ | `e[38;2;58;120;255m |
| Documents | \.((pdf) (doc(x)?) (md) (xls(x)?) (ppt(x)?) (xps))\$ | `e[38;5;121m |
| Temporary | \.((tmp) (bak) (sav) (temp))\$ | `e[38;5;200m |
| DataFile | \.((json) (xml) (csv) (db) (mof) (mdb) (dat))\$ | `e[38;2;249;241;165m |
| Executable | \.((exe) (bat) (cmd) (sh) (py))\$ | `e[38;2;197;15;31m |
| System | \.((sys) (dll) (bin) (conf*) (ini))\$ | `e[38;5;204m |
| Graphics | \.((jpg) (png) (gif) (bmp) (jpeg) (ico) (svg) (tif) (raw))\$ | `e[38;2;255;0;255m |
| Media | \.((mp[34]) (m4v) (wav) (au) (flac) (mov)))\$ | `e[38;2;255;199;6m |
| Archive | \.((zip) (rar) (tar) (gz(ip)?dir) (7z) (wim) (bzip2) (msi(x)?) (rpm))\$ | `e[38;5;75m |
| TopContainer | | `e[38;2;0;255;255m |
| ChildContainer | | `e[38;2;255;255;0m |

PS C:\> -

Set-PSAnsifileMap

Use this command to modify an existing entry. You need to specify a regular expression pattern to match on the filename and/or an ANSI escape sequence. If the entry description doesn't exist, you will need to specify the regex pattern and the ANSI sequence to add the entry to \$PSAnsifileMap.

```
Set-PSAnsifileMap Archive -Ansi ``e[38;5;75m"
```

Remove-PSAnsifileEntry

If you need to, you can remove an entry from \$PSAnsifileMap.

```
Remove-PSAnsifileEntry DevFiles
```

Export-PSAnsifileMap

Any changes you make to \$PSAnsifileMap will only last until you import the module again. To make the change permanent, use [Export-PSAnsifileMap](#). This will create the `psansifilemap.json` file in your `$HOME` directory. When you import the PSScriptTools module, if this file is found, it will be imported. Otherwise, the default module file will be used.

Other Module Features

From time to time I will include additional items that you might find useful in your PowerShell work.

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"
■ 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`.

dir \$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 |
| Memory Compression | 3044 | 0 | 1249 | 09.69 |
| firefox | 18936 | 1630 | 798 | 06.18 |
| LenovoVantageService | 5724 | 1273 | 784 | 06.08 |
| dwm | 1532 | 2835 | 382 | 02.96 |
| firefox | 18368 | 3187 | 349 | 02.71 |
| firefox | 21912 | 1573 | 338 | 02.62 |
| pwsh | 25220 | 1183 | 311 | 02.41 |
| thunderbird | 23268 | 2032 | 247 | 01.91 |
| powershell_ise | 4896 | 946 | 244 | 01.89 |
| firefox | 28208 | 901 | 224 | 01.74 |
| Code | 34948 | 598 | 213 | 01.65 |
| powershell | 24608 | 917 | 209 | 01.62 |
| pwsh | 21864 | 1219 | 203 | 01.57 |

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 PowerShell tools for:

- [Creating and managing custom type extensions](#)
- [Managing scheduled jobs](#)
- [Running remote commands outside of PowerShell Remoting.](#)
- [Getting memory status and information](#)
- [Keeping up to date with PowerShell 7.x releases](#)
- [Inventorying your PowerShell script library](#)
- [A simple command-line task and to-do manager](#)

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 these commands on any edition of PowerShell Core or any beta releases of PowerShell 7.

Last Updated 2020-12-28 18:38:04Z

Module Commands

This section contains the same help content you would get from a PowerShell prompt using `Get-Help`. Note that most code examples have been formatted to fit the 80 character page width and sometimes with artificial formatting. Don't assume you can run examples *exactly* as they are shown. Some of the help examples might also use special or custom characters that might not render properly in the PDF.

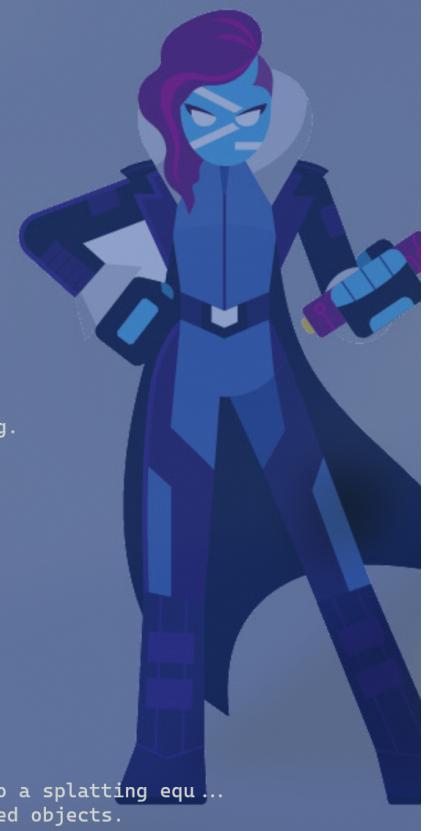
Remember, you can also view the online help for each command:

[Help](#) Convert to WPF Grid [-online](#)

If you can't remember what commands are in this module, you can always ask PowerShell.

```
Get-Command -module PSScriptTools
```

Or use the `Get-PSScriptTools` command.



You can also filter by verb.

```
PS C:\>
PS C:\> Get-PSScriptTools -Verb Test
[---] [---] [---]
[---] [---] [---] [---] [---] [---] [---] [---]
[---] [---] [---] [---] [---] [---] [---] [---] [---]
[---] [---] [---] [---] [---] [---] [---] [---] [---]
[---] [---] [---] [---] [---] [---] [---] [---] [---]
[---] [---] [---] [---] [---] [---] [---] [---] [---]
[---] [---] [---] [---] [---] [---] [---] [---] [---]
[---] [---] [---] [---] [---] [---] [---] [---] [---]
```

Verb: Test

| Name | Alias | Synopsis |
|---------------------|-------|--|
| Test-EmptyFolder | | Test if a folder is empty of files. |
| Test-Expression | tex | Test a PowerShell expression over a period of time. |
| Test-ExpressionForm | texf | Display a graphical test form for Test-Expression. |
| Test-IsPSWindows | | Test if running PowerShell on a Windows platform. |
| Test-WithCulture | | Test your PowerShell code under a different culture. |



Add-Border

Synopsis

Create a text border around a string.

Syntax

single (Default)

```
Add-Border [-Text] <String> [-Character <String>] [-InsertBlanks]
[-Tab <Int32>] [-ANSIBorder <String>] [-ANSIText <String>] [<CommonParameters>]
```

block

```
Add-Border [-TextBlock] <String[]> [-Character <String>] [-InsertBlanks]
[-Tab <Int32>] [-ANSIBorder <String>] [-ANSIText <String>] [<CommonParameters>]
```

Description

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.

Examples

EXAMPLE 1

```
PS C:\> Add-Border "PowerShell Wins!"  
*****  
* PowerShell Wins! *  
*****
```

EXAMPLE 2

```
PS C:\> Add-Border "PowerShell Wins!" -tab 1  
*****  
* PowerShell Wins! *  
*****
```

Note that this example may not format properly in all consoles.

EXAMPLE 3

```
PS C:\> Add-Border "PowerShell Wins!" -character "-" -insertBlanks

-----
-
- PowerShell Wins!
-
-----
```

EXAMPLE 4

```
PS C:\> Add-Border -textblock (Get-Service win* | Out-String).trim()

*****
* Status      Name          DisplayName      *
* -----      ----          -----
* Stopped     WinDefend    Windows Defender Antivirus Service   *
* Running     WinHttpAutoProx... WinHTTP Web Proxy Auto-Discovery Se... *
* Running     Winmgmt       Windows Management Instrumentation   *
* Stopped     WinRM        Windows Remote Management (WS-Manag... *
*****
```

Create a border around the output of a Get-Service command.

EXAMPLE 5

```
PS C:\> Add-Border -Text $t -ANSIBorder "$([char]0x1b)[38;5;47m"
-ANSIText "$([char]0x1b)[93m" -InsertBlanks

*****
*           *
* I am the walrus *
*           *
*****
```

This will write a color version of the text and border. You would this type of ANSI syntax for Windows PowerShell. In PowerShell 7, you can use the same syntax or the much easier "`e[38;5;47m".

EXAMPLE 6

```
PS C:\> Add-Border -textblock (Get-PSWho -AsString).trim() -ANSIBorder "e[38;5;214m" -Character ([char]0x25CA) -ANSIText "`e[38;5;225m"

oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
❖ 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) ❖
oooooooooooooooooooooooooooooooooooooooooooooooooooo
```

This example requires PowerShell 7 because of the way the escape sequence is defined. The border character is a diamond. Depending on how you are viewing this help content, it may not display properly.

Parameters

-Text

A single line of text that will be wrapped in a border.

Type: **String**
Parameter Sets: single
Aliases:

Required: True
Position: 1
Default value: None
Accept pipeline input: True (**ByValue**)
Accept wildcard characters: False

-TextBlock

A multi-line block of text. You might want to trim blank lines from the beginning, end or both.

Type: `String[]`
Parameter Sets: `block`
Aliases: `tb`

Required: True
Position: 1
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Character

The character to use for the border. It must be a single character.

```
Type: String
Parameter Sets: (All)
Aliases: border

Required: False
Position: Named
Default value: *
Accept pipeline input: False
Accept wildcard characters: False
```

-InsertBlanks

Insert blank lines before and after the text. The default behavior is to create a border box close to the text. See examples.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-Tab

Insert X number of tabs.

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: 0
Accept pipeline input: False
Accept wildcard characters: False
```

-ANSIBorder

Enter an ANSI escape sequence to color the border characters.

Type: **String**

Parameter Sets: ([All](#))

Aliases:

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

-ANSIText

Enter an ANSI escape sequence to color the text.

Type: **String**

Parameter Sets: ([All](#))

Aliases:

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[New-ANSIBar](#)

Compare-Module

Synopsis

Compare PowerShell module versions.

Syntax

```
Compare-Module [[-Name] <String>] [-Gallery <String>] [<CommonParameters>]
```

Description

Use this command to compare module versions between what is installed against an online repository like the PSGallery. Results will be automatically sorted by module name.

Examples

EXAMPLE 1

```
PS C:\> Compare-Module | Where-object {$_.UpdateNeeded}

Name          : DNSSuffix
OnlineVersion : 0.4.1
InstalledVersion : 0.2.0
PublishedDate : 10/22/2018 8:21:46 PM
UpdateNeeded   : True

Name          : InvokeBuild
OnlineVersion : 5.4.2
InstalledVersion : 3.2.2
PublishedDate : 12/7/2018 1:30:46 AM
UpdateNeeded   : True
...
```

List all modules that could be updated.

EXAMPLE 2

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

Compare modules and send results to Out-GridView. Use Out-GridView as an object picker to decide what modules to update.

EXAMPLE 3

```
PS C:\> Compare-Module -name xWin* | Format-Table
```

| Name | OnlineVersion | InstalledVersion | PublishedDate | UpdateNeeded |
|----------------|---------------|------------------|-----------------------|--------------|
| xWindowsUpdate | 2.7.0.0 | 2.7.0.0,2.5.0.0 | 7/12/2017 10:43:54 PM | False |
| xWinEventLog | 1.2.0.0 | 1.2.0.0 | 6/13/2018 8:06:45 PM | False |

Compare all modules that start with xWin* and display results in a table format.

EXAMPLE 4

```
PS C:\> get-dscresource xAD* | Select-Object moduleName -Unique | Compare-Module
```

| | |
|-------------------------|--------------------------|
| Name | : xActiveDirectory |
| OnlineVersion | : 2.22.0.0 |
| InstalledVersion | : 2.16.0.0,2.14.0.0 |
| PublishedDate | : 10/25/2018 5:25:24 PM |
| UpdateNeeded | : True |
| Name | : xAdcsDeployment |
| OnlineVersion | : 1.4.0.0 |
| InstalledVersion | : 1.1.0.0,1.0.0.0 |
| PublishedDate | : 12/20/2017 10:10:43 PM |
| UpdateNeeded | : True |

Get all DSC Resources that start with xAD and select the corresponding module name. Since the module name will be listed for every resource, get a unique list and pipe that to Compare-Module.

Parameters

-Name

The name of a module to check. Wildcards are permitted.

```
Type: String
Parameter Sets: (All)
Aliases: modulename

Required: False
Position: 1
Default value: None
Accept pipeline input: True (ByPropertyName)
Accept wildcard characters: True
```

-Gallery

Specify the remote repository or gallery to check.

Type: **String**

Parameter Sets: (**All**)

Aliases:

Required: False

Position: Named

Default value: PSGallery

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

PSCustomObject

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Find-Module](#)

[Get-Module](#)

[Update-Module](#)

Compare-Script

Synopsis

Compare PowerShell script versions.

Syntax

```
Compare-Script [[-Name] <String>] [-Gallery <String>] [<CommonParameters>]
```

Description

Use this command to compare script versions between what is installed against an online repository like the PSGallery. Results will be automatically sorted by the script name.

Examples

EXAMPLE 1

```
PS C:\> Compare-Script | Where-object {$_.UpdateNeeded}

Name          : DNSSuffix
OnlineVersion : 0.4.1
InstalledVersion : 0.2.0
PublishedDate : 10/22/2020 8:21:46 PM
UpdateNeeded   : True

Name          : InvokeBuild
OnlineVersion : 5.4.2
InstalledVersion : 3.2.2
PublishedDate : 12/7/2020 1:30:46 AM
UpdateNeeded   : True
...
```

List all scripts that could be updated.

EXAMPLE 2

```
PS C:\> Compare-Script | Where UpdateNeeded |
Out-GridView -Title "Select scripts to update" -OutputMode multiple |
Foreach-Object { Update-Script $_.name }
```

Compare scripts and send results to Out-GridView. Use Out-GridView as an object picker to decide what scripts to update.

Parameters

-Name

The name of a script to check. Wildcards are permitted.

```
Type: String
Parameter Sets: (All)
Aliases: scriptname

Required: False
Position: 1
Default value: None
Accept pipeline input: True (ByPropertyName)
Accept wildcard characters: True
```

-Gallery

Specify the remote repository or gallery to check.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: PSGallery
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

PSCustomObject

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Find-Script](#)

[Get-InstalledScript](#)

[Update-Script](#)

Convert-CommandToHashtable

Synopsis

Convert a PowerShell expression into a splatting equivalent.

Syntax

```
Convert-CommandToHashtable [-Text] <String> [<CommonParameters>]
```

Description

This command is intended to convert a long PowerShell expression with named parameters into a splatting alternative. The central concept is that you are editing a script file with a lengthy PowerShell expression with multiple parameters and you would like to turn it into splatting code.

Examples

Example 1

```
PS C:\> $text ="Get-Winevent -listlog p* -computername SRV1 -erroraction stop"
PS C:\> Convert-CommandToHashtable -Text $text | Set-Clipboard
```

The \$text variable might be a line of code from your script. The second line converts into a splatting sequence and copies it to the Windows clipboard so you can paste it back into your script. You could create a VS Code task sequence using this function.

Parameters

-Text

A PowerShell command using a single cmdlet or function, preferably with named parameters.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

Hashtable

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Convert-HashtableToCode](#)

Convert-EventLogRecord

Synopsis

Convert EventLogRecords to structured objects.

Syntax

```
Convert-EventLogRecord [-LogRecord] <EventLogRecord[]> [<CommonParameters>]
```

Description

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 the results of a Get-WinEvent command into a PowerShell custom object with additional information. For best results, you should pipe the same event IDs to this command.

Note that not every event record exposes data that is compatible with this command. For those types of event log records, you will see a RawProperties property with most likely an array of strings. Use the Message property for more information.

Examples

EXAMPLE 1

```
PS C:\> Get-WinEvent -FilterHashtable @{Logname = 'security';ID=5059} |  
Convert-EventLogRecord | Select-Object -Property TimeCreated,Subject*,  
Computername  
  
TimeCreated      : 1/20/2020 10:48:45 AM  
SubjectUserId    : S-1-5-83-1-2951761591-1086169693-630393256-923523501  
SubjectUserName   : AFF04EB7-A25D-40BD-A809-9325ADD90B37  
SubjectDomainName : NT VIRTUAL MACHINE  
SubjectLogonId    : 0x7cbf5  
Computername     : Bovine320  
  
TimeCreated      : 1/20/2020 10:48:45 AM  
SubjectUserId    : S-1-5-83-1-2951761591-1086169693-630393256-923523501  
SubjectUserName   : AFF04EB7-A25D-40BD-A809-9325ADD90B37  
SubjectDomainName : NT VIRTUAL MACHINE  
SubjectLogonId    : 0x7cbf5  
Computername     : Bovine320
```

EXAMPLE 2

```
PS C:\> Get-WinEvent -FilterHashtable @{Logname = 'security';ID=4624} ` 
-MaxEvents 100 -computername win10 | Convert-EventLogRecord | 
Where-Object {$_.LogonType -eq 3} | 
Select-Object -first 10 -property TargetUsername,IPAddress, 
TimeCreated,Computername | Format-Table
```

| TargetUserName | IpAddress | TimeCreated | Computername |
|----------------|---------------------------|-----------------------|-------------------|
| ArtD | fe80::ddae:8ade:c3ff:e584 | 1/20/2020 12:05:12 PM | WIN10.Company.Pri |
| WIN10\$ | - | 1/20/2020 11:56:52 AM | WIN10.Company.Pri |
| WIN10\$ | - | 1/20/2020 11:56:52 AM | WIN10.Company.Pri |
| WIN10\$ | - | 1/20/2020 11:56:52 AM | WIN10.Company.Pri |
| WIN10\$ | - | 1/20/2020 11:56:51 AM | WIN10.Company.Pri |
| ArtD | 192.168.3.10 | 1/20/2020 11:45:31 AM | WIN10.Company.Pri |
| WIN10\$ | ::1 | 1/20/2020 11:39:52 AM | WIN10.Company.Pri |
| ArtD | 192.168.3.10 | 1/20/2020 11:35:49 AM | WIN10.Company.Pri |
| ArtD | 192.168.3.10 | 1/20/2020 11:34:36 AM | WIN10.Company.Pri |
| ArtD | 192.168.3.10 | 1/20/2020 11:32:06 AM | WIN10.Company.Pri |

This example filters on a property added by this command to only show interactive logons.

EXAMPLE 3

```
PS C:\> Get-WinEvent -FilterHashtable @{Logname = 'system'; 
ID =7040} -MaxEvent 10 | Convert-EventLogRecord | 
Select-Object -Property TimeCreated,@{Name="Service";Expression={$_.param4}}, 
@{Name="OriginalState";Expression = "{$_.param2}"}, 
@{Name="NewState";Expression={$_.param3}},Computername | Format-Table
```

| TimeCreated | Service | OriginalState | NewState | Computername |
|----------------------|------------------|---------------|--------------|--------------|
| 1/20/2020 9:26:08 AM | BITS | demand start | auto start | Bovine320 |
| 1/20/2020 5:47:17 AM | BITS | auto start | demand start | Bovine320 |
| 1/20/2020 5:45:11 AM | BITS | demand start | auto start | Bovine320 |
| 1/20/2020 1:44:31 AM | BITS | auto start | demand start | Bovine320 |
| 1/20/2020 1:42:30 AM | BITS | demand start | auto start | Bovine320 |
| 1/19/2020 8:53:37 PM | BITS | auto start | demand start | Bovine320 |
| 1/17/2020 8:27:10 PM | TrustedInstaller | demand start | auto start | Bovine320 |
| 1/17/2020 8:27:10 PM | TrustedInstaller | auto start | demand start | Bovine320 |
| 1/17/2020 8:26:29 PM | TrustedInstaller | demand start | auto start | Bovine320 |
| 1/17/2020 8:26:20 PM | TrustedInstaller | auto start | demand start | Bovine320 |

Once you know the type of data, you can customize the output or build a script around it.

EXAMPLE 4

```
PS C:\> Get-WinEvent -FilterHashtable @{Logname = "Application";  
ID=17137} -MaxEvents 1 | Convert-EventLogRecord  
  
LogName      : Application  
RecordType   : Information  
TimeCreated  : 1/20/2020 2:31:52 PM  
ID          : 17137  
RawProperties : {TickleEventDB}  
Message      : Starting up database 'TickleEventDB'.  
Keywords     : {Classic}  
Source       : MSSQL$SQLEXPRESS  
Computername : Bovine320
```

This record doesn't have structured extra data. The replacement strings are stored as text so the command displays the data using the RawProperties property.

EXAMPLE 5

```
PS C:\> $all = New-PSSession -ComputerName 'win10','srv1','srv2','dom1'
PS C:\> $local = Get-Item Function:\Convert-EventLogRecord
PS C:\> Invoke-Command -ScriptBlock {
    New-item -Path Function: -Name $using:local.name -Value $using:local.ScriptBlock
} -Session $all
PS C:\> Invoke-Command {
    Get-WinEvent -FilterHashtable @{Logname='security';id=4624} -MaxEvents 10 |
    Convert-EventLogRecord |
    Select-Object -Property Computername,Time*,TargetUser*,
    TargetDomainName,Subject*} -session $all -HideComputerName |
    Select-Object -Property * -ExcludeProperty runspaceID

Computername      : WIN10.Company.Pri
TimeCreated       : 1/20/2020 5:21:17 PM
TargetUserSid     : S-1-5-18
TargetUserName   : SYSTEM
TargetDomainName : NT AUTHORITY
SubjectUserSid    : S-1-5-18
SubjectUserName  : WIN10$
SubjectDomainName: COMPANY
SubjectLogonId    : 0x3e7

Computername      : WIN10.Company.Pri
TimeCreated       : 1/20/2020 5:18:51 PM
TargetUserSid     : S-1-5-18
TargetUserName   : SYSTEM
TargetDomainName : NT AUTHORITY
SubjectUserSid    : S-1-5-18
SubjectUserName  : WIN10$
SubjectDomainName: COMPANY
SubjectLogonId    : 0x3e7

Computername      : WIN10.Company.Pri
TimeCreated       : 1/20/2020 5:16:07 PM
TargetUserSid     : S-1-5-21-278538743-3177530655-100218012-1105
TargetUserName   : ArtD
TargetDomainName : COMPANY.PRI
SubjectUserSid    : S-1-0-0
SubjectUserName  : -
SubjectDomainName: -
SubjectLogonId    : 0x0
...
```

The first command creates PSSessions to several remote computers. The local copy of this command is created in the remote PSSessions. Then event log data is retrieved in the remote sessions and converted using the Convert-EventlogRecord function in each session.

Parameters

-LogRecord

An event log record from the Get-WinEvent command.

Type: `EventLogRecord[]`

Parameter Sets: `(All)`

Aliases:

Required: True

Position: 1

Default value: None

Accept pipeline input: True (`ByValue`)

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: `-Debug`, `-ErrorAction`, `-ErrorVariable`, `-InformationAction`, `-InformationVariable`, `-OutVariable`, `-OutBuffer`, `-PipelineVariable`, `-Verbose`, `-WarningAction`, and `-WarningVariable`. For more information, see [about_CommonParameters](#).

Inputs

`System.Diagnostics.Eventing.Reader.EventLogRecord`

Outputs

`PSCustomObject`

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-WinEvent](#)

Convert-HashtableString

Synopsis

Convert a hashtable string into a hashtable object.

Syntax

```
Convert-HashtableString [-Text] <String> [<CommonParameters>]
```

Description

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.

Examples

Example 1

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

Name          Value
----          -----
CreatedBy     BOVINE320\Jeff
CreatedAt     10/02/2020 21:28:47 UTC
Computername  Think51
Error
Completed    True
Date          10/02/2020 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.

Parameters

-Text

Enter your hashtable string.

Type: **String**

Parameter Sets: (**All**)

Aliases:

Required: True

Position: 0

Default value: None

Accept pipeline input: True (**ByValue**)

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

hashtable

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Import-PowerShellDatafile](#)

[Convert-HashtableToCode](#)

Convert-HashtableToCode

Synopsis

Convert a hashtable to a string representation.

Syntax

psd1 (Default)

```
Convert-HashtableToCode [-Hashtable] <Hashtable> [-Indent <Int32>]  
[<CommonParameters>]
```

inline

```
Convert-HashtableToCode [-Hashtable] <Hashtable> [-Inline] [<CommonParameters>]
```

Description

Use this command to convert a hashtable into its text or string equivalent. It is assumed that any array values contain items of the same type. This command has not been tested with large or complex hashtables, so you might need to manually edit the output to meet your tastes or requirements.

Examples

Example 1

```
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.

Example 2

```
PS C:\> Convert-HashtableToCode $h -inline  
  
@{Name = 'SRV1';Asset = 123454;Location = 'Omaha'}
```

Create an inline string version of the hashtable.

Parameters

-Hashtable

A hashtable to convert. It can be standard or ordered hashtable.

```
Type: Hashtable
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Indent

Specify the number of tabs to indent. You shouldn't need to specify this parameter. It exists for situations where there are nested hashtables.

```
Type: Int32
Parameter Sets: psd1
Aliases: tab

Required: False
Position: Named
Default value: 1
Accept pipeline input: False
Accept wildcard characters: False
```

-Inline

Write the hashtable as an inline expression.

```
Type: SwitchParameter
Parameter Sets: inline
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Collections.Hashtable

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Convert-HashtableString](#)

ConvertFrom-LexicalTimespan

Synopsis

Convert a lexical timespan into a PowerShell timespan.

Syntax

```
ConvertFrom-LexicalTimespan [-String] <String> [-AsString] [<CommonParameters>]
```

Description

When working with some XML data, such as that from scheduled tasks, timespans or durations are stored in a lexical format like P0DT0H0M47S. You can use this command to convert that value into a timespan object.

Examples

Example 1

```
PS C:\> ConvertFrom-LexicalTimespan P0DT0H0M47S
```

```
Days          : 0
Hours         : 0
Minutes       : 0
Seconds       : 47
Milliseconds  : 0
Ticks         : 470000000
TotalDays     : 0.000543981481481481
TotalHours    : 0.01305555555555556
TotalMinutes  : 0.7833333333333333
TotalSeconds  : 47
TotalMilliseconds : 47000
```

Example 2

```
PS C:\> Get-ScheduledTask -TaskName DailyWatcher |
Select-Object Taskname,
@{Name="ExecutionLimit";Expression = {
ConvertFrom-LexicalTimespan $_.settings.ExecutionTimeLimit } }

Taskname      ExecutionLimit
-----        -----
DailyWatcher  3.00:00:00
```

Parameters

-AsString

Format the timespan as a string

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-String

Enter a lexical time string like P23DT3H43M. This is case-sensitive.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String

Outputs

String

Timespan

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[ConvertTo-LexicalTimespan](#)

ConvertFrom-Text

Synopsis

Convert structured text to objects.

Syntax

File (Default)

```
ConvertFrom-Text [-Pattern] <Regex> [-Path] <String> [-TypeName <String>]
[-NoProgress] [<CommonParameters>]
```

Inputobject

```
ConvertFrom-Text [-Pattern] <Regex> [-InputObject] <String>
[-TypeName <String>] [-NoProgress] [<CommonParameters>]
```

Description

This command will take structured text such as from a log file and convert it to objects that you can use in the PowerShell pipeline. You can specify the path to a text file, or pipe content directly into this command. The piped content could even be output from command-line tools. You have to specify a regular expression pattern that uses named captures. The names will become property names in the custom objects.

The command will write a generic custom object to the pipeline. However, you can specify a custom type name. You might want to do this if you have your own format ps1xml file and want to handle formatting through that file.

Examples

EXAMPLE 1

```
PS C:\> $b = "(?<Date>\d{2}-\d{2}-\d{4})\s\d{2}:\d{2}.*(?<Error>\d+),\s+(?<Step>.*):\s+(?<Action>\w+),\s+(?<Path>(\w+\\"*\w+\.\w+))"
PS C:\> ConvertFrom-Text -pattern $b -path C:\windows\DtcInstall.log

Date    : 10-18-2020 10:49
Error   : 0
Step    : CMsdtcUpgradePlugin::PostApply
Action   : Enter
Path    : com\complus\dtc\dtc\msdtcstp\msdtcplugin.cpp

Date    : 10-18-2020 10:49
Error   : 0
Step    : CMsdtcUpgradePlugin::PostApply
Action   : Exit
Path    : com\complus\dtc\dtc\msdtcstp\msdtcplugin.cpp
...
```

The first command creates a variable to hold the regular expression pattern that defines named captures for content in the DtcInstall.log. The second line runs the command using the pattern and the log file.

EXAMPLE 2

```
PS C:\> $wu = "(?<Date>\d{4}-\d{2}-\d{2})\s+(?<Time>(\d{2}:\d{2}:\d{2}))\s+(?<PID>\d+)\s+(?<TID>\w+)\s+(?<Component>\w+)\s+(?<Message>.*)"  
PS C:\> $out = ConvertFrom-Text -pattern $wu -path C:\Windows\WindowsUpdate.log -noprogress  
PS C:\> $out | Group-Object Component | Sort-Object Count  
  
Count Name Group  
---- -- --  
 20 DtaStor {@[Date=2020-01-27; Time=07:19:19; PID=1...  
 72 Setup {@[Date=2020-01-27; Time=07:19:05; PID=1...  
148 SLS {@[Date=2020-01-27; Time=07:19:05; PID=1...  
150 PT {@[Date=2020-01-27; Time=07:19:08; PID=1...  
209 WuTask {@[Date=2020-01-26; Time=20:05:28; PID=1...  
256 EP {@[Date=2020-01-26; Time=21:21:23; PID=1...  
263 Handler {@[Date=2020-01-27; Time=07:19:42; PID=3...  
837 Report {@[Date=2020-01-26; Time=21:21:23; PID=1...  
900 IdleTmr {@[Date=2020-01-26; Time=21:21:23; PID=1...  
903 Service {@[Date=2020-01-26; Time=20:05:29; PID=1...  
924 Misc {@[Date=2020-01-26; Time=21:21:23; PID=1...  
1062 DnldMgr {@[Date=2020-01-26; Time=21:21:23; PID=1...  
2544 AU {@[Date=2020-01-26; Time=19:55:27; PID=1...  
2839 Agent {@[Date=2020-01-26; Time=21:21:23; PID=1...  
  
PS C:\> $out | Where-Object {[datetime]$_.date -ge [datetime]"2/10/2020" -AND $_.component -eq "AU"} | Format-Table Date,Time,Message -wrap  
  
Date Time Message  
---- -- --  
2020-02-10 05:36:44:183 ##### AU: Initializing Automatic Updates #####  
2020-02-10 05:36:44:184 Additional Service {117CAB2D-82B1-4B5A-A08C-4D62DBEE7782} with Approval  
type {Scheduled} added to AU services list  
2020-02-10 05:36:44:184 AIR Mode is disabled  
2020-02-10 05:36:44:185 # Approval type: Scheduled (User preference)  
2020-02-10 05:36:44:185 # Auto-install minor updates: Yes (User preference)  
2020-02-10 05:36:44:185 # ServiceTypeDefault: Service 117CAB2D-82B1-4B5A-A08C-4D62DBEE7782  
Approval type: (Scheduled)  
2020-02-10 05:36:44:185 # WILL interact with non-admins (Non-admins are elevated (User preference))  
2020-02-10 05:36:44:204 WARNING: Failed to get Wu Exemption info from NLM, assuming not exempt,  
error = 0x80070490  
2020-02-10 05:36:44:213 AU finished delayed initialization  
2020-02-10 05:38:01:000 #####  
...
```

In this example, the WindowsUpdate log is converted from text to objects using the regular expression pattern. Given the size of the log file this process can take some time to complete so the progress bar is turned off to improve performance.

EXAMPLE 3

```
PS C:\> Get-Content c:\windows\windowsupdate.log -totalcount 50 |  
ConvertFrom-Text $wu
```

This example gets the first 50 lines from the Windows update log and converts that to objects using the pattern from the previous example.

EXAMPLE 4

```
PS C:\> $c = "(?<Protocol>\w{3})\s+(?<LocalIP>(\d{1,3}\.){3}\d{1,3}):\s+ (?<LocalPort>\d+)\s+ (?<ForeignIP>.*):\s+ (?<ForeignPort>\d+)\s+ (?<State>\w+)?"
PS C:\> netstat | select -skip 4 | ConvertFrom-Text $c |
Format-Table -AutoSize
```

| Protocol | LocalIP | LocalPort | ForeignIP | ForeignPort | State |
|----------|--------------|-----------|----------------|-------------|-------------|
| TCP | 127.0.0.1 | 19872 | Novo8 | 50835 | ESTABLISHED |
| TCP | 127.0.0.1 | 50440 | Novo8 | 50441 | ESTABLISHED |
| TCP | 127.0.0.1 | 50441 | Novo8 | 50440 | ESTABLISHED |
| TCP | 127.0.0.1 | 50445 | Novo8 | 50446 | ESTABLISHED |
| TCP | 127.0.0.1 | 50446 | Novo8 | 50445 | ESTABLISHED |
| TCP | 127.0.0.1 | 50835 | Novo8 | 19872 | ESTABLISHED |
| TCP | 192.168.6.98 | 50753 | 74.125.129.125 | 5222 | ESTABLISHED |

The first command creates a variable to be used with output from the Netstat command which is used in the second command.

EXAMPLE 5

```
PS C:\> $arp = "(?<IPAddress>(\d{1,3}\.){3}\d{1,3})\s+(?<MAC>(\w{2}-){5}\w{2})\s+(?<Type>\w+)"
PS C:\> arp -g -N 172.16.10.22 | Select-Object -skip 3 |
ForEach-Object {$_.Trim()} |
ConvertFrom-Text $arp -noprogress -typename arpData
```

| IPAddress | MAC | Type |
|---------------|-------------------|---------|
| 172.16.10.1 | 00-13-d3-66-50-4b | dynamic |
| 172.16.10.100 | 00-0d-a2-01-07-5d | dynamic |
| 172.16.10.101 | 2c-76-8a-3d-11-30 | dynamic |
| 172.16.10.121 | 00-0e-58-ce-8b-b6 | dynamic |
| 172.16.10.122 | 1c-ab-a7-99-9a-e4 | dynamic |
| 172.16.10.124 | 00-1e-2a-d9-cd-b6 | dynamic |
| 172.16.10.126 | 00-0e-58-8c-13-ac | dynamic |
| 172.16.10.128 | 70-11-24-51-84-60 | dynamic |
| ... | | |

The first command creates a regular expression for the ARP command. The second prompt shows the ARP command being used to select the content, trimming each line, and then converting the output to text using the regular expression named pattern. This example also defines a custom type name for the output.

Parameters

-InputObject

Any text that you want to pipe into this command. It can be a certain number of lines from a large text or log file. Or the output of a command line tool. Be sure to filter out blank lines.

Type: String
Parameter Sets: Inputobject
Aliases:

Required: True
Position: 1
Default value: None
Accept pipeline input: True (**ByValue**)
Accept wildcard characters: False

-NoProgress

By default this command will display a progress bar to inform the user on the status. For large data sets this can impact performance. Use this parameter to suppress the progress messages.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-Path

The filename and path to the text or log file.

```
Type: String
Parameter Sets: File
Aliases: file

Required: True
Position: 1
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Pattern

A regular expression pattern that uses named captures. This parameter has an aliases of regex and rx.

```
Type: Regex
Parameter Sets: (All)
Aliases: regex, rx

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-TypeName

Enter an optional typename for the object output. If you don't use one, the command will write a generic custom object to the pipeline.

Type: **String**

Parameter Sets: (**All**)

Aliases:

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String

Outputs

PSCustomObject

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Content](#)

[About_Regular_Expressions](#)

ConvertFrom-UTCTime

Synopsis

Convert a datetime value from universal.

Syntax

```
ConvertFrom-UTCTime [-DateTime] <DateTime> [<CommonParameters>]
```

Description

Use this command to convert a universal datetime object into local time.

This command was introduced in v2.3.0.

Examples

Example 1

```
PS C:\> ConvertFrom-UTCTime "18:00"
Monday, March 4, 2020 1:00:00 PM
```

Covert the time 18:00 for the current day from universal time to local time. This result reflects Eastern Time which on this date is UTC-5.

Parameters

-DateTime

Enter a Universal Datetime value

Type: `DateTime`
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: True ([ByValue](#))
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.DateTime

Outputs

System.DateTime

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[ConvertTo-UTCTime](#)

[Get-Date](#)

ConvertTo-ASCIIArt

Synopsis

Convert text to ASCII art.

Syntax

```
ConvertTo-ASCIIArt [-Text] <String> [[-Font] <String>] [<CommonParameters>]
```

Description

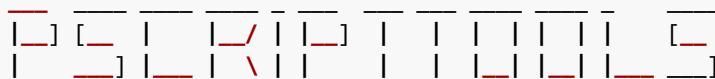
This command can be used to transform a string of text into ASCII art. ConvertTo-ASCIIArt utilizes the web service at <https://artii.herokuapp.com> which allows you to transform text. You might use this to create headers for your scripts or PowerShell profile.

Expect a period of trial and error to find a good font that works with your text. The shorter your text, the better.

Examples

Example 1

```
PS C:\> ConvertTo-ASCIIArt -text PSScriptTools -Font cybermedium
```

A block of ASCII characters representing the word "PSScriptTools" in a stylized font. The letters are composed of various horizontal and vertical lines, with some characters having diagonal strokes. The font is characterized by its bold, blocky appearance.

Convert the string 'PSScriptTools' with the cybermedium font. The font parameter supports auto-completion so you can tab-complete to see all possible options.

Example 2

```
PS C:\> ConvertTo-ASCIIArt -Text $env:COMPUTERNAME -Font ogre |  
Out-File c:\scripts\computername-art.txt
```

Convert the computer name to ASCII art and save the results to a text file. Later you can use Get-Content to display the "artistic" version of the computer name.

Parameters

-Font

Specify a font from https://artii.herokuapp.com/fonts_list. Font names are case-sensitive. You can use tab-completion to cycle through the list.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: 1
Default value: big
Accept pipeline input: False
Accept wildcard characters: False

-Text

Enter a short string of text to convert.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: True ([ByValue](#))
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String

Outputs

System.String

Notes

This command has an alias of cart.

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

ConvertTo-Hashtable

Synopsis

Convert an object into a hashtable.

Syntax

```
ConvertTo-Hashtable [-InputObject] <Object> [-NoEmpty] [-Exclude <String[]>]
[-Alphabetical] [<CommonParameters>]
```

Description

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.

Examples

EXAMPLE 1

```
PS C:\> Get-Process -id $pid |
Select-Object name,id,handles,workingset |
ConvertTo-Hashtable

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

EXAMPLE 2

```
PS C:\> $hash = Get-Service spooler |
ConvertTo-Hashtable -Exclude CanStop,CanPauseAndContinue -NoEmpty
PS C:\> $hash

Name          Value
----          -----
ServiceType   Win32OwnProcess, InteractiveProcess
ServiceName   spooler
ServiceHandle SafeServiceHandle
DependentServices {Fax}
ServicesDependedOn {RPCSS, http}
Name          spooler
Status         Running
MachineName   .
RequiredServices {RPCSS, http}
DisplayName   Print Spooler
```

This created a hashtable from the Spooler service object, skipping empty properties and excluding CanStop and CanPauseAndContinue.

EXAMPLE 3

```
PS C:\> Get-Service bits |  
Select-Object Name,Displayname,Status,  
@{Name="Computername";Expression={$_.Machinename}} |  
ConvertTo-Hashtable -Alphabetical  
  
Name                        Value  
----  
Computername               .  
DisplayName               Background Intelligent Transfer Service  
Name                       bits  
Status                     Running
```

Convert an object to a hashtable and order the properties alphabetically.

Parameters

-InputObject

A PowerShell object to convert to a hashtable.

```
Type: Object  
Parameter Sets: (All)  
Aliases:  
  
Required: True  
Position: 1  
Default value: None  
Accept pipeline input: True (ByValue)  
Accept wildcard characters: False
```

-NoEmpty

Do not include object properties that have no value.

```
Type: SwitchParameter  
Parameter Sets: (All)  
Aliases:  
  
Required: False  
Position: Named  
Default value: False  
Accept pipeline input: False  
Accept wildcard characters: False
```

-Exclude

An array of property names to exclude from the hashtable.

```
Type: String[]
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Alphabetical

Create a hashtable with property names arranged alphabetically.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.Object

Outputs

System.Collections.Specialized.OrderedDictionary

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

This was originally described at: <http://jdhitsolutions.com/blog/2013/01/convert-powershell-object-to-hashtable-revised>

Related Links

[About_Hash_Tables](#)

[Get-Member](#)

ConvertTo-LexicalTimespan

Synopsis

Convert a timespan to lexical time.

Syntax

```
ConvertTo-LexicalTimespan [-Timespan] <TimeSpan> [<CommonParameters>]
```

Description

Convert a timespan into a lexical version that you can insert into an XML document.

Examples

Example 1

```
PS C:\> ConvertTo-LexicalTimespan (New-Timespan -Days 7)  
P7D
```

You can insert this value into an XML document where you need to represent a time-span.

Parameters

-Timespan

Enter a timespan object

Type: `TimeSpan`
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: True ([ByValue](#))
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: `-Debug`, `-ErrorAction`, `-ErrorVariable`, `-InformationAction`, `-InformationVariable`, `-OutVariable`, `-OutBuffer`, `-PipelineVariable`, `-Verbose`, `-WarningAction`, and `-WarningVariable`. For more information, see [about_CommonParameters](#).

Inputs

System.TimeSpan

Outputs

String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[ConvertFrom-LexicalTimespan](#)

ConvertTo-LocalTime

Synopsis

Convert a foreign time to local.

Syntax

```
ConvertTo-LocalTime [-Datetime] <DateTime> [-UTCOffset] <TimeSpan>
[-DaylightSavingTime] [<CommonParameters>]
```

Description

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. You can use Get-Timezone or Get-TZData to help. See examples.

The parameter for DaylightSavingTime is to indicate that the remote location is observing DST. You can use this with the location's standard UTC offset, or you can specify an offset that takes DST into account.

Examples

Example 1

```
PS C:\> ConvertTo-LocalTime "3/15/2019 7:00AM" 8:00:00
Thursday, March 14, 2019 7:00:00 PM
```

Convert a time that is in Singapore to local (Eastern) time.

Example 2

```
PS C:\> Get-TimeZone -ListAvailable | where-object 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 is first determining the UTC offset for Hawaii. Then 10:00AM in Honolulu, is converted to local time which in this example is in the Eastern Time zone.

Parameters

-Datetime

Enter a non-local date time

```
Type: DateTime
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-UTCOffset

Enter the location's' UTC Offset. You can use Get-Timezone to discover it.

```
Type: TimeSpan
Parameter Sets: (All)
Aliases: offset

Required: True
Position: 1
Default value: None
Accept pipeline input: True (ByPropertyName)
Accept wildcard characters: False
```

-DaylightSavingTime

Indicate that the foreign location is using Daylight Saving Time

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: dst

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

DateTime

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-TimeZone](#)

[Get-Date](#)

[Get-MyTimeInfo](#)

[Get-TZList](#)

[ConvertFrom-UTCTime](#)

[ConvertTo-UTCTime](#)

ConvertTo-Markdown

Synopsis

Convert pipeline output to a markdown document.

Syntax

```
ConvertTo-Markdown [[-InputObject <Object>] [-Title <String>]
[-PreContent <String[]>] [-PostContent <String[]>] [-Width <Int32>] [-AsTable]
[<CommonParameters>]
```

Description

This command is designed to accept pipelined output and create a generic markdown document. The pipeline output will be formatted as a text block or you can specify a table. You can optionally define a title, content to appear before the output, and content to appear after the output. Best efforts have been made to produce markdown output that meets basic standards.

The command does not create a text file. You need to pipe results from this command to a cmdlet like Out-File or Set-Content. See examples.

Examples

EXAMPLE 1

```
PS C:\> Get-Service Bits,Winrm |  
ConvertTo-Markdown -title "Service Check" -precontent "## $($env:computername)"  
  
# Service Check  
  
## THINK51  
  
\\\\\\text  
  


Status	Name	DisplayName
Running	Bits	Background Intelligent Transfer Ser...
Running	Winrm	Windows Remote Management (WS-Manag... \\\\\\


```

Create markdown output from a Get-Service command.

EXAMPLE 2

```
PS C:\> Get-Service Bits,Winrm |  
ConvertTo-Markdown -title "Service Check" -precontent "## $($env:computername)"`  
-postcontent "_report $(Get-Date)_" | Out-File c:\work\svc.md
```

Re-run the previous command and save the output to a file.

EXAMPLE 3

```
PS C:\> $computers = "srv1","srv2","srv4"
PS C:\> $Title = "System Report"
PS C:\> $footer = "_report run by $($env:USERDOMAIN)\$($env:USERNAME)_"
PS C:\> $sb = {
    $os = Get-CimInstance -classname win32_operatingsystem -property caption,
    lastbootUptime
    \[PSCustomObject\]@{
        PSVersion = $PSVersionTable.PSVersion
        OS = $os.caption
        Uptime = (Get-Date) - $os.lastbootUpTime
        SizeFreeGB = (Get-Volume -DriveLetter C).SizeRemaining /1GB
    }
}
PS C:\> $out == ConvertTo-Markdown -title $Title
PS C:\> foreach ($computer in $computers) {
    $out+== Invoke-command -scriptblock $sb -Computer $computer -HideComputerName |
    Select-Object -Property * -ExcludeProperty RunspaceID |
    ConvertTo-Markdown -PreContent "## $($computer.ToUpper())"
}
PS C:\>$out +== ConvertTo-Markdown -PostContent $footer
PS C:\>$out | Set-Content c:\work\report.md
```

Here is an example that creates a series of markdown fragments for each computer and in the end creates a markdown document.

Parameters

-Inputobject

Typically the results of a PowerShell command or expression.

```
Type: Object
Parameter Sets: (All)
Aliases:

Required: False
Position: 1
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Title

Specify a top-level title. You do not need to include any markdown.

Type: **String**
Parameter Sets: ([All](#))

Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-PreContent

Enter whatever content you want to appear before converted input. You can use whatever markdown you wish.

Type: **String[]**
Parameter Sets: ([All](#))

Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-PostContent

Enter whatever content you want to appear after converted input. You can use whatever markdown you wish.

Type: **String[]**
Parameter Sets: ([All](#))

Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Width

Specify the document width. Depending on what you intend to do with the markdown from this command you may want to adjust this value.

Type: **Int32**
Parameter Sets: ([All](#))

Aliases:

Required: False
Position: Named
Default value: 80
Accept pipeline input: False
Accept wildcard characters: False

-AsTable

Format the incoming data as a markdown table. This works best with similar content such as the result of running a PowerShell command.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

[System.Object](#)

Outputs

[System.String](#)

Notes

Learn more about PowerShell: <https://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Convertto-HTML](#)

[Out-File](#)

ConvertTo-UTCTime

Synopsis

Convert a local datetime to universal time.

Syntax

```
ConvertTo-UTCTime [[-DateTime] <DateTime>] [-AsString] [<CommonParameters>]
```

Description

Convert a local datetime to universal time. The default is now but you can specify a datetime value. You also have an option to format the result as a sortable string.

This command was introduced in v2.3.0.

Examples

Example 1

```
PS C:\> Get-Date  
Monday, December 28, 2020 7:43:13 PM  
  
PS C:\> ConvertTo-UTCTime  
Tuesday, December 29, 2020 12:43:37 AM
```

Example 2

```
PS C:\> ConvertTo-UTCTime -asString  
2020-12-29 00:44:01Z
```

Parameters

-DateTime

Enter a Datetime value

Type: **DateTime**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: 0
Default value: now
Accept pipeline input: True ([ByValue](#))
Accept wildcard characters: False

-AsString

Convert the date-time value to a sortable string. This is the same thing as running a command like "{0:u}" -f (Get-Date).ToUniversalTime()

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.DateTime

Outputs

System.DateTime

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[ConvertFrom-UTCTime](#)

[Get-Date](#)

ConvertTo-WPFGrid

Synopsis

Send command output to an interactive WPF-based grid.

Syntax

input (Default)

```
ConvertTo-WPFGrid [[-Title] <String>] [[-Timeout] <Int32>] [-Refresh]
[-Gridlines <String>] [-InitializationScript <ScriptBlock>]
[-UseLocalVariable <String[]>] [-UserProfile] [<CommonParameters>]
```

Input

```
ConvertTo-WPFGrid [[-InputObject] <PSObject>] [[-Title] <String>]
[[-Timeout] <Int32>] [-Refresh] [-Gridlines <String>]
[-InitializationScript <ScriptBlock>] [-UseLocalVariable <String[]>]
[-UserProfile] [<CommonParameters>]
```

scriptblock

```
ConvertTo-WPFGrid [-Scriptblock <ScriptBlock>] [[-Title] <String>]
[[-Timeout] <Int32>] [-Refresh] [-Gridlines <String>]
[-InitializationScript <ScriptBlock>] [-UseLocalVariable <String[]>]
[-UserProfile] [<CommonParameters>]
```

Description

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. You will want to be selective about which properties you pipe through to this command. See examples.

You can specify a timeout value which will automatically close the form. If you specify a timeout and the Refresh parameter, then the contents of the datagrid will automatically refresh using the timeout value as an integer. This will only work when you pipe a PowerShell expression to ConvertTo-WPFGrid as one command. This will fail if you break the command in the PowerShell ISE or use a nested prompt. Beginning with v2.4.0 the form now has a Refresh button which will automatically refresh the datagrid. You should set a refresh interval that is greater than the time it takes to complete the command.

Because the grid is running in a new background runspace, it does not automatically inherit anything from your current session. However, you can use the -UserProfile parameter which will load your user profile scripts into the runspace. You can specify a list of locally defined variables to be used in the form. Use the variable name without the \$. Finally, you can also use the -InitializationScript parameter and specify a scriptblock of

PowerShell code to initialize the runspace. This is helpful when you need to dot source external scripts or import modules not in your module path.

This command runs the WPF grid in a new runspace so your PowerShell prompt will not be blocked. However, after closing the form you may be left with the runspace. You can use Remove-Runspace to clean up or wait until you restart PowerShell.

This command requires a Windows platform.

Examples

EXAMPLE 1

```
PS C:\> Get-Process | Sort-Object WS -Descending |
Select-object -first 20 ID,Name,WS,VM,PM,Handles,StartTime |
ConvertTo-WPFGrid -Refresh -timeout 20 -Title "Top Processes"
```

Get the top 20 processes based on the value of the WorkingSet property and display selected properties in the WPF Grid. The contents will automatically refresh every 20 seconds. You will need to manually close the form.

EXAMPLE 2

```
PS C:\> $vmhost = "CHI-HVR2"
PS C:\> Get-VM -computername $VMHost | Select Name,State,Uptime,
@{Name="AssignedMB";Expression={$_.MemoryAssigned/1mb -as [int]}},
@{Name="DemandMB";Expression={$_.MemoryDemand/1mb -as [int]}} |
ConvertTo-WPFGrid -title "VM Report $VMHost" -timeout 30 -refresh
-uselocalvariable VMHost
```

Get Hyper-V virtual machine information and refresh every 30 seconds. Because the command is using a locally defined variable it is also being used in the form. Note that this would be written as one long pipelined expression. It is formatted here for the sake of the help documentation.

EXAMPLE 3

```
PS C:\> Get-VMData -host CHI-HVR2 |
ConvertTo-WPFGrid -title "VM Data" -refresh -timeout 60 -useprofile
```

This example uses a hypothetical command that might be defined in a PowerShell profile script. ConvertTo-WPFGrid will load the profile scripts so that the data can be updated every 60 seconds.

EXAMPLE 4

```
PS C:\> (Get-Processdata -Computername $computers).where({$_.WorkingSet -ge 100mb}) |
ConvertTo-WPFGrid -Title "Process Report" -UseLocalVariable computers -InitializationScript { . C:\scripts\Get-ProcessData.ps1 } -Refresh -Timeout 30
```

This command runs a function that is defined in a script file. In order for the form to refresh, it must also dot source the script which is happening with the InitializationScript parameter. The example is also loading the

local \$computers variable so that it too is available upon refresh.

Parameters

-InputObject

Typically the results of a PowerShell command or expression. You should select the specific properties you wish to display.

```
Type: PSObject
Parameter Sets: Input
Aliases:

Required: False
Position: 1
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Title

Specify a title for your form.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 2
Default value: ConvertTo-WPFGrid
Accept pipeline input: False
Accept wildcard characters: False
```

-Timeout

By default, the grid will remain displayed until you manually close it. But you can specify a timeout interval in seconds. The minimum accepted value is 5 seconds. If you use this parameter with -Refresh, then the datagrid will be refreshed with results of the PowerShell expression you piped to ConvertTo-WPFGrid.

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: False
Position: 3
Default value: 0
Accept pipeline input: False
Accept wildcard characters: False
```

-Refresh

If you specify this parameter and a Timeout value, this command will refresh the datagrid with the PowerShell expression piped into ConvertTo-WPFGrid. You should use a value that is longer than the time it takes to complete the command that generates your data.

This parameter will only work if you are using ConvertTo-WPFGrid at the end of a pipelined expression. See examples.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-UseProfile

Load your PowerShell profiles into the background runspace.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: profile

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Scriptblock

Enter a scriptblock that will generate data to be populated in the form

```
Type: ScriptBlock
Parameter Sets: scriptblock
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-UseLocalVariable

Load locally defined variables into the background runspace

```
Type: String[]
Parameter Sets: (All)
Aliases: var

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-InitializationScript

Run this scriptblock to initialize the background runspace. You might need to dot source a script file or load a non-standard module.

```
Type: ScriptBlock
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Gridlines

Control how grid lines are displayed in the form. You may not want to have any or perhaps only vertical or horizontal lines.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.Object

Outputs

None

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Out-GridView](#)

[ConvertTo-HTML](#)

[ConvertTo-Markdown](#)

Copy-Command

Synopsis

Copy a PowerShell command.

Syntax

```
Copy-Command [-Command] <String> [[-NewName] <String>] [-IncludeDynamic]
[-AsProxy] [-UseForwardHelp] [<CommonParameters>]
```

Description

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.

For best results, run this in the PowerShell ISE or Visual Studio code, the copied command will be opened in a new tab or file.

Examples

EXAMPLE 1

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

Create a copy of Get-Process called Get-MyProcess.

EXAMPLE 2

```
PS C:\> Copy-Command Get-Eventlog -asproxy -useforwardhelp
```

Create a proxy function for Get-Eventlog and use forwarded help links.

EXAMPLE 3

```
PS C:\> Copy-Command Get-ADComputer Get-MyADComputer -includedynamic
```

Create a wrapper function for Get-ADComputer called Get-MyADComputer. Due to the way the Active Directory cmdlets are written, most parameters appear to be dynamic so you need to include dynamic parameters otherwise there will be no parameters in the final function.

Parameters

-Command

The name of a PowerShell command, preferably a cmdlet but that is not a requirement. You can specify an alias and it will be resolved.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: True
Position: 1
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-NewName

Specify a name for your copy of the command. If no new name is specified, the original name will be used.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 2
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-IncludeDynamic

The command will only copy explicitly defined parameters unless you specify to include any dynamic parameters as well. If you copy a command and it seems to be missing parameters, re-copy and include dynamic parameters.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-AsProxy

Create a traditional proxy function.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-UseForwardHelp

By default the copy process will create a comment-based help block with the original command's help which you can then edit to meet your requirements. Or you can opt to retain the forwarded help links to the original command.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Command](#)

Copy-HelpExample

Synopsis

Copy code snippet from help examples.

Syntax

```
Copy-HelpExample [-Name] <String> [-Path <String>] [-UseGridView]  
[<CommonParameters>]
```

Description

This command is intended to make it easier to copy code snippets from help examples to the clipboard. You can select one or more examples which have been trimmed of comments, blank lines and most prompts. Some code examples contain the output or have several lines of code. You will need to manually delete what you don't want. If this command is run on a Windows system you have a dynamic parameter to use Out-GridView to display your choices. When prompted enter a comma-separated list of the examples you wish to copy. Otherwise, the command will display a console-based menu. Note that if you are using the PowerShell ISE you will be forced to use Out-GridView.

Examples

Example 1

```
PS C:\> Copy-HelpExample -Name Stop-Process

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 all instances of a process

Stop-Process -Name "notepad"

[2] Example 2: Stop a specific instance of a process

Stop-Process -Id 3952 -Confirm -PassThru
Confirm
Are you sure you want to perform this action?
Performing operation "Stop-Process" on Target "notepad (3952)".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help
(default is "Y"):
Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName
----- ----- ----- -----
41 2 996 3212 31 3952 notepad

[3] Example 3: Stop a process and detect that it has stopped

calc
$p = Get-Process -Name "calc"
Stop-Process -InputObject $p
Get-Process | Where-Object {$_.HasExited}

[4] Example 4: Stop a process not owned by the current user

Get-Process -Name "lsass" | Stop-Process

Stop-Process : Cannot stop process 'lsass (596)' because of the following error
: Access is denied
At line:1 char:34
+ Get-Process -Name "lsass" | Stop-Process <<<
[ADMIN]: Get-Process -Name "lsass" | Stop-Process

Warning!
Are you sure you want to perform this action?
Performing operation 'Stop-Process' on Target 'lsass(596)'.
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"):
[ADMIN]: Get-Process -Name "lsass" | Stop-Process -Force
[ADMIN]:

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

The console menu will be displayed using ANSI. Enter a comma separated list of numbers for the items to copy to the clipboard.

Parameters

-Name

Enter the name of the PowerShell command.

Type: `String`
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Path

Gets help that explains how the cmdlet works in the specified provider path. Enter a PowerShell provider path.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-UseGridView

Select help examples using Out-GridView. This parameter is only available on Windows systems. The parameter has an alias of 'ogv'.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: ogv

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

None

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Help](#)

Copy-PSFunction

Synopsis

Copy a local PowerShell function to a remote session.

Syntax

```
Copy-PSFunction [-Name] <String[]> -Session <PSSession> [-Force]
[<CommonParameters>]
```

Description

This command is designed to solve the problem when you want to run a function loaded locally on a remote computer. Copy-PSFunction will copy a PowerShell function that is loaded in your current PowerShell session to a remote PowerShell session. The remote session must already be created. The copied function only exists remotely for the duration of the remote PowerShell session.

If the function relies on external or additional files, you will have to copy them to the remote session separately.

Examples

Example 1

```
PS C:\> "Get-LastBoot", "Get-DiskFree" | Copy-PSFunction -session $S
```

Copy the local functions Get-LastBoot and Get-DiskFree to a previously created PSSession saved as \$S. You could then run the function remotely using Invoke-Command.

Parameters

-Force

Overwrite an existing function with the same name. The default behavior is to skip existing functions.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Name

Enter the name of a local function.

```
Type: String[]
Parameter Sets: (All)
Aliases:

Required: True
Position: 1
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Session

Specify an existing PSSession.

```
Type: PSSession
Parameter Sets: (All)
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String[]

Outputs

Deserialized.System.Management.Automation.FunctionInfo

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Copy-Item](#)

Export-PSAnsifileMap

Synopsis

Export a PSAnsifileMap to a file.

Syntax

```
Export-PSAnsifileMap [-Passthru] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

The PSScriptTools module includes a JSON file that is automatically imported as the global PSAnsifileMap variable. This variable is used for the custom ANSI formatted table view, among other module commands. If you wish to customize the file map, you can use the Set-PSAnsifileMap command. These changes are not permanent and will be overwritten the next time you import the PSScriptTools module. To use your customized settings, you need to export your modified \$PSAnsifileMap object with this command.

The command will export the settings to a JSON file called psansifilemap.json in the root of \$HOME. The next time you import the PSScriptTools module, it will use this file if found. To revert to the default file map either rename or delete the file in \$HOME.

Examples

Example 1

```
PS C:\>Export-PSAnsifileMap -Passthru

Directory: C:\Users\Jeff

Mode          LastWriteTime        Length Name
----          -----          1631 psansifilemap.json
-a--- 12/28/2020 1:02 PM
```

Parameters

-Confirm

Prompts you for confirmation before running the cmdlet.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Passthru

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: wi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

None

System.IO.FileInfo

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-PSAnsFileMap](#)

[Set-PSAnsFileMap](#)

Find-CimClass

Synopsis

Search CIM for a class.

Syntax

```
Find-CimClass [-Classname] <String> [-Exclude <String>]
[-Computername <String>] [<CommonParameters>]
```

Description

This function is designed to search an entire CIM repository for a class name. Sometimes, you can guess a CIM/WMI 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.

This command requires a Windows platform.

Examples

Example 1

```
PS C:\> Find-CimClass -Classname *protection*
```

NameSpace: Root/CIMV2/mdm/dmmap

| CimClassName | CimClassMethods | CimClassProperties |
|-------------------------------------|-----------------|------------------------|
| MDM_AppLocker_EnterpriseDataProt... | {} | {InstanceId, Parent... |
| MDM_AppLocker_EnterpriseDataProt... | {} | {InstanceId, Parent... |
| MDM_EnterpriseDataProtection | {} | {InstanceId, Parent... |
| MDM_EnterpriseDataProtection_Set... | {} | {AllowAzureRMSForED... |
| MDM_Policy_Config01_DataProtecti... | {} | {AllowDirectMemoryA... |
| MDM_Policy_Result01_DataProtecti... | {} | {AllowDirectMemoryA... |
| MDM_Reportng_EnterpriseDataProt... | {} | {InstanceId, LogCou... |
| MDM_Reportng_EnterpriseDataProt... | {} | {InstanceId, Logs, ... |
| MDM_WindowsAdvancedThreatProtect... | {} | {InstanceId, Offboa... |
| MDM_WindowsAdvancedThreatProtect... | {} | {GroupIds, Instance... |
| MDM_WindowsAdvancedThreatProtect... | {} | {Criticality, Grou ... |
| MDM_WindowsAdvancedThreatProtect... | {} | {InstanceId, LastCo... |

NameSpace: Root/Microsoft/SecurityClient

| CimClassName | CimClassMethods | CimClassProperties |
|----------------------------|-----------------|------------------------|
| ProtectionTechnologyStatus | {} | {PackedXml, SchemaV... |
| ... | | |

Example 2

```
PS C:\> Find-CimClass -Classname *volume* -Exclude "win32_Perf*"
```

Search for any class with 'volume' in the name but exclude anything that starts with 'win32_Perf'.

Parameters

-Classname

Enter the name of a CIM/WMI class. Wildcards are permitted.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: True
```

-Computername

Enter the name of a computer to search.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: localhost
Accept pipeline input: False
Accept wildcard characters: False
```

-Exclude

Enter a pattern for class names to EXCLUDE from the results. You can use wildcards or regular expressions.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

Microsoft.Management.Infrastructure.CimClass

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources>

Related Links

[Get-CimClass](#)

Format-Percent

Synopsis

Format a value as a percentage.

Syntax

None (Default)

```
Format-Percent [-Value] <Object> [-Total] <Object> [-Decimal <Int32>]
[<CommonParameters>]
```

String

```
Format-Percent [-Value] <Object> [-Total] <Object> [-Decimal <Int32>]
[-AsString] [<CommonParameters>]
```

Description

This command calculates a percentage of a value from a total, with the formula: (value/total)*100. The default is to return a value to 2 decimal places but you can configure that with -Decimal. There is also an option to format the percentage as a string which will include the % symbol.

Examples

EXAMPLE 1

```
PS C:\> Format-Percent -value 1234.567 -total 5000 -decimal 4
24.6913
```

Calculate a percentage from 1234.567 out of 5000 (i.e. 1234.567/5000) to 4 decimal points.

EXAMPLE 2

```
PS C:\> Get-CimInstance win32_operatingsystem -computer chi-dc04 |
Select-Object PSComputerName,TotalVisibleMemorySize,
@{Name="PctFreeMem";Expression={
    Format-Percent $_.FreePhysicalMemory $_.TotalVisibleMemorySize}}
```

| PSComputerName | TotalVisibleMemorySize | PctFreeMem |
|----------------|------------------------|------------|
| chi-dc04 | 1738292 | 23.92 |

EXAMPLE 3

```
PS C:\> Get-CimInstance win32_operatingsystem -computer chi-dc04 |  
Select-Object PSComputerName,TotalVisibleMemorySize,  
@{Name="PctFreeMem";Expression={  
Format-Percent $_.FreePhysicalMemory $_.TotalVisibleMemorySize -asString}}}  
  


PSComputerName	TotalVisibleMemorySize	PctFreeMem
chi-dc04	1738292	23.92%


```

Parameters

-Value

The numerator value.

```
Type: Object  
Parameter Sets: (All)  
Aliases: X, Numerator  
  
Required: True  
Position: 1  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-Total

The denominator value.

```
Type: Object  
Parameter Sets: (All)  
Aliases: Y, Denominator  
  
Required: True  
Position: 2  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-Decimal

The number of decimal places to return between 0 and 15.

Type: `Int32`Parameter Sets: ([All](#))**Aliases:****Required:** False**Position:** Named**Default value:** 2**Accept pipeline input:** False**Accept wildcard characters:** False

-AsString

Write the result as a string.

Type: `SwitchParameter`

Parameter Sets: String

Aliases:**Required:** False**Position:** Named**Default value:** False**Accept pipeline input:** False**Accept wildcard characters:** False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#) (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.Object

Outputs

System.Double

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Format-Value](#)

[Format-String](#)

Format-String

Synopsis

Options for formatting strings.

Syntax

```
Format-String [-Text] <String> [-Reverse] [-Case <String>]  
[-Replace <Hashtable>] [-Randomize] [<CommonParameters>]
```

Description

Use this command to apply different types of formatting to strings. You can apply multiple transformations.

They are applied in this order:

- 1) Reverse
- 2) Randomization
- 3) Replace
- 4) Case

Examples

EXAMPLE 1

```
PS C:\> "P@ssw0rd" | Format-String -Reverse  
  
dr0wss@P
```

EXAMPLE 2

```
PS C:\> "P@ssw0rd" | Format-String -Reverse -Randomize  
  
rs0Pd@ws
```

EXAMPLE 3

```
PS C:\> $env:computername | Format-String -Case Lower  
  
win81-ent-01
```

EXAMPLE 4

```
PS C:\> Format-String "p*wer2she!!" -Case Alternate  
  
P*WeR2ShE!!
```

EXAMPLE 5

```
PS C:\> Format-String "alphabet" -Randomize -Replace @{a="@";e=3} `n -Case Alternate

3bP1@tH@
```

EXAMPLE 6

```
PS C:\> "pOWERSHELL" | Format-String -Case Toggle

PowerShell
```

Parameters

-Text

Any string you want to format.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: True
Position: 1
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Reverse

Reverse the text string.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-Case

Valid values are Upper, Lower, Proper, Alternate, and Toggle.

Proper case will capitalize the first letter of the string.

Alternate case will alternate between upper and lower case, starting with upper case, e.g. PoWeRsHeLl

Toggle case will make upper case lower and vice versa, e.g. Powershell -> pOWERSHELL

```
Type: String
Parameter Sets: (All)
Aliases:
```

```
Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Replace

Specify a hashtable of replacement values. The hashtable key is the string you want to replace and the value is the replacement (see examples). Replacement keys are CASE SENSITIVE.

```
Type: Hashtable
Parameter Sets: (All)
Aliases:
```

```
Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Randomize

Re-arrange the text in a random order.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:
```

```
Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Format-Value](#)

[Format-Percent](#)

Format-Value

Synopsis

Format a numeric value.

Syntax

Default (Default)

```
Format-Value [-InputObject] <Object> [[-Unit] <String>] [-Decimal <Int32>]  
[<CommonParameters>]
```

Number

```
Format-Value [-InputObject] <Object> [-Decimal <Int32>] [-AsNumber]  
[<CommonParameters>]
```

Auto

```
Format-Value [-InputObject] <Object> [-Decimal <Int32>] [-Autodetect]  
[<CommonParameters>]
```

Currency

```
Format-Value [-InputObject] <Object> [-AsCurrency] [<CommonParameters>]
```

Description

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.

You can let the command auto-detect the value and divide with an appropriate value.

Examples

Example 1

```
PS C:\> Get-CimInstance -class win32_logicaldisk -filter "DriveType=3" |  
Select-Object DeviceID,  
@{Name="SizeGB";Expression={$_.size | Format-Value -unit GB}},  
@{Name="FreeGB";Expression={$_.freespace | Format-Value -unit GB -decimal 2}}  
  
DeviceID          SizeGB        FreeGB  
-----          -----  
C:                200          124.97  
D:                437          29.01  
E:                25           9.67
```

Example 2

```
PS C:\> (Get-Process chrome | measure ws -sum ).sum |  
Format-Value -Autodetect -verbose -Decimal 4  
  
VERBOSE: Starting: Format-Value  
VERBOSE: Status: Using parameter set Auto  
VERBOSE: Status: Formatting 965332992  
VERBOSE: Status: Using Autodetect  
VERBOSE: ..as MB  
VERBOSE: Status: Reformatting 920.61328125  
VERBOSE: ..to 4 decimal places  
920.6133  
VERBOSE: Ending: Format-Value
```

Example 3

```
PS C:\> 3456.5689 | Format-Value -AsCurrency  
$3,456.57
```

Format a value as currency.

Example 4

```
PS C:\> 1234567.8973 | Format-Value -AsNumber -Decimal 2  
1,234,567.90
```

Format the value as a number to 2 decimal points.

Parameters

-InputObject

Type: Object

Parameter Sets: (All)

Aliases:**Required:** True**Position:** 2**Default value:** None**Accept pipeline input:** True (**ByValue**)**Accept wildcard characters:** False

-Unit

The unit of measurement for your value. Valid choices are "KB", "MB", "GB", "TB", and "PB".

If you don't specify a unit, the value will remain as is, although you can still specify the number of decimal places.

Type: String

Parameter Sets: Default

Aliases:**Required:** False**Position:** 1**Default value:** None**Accept pipeline input:** False**Accept wildcard characters:** False

-Decimal

The number of decimal places to return between 0 and 15.

Type: Int32

Parameter Sets: Default, Number, Auto

Aliases:**Required:** False**Position:** Named**Default value:** 0**Accept pipeline input:** False**Accept wildcard characters:** False

-Autodetect

Attempt to autodetect and format the value.

Type: **SwitchParameter**
Parameter Sets: Auto
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-AsCurrency

Format the numeric value as currency using detected cultural settings. The output will be a string.

Type: **SwitchParameter**
Parameter Sets: Currency
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-AsNumber

Format the numeric value as a number using detected cultural settings for a separator like a comma. If the incoming value contains decimal points, by default they will be removed unless you use -Decimal.

The output will be a string.

Type: **SwitchParameter**
Parameter Sets: Number
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.Object

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Format-String](#)

[Format-Percent](#)

Get-CommandSyntax

Synopsis

Get provider-specific command syntax.

Syntax

```
Get-CommandSyntax [-Name] <String> [<CommonParameters>]
```

Description

Some PowerShell commands are provider aware and may have special syntax or parameters depending on what PSDrive you are using when you run the command. In Windows PowerShell, the help system could show you syntax based on a given path. However, this no longer appears to work. This command is intended as an alternative. Specify a cmdlet or function name, and the output will display the syntax detected when using different providers. Dynamic parameters will be highlighted with an ANSI-escape sequence.

Examples

Example 1

```
PS C:\> Get-CommandSyntax -Name Get-Item

Registry

Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>]
[-Exclude <string[]>] [-Force] [-Credential <pscredential>]
[<CommonParameters>]

Get-Item -LiteralPath <string[]> [-Filter <string>] [-Include <string[]>]
[-Exclude <string[]>] [-Force] [-Credential <pscredential>]
[<CommonParameters>]
```

Alias

```
Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>]
[-Exclude <string[]>] [-Force] [-Credential <pscredential>]
[<CommonParameters>]

Get-Item -LiteralPath <string[]> [-Filter <string>] [-Include <string[]>]
[-Exclude <string[]>] [-Force] [-Credential <pscredential>]
[<CommonParameters>]

...
```

The output will show each PowerShell Provider and the corresponding command syntax. Dynamic parameters will be highlighted by color.

Parameters

-Name

Enter the name of a PowerShell cmdlet or function. Ideally, it has been loaded into the current PowerShell session.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Help](#)

[Get-Command](#)

Get-DirectoryInfo

Synopsis

Get directory information.

Syntax

```
Get-DirectoryInfo [[-Path] <String>] [-Depth <Int32>] [<CommonParameters>]
```

Description

This command is designed to provide quick access to top-level directory information. The default behavior is to show the total number of files in the immediate directory. Although the command will also capture the total file size in the immediate directory. You can use the Depth parameter to recurse through a specified number of levels.

The command output will use a wide format by default. However, other wide views are available. See Examples.

Examples

Example 1

```
PS C:\> Get-DirectoryInfo

Path: C:\

gemfonts [15]           PerfLogs [0]
Pluralsight [17]         Presentations [1]
Program Files [0]         Program Files (x86) [0]
Ruby27-x64 [3]           Scripts [3652]
Thunderbird [0]           Training [3]
Users [0]                 Windows [38]
Windows.old [0]           Windows10Upgrade [23]
work [13]
```

The default output will use ANSI escape sequences.

Example 2

```
PS C:\> Get-DirectoryInfo -Path D:\ | Format-Wide -View sizemb
```

Path: D:\

| | |
|-------------------|--------------------|
| autolab [0MB] | backtemp [0MB] |
| Backup [0.01MB] | Backups [140.49MB] |
| bovine320 [0MB] | Databases [0MB] |
| Exports [0MB] | iso [16137.65MB] |
| JDHIT [35.58MB] | Logitech [0MB] |
| OneDrive [0MB] | rip [60.99MB] |
| temp [10.67MB] | video [83.56MB] |
| VMDisks [68053MB] | VMs [0MB] |

Using one of the alternate Format-Wide views. Other views are size and sizekb.

Example 3

```
PS C:\> Get-DirectoryInfo D:\Autolab\ -Depth 2 |
Format-Table -GroupBy parent -Property Name,File* -wrap
```

Parent: D:\Autolab

| Name | FileCount | FileSize |
|------------------------|-----------|-------------|
| Configurations | 0 | 0 |
| Hotfixes | 0 | 0 |
| ISOs | 6 | 16838768742 |
| MasterVirtualHardDisks | 3 | 22326280192 |
| Resources | 0 | 0 |
| VMVirtualHardDisks | 0 | 0 |

Parent: D:\Autolab\Configurations

| Name | FileCount | FileSize |
|---------------------------------------|-----------|----------|
| Implement-Windows-Server-DHCP-2016 | 7 | 65126 |
| Jason-DSC-Env | 7 | 66933 |
| microsoft-powershell-implementing-jea | 7 | 65462 |
| MultiRole | 7 | 65820 |
| MultiRole-Server-2016 | 7 | 62063 |
| PowerShellLab | 7 | 83541 |
| SingleServer | 4 | 15784 |
| SingleServer2012R2 | 4 | 15937 |
| SingleServer2012R2-GUI | 4 | 16005 |
| SingleServer-GUI-2016 | 4 | 16397 |
| SingleServer-GUI-2019 | 4 | 15845 |
| Windows10 | 4 | 20695 |

Parent: D:\Autolab\Configurations\PowerShellLab

| Name | FileCount | FileSize |
|-----------|-----------|----------|
| PostSetup | 5 | 15275 |

Here's an example using the `DirectoryStat` object with different formatting.

Parameters

-Depth

The Depth parameter determines the number of subdirectories to recursively query.

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Path

Specify the top-level path.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: `-Debug`, `-ErrorAction`, `-ErrorVariable`, `-InformationAction`, `-InformationVariable`, `-OutVariable`, `-OutBuffer`, `-PipelineVariable`, `-Verbose`, `-WarningAction`, and `-WarningVariable`. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

DirectoryStat

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-ChildItem](#)

Get-FileItem

Synopsis

A PowerShell version of the Where CLI command.

Syntax

Default (Default)

```
Get-FileItem [-Pattern] <String[]> [-Regex] [-Full] [-Quiet] [-First]  
[<CommonParameters>]
```

Path

```
Get-FileItem [-Pattern] <String[]> [-Regex] [-Path <String[]>] [-Recurse]  
[-Full] [-Quiet] [-First] [<CommonParameters>]
```

Description

This is an enhanced, PowerShell version of the WHERE command from the traditional CLI which will find files in %PATH% that match a particular pattern.

Examples

EXAMPLE 1

```
PS C:\> Get-Fileitem notepad.exe  
  
C:\Windows\system32\notepad.exe  
C:\Windows\notepad.exe
```

Find notepad.exe in %PATH% and return the full file name. This is the default behavior.

EXAMPLE 2

```
PS C:\> PSWhere foo.exe -quiet  
  
False
```

Search for foo.exe and return \$True if found. This command is using the PSWhere alias.

EXAMPLE 3

```
PS C:\> Get-FileItem "^\d+\S+.txt" -Regex -path c:\scripts -full

Directory: C:\scripts
```

| Mode | LastWriteTime | Length | Name |
|-------|---------------|----------|----------------------------|
| -a--- | 12/5/2007 | 2:19 PM | 30146 1000FemaleNames.txt |
| -a--- | 12/5/2007 | 2:19 PM | 29618 1000MaleNames.txt |
| -a--- | 6/2/2010 | 11:02 AM | 31206 1000names.txt |
| -a--- | 6/3/2010 | 8:52 AM | 3154 100names.txt |
| -a--- | 4/13/2012 | 10:27 AM | 3781 13ScriptBlocks-v2.txt |
| -a--- | 8/13/2010 | 10:41 AM | 3958 13ScriptBlocks.txt |
| -a--- | 2/7/2011 | 1:37 PM | 78542 2500names.txt |
| -a--- | 2/8/2011 | 9:43 AM | 157396 5000names.txt |

Find all TXT files in C:\Scripts that start with a number and display full file information.

Parameters

-Pattern

The name of the file to find. Separate multiple entries with a comma. Wildcards are allowed. You can also specify a regular expression pattern by including the -REGEX parameter.

Type: `String[]`
 Parameter Sets: `(All)`
 Aliases:

Required: True
Position: 1
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Regex

Indicates that the pattern is a regular expression.

Type: `SwitchParameter`
 Parameter Sets: `(All)`
 Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-Path

The folders to search other than %PATH%.

```
Type: String[]
Parameter Sets: Path
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Recurse

Used with -Path to indicate a recursive search.

```
Type: SwitchParameter
Parameter Sets: Path
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-Full

Write the full file object to the pipeline. The default is just the full name.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-Quiet

Returns True if a match is made. This parameter will override -Full.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-First

Stop searching after the pattern is found. Don't search any more paths.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

String

Outputs

String

Boolean

System.IO.FileInfo

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-ChildItem](#)

[Where.exe](#)

Get-FolderSizeInfo

Synopsis

Get folder size information.

Syntax

```
Get-FolderSizeInfo [-Path] <String[]> [-Hidden] [<CommonParameters>]
```

Description

This command is an alternative to discovering the size of a folder, or at least an easier method. Use the -Hidden parameter to include hidden files in the output. The measurement will include all files in all sub-folders.

Note that this command has been optimized for performance, but if you have a lot of files to count that will take time, especially when using Windows PowerShell. When querying system folders like C:\Windows on a Windows PowerShell platform, you might get better results including hidden files. Due to the nature of the .NET Framework changes, you might see different results for the same folder when run in PowerShell 7 compared to Windows PowerShell 5.1.

Examples

Example 1

```
PS C:\> Get-FolderSizeInfo -Path d:\temp
```

| Computername | Path | TotalFiles | TotalSize |
|--------------|---------|------------|-----------|
| BOVINE320 | D:\temp | 48 | 121824451 |

Example 2

```
PS C:\> Get-FolderSizeInfo -Path d:\temp -hidden
```

| Computername | Path | TotalFiles | TotalSize |
|--------------|---------|------------|-----------|
| BOVINE320 | D:\temp | 146 | 125655552 |

Include hidden files.

Example 3

```
PS C:\> Get-ChildItem d:\ -Directory | Get-FolderSizeInfo |
Where-Object TotalSize -gt 1MB | Sort-Object TotalSize -Descending |
Format-Table -View mb
```

| Computername | Path | TotalFiles | TotalSizeMB |
|--------------|-----------------------|------------|-------------|
| BOVINE320 | D:\VMDisks | 18 | 114873.7246 |
| BOVINE320 | D:\ISO | 17 | 42526.8204 |
| BOVINE320 | D:\SQLServer2017Media | 1 | 710.8545 |
| BOVINE320 | D:\officeViewers | 4 | 158.9155 |
| BOVINE320 | D:\Temp | 48 | 116.1809 |
| BOVINE320 | D:\Sysinternals | 153 | 59.6169 |
| BOVINE320 | D:\blog | 41 | 21.9948 |
| BOVINE320 | D:\BackTemp | 2 | 21.6734 |
| BOVINE320 | D:\rip | 3 | 11.1546 |
| BOVINE320 | D:\logs | 134 | 3.9517 |
| BOVINE320 | D:\2016 | 5 | 1.5608 |

Get the top-level directories from D and pipe them to Get-FolderSizeInfo. Items with a total size of greater than 1MB are sorted on the total size and then formatted as a table using a built-in view called MB which formats the total size in MB. There are also views named KB,GB and TB to display formatted results accordingly.

Example 4

```
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 |

Get all sub-folders under C:\work greater than 2MB in size and display using the Name table view.

Parameters

-Hidden

Include hidden directories.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Path

Enter a file system path like C:\Scripts.

```
Type: String[]
Parameter Sets: (All)
Aliases: PSPath

Required: True
Position: 0
Default value: None
Accept pipeline input: True (ByPropertyName, ByValue)
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String[]

Outputs

FolderSizeInfo

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Test-EmptyFolder](#)

[Get-ChildItem](#)

Measure-Object

Get-FormatView

Synopsis

Get defined format views.

Syntax

```
Get-FormatView [[ -TypeName] <String>] [[-PowerShellVersion] <Version>] [<CommonParameters>]
```

Description

PowerShell's formatting system includes custom views that display objects in different ways. Unfortunately, this information is not readily available to a typical PowerShell user. Get-FormatView displays the available views for a given object type. You might get additional views when importing modules such as the PSScriptTools module. The result is there might be different views depending on if you use Format-Table, or Format-List. If you only see a single defined view, that is the default for that type of control.

Examples

Example 1

```
PS C:\> Get-FormatView system.diagnostics.process
```

Type: **System.Diagnostics.Process**

| Format | Name |
|--------|-----------|
| ----- | ----- |
| Table | process |
| Table | Priority |
| Table | StartTime |
| Wide | process |
| Table | WS |

The default view should be the first one listed for each format type. With this information, you can now run a command like `Get-Process | Format-Table -view Priority`. The WS view is added when you import the PSScriptTools module.

Example 2

```
PS C:\> (Get-Service bits).gettype() | Get-FormatView
```

Type: System.ServiceProcess.ServiceController

| | |
|---------------|---|
| Format | Name |
| ----- | ----- |
| Table | service |
| List | System.ServiceProcess.ServiceController |
| Table | service |
| Table | Ansi |

You can pipe a type name to the command.

Example 3

```
PS C:\> Get-FormatView | Where-Object Format -eq Table |
Group-Object typename | Where-Object count -gt 1 | Select-Object Name,
@{Name="Names";Expression = {$_.group.name}}
```

| Name | Names |
|---|---|
| --- | ----- |
| FolderSizeInfo | { default , MB, GB, KB...} |
| gitsize | { mb , default} |
| ModuleCommand | { default , verb} |
| System.Diagnostics.Process | {process, Priority , StartTime..} |
| System.IO.DirectoryInfo | {children, ansi} |
| System.IO.FileInfo | {children, ansi} |
| System.Management.Automation.AliasInfo | { CommandInfo , AliasInfo, opti..} |
| System.Management.Automation.ApplicationInfo | { CommandInfo , ApplicationInfo} |
| System.Management.Automation.ExternalScriptInfo | { CommandInfo , ExternalScriptI..} |
| System.Management.Automation.FilterInfo | { CommandInfo , FilterInfo} |
| System.Management.Automation.FunctionInfo | { CommandInfo , FunctionInfo} |
| System.Management.Automation.ScriptInfo | { CommandInfo , ScriptInfo} |
| System.ServiceProcess.ServiceController | {service, service, Ansi} |

This example expression is getting all Table format views for types that have more than 1 defined. If a type only has a single view, that is the default which you are seeing already. The output you see here shows additional table views for different object types.

Parameters

-TypeName

Specify a typename such as System.Diagnostics.Process.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: 0
Default value: *
Accept pipeline input: True ([ByValue](#))
Accept wildcard characters: True

-PowerShellVersion

Specify the version of PowerShell this cmdlet gets for the formatting data. Enter a two-digit number separated by a period.

Type: **Version**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: 1
Default value: your current PowerShell version
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

PSFormatView

Notes

This command relies on data provided by Get-FormatData. Some object types might be stored in PowerShell in unexpected ways. This command should have an alias of gfv.

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-FormatData](#)

[Get-Member](#)

Get-GitSize

Synopsis

Get the size of .git folder.

Syntax

```
Get-GitSize [[-Path] <String>] [<CommonParameters>]
```

Description

When using git, it creates a hidden folder for change tracking. Because the file is hidden it is easy to overlook how large it might become. The command uses a formatting file to display a default view. There is an additional table view called MB that you can use.

Examples

EXAMPLE 1

```
PS C:\Scripts\PiedPiper> Get-GitSize

Path                                Files      SizeKB
---                                ----      -----
C:\scripts\PiedPiper                  751       6859.9834
```

Get the size of the .git folder from the current path.

EXAMPLE 2

```
PS C:\> Get-ChildItem c:\scripts -Directory | Get-GitSize |
Sort-Object -property Size -descending |
Select-Object -first 5 -property Computername,Name,Files,Size

Computername Name      Files      Size
----- ----      ----      -----
WIN10DSK2    PSAutoLab   526 193760657
WIN10DSK2    DevOps-Courses 29 53298180
WIN10DSK2    PSScriptTools 751 7024623
WIN10DSK2    PSGUI        32 6705894
WIN10DSK2    DscWorkshop  24 5590511
```

Get the directories under C:\Scripts that have a .git folder and sort on the Size property in descending order. Then select the first 5 directories and use the specified properties.

EXAMPLE 3

```
PS S:\PSReleaseTools> Get-GitSize | Format-Table -view mb
```

| Path | Files | SizeMB |
|---------------------------|-------|--------|
| C:\scripts\PSReleaseTools | 440 | 3.0588 |

Get the git folder size and format using the MB table view.

Parameters

-Path

The path to the parent folder, not the .git folder.

```
Type: String
Parameter Sets: (All)
Aliases: pspath

Required: False
Position: 1
Default value: current location
Accept pipeline input: True (ByPropertyName, ByValue)
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String

Outputs

gitSize

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

This is a variation of code posted at <https://gist.github.com/jdhitsolutions/cbdc7118f24ba551a0bb325664415649>

Related Links

[Get-ChildItem](#)

[Measure-Object](#)

[Remove-MergedBranch](#)

Get-ModuleCommand

Synopsis

Get a summary of module commands.

Syntax

name (Default)

```
Get-ModuleCommand [-Name] <String> [-ListAvailable] [<CommonParameters>]
```

fqdn

```
Get-ModuleCommand -FullyQualifiedNamespace <ModuleSpecification> [-ListAvailable] [<CommonParameters>]
```

Description

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 the module but not currently loaded. Note that if the help file is malformed or missing, you might get oddly formatted results.

Examples

Example 1

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

ModuleName: PSCalendar

Name          Alias Synopsis
----          ----
Get-Calendar   cal   Displays a visual representation of a calendar.
Show-Calendar  scal  Display a colorized calendar month in the console.
Show-GuiCalendar gcal  Display a WPF-based calendar
```

Get module commands using the default formatted view. You can install this module from the PowerShell Gallery.

Example 2

```
PS C:\> Get-ModuleCommand smbshare -ListAvailable | Format-List
```

| | | |
|--|---|----------------------|
| Name | : | Block-SmbShareAccess |
| Alias | : | blsmba |
| Synopsis : Adds a deny ACE for a trustee to the security descriptor of the SMB share. | | |
| Name | : | Close-SmbOpenFile |
| Alias | : | cssmbo |
| Synopsis : Closes a file that is open by one of the clients of the SMB server. | | |
| Name | : | Close-SmbSession |
| Alias | : | cssmbse |
| Synopsis : Ends forcibly the SMB session. | | |
| ... | | |

Using the default list view.

Example 3

```
PS C:\> Get-ModuleCommand PSScriptTools | Format-Table -view verb
```

| | | | |
|------------------|--------------|-------------|---------------------------------------|
| Verb: Add | | | |
| Name | Alias | Type | Synopsis |
| --- | --- | --- | --- |
| Add-Border | | Function | Create a text border around a string. |

| | | | |
|----------------------|--------------|-------------|-------------------------------------|
| Verb: Compare | | | |
| Name | Alias | Type | Synopsis |
| --- | --- | --- | --- |
| Compare-Module | cmo | Function | Compare PowerShell module versions. |

...

Display commands using a custom table view called 'Verb'.

Parameters

-FullyQualifiedName

Specifies names of modules in the form of ModuleSpecification objects. The FullyQualifiedName parameter accepts a module name that is specified in the following formats:

`@{ModuleName = "modulename"; ModuleVersion = "version_number"}`

`@{ModuleName = "modulename"; ModuleVersion = "version_number"; Guid = "GUID"}`

ModuleName and ModuleVersion are required, but Guid is optional.

You cannot specify the FullyQualifiedName parameter in the same command as a Name parameter.

```
Type: ModuleSpecification
```

```
Parameter Sets: fqdn
```

```
Aliases:
```

```
Required: True
```

```
Position: Named
```

```
Default value: None
```

```
Accept pipeline input: False
```

```
Accept wildcard characters: False
```

-ListAvailable

Indicates that this cmdlet gets all installed modules. Get-Module finds modules in paths listed in the PSModulePath environment variable. Without this parameter, Get-ModuleCommand gets only the modules that are both listed in the PSModulePath environment variable, and that are loaded in the current session.

ListAvailable does not return information about modules that are not found in the PSModulePath environment variable, even if those modules are loaded in the current session.

```
Type: SwitchParameter
```

```
Parameter Sets: (All)
```

```
Aliases:
```

```
Required: False
```

```
Position: Named
```

```
Default value: None
```

```
Accept pipeline input: False
```

```
Accept wildcard characters: False
```

-Name

The name of an installed module.

```
Type: String
```

```
Parameter Sets: name
```

```
Aliases:
```

```
Required: True
```

```
Position: 0
```

```
Default value: None
```

```
Accept pipeline input: False
```

```
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

ModuleCommand

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Command](#)

[Get-Module](#)

Get-MyAlias

Synopsis

Get non-default aliases defined in the current session.

Syntax

```
Get-MyAlias [-NoModule] [<CommonParameters>]
```

Description

Often you might define aliases for functions and scripts you use often. It may difficult sometimes to remember them all or to find them in the default Get-Alias output. This command will list all currently defined aliases that are not part of the initial PowerShell state.

The PSScriptTools module also includes a custom formatting file for alias objects which you can use with Get-Alias or Get-MyAlias. See examples.

Examples

Example 1

| CommandType | Name | Version | Source |
|-------------|--------------------------------|---------|-----------------|
| Alias | abt -> Get-AboutInfo | | |
| Alias | bv -> Brave | | |
| Alias | cal -> Get-Calendar | 1.11.0 | PSCalendar |
| Alias | cc -> Copy-Command | 2.27.0 | PSScriptTools |
| Alias | cfn -> New-CustomFileName | 2.27.0 | PSScriptTools |
| Alias | CFS -> ConvertFrom-String | 3.1.0.0 | Microsoft.Po... |
| Alias | cft -> ConvertFrom-Text | 2.27.0 | PSScriptTools |
| Alias | chc -> Convert-HashTableToCode | 2.27.0 | PSScriptTools |
| Alias | che -> Copy-HelpExample | 2.27.0 | PSScriptTools |
| Alias | cl -> Create-List | | |
| Alias | clr -> Convert-EventLogRecord | 2.27.0 | PSScriptTools |
| Alias | clt -> ConvertTo-LocalTime | 2.27.0 | PSScriptTools |
| Alias | cmo -> Compare-Module | 2.27.0 | PSScriptTools |
| ... | | | |

Get all aliases that aren't par of the initial session state. This will include aliases defined in any modules you have loaded.

Example 2

```
PS C:\> Get-MyAlias -NoModule
```

| CommandType | Name | Version | Source |
|-------------|----------------------|---------|--------|
| Alias | abt -> Get-AboutInfo | | |
| Alias | bv -> Brave | | |
| Alias | cl -> Create-List | | |

Get defined aliases that don't belong to a module. These should be aliases you have defined in stand-alone scripts or your profile.

Example 3

```
PS C:\> Get-MyAlias -NoModule | Format-Table -View options
```

| Name | Definition | Options | ModuleName | Version |
|------|---------------|----------|------------|---------|
| abt | Get-AboutInfo | None | | |
| bv | Brave | None | | |
| cl | Create-List | None | | |
| np | notepad | ReadOnly | | |

Get your aliases and pipe to format table using a custom view defined by the PSScriptTools module.

Parameters

-NoModule

Only show aliases that DO NOT belong to a module.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.Management.Automation.AliasInfo

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Alias](#)

Get-MyCounter

Synopsis

Get performance counter data.

Syntax

```
Get-MyCounter [[-Counter] <String[]>] [-SampleInterval <Int32>]
[-MaxSamples <Int64>] [-Continuous] [-ComputerName <String[]>]
[<CommonParameters>]
```

Description

Get-MyCounter is an enhanced version of Get-Counter which is available on Windows platforms to retrieve performance counter data. One of the challenges with Get-Counter is how it formats results. Get-MyCounter takes the same information and writes a custom object to the pipeline that is easier to work with. You can pipe counters from Get-Counter to this command.

The custom object has an associated formatting file with custom views. See examples.

Examples

Example 1

```
PS C:\> Get-Counter -list "system" | Get-MyCounter

Computername: SERVER18

Timestamp          Category Counter           Value
-----          -----
11/4/2020 10:48:47 AM system file read operations/sec    203.3096
11/4/2020 10:48:47 AM system file write operations/sec   252.6566
11/4/2020 10:48:47 AM system file control operations/sec 197.3879
11/4/2020 10:48:47 AM system file read bytes/sec       206336.5281
11/4/2020 10:48:47 AM system file write bytes/sec      56409.5271
11/4/2020 10:48:47 AM system file control bytes/sec    10452.6787
11/4/2020 10:48:47 AM system context switches/sec      6068.6924
11/4/2020 10:48:47 AM system system calls/sec        17854.7266
11/4/2020 10:48:47 AM system file data operations/sec   455.9662
11/4/2020 10:48:47 AM system system up time          73056.4005
11/4/2020 10:48:47 AM system processor queue length    0
11/4/2020 10:48:47 AM system processes                  301
11/4/2020 10:48:47 AM system threads                   4502
11/4/2020 10:48:47 AM system alignment fixups/sec      0
11/4/2020 10:48:47 AM system exception dispatches/sec  6.9086
11/4/2020 10:48:47 AM system floating emulations/sec   0
11/4/2020 10:48:47 AM system % registry quota in use   4.0327
```

Get all of the System counters with Get-Counter and pipe them to Get-MyCounter.

Example 2

```
PS C:\> Get-MyCounter -computername server18 | Format-table -view category
```

Category: network interface(intel[r] ethernet connection [11] i219-lm)

| Computername | Timestamp | Counter | Value |
|--------------|-----------------------|-----------------|-----------|
| SERVER18 | 11/4/2020 11:20:09 AM | bytes total/sec | 2662.0477 |

Category: network interface(intel[r] wi-fi 6 ax201 160mhz)

| Computername | Timestamp | Counter | Value |
|--------------|-----------------------|-----------------|-------|
| SERVER18 | 11/4/2020 11:20:09 AM | bytes total/sec | 0 |

Category: processor(_total)

| Computername | Timestamp | Counter | Value |
|--------------|-----------------------|------------------|--------|
| SERVER18 | 11/4/2020 11:20:09 AM | % processor time | 1.4158 |

Category: memory

| Computername | Timestamp | Counter | Value |
|--------------|-----------------------|--------------------------|---------|
| SERVER18 | 11/4/2020 11:20:09 AM | % committed bytes in use | 40.5214 |
| SERVER18 | 11/4/2020 11:20:09 AM | cache faults/sec | 0 |

Category: physicaldisk(_total)

| Computername | Timestamp | Counter | Value |
|--------------|-----------------------|---------------------------|--------|
| SERVER18 | 11/4/2020 11:20:09 AM | % disk time | 0.0217 |
| SERVER18 | 11/4/2020 11:20:09 AM | current disk queue length | 0 |

Get the default counter set and pipe to Get-MyCounter to get values for the local host.

Example 3

```
PS C:\> $c = (Get-Counter -list logicaldisk).PathsWithInstances |
Where-Object {$_. -match "\(\w+\)\\\%\"}
PS C:\> Get-MyCounter -Counter $c -ComputerName SERVER18,SERVER2 |
Format-Table -view category
```

Category: logicaldisk(c:)

| Computername | Timestamp | Counter | Value |
|--------------|-----------------------|-------------------|---------|
| SERVER18 | 11/4/2020 10:50:03 AM | % free space | 48.3822 |
| SERVER2 | 11/4/2020 10:50:04 AM | % free space | 54.5916 |
| SERVER18 | 11/4/2020 10:50:03 AM | % disk time | 1.4669 |
| SERVER2 | 11/4/2020 10:50:04 AM | % disk time | 5.3787 |
| SERVER18 | 11/4/2020 10:50:03 AM | % disk read time | 0.8467 |
| SERVER2 | 11/4/2020 10:50:04 AM | % disk read time | 0 |
| SERVER18 | 11/4/2020 10:50:03 AM | % disk write time | 0.6203 |
| SERVER2 | 11/4/2020 10:50:04 AM | % disk write time | 5.3787 |
| SERVER18 | 11/4/2020 10:50:03 AM | % idle time | 98.5846 |
| SERVER2 | 11/4/2020 10:50:04 AM | % idle time | 93.3567 |

```
PS C:\> Get-MyCounter -Counter $c -ComputerName SERVER18,SERVER2 |
Sort-Object Computername
```

Computername: SERVER18

| Timestamp | Category | Counter | Value |
|-----------------------|-----------------|-------------------|---------|
| 11/4/2020 10:50:35 AM | logicaldisk(c:) | % free space | 48.3822 |
| 11/4/2020 10:50:35 AM | logicaldisk(c:) | % disk time | 0.0263 |
| 11/4/2020 10:50:35 AM | logicaldisk(c:) | % disk read time | 0 |
| 11/4/2020 10:50:35 AM | logicaldisk(c:) | % disk write time | 0.0263 |
| 11/4/2020 10:50:35 AM | logicaldisk(c:) | % idle time | 99.9435 |

Computername: SERVER2

| Timestamp | Category | Counter | Value |
|-----------------------|-----------------|-------------------|---------|
| 11/4/2020 10:50:37 AM | logicaldisk(c:) | % free space | 54.5916 |
| 11/4/2020 10:50:37 AM | logicaldisk(c:) | % disk time | 0 |
| 11/4/2020 10:50:37 AM | logicaldisk(c:) | % disk read time | 0 |
| 11/4/2020 10:50:37 AM | logicaldisk(c:) | % disk write time | 0 |
| 11/4/2020 10:50:37 AM | logicaldisk(c:) | % idle time | 99.0114 |

The first command gets a collection of logical disk counters for drive C. The second command gets performance counter data for two remote computers and formats the results using a custom view. The last command repeats the process but sorts the result by the computer name.

Example 4

```
PS C:\> $p == Get-MyCounter -Counter "\IPv4\Datagrams/sec" -ComputerName SERVER2
-SampleInterval 5 -MaxSamples 30
```

This command will get the specified counter value every 5 seconds for a total of 30 samples.

Parameters

-ComputerName

The name of a remote computer. Querying a remote computer does not use PowerShell remoting and requires administrator-level permissions. Typically, the RemoteRegistry service must also be running.

```
Type: String[]
Parameter Sets: (All)
Aliases: Cn

Required: False
Position: Named
Default value: localhost
Accept pipeline input: False
Accept wildcard characters: False
```

-Continuous

Gets samples continuously until you press CTRL+C. By default, Get-MyCounter gets only one counter sample. You can use the SampleInterval parameter to set the interval for continuous sampling.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Counter

Gets data from the specified performance counters. Enter one or more counter paths. Wildcards are permitted only in the Instance value. You can also pipe counter path strings to Get-MyCounter.

Each counter path has the following format:

\ComputerName\CounterSet(Instance)\CounterName

For example:

\Server01\Processor(2)\% User Time

The ComputerName element is optional. If you omit it, Get-MyCounter uses the value of the ComputerName parameter.

Type: `String[]`
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: 0
Default value: None
Accept pipeline input: True ([ByPropertyName](#), [ByValue](#))
Accept wildcard characters: False

-MaxSamples

Specifies the number of samples to get from each counter. The default is 1 sample. To get samples continuously (no maximum sample size), use the Continuous parameter.

Type: `Int64`
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-SampleInterval

Specifies the time between samples in seconds. The minimum value and the default value are 1 second

Type: `Int32`
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String[]

Outputs

myCounter

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Counter](#)

Get-MyTimeInfo

Synopsis

Display time settings for a collection of locations.

Syntax

```
Get-MyTimeInfo [[-Locations] <OrderedDictionary>] [-HomeTimeZone <String>]
[-DateTime <DateTime>] [-AsTable] [-AsList] [<CommonParameters>]
```

Description

This command is designed to present a console-based version of a world clock. You provide a hashtable of locations and their respective time zones and the command will write a custom object to the pipeline. Be aware that TimeZone names may vary depending on the .NET Framework version. You may need to enumerate using a command like [System.TimeZoneInfo]::GetSystemTimeZones().ID or the Get-TZList command.

A Note on Formatting:

Normally, a PowerShell command should write an object to the pipeline and then you could use Format-Table or Format-List as you wanted. Those commands will continue to work. However, given the way this command writes to the pipeline, that is with dynamically generated properties, it is difficult to create the usual format ps1xml file. To provide some nicer formatting this command has optional parameters to help your format the output. Note that even though it may look like a table, the output object will be a string.

This command was added in v2.3.0.

Examples

EXAMPLE 1

```
P{S C:\>Get-MyTimeInfo

Now : 3/4/2020 1:28:43 PM
Home : 3/4/2020 1:28:43 PM
UTC : 3/4/2020 6:28:43 PM
Singapore : 3/5/2020 2:28:43 AM
Seattle : 3/4/2020 10:28:43 AM
Stockholm : 3/4/2020 7:28:43 PM
IsDaylightSavings : False
```

The default output is a custom object with each timezone as a property.

EXAMPLE 2

```
Get-MyTimeInfo -AsTable
```

Now: 03/04/2020 13:28:11
UTC: 03/04/2020 18:28:11

| Home | Singapore | Seattle | Stockholm | IsDaylightSavings |
|---------------------|---------------------|----------------------|---------------------|-------------------|
| --- | --- | --- | --- | --- |
| 3/4/2020 1:28:11 PM | 3/5/2020 2:28:11 AM | 3/4/2020 10:28:11 AM | 3/4/2020 7:28:11 PM | False |

Display current time information as a table. The output is a string.

EXAMPLE 3

```
PS C:\> Get-MyTimeInfo -AsList
```

Now: 03/04/2020 13:27:03
UTC: 03/04/2020 18:27:03

| | | |
|--------------------------|---|----------------------|
| Home | : | 3/4/2020 1:27:03 PM |
| Singapore | : | 3/5/2020 2:27:03 AM |
| Seattle | : | 3/4/2020 10:27:03 AM |
| Stockholm | : | 3/4/2020 7:27:03 PM |
| IsDaylightSavings | : | False |

Get current time info formatted as a list.

EXAMPLE 4

```
PS C:\> $loc = [ordered]@{ "Hong Kong" = "China Standard Time"; Honolulu = "Hawaiian Standard Time"; Mumbai = "India Standard Time" }
```

```
PS C:\> Get-MyTimeInfo -Locations $loc -ft
```

Now: 03/04/2020 13:26:23
UTC: 03/04/2020 18:26:23

| Home | Hong Kong | Honolulu | Mumbai | IsDaylightSavings |
|---------------------|---------------------|---------------------|----------------------|-------------------|
| --- | --- | --- | --- | --- |
| 3/4/2020 1:26:23 PM | 3/5/2020 2:26:23 AM | 3/4/2020 8:26:23 AM | 3/4/2020 11:56:23 PM | False |

Using a custom location hashtable, get time zone information formatted as a table. This example is using the **-ft** alias for the **AsTable** parameter. Even though this is formatted as a table the actual output is a string.

EXAMPLE 5

```
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 |
|---------------------|----------------------|----------------------|---------------------|
| --- | --- | --- | --- |
| 3/4/2020 1:18:36 PM | 3/4/2020 12:18:36 PM | 3/4/2020 10:18:36 AM | 3/5/2020 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 PSDefaultParameterValue value for the HomeTimeZone parameter in your PowerShell profile.

Parameters

-Locations

Use an ordered hashtable of location names and timezones. You can find timezones with the Get-TimeZone cmdlet or through the .NET Framework with an expression like

```
[System.TimeZoneInfo]::GetSystemTimeZones()
```

The hashtable key should be the location or city name and the value should be the time zone ID. Be careful as it appears time zone IDs are case-sensitive.

The default value is:

```
[ordered]@{  
    Singapore = "Singapore Standard Time";  
    Seattle   = "Pacific Standard Time";  
    Stockholm = "Central Europe Standard Time";  
}
```

You might want to define a default value in \$PSDefaultParameterValues with your own defaults.

It is recommended you limit this hashtable to no more than 5 locations, especially if you want to format the results as a table.

Type: **OrderedDictionary**
Parameter Sets: **(All)**
Aliases:

Required: False
Position: 1
Default value: see note
Accept pipeline input: False
Accept wildcard characters: False

-HomeTimeZone

Specify the timezone ID of your home location. You might want to set this as a PSDefaultParameterValue

Type: **String**
Parameter Sets: **(All)**
Aliases:

Required: False
Position: Named
Default value: Eastern Standard Time
Accept pipeline input: False
Accept wildcard characters: False

-DateTime

Specify the datetime value to use. The default is now.

```
Type: DateTime
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: $(Get-Date)
Accept pipeline input: False
Accept wildcard characters: False
```

-AsTable

Display the results as a formatted table. This parameter has an alias of ft.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: ft

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-AsList

Display the results as a formatted list. This parameter has an alias of fl.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: fl

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

Datetime

Outputs

myTimeInfo

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-TimeZone](#)

Get-MyVariable

Synopsis

Get all user-defined variables.

Syntax

```
Get-MyVariable [[-Scope] <String>] [-NoTypeInformation] [<CommonParameters>]
```

Description

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. The command will also display the value type for each variable. If you want to suppress this output use the -NoTypeInformation switch.

This function is designed to work with the PowerShell console, NOT the PowerShell ISE.

Examples

EXAMPLE 1

```
PS C:\> Get-MyVariable

Name      Value          Type
----      ---
a        bits           ServiceController
dt       10/22/2020 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.

EXAMPLE 2

```
PS C:\> Get-MyVariable | Select-Object name,type

Name      Type
----      --
a        ServiceController
dt       DateTime
foo      Int32
r        Object[]
```

EXAMPLE 3

```
PS C:\> Get-MyVariable | Export-Clixml myvar.xml
PS C:\> import-clixml .\myvar.xml |
ForEach-Object {set-variable -Name $_.name -Value $_.value}
```

You can then import this XML file in another session to restore these variables.

EXAMPLE 4

```
PS C:\> function foo {
    c:\scripts\Get-MyVariable2.ps1;
    $a=4;$b=2;$c=$a*$b;
    Get-MyVariable -notypeinformation -scope 1 -verbose;
    $c
}

PS C:\> foo
VERBOSE: Getting system defined variables
VERBOSE: Found 49
VERBOSE: Getting current variables in 1 scope
VERBOSE: Found 27
VERBOSE: Filtering variables

Name          Value
----          -----
a              4
b              2
c              8
VERBOSE: Finished getting my variables
8
```

This sample function dot sources the script with this function. Within the function, Get-MyVariable is called specifying scope 1, or the parent scope. Scope 0 would be the scope of the Get-MyVariable function. Here's the result.

EXAMPLE 5

```
PS C:\> Get-MyVariable | where {$_._type -eq "Scriptblock"} |
Select-Object name,value

Name          Value
----          -----
bigp          ps | where {$_.ws -gt 100mb}
dirt          Param(\[string\]$Path=$env:temp) Get-C...
disk          Param (\[string\]$computername=$env:co...
run           gsv | where {$_.status -eq "running"}
up            Param(\[string\]$computername=$env:com...
```

Get all my variables that are scriptblocks.

Parameters

-Scope

The scope to query. The default is the Global scope but you can also specify Local, Script, Private or a number between 0 and 3 where 0 is the current scope, 1 is the parent scope, 2 is the grandparent scope, and so on.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 1
Default value: Global
Accept pipeline input: False
Accept wildcard characters: False
```

-NoTypeInformation

If specified, suppress the type information for each variable value.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

System.Management.Automation.PSVariable

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

An earlier version of this function is described at <http://jdhitsolutions.com/blog/2012/05/get-my-variable-revisited>

Related Links

[Get-Variable](#)

[About_Variables](#)

[About_Scope](#)

Get-ParameterInfo

Synopsis

Retrieve command parameter information.

Syntax

```
Get-ParameterInfo [-Command] <String> [-Parameter <String>]  
[<CommonParameters>]
```

Description

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. See examples.

Examples

EXAMPLE 1

```
PS C:\> Get-ParameterInfo Get-Service
```

ParameterSet: Default

```
Name : Name
Aliases : ServiceName
Mandatory : False
IsDynamic : False
Position : 0
Type : System.String[]
ValueFromPipeline : True
ValueFromPipelineByPropertyName : True
```

ParameterSet: __AllParameterSets

```
Name : ComputerName
Aliases : Cn
Mandatory : False
IsDynamic : False
Position : Named
Type : System.String[]
ValueFromPipeline : False
ValueFromPipelineByPropertyName : True

Name : DependentServices
Aliases : DS
Mandatory : False
IsDynamic : False
Position : Named
Type : System.Management.Automation.SwitchParameter
ValueFromPipeline : False
ValueFromPipelineByPropertyName : False
...
```

Return parameter information for Get-Service using the default list view.

EXAMPLE 2

```
PS C:\> Get-ParameterInfo mkdir |
Select-Object Name,Type,Position,parameterset
```

| Name | Type | Position | ParameterSet |
|----------------|--|----------|--------------------|
| Path | System.String[] | 0 | pathSet |
| Path | System.String[] | 0 | nameSet |
| Name | System.String | Named | nameSet |
| Value | System.Object | Named | __AllParameterSets |
| Force | System.Management.Automation.Switch... | Named | __AllParameterSets |
| Credential | System.Management.Automation.PSCred... | Named | __AllParameterSets |
| UseTransaction | System.Management.Automation.Switch... | Named | __AllParameterSets |

Get selected parameter information for the mkdir command.

EXAMPLE 3

```
PS C:\> Get-ParameterInfo Test-WSMan | Sort ParameterSet | Format-Table
```

ParameterSet: __AllParameterSets

| Name | Aliases | Mandatory | Position | Type |
|-----------------------|---------|-----------|----------|---------------------------------|
| <hr/> | | | | |
| CertificateThumbprint | | False | Named | System.String |
| Credential | cred,c | False | Named | System.Management.Automation... |
| ComputerName | cn | False | 0 | System.String |
| Authentication | auth,am | False | Named | Microsoft.WSMan.Management.... |

ParameterSet: ComputerName

| Name | Aliases | Mandatory | Position | Type |
|-----------------|---------|-----------|----------|--------------------------------------|
| <hr/> | | | | |
| UseSSL | | False | Named | System.Management.Automation.Swit... |
| Port | | False | Named | System.Int32 |
| ApplicationName | | False | Named | System.String |

Get all parameters from Test-WSMan and display details as a formatted table. The object type from Get-ParameterInfo has a default table view.

Example 4

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

ParameterSet: __AllParameterSets

| | | |
|---------------------------------|---|-----------------|
| Name | : | computername |
| Aliases | : | Cn |
| Mandatory | : | False |
| IsDynamic | : | False |
| Position | : | Named |
| Type | : | System.String[] |
| ValueFromPipeline | : | False |
| ValueFromPipelineByPropertyName | : | False |

Get details on the Computername parameter of the Get-Counter cmdlet.

Parameters

-Command

The name of a cmdlet or function. The parameter has an alias of Name.

```
Type: String
Parameter Sets: (All)
Aliases: name

Required: True
Position: 1
Default value: None
Accept pipeline input: True (ByPropertyName, ByValue)
Accept wildcard characters: False
```

-Parameter

{{Fill Parameter Description}}

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String

Outputs

PSPrinterInfo

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Get-Command

Get-PathVariable

Synopsis

Get information from locations in %PATH%.

Syntax

```
Get-PathVariable [[-Scope] <String>] [<CommonParameters>]
```

Description

Use this command to test the locations specified in the %PATH% environment variable. On Windows platforms, you can distinguish between settings set per machine and those set per user. On non-Windows platforms, the scope will be Process.

Examples

Example 1

```
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
...
Machine Jeff     C:\WINDOWS                         True
Machine Jeff     C:\WINDOWS\system32                True
Machine Jeff     C:\WINDOWS\System32\Wbem           True
...
```

Example 2

```
PS /home/jeff> Get-PathVariable | Where-Object {-Not $_.exists}

Scope      : Process
Computername : Bovine320
UserName   : jeff
Path       : /snap/bin
Exists     : False
```

This example is on a Linux platform, finding locations that don't exist or can be verified. You could run the same command on Windows.

Parameters

-Scope

On Windows platforms you can distinguish between Machine and User specific settings.

```
Type: String
Parameter Sets: (All)
Aliases:
Accepted values: All, User, Machine

Required: False
Position: 0
Default value: All
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

EnvPath

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Get-PowerShellEngine

Synopsis

Get the path to the current PowerShell engine.

Syntax

```
Get-PowerShellEngine [-Detail]
```

Description

Use this command to find the path to the PowerShell executable, or engine that is running your current session. The default is to provide the path only. But you can also get detailed information

Examples

EXAMPLE 1

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

EXAMPLE 2

```
PS C:\> Get-PowerShellEngine -Detail
```

| | | |
|-----------------------|---|---|
| Path | : | C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe |
| FileVersion | : | 10.0.15063.0 (WinBuild.160101.0800) |
| PSVersion | : | 5.1.15063.502 |
| ProductVersion | : | 10.0.15063.0 |
| Edition | : | Desktop |
| Host | : | Visual Studio Code Host |
| Culture | : | en-US |
| Platform | : | |

This result is from running in the Visual Studio Code integrated PowerShell terminal.

EXAMPLE 3

```
PS C:\> Get-PowerShellEngine -Detail

Path          : C:\Program Files\PowerShell\7\pwsh.exe
FileVersion   : 7.1.0.0
PSVersion     : 7.1.0
ProductVersion: 7.1.0 SHA: d2953dcaf8323b95371380639ced00dac4ed209f
Edition       : Core
Host          : ConsoleHost
Culture       : en-US
Platform      : Win32NT
```

This result is from running in a PowerShell 7 session on Windows 10

Parameters

-Detail

Include additional information. Not all properties may have values depending on operating system and PowerShell version.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

Inputs

Outputs

System.String

PSCustomObject

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[\\$PSVersionTable](#)

[\\$Host](#)

Get-Process

Get-PSAnsifileMap

Synopsis

Display the PSAnsifileMap

Syntax

```
Get-PSAnsifileMap [CommonParameters]
```

Description

Use this command to display the PSAnsifileMap global variable. The Ansi pattern will be shown using the pattern.

Examples

Example 1

```
PS C:\> Get-PSAnsifileMap

Description  Pattern          ANSI
-----
PowerShell   \.((ps(d|m)?1)|(ps1xml))$  \e[38;2;252;127;12m` e[38;2;252;127;12m
Text         \.((txt)|(log)|(htm(1)?))$  \e[38;2;58;120;255m` e[38;2;58;120;255m
...
...
```

The output will display the ANSI sequence using the sequence itself. The escape character will be based on the version of PowerShell you are using.

Parameters

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

PSAnsiFileEntry

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Set-PSAnsiFileMap](#)

Get-PSLocation

Synopsis

Get common location values.

Syntax

```
Get-PSLocation [<CommonParameters>]
```

Description

This command will write an object to the pipeline that displays the values of common file locations. You might find this helpful when scripting cross-platform.

Examples

EXAMPLE 1

```
PS C:\> Get-PSLocation

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

Results on a Windows system.

EXAMPLE 2

```
PS C:\> Get-PSLocation

Temp      : /tmp/
Home      : /home/jeff
Desktop   :
PowerShell : /home/jeff/.config/powershell
```

Results on a Linux system running PowerShell.

Parameters

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Location](#)

[Set-Location](#)

Get-PSProfile

Synopsis

Get PowerShell profile locations

Syntax

```
Get-PSProfile [<CommonParameters>]
```

Description

This command is designed for Windows-based systems to show all possible PowerShell profile scripts. Including those for VS Code and the PowerShell ISE.

Examples

Example 1

| PS C:\> Get-PSProfile | | |
|--------------------------|---|--------|
| Name: PowerShell | | |
| Scope | Path | Exists |
| AllUsersCurrentHost | C:\Program Files\PowerShell\7\Microsoft.PowerShell_profile.ps1 | False |
| AllUsersAllHosts | C:\Program Files\PowerShell\7\profile.ps1 | False |
| CurrentUserAllHosts | C:\Users\Jeff\Documents\PowerShell\profile.ps1 | True |
| CurrentUserCurrentHost | C:\Users\Jeff\Documents\PowerShell\Microsoft.PowerShell_profile.ps1 | True |
| Name: Windows PowerShell | | |
| Scope | Path | Exists |
| AllUsersCurrentHost | C:\WINDOWS\System32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1 | True |
| AllUsersAllHosts | C:\WINDOWS\System32\WindowsPowerShell\v1.0\profile.ps1 | True |
| CurrentUserAllHosts | C:\Users\Jeff\Documents\WindowsPowerShell\profile.ps1 | True |
| CurrentUserCurrentHost | C:\Users\Jeff\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1 | True |
| Name: VSCode PowerShell | | |
| Scope | Path | Exists |
| CurrentUserCurrentHost | C:\Users\Jeff\Documents\PowerShell\Microsoft.VSCode_profile.ps1 | True |
| AllUsersCurrentHost | C:\Program Files\PowerShell\7\Microsoft.VSCode_profile.ps1 | False |
| ... | | |

The command has a default formatted table view.

Example 2

```
PS C:\> Get-PSProfile | Where-Object Exists | Format-List

Name: PowerShell

Scope      : CurrentUserAllHosts
Path       : C:\Users\Jeff\Documents\PowerShell\profile.ps1
Exists     : True
LastModified : 9/9/2020 2:35:45 PM

Scope      : CurrentUserCurrentHost
Path       : C:\Users\Jeff\Documents\PowerShell\Microsoft.PowerShell_profile.ps1
Exists     : True
LastModified : 9/9/2020 2:03:44 PM
```

Name: Windows PowerShell

```
Scope      : AllUsersCurrentHost
Path       : C:\WINDOWS\System32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1
Exists     : True
LastModified : 10/9/2020 4:08:35 PM
...
```

The command has a default list view.

Parameters

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

PSProfilePath

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Get-PSScriptTools

Synopsis

Get a summary of PSScriptTools commands.

Syntax

```
Get-PSScriptTools [-Verb <String>] [<CommonParameters>]
```

Description

You can use this command to get a summary display of functions included in the PSScriptTools module. Use the -Verb parameter to filter the output.

Examples

Example 1

```
PS C:\> Get-PSScriptTools

Verb: Add
Name          Alias          Synopsis
----          ----          -----
Add-Border               Create a text border around a string.

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 struct...
Convert-HashtableString
Convert-HashtableToCode
...
Convert a PowerShell expression i...
Convert a hashtable string into a...
Convert a hashtable to a string r...
```

Example 2

```
PS C:\> Get-PSScriptTools | Where alias
```

Verb: Compare

| Name | Alias | Synopsis |
|----------------|-------|-------------------------------------|
| Compare-Module | cmo | Compare PowerShell module versions. |
| Compare-Script | csc | Compare PowerShell script versions. |
| ... | | |

Verb: Convert

| Name | Alias | Synopsis |
|-------------------------|-------|--|
| Convert-EventLogRecord | clr | Convert EventLogRecords to structured... |
| Convert-HashtableToCode | chc | Convert a hashtable to a string repre... |
| ... | | |

List commands with defined aliases in the PSScriptTools module.

Example 3

```
PS C:\> Get-PSScriptTools -Verb Select
```

Verb:Select

| Name | Alias | Synopsis |
|---------------|--------|--------------------------------|
| Select-After | after | Select objects after a give... |
| Select-Before | before | Select objects before a giv... |
| Select-First | First | Select the first X number o... |
| Select-Last | Last | Select the last X number of... |
| Select-Newest | newest | Select the newest X number ... |
| Select-Oldest | oldest | Select the oldest X number ... |

Get all module commands that use the Select verb.

Parameters

-Verb

Filter commands based on a standard PowerShell verb.

Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

PSScriptTool

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Command](#)

[Get-Module](#)

[Open-PSScriptToolsHelp](#)

Get-PSWho

Synopsis

Get PowerShell user summary information.

Syntax

```
Get-PSWho [-AsString] [<CommonParameters>]
```

Description

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.

The default behavior is to write an object to the pipeline, but you can use the -AsString parameter to force the command to write a string. This makes it easier to use in your scripts with Write-Verbose.

Examples

EXAMPLE 1

```
PS C:\> Get-PSWho

User          : WIN10\Jeff
Elevated      : True
Computername  : WIN10
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)
```

EXAMPLE 2

```
PS /home/jhicks> Get-PSWho

User          : jeff
Elevated      : NA
Computername  : WIN10
OperatingSystem : Linux 4.4.0-18362-Microsoft #476-Microsoft Fri Nov 01 16:53:00 PST 2019
OSVersion     : Ubuntu 18.04.3 LTS
PSVersion     : 7.0.0-rc.2
Edition       : Core
PSHost        : ConsoleHost
WSMan         : 3.0
ExecutionPolicy : Unrestricted
Culture       : Invariant Language (Invariant Country)
```

EXAMPLE 3

```
PS C:\> Get-PSWho

User          : WIN10\Jeff
Elevated      : True
Computername  : WIN10
OperatingSystem : Microsoft Windows 10 Pro [64-bit]
OSVersion     : 10.0.19042
PSVersion     : 7.1.0
Edition       : Core
PSHost        : ConsoleHost
WSMan         : 3.0
ExecutionPolicy : RemoteSigned
Culture       : English (United States)
```

EXAMPLE 4

```
PS C:\> Get-PSWho -asString | Set-Content c:\test\who.txt
```

Parameters

-AsString

Write the summary object as a string. This can be useful when you want to save the information in a log file.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

PSWho

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-CimInstance](#)

[Get-ExecutionPolicy](#)

[\\$PSVersionTable](#)

[\\$Host](#)

Get-TZData

Synopsis

Get time zone details.

Syntax

```
Get-TZData [-TimeZoneArea] <String> [-Raw] [<CommonParameters>]
```

Description

This command uses a free and publicly available REST API offered by <http://worldtimeapi.org> to get information about a time zone. You can use Get-TZList to find an area and this command to display the details. The time zone area name is case-sensitive. The default is to write a custom object to the pipeline, but you also have an option of seeing the raw data that is returned from the API. On PowerShell Core, the raw data will be slightly different.

Note that if the site is busy you may get an error. If that happens, wait a minute and try again.

Examples

Example 1

```
PS C:\> Get-TZData Australia/Hobart
PS C:\> Get-TZData Australia/Hobart
Timezone          Label      Offset    DST        Time
-----          -----      -----    --  -----  -----
Australia/Hobart AEDT    11:00:00  True  3/16/2020 5:35:46 AM
```

Get time zone information for Hobart.

Example 2

```
PS C:\> Get-TZData Asia/Tokyo -Raw
```

```
week_number : 11
utc_offset   : +09:00
unixtime     : 1552674997
timezone     : Asia/Tokyo
dst_until    :
dst_from     :
dst          : False
day_of_year  : 75
day_of_week  : 6
datetime     : 2020-03-16T03:36:37.829505+09:00
abbreviation : JST
```

Get time zone information for Tokyo as a raw format.

Example 3

```
PS C:\> Get-TZList Antarctica | Get-TZData | Sort-Object Offset
```

| Timezone | Label | Offset | DST | Time |
|---------------------------|-------|-----------|-------|-----------------------|
| Antarctica/Rothera | -03 | -03:00:00 | False | 3/15/2020 3:39:59 PM |
| Antarctica/Palmer | -03 | -03:00:00 | False | 3/15/2020 3:39:59 PM |
| Antarctica/Troll | +00 | 00:00:00 | False | 3/15/2020 6:40:00 PM |
| Antarctica/Syowa | +03 | 03:00:00 | False | 3/15/2020 9:39:59 PM |
| Antarctica/Mawson | +05 | 05:00:00 | False | 3/15/2020 11:39:59 PM |
| Antarctica/Vostok | +06 | 06:00:00 | False | 3/16/2020 12:40:00 AM |
| Antarctica/Davis | +07 | 07:00:00 | False | 3/16/2020 1:39:58 AM |
| Antarctica/Casey | +08 | 08:00:00 | False | 3/16/2020 2:39:58 AM |
| Antarctica/DumontDUrville | +10 | 10:00:00 | False | 3/16/2020 4:39:58 AM |
| Antarctica/Macquarie | +11 | 11:00:00 | False | 3/16/2020 5:39:58 AM |

Get all time zone areas in Antarctica and pipe them to Get-TZData to retrieve the details.

Example 4

```
PS C:\> Get-TZData Europe/Rome | ConvertTo-LocalTime -Datetime "3/15/2020 4:00PM"
```

```
Friday, March 15, 2020 11:00:00 AM
```

Convert the datetime in Rome to local time, which in this example is Eastern time.

Parameters

-Raw

Return raw, unformatted data. Due to the way PowerShell Core automatically wants to format date time strings, raw output had to be slightly adjusted.

Type: **SwitchParameter**

Parameter Sets: (**All**)

Aliases:

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

-TimeZoneArea

Enter a timezone location like Pacific/Auckland. It is case sensitive. Use Get-TZList to retrieve a list of areas.

Type: **String**

Parameter Sets: (**All**)

Aliases:

Required: True

Position: 0

Default value: None

Accept pipeline input: True (**ByValue**)

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

PSCustomObject

TimeZoneData

Notes

Learn more about PowerShell:<http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-TZList](#)

Get-TZList

Synopsis

Get a list of time zone areas.

Syntax

zone (Default)

```
Get-TZList [-TimeZoneArea] <String> [<CommonParameters>]
```

all

```
Get-TZList [-All] [<CommonParameters>]
```

Description

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. You must have Internet access for this command to work. Note that if the site is busy you may get an error. If that happens, wait a minute and try again.

Examples

Example 1

```
PS C:\> Get-TZList -all
```

```
Africa/Abidjan
Africa/Accra
Africa/Algiers
Africa/Bissau
Africa/Cairo
...
...
```

Get a list of all time zone areas.

Example 2

```
PS C:\> Get-TZList Atlantic
```

Atlantic/Azores
Atlantic/Bermuda
Atlantic/Canary
Atlantic/Cape_Verde
Atlantic/Faroe
Atlantic/Madeira
Atlantic/Reykjavik
Atlantic/South_Georgia
Atlantic/Stanley

Get all time zone areas in the Atlantic region.

Parameters

-All

Get a list of all timezone areas

```
Type: SwitchParameter
Parameter Sets: all
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-TimeZoneArea

Specify a time zone region.

```
Type: String
Parameter Sets: zone
Aliases:
Accepted values: Africa, America, Antarctica, Asia, Atlantic, Australia, Europe, Indian, Pacific

Required: True
Position: 0
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

string

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-TZData](#)

Get-WhoIs

Synopsis

Lookup WhoIS data for a given IPv4 address.

Syntax

```
Get-WhoIs [-IPAddress] <String> [<CommonParameters>]
```

Description

This command queries the ARIN database to lookup WhoIs information for a given IPv4 address.

Examples

Example 1

```
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 |

Example 2

```
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 |

Parameters

-IPAddress

Enter a valid IPV4 address to lookup with WhoIs. It is assumed all of the octets are less than 254.

Type: **String**
Parameter Sets: (**All**)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: True (**ByPropertyName**, **ByValue**)
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String

Outputs

WhoIsResult

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Invoke-RestMethod](#)

Get-WindowsVersion

Synopsis

Get Windows version information.

Syntax

```
Get-WindowsVersion [[-Computername] <String[]>] [-Credential <PSCredential>]
[-UseSSL] [-ThrottleLimit <Int32>] [-Authentication <String>] [<CommonParameters>]
```

Description

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. The parameters are the same as in Invoke-Command.

If you are querying the local computer, all other parameters will be ignored.

Examples

EXAMPLE 1

```
PS C:\>Get-WindowsVersion

Computername: DESK109

ProductName      EditionID      ReleaseID  Build  InstalledUTC
-----          -----          -----      -----  -----
Windows 10 Pro    Professional   2009       19042  10/16/2020 3:09:01 PM
```

Query the local host.

EXAMPLE 2

```
PS C:\> Get-WindowsVersion -Computername srv1,srv2,win10 -Credential $art
```

Computername: WIN10

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

Computername: SRV1

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

Computername: SRV2

| ProductName | EditionID | ReleaseID | Build | InstalledUTC |
|------------------------------|----------------|-----------|-------|---------------------|
| Windows Server 2016 Standard | ServerStandard | 1607 | 14393 | 2/6/2020 6:47:12 PM |

Get Windows version information from remote computers using an alternate credential.

Example 3

```
PS C:\> Get-WindowsVersion -Computername Dom1 | Select-Object *
```

| | |
|---------------------|---|
| ProductName | : Windows Server 2016 Standard Evaluation |
| EditionID | : ServerStandardEval |
| ReleaseID | : 1607 |
| Build | : 14393.3474 |
| Branch | : rs1_release |
| InstalledUTC | : 2/6/2020 5:18:50 PM |
| Computername | : DOM1 |

Parameters

-Computername

Specifies the computers on which the command runs. The default is the local computer.

When you use the ComputerName parameter, Windows PowerShell creates a temporary connection that is used only to run the specified command and is then closed. If you need a persistent connection, use the Session parameter.

Type the NETBIOS name, IP address, or fully qualified domain name of one or more computers in a comma-separated list. To specify the local computer, type the computer name, localhost, or a dot (.).

To use an IP address in the value of ComputerName , the command must include the Credential parameter. Also, the computer must be configured for HTTPS transport or the IP address of the remote computer must be included in the WinRM TrustedHosts list on the local computer. For instructions for adding a computer name to

the TrustedHosts list, see "How to Add a Computer to the Trusted Host List" in about_Remote_Troubleshooting.

On Windows Vista and later versions of the Windows operating system, to include the local computer in the value of ComputerName , you must open Windows PowerShell by using the Run as administrator option.

```
Type: String[]
Parameter Sets: (All)
Aliases:

Required: False
Position: 1
Default value: $env:COMPUTERNAME
Accept pipeline input: True (ByPropertyName, ByValue)
Accept wildcard characters: False
```

-Credential

Specifies a user account that has permission to perform this action. The default is the current user.

Type a user name, such as User01 or Domain01\User01. Or, enter a PSCredential object, such as one generated by the Get-Credential cmdlet. If you type a user name, this cmdlet prompts you for a password.

```
Type: PSCredential
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-UseSSL

Indicates that this cmdlet uses the Secure Sockets Layer (SSL) protocol to establish a connection to the remote computer. By default, SSL is not used.

WS-Management encrypts all Windows PowerShell content transmitted over the network. The UseSSL parameter is an additional protection that sends the data across an HTTPS, instead of HTTP.

If you use this parameter, but SSL is not available on the port that is used for the command, the command fails.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-ThrottleLimit

Specifies the maximum number of concurrent connections that can be established to run this command. If you omit this parameter or enter a value of 0, the default value, 32, is used.

The throttle limit applies only to the current command, not to the session or to the computer.

Type: **Int32**

Parameter Sets: (**A11**)

Aliases:

Required: False

Position: Named

Default value: 0

Accept pipeline input: False

Accept wildcard characters: False

-Authentication

Specifies the mechanism that is used to authenticate the user's credentials. The acceptable values for this parameter are:

- Default
- Basic
- Credssp
- Digest
- Kerberos
- Negotiate
- NegotiateWithImplicitCredential

The default value is Default.

CredSSP authentication is available only in Windows Vista, Windows Server 2008, and later versions of the Windows operating system.

For information about the values of this parameter, see the description of the AuthenticationMechanismEnumeration (<http://go.microsoft.com/fwlink/?LinkID=144382>) in the Microsoft Developer Network (MSDN) library.



Credential Security Support Provider (CredSSP) authentication, in which the user's credentials are passed to a remote computer to be authenticated, is designed for commands that require authentication on more than one resource, such as accessing a remote network share. This mechanism increases the security risk of the remote operation. If the remote computer is compromised, the credentials that are passed to it can be used to control the network session.

Type: **String**

Parameter Sets: (**All**)

Aliases:

Required: False

Position: Named

Default value: Default

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

WindowsVersion

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-WindowsVersionString](#)

[WinVer.exe](#)

[Invoke-Command](#)

Get-WindowsVersionString

Synopsis

Get Windows version information.

Syntax

```
Get-WindowsVersionString [[-Computername] <String[]>]
[-Credential <PSCredential>] [-UseSSL] [-ThrottleLimit <Int32>]
[-Authentication <String>] [<CommonParameters>]
```

Description

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. The parameters are the same as in Invoke-Command. The command writes a string of version information.

If you are querying the local computer, all other parameters will be ignored.

Examples

EXAMPLE 1

```
PS C:\> Get-WindowsVersionString -Computername win10 -credential company\artd
WIN10 Windows 10 Enterprise (OS Build 15063.1418)
```

Get a string version of Windows version information from a remote computer and use an alternate credential.

EXAMPLE 2

```
PS C:\> Get-WindowsVersionString
BOVINE320 Windows 10 Pro Version Professional (OS Build 17763.253)
```

Get version information for the local host.

Parameters

-Computername

Specifies the computers on which the command runs. The default is the local computer.

When you use the ComputerName parameter, Windows PowerShell creates a temporary connection that is used only to run the specified command and is then closed. If you need a persistent connection, use the Session parameter.

Type the NETBIOS name, IP address, or fully qualified domain name of one or more computers in a comma-separated list. To specify the local computer, type the computer name, localhost, or a dot (.).

To use an IP address in the value of ComputerName , the command must include the Credential parameter. Also, the computer must be configured for HTTPS transport or the IP address of the remote computer must be included in the WinRM TrustedHosts list on the local computer. For instructions for adding a computer name to the TrustedHosts list, see "How to Add a Computer to the Trusted Host List" in about_Remote_Troubleshooting.

On Windows Vista and later versions of the Windows operating system, to include the local computer in the value of ComputerName , you must open Windows PowerShell by using the Run as administrator option.

```
Type: String[]
Parameter Sets: (All)
Aliases:

Required: False
Position: 1
Default value: $env:COMPUTERNAME
Accept pipeline input: True (ByPropertyName, ByValue)
Accept wildcard characters: False
```

-Credential

Specifies a user account that has permission to perform this action. The default is the current user.

Type a user name, such as User01 or Domain01\User01. Or, enter a PSCredential object, such as one generated by the Get-Credential cmdlet. If you type a user name, this cmdlet prompts you for a password.

```
Type: PSCredential
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-UseSSL

Indicates that this cmdlet uses the Secure Sockets Layer (SSL) protocol to establish a connection to the remote computer. By default, SSL is not used.

WS-Management encrypts all Windows PowerShell content transmitted over the network. The UseSSL parameter is an additional protection that sends the data across an HTTPS, instead of HTTP.

If you use this parameter, but SSL is not available on the port that is used for the command, the command fails.

Type: `SwitchParameter`**Parameter Sets:** ([All](#))**Aliases:****Required:** False**Position:** Named**Default value:** False**Accept pipeline input:** False**Accept wildcard characters:** False

-ThrottleLimit

Specifies the maximum number of concurrent connections that can be established to run this command. If you omit this parameter or enter a value of 0, the default value, 32, is used.

The throttle limit applies only to the current command, not to the session or to the computer.

Type: `Int32`**Parameter Sets:** ([All](#))**Aliases:****Required:** False**Position:** Named**Default value:** 0**Accept pipeline input:** False**Accept wildcard characters:** False

-Authentication

Specifies the mechanism that is used to authenticate the user's credentials. The acceptable values for this parameter are:

- Default
- Basic
- Credssp
- Digest
- Kerberos
- Negotiate
- NegotiateWithImplicitCredential

The default value is Default.

CredSSP authentication is available only in Windows Vista, Windows Server 2008, and later versions of the Windows operating system.

For information about the values of this parameter, see the description of the `AuthenticationMechanismEnumeration` (<http://go.microsoft.com/fwlink/?LinkID=144382>) in the Microsoft Developer Network (MSDN) library.



Credential Security Support Provider (CredSSP) authentication, in which the user's credentials are passed to a remote computer to be authenticated, is designed for commands that require authentication on more than one resource, such as accessing a remote network share. This mechanism increases the security risk of the remote operation. If the remote computer is compromised, the credentials that are passed to it can be used to control the network session.

Type: `String`

Parameter Sets: ([All](#))

Aliases:

Required: False

Position: Named

Default value: Default

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#) (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-WindowsVersion](#)

Winver.exe

Invoke-InputBox

Synopsis

Launch a graphical input box.

Syntax

plain (Default)

```
Invoke-InputBox [-Title <String>] [-Prompt <String>]  
[-BackgroundColor <String>] [<CommonParameters>]
```

secure

```
Invoke-InputBox [-Title <String>] [-Prompt <String>] [-AsSecureString]  
[-BackgroundColor <String>] [<CommonParameters>]
```

Description

Use this command as a graphical replacement for Read-Host. The command will write either a string or a secure string to the pipeline. You can customize the prompt, title and background color.

This command requires a Windows platform.

Examples

EXAMPLE 1

```
PS C:\> $name == Invoke-InputBox -prompt "Enter a user name" -title "New User"
```

Display an graphical inputbox with a given prompt and title. The entered value will be saved to \$name.

EXAMPLE 2

```
PS C:\> $pass == Invoke-InputBox -prompt "Enter a new password"  
-title "New User" -asSecureString -background red
```

Get a secure string value from the user. This example also changes the form background to red.

Parameters

-AsSecureString

Use to mask the entry and return a secure string.

```
Type: SwitchParameter
Parameter Sets: secure
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-BackgroundColor

Set the form background color. You can use a value like 'red' or a '#c0c0c0'.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: White
Accept pipeline input: False
Accept wildcard characters: False
```

-Prompt

Enter a prompt. No more than 50 characters.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: "Please enter a value"
Accept pipeline input: False
Accept wildcard characters: False
```

-Title

Enter the title for the input box. No more than 25 characters.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: "User Input"
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.String

System.Security.SecureString

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Read-Host](#)

[New-WPFMessageBox](#)

Join-Hashtable

Synopsis

Combine two hashtables into one.

Syntax

```
Join-Hashtable [[-First] <Hashtable>] [[-Second] <Hashtable>] [-Force]
[<CommonParameters>]
```

Description

This command will combine two hashtables into a single hashtable. Normally this is as easy as \$hash1+\$hash2. But if there are duplicate keys, this will fail. 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.

The original hashtables will not be modified.

Examples

EXAMPLE 1

```
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 |

EXAMPLE 2

```
PS C:\>$c == Join-Hashtable $a $b -force  
PS C:\> $c
```

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

Parameters

-First

The primary hashtable. If there are any duplicate keys and you use -Force, values from this hashtable will be kept.

Type: **Hashtable**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: 1
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Second

The secondary hashtable.

Type: **Hashtable**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: 2
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Force

Do not prompt for conflicts. Always keep the key from the first hashtable.

Type: **SwitchParameter**

Parameter Sets: (**All**)

Aliases:

Required: False

Position: Named

Default value: False

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

hashtable

Outputs

hashtable

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[About_Hash_Tables](#)

New-ANSIBar

Synopsis

Display an ANSI colored bar.

Syntax

standard (Default)

```
New-ANSIBar -Range <Int32[]> [-Spacing <Int32>] [-Character <String>]  
[-Gradient] [<CommonParameters>]
```

custom

```
New-ANSIBar -Range <Int32[]> [-Spacing <Int32>] [-Custom <Char>] [-Gradient]  
[<CommonParameters>]
```

Description

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.

You can learn more about ANSI escape codes at https://en.wikipedia.org/wiki/ANSI_escape_code.

Examples

Example 1

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

This will create a grayscale gradient bar that goes from dark to light to dark.

Example 2

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

Example 3

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

Parameters

-Character

Specify a character to use for the bar.

```
Type: String  
Parameter Sets: standard  
Aliases:  
Accepted values: FullBlock, LightShade, MediumShade, DarkShade, BlackSquare, WhiteSquare  
  
Required: False  
Position: Named  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-Custom

Specify a custom character.

```
Type: Char  
Parameter Sets: custom  
Aliases:  
  
Required: False  
Position: Named  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-Gradient

Display as a single gradient from the first value to the last.

```
Type: SwitchParameter  
Parameter Sets: (All)  
Aliases:  
  
Required: False  
Position: Named  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-Range

Enter a range of 256 color values, e.g. (232..255)

Type: `Int32[]`
Parameter Sets: ([All](#))
Aliases:

`Required:` True
`Position:` Named
`Default value:` None
`Accept pipeline input:` False
`Accept wildcard characters:` False

-Spacing

How many characters do you want in the bar of each value? This will increase the overall length of the bar.

Type: `Int32`
Parameter Sets: ([All](#))
Aliases:

`Required:` False
`Position:` Named
`Default value:` None
`Accept pipeline input:` False
`Accept wildcard characters:` False

CommonParameters

This cmdlet supports the common parameters: `-Debug`, `-ErrorAction`, `-ErrorVariable`, `-InformationAction`, `-InformationVariable`, `-OutVariable`, `-OutBuffer`, `-PipelineVariable`, `-Verbose`, `-WarningAction`, and `-WarningVariable`. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[New-RedGreenGradient](#)

[Write-ANSIProgress](#)

New-CustomFileName

Synopsis

Create a custom file name based on a template.

Syntax

```
New-CustomFileName [-Template] <String> [-Case <String>] [<CommonParameters>]
```

Description

This command will generate a custom file name based on a template string that you provide. 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 - the hour of the day in 12-hour format to 2 digits
- %hour24 - the hour of the day in 24-hour format to 2 digits
- %minute
- %seconds
- %time - A compact string of HourMinuteSecond
- %string - A random string
- %guid

You can also insert a random number using %# with a # character for each digit. If you want a 2 digit random number use %. If you want 6 digits, use %.

The command will attempt to preserve case for any non-pattern string, but you should separate it from other placeholder patterns with one of these characters: - () [] or a . Using an underscore will not work.

Another option, is to turn the entire custom name into upper or lower case.

Examples

EXAMPLE 1

```
PS C:\> New-CustomFileName %computername_%day%monthname%yr-%time.log  
COWPC_28Nov20-142138.log
```

EXAMPLE 2

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

Create a custom file name using the day of the week and a 4 digit random number.

EXAMPLE 3

```
PS C:\> New-CustomFileName %username-%string.tmp -Case Upper  
JEFF-Z0XUXMFS.TMP
```

Create an upper case custom file name. The %string placeholder will be replaced with a random 8 character string.

EXAMPLE 4

```
PS C:\> Join-Path c:\work (New-CustomFilename "%Year%Monthname-LOG-%computername[%username].txt" -case lower)  
c:\work\2020nov-log-bovine320[jeff].txt
```

Create a lower case filename using Join-Path. This command does not create the file, it only generates a name for you to use.

EXAMPLE 5

```
PS C:\> 1..10 | foreach-object {
    $file = New-Item (Join-Path c:\work\data (New-CustomFileName %string-%####.dat))
    $stream = $file.open("OpenOrCreate")
    $stream.Seek((Get-Random -minimum 250 -Maximum 2KB), "Begin") | Out-Null
    $stream.WriteByte(0)
    $stream.Close()
    $file
}
```

Directory: C:\work\data

| Mode | LastWriteTime | Length | Name |
|--------|-------------------|--------|--------------------|
| -a---- | 3/15/2020 4:46 PM | 976 | rcphz2nj-6431.dat |
| -a---- | 3/15/2020 4:46 PM | 1797 | viz32er5-0526.dat |
| -a---- | 3/15/2020 4:46 PM | 1775 | k2mukuv4-8267.dat |
| -a---- | 3/15/2020 4:46 PM | 666 | 0encqdlt-8753.dat |
| -a---- | 3/15/2020 4:46 PM | 513 | dbswpujf-6314.dat |
| -a---- | 3/15/2020 4:46 PM | 371 | qlkdufp0-0481.dat |
| -a---- | 3/15/2020 4:46 PM | 2010 | 5cxq3tb5-5624.dat |
| -a---- | 3/15/2020 4:46 PM | 2043 | mcvoh4n5-8041.dat |
| -a---- | 3/15/2020 4:46 PM | 1048 | 4iwibnmf-1584.dat |
| -a---- | 3/15/2020 4:46 PM | 378 | fgsj0rttd-2894.dat |

Create 10 dummy files with random names and sizes.

Parameters

-Case

Some values like username or computername might be in a different case than what you want. You can use the default value, or return a value that is all upper or lower case.

Type: String
 Parameter Sets: (All)
Aliases:
 Accepted values: Lower, Upper, Default

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Template

A string that defines the naming pattern based on a set of placeholders. You can create a template string using any of these variables, including the % symbol.

- %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 - the hour of the day in 12-hour format to 2 digits
- %hour24 - the hour of the day in 24-hour format to 2 digits
- %minute
- %seconds
- %time - A compact string of HourMinuteSecond
- %string - A random string
- %guid
- %# - a random number matching the number of # characters

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[New-RandomFileName](#)

New-PSDriveHere

Synopsis

Create a new PSDrive at the current location.

Syntax

Folder (Default)

```
New-PSDriveHere [[-Path] <String>] [-First] [-SetLocation] [-Passthru]
[-WhatIf] [-Confirm] [<CommonParameters>]
```

Name

```
New-PSDriveHere [[-Path] <String>] [[-Name] <String>] [-SetLocation]
[-Passthru] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

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. If you prefer to use the first word of the location, use -First. If you prefer to specify a completely different name, then use the -Name parameter.

This command will not write anything to the pipeline unless you use -Passthru.

Examples

EXAMPLE 1

```
PS C:\users\jeff\documents\Enterprise Mgmt Webinar\> New-PSDriveHere
```

This will create a new PSDrive called Webinar rooted to the current location.

EXAMPLE 2

```
PS C:\users\jeff\documents\Enterprise Mgmt Webinar\> New-PSDriveHere -first
```

This will create a new PSDrive called Enterprise rooted to the current location.

EXAMPLE 3

```
PS C:\> New-PSDriveHere HKLM:\software\microsoft -passthru |  
Select-Object -ExpandProperty Name  
  
microsoft
```

EXAMPLE 4

```
PS C:\> New-PSDriveHere -Path "\\\NAS\files\powershell" -Name PSFiles
```

Create a new PSDrive called PSFiles rooted to the specified path.

EXAMPLE 5

```
PS C:\Users\Jeff\Documents\DeepDive\> New-PSDriveHere . DeepDive -setlocation  
PS DeepDive:\>
```

Create a new PSDrive and change location to it.

Parameters

-Path

The path for the new PSDrive. The default is the current location.

Type: **String**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: 1
Default value: .
Accept pipeline input: False
Accept wildcard characters: False

-Name

The name for the new PSDrive. The default is the last word in the specified location, unless you use -First.

Type: **String**
Parameter Sets: Name
Aliases:

Required: False
Position: 2
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-First

Use the first word of the current location for the new PSDrive.

Type: **SwitchParameter**
Parameter Sets: Folder
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-SetLocation

Set location to this new drive. This parameter has an alias of CD.

Type: **SwitchParameter**
Parameter Sets: **(All)**
Aliases: cd

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

Type: **SwitchParameter**
Parameter Sets: **(All)**
Aliases: wi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Confirm

Prompts you for confirmation before running the cmdlet.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Passthru

Pass the new PSDrive object to the pipeline.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

None

System.Management.Automation.PSDrive

Notes

Originally published at <http://jdhitsolutions.com/blog/2010/08/New-PSDriveHere/>

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-PSDrive](#)

[New-PSDrive](#)

New-PSFormatXML

Synopsis

Create or modify a format.ps1xml file.

Syntax

```
New-PSFormatXML [-InputObject] <Object> [[-Properties] <Object[]>]
[-Typename <String>] [[-FormatType] <String>] [[-ViewName] <String>]
[-Path] <String> [-GroupBy <String>] [-Wrap] [-Append]
[-Passthru] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

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, such as a table or list. Or you may want to have different views that display the object differently. Format directives are stored in format.ps1xml files which can be tedious to create. This command simplifies that process.

Note that the table and wide views are set to Autosize. However, the table definition will include best guesses for column widths. If you prefer a more granular approach you can delete the Autosize tag and experiment with varying widths. Don't forget to run Update-FormatData to load your new file. You may need to start a new PowerShell session to fully test changes.

Pipe an instance of your custom object to this function and it will generate a format.ps1xml file based on either all the properties or a subset that you provide. You can repeat the process to add additional views. When finished, edit the format.ps1xml file and fine-tune it. The file will have notes on how to substitute script blocks. Although, beginning with v2.31.0, you can specify a hashtable as a custom property name just as you can with Select-Object.

Even though this command was written to make it easier when writing modules that might use custom objects, you can use this command to define additional views for standard objects such as files and processes. See Examples.

If you run this command inside the Visual Studio Code PowerShell Integrated Console and use -Passthru, the new file will automatically be opened in your editor.

Examples

Example 1

```

PS C:\> $tname = "myThing"
PS C:\> $obj = [PSCustomObject]@{
    PSTypeName      = $tname
    Name            = "Jeff"
    Date            = (Get-Date)
    Computername   = $env:computername
    OS              = (Get-Ciminstance win32_operatingsystem).caption
}
PS C:\> $upParams = @{
    TypeName        = $tname
    MemberType      = "ScriptProperty"
    MemberName      = "Runtime"
    Value           = {(Get-Date) - [datetime]"1/1/2020"}
    Force           = $True
}
PS C:\> Update-TypeData @upParams
PS C:\> $obj

Name      : Jeff
Date      : 2/10/2020 8:49:10 AM
Computername : BOVINE320
OS        : Microsoft Windows 10 Pro
Runtime   : 40.20:49:43.9205882

```

This example begins by creating a custom object. You might normally do this in a script or module.

Example 2

```

PS C:\> $fmt = "C:\scripts\$tname.format.ps1xml"
PS C:\> $obj | New-PSFormatXML -Prop Name,Date,Computername,OS -path $fmt
PS C:\> $obj | New-PSFormatXML -Prop Name,OS,Runtime -view runtime -path $fmt -append
PS C:\> $obj | New-PSFormatXML -FormatType List -path $fmt -append

```

The object is then piped to New-PSFormatXML to generate a new format.ps1xml file. Subsequent commands add more formatted views. When the file is completed it can be modified. Note that these examples are using shortened parameter names.

Example 3

```

PS C:\> Update-FormatData -appendpath "C:\work\$tname.format.ps1xml"
PS C:\> $obj

Name Date Computername Operating System
-----
Jeff 2/10/2020 8:49:10 AM 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, 2020
Computername : BOVINE320
OperatingSystem : Microsoft Windows 10 Pro
Runtime : 40.21:12:01

```

After the format.ps1xml file is applied, the object can be formatted as designed.

Example 4

```

PS C:\> $obj | New-PSFormatXML -view computer -Group Computername
-path "c:\work\$tname.format.ps1xml" -append
PS C:\> Update-FormatData -appendpath "C:\work\$tname.format.ps1xml"
PS C:\> $obj | Format-Table -View computer

Computername: BOVINE320

Name Date OS Runtime
-----
Jeff 2/10/2020 8:49:10 AM Microsoft Windows 10 Pro 40.20:56:24.5411481

```

This adds another view called Computer that groups objects on the Computername property.

Example 5

```

PS C:\>$params = @{
Properties = "DisplayName"
FormatType = "Wide"
Path = "C:\work\svc.format.ps1xml"
GroupBy = "Status"
ViewName ="Status"
}
PS C:\> Get-Service bits | New-PSFormatXML @params
PS C:\> Update-FormatData $params.path

```

This will create a custom format file for service objects. This will create a wide display using the DisplayName property. Once loaded into PowerShell, you can run a command like this:

| | | |
|-------------|--------------------|--------------------------|
| Get-Service | Sort-Object Status | Format-Wide -view Status |
|-------------|--------------------|--------------------------|

Example 6

```
PS C:\> '' | Select-Object -Property Name,Size,Date,Count,Age |
New-PSFormatXML -Typename myThing -Path c:\scripts\mything.format.ps1xml
```

This is an example of creating a formatting file from an empty object. Normally, you would first define your object and verify it has all the properties you need, and then you would create the formatting file. But you may want to create the formatting file in parallel using an older technique like this.

Example 7

```
PS C:\> $p = @{
FormatType = "List"
ViewName = "run"
Path = "c:\scripts\run.ps1xml"
Properties = "ID","Name","Path","StartTime",
@{Name="Runtime";Expression={(Get-Date) - $_.starttime}}
}
PS C:\> Get-Process -id $pid | New-PSFormatXML @p
```

Beginning with v2.31.0 of the PSScriptTools module, you can specify a property defined as a scriptblock, just as you do with Select-Object. The XML file will be automatically created using the script block.

Parameters

-Append

Append the new view to an existing format.ps1xml file. You need to make sure that view names are unique. With the exception of default. You can have multiple default views as long as they are different types, such as table and list.

Type: **SwitchParameter**
Parameter Sets: **(All)**
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Confirm

Prompts you for confirmation before running the cmdlet.

Type: **SwitchParameter**

Parameter Sets: ([All](#))

Aliases: cf

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

-FormatType

Specify whether to create a table, list, or wide view.

Type: **String**

Parameter Sets: ([All](#))

Aliases:

Accepted values: Table, List, Wide

Required: False

Position: 2

Default value: Table

Accept pipeline input: False

Accept wildcard characters: False

-InputObject

Specify an object to analyze and generate or update a ps1xml file. All you need is one instance of the object. Ideally, the object will have values for all properties.

Type: **Object**

Parameter Sets: ([All](#))

Aliases:

Required: True

Position: 0

Default value: None

Accept pipeline input: True ([ByValue](#))

Accept wildcard characters: False

-Passthru

Write the ps1xml file object to the pipeline. If you run this command inside the VS Code PowerShell integrated console and use this parameter, the file will be opened in the editor.

Type: `SwitchParameter`**Parameter Sets:** ([All](#))**Aliases:****Required:** False**Position:** Named**Default value:** None**Accept pipeline input:** False**Accept wildcard characters:** False

-Path

Enter full filename and path for the format.ps1xml file.

Type: `String`**Parameter Sets:** ([All](#))**Aliases:****Required:** True**Position:** 4**Default value:** None**Accept pipeline input:** False**Accept wildcard characters:** False

-Properties

Enter a set of properties to include. If you don't specify anything then all properties will be used. When creating a Wide view you should only specify a single property.

Type: `Object[]`**Parameter Sets:** ([All](#))**Aliases:****Required:** False**Position:** 1**Default value:** None**Accept pipeline input:** False**Accept wildcard characters:** False

-ViewName

Enter the name of your view.

Type: `String`**Parameter Sets:** ([All](#))**Aliases:****Required:** False**Position:** 3**Default value:** default**Accept pipeline input:** False**Accept wildcard characters:** False

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: wi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Typename

Specify the object typename. If you don't, then the command will use the detected object type from the InputObject.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-GroupBy

Specify a property name to group objects on. You can edit the file if you need to change how it is displayed and/or calculated.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Wrap

Wrap long lines. This only applies to Tables.

Type: **SwitchParameter**

Parameter Sets: (**All**)

Aliases:

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Object

Outputs

None

System.IO.FileInfo

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Update-FormatData](#)

New-RandomFileName

Synopsis

Create a random file name.

Syntax

none (Default)

```
New-RandomFileName [-Extension <String>] [<CommonParameters>]
```

temp

```
New-RandomFileName [-Extension <String>] [-UseTempFolder] [<CommonParameters>]
```

home

```
New-RandomFileName [-Extension <String>] [-UseHomeFolder] [<CommonParameters>]
```

Description

Create a new random file name. The default is a completely random name including the extension. But you can also create a filename that includes either the TEMP folder or the user's home folder. In the case of a Windows system, the home folder will be the documents folder.

This command does not create the file, it only generates a name for you to use.

Examples

EXAMPLE 1

```
PS C:\> New-RandomFileName  
fykxecvh.ipw
```

EXAMPLE 2

```
PS C:\> New-RandomFileName -extension dat  
emevgq3r.dat
```

Specify a file extension.

EXAMPLE 3

```
PS C:\> New-RandomFileName -extension log -UseHomeFolder  
C:\Users\Jeff\Documents\kbyw4fda.log
```

Create a random file name using the user's home folder. In Windows, this will be the Documents folder.

EXAMPLE 4

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

Create a random file name using the user's home folder on a Linux installation.

Parameters

-Extension

Use a specific extension. Do not include the period.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-UseHomeFolder

Include the user's HOME folder.

Type: **SwitchParameter**
Parameter Sets: home
Aliases: home

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-UseTempFolder

Include the TEMP folder.

Type: **SwitchParameter**
Parameter Sets: temp
Aliases: temp

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[New-CustomFileName](#)

New-RedGreenGradient

Synopsis

Create an ANSI gradient from red to green.

Syntax

```
New-RedGreenGradient [[-Percent <Double>] [-Step <Int32>] [-Character <Char>]
[<CommonParameters>]]
```

Description

You can use this command to create an ANSI colored gradient bar running from red to green. By specifying a percentage, you can provide a visual representation. The closer the percent value is to 1 the more green will be displayed. Use the -Step parameter to adjust the bar length. The smaller the step the longer the bar.

Examples

Example 1

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

This will display a red to green gradient bar.

Example 2

```
PS C:\> Get-Volume |
Where {$_.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      92 38.84 ██████████
D Data         104 21.82 ████████
```

The bar graph will be colored from red towards green. This example is using the Format-Percent and Format-Value commands from the PSScriptTools module.

Parameters

-Character

Specify a character to use for the gradient bar

```
Type: Char
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: [char]0x2588
Accept pipeline input: False
Accept wildcard characters: False
```

-Percent

Specify a percentage as a decimal value like .35

```
Type: Double
Parameter Sets: (All)
Aliases:

Required: False
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Step

Specify a relative bar length between 2 and 10. The smaller the number the longer the bar.

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: 5
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[New-ANSIBar](#)

[Write-ANSIProgress](#)

New-WPFMessageBox

Synopsis

Display a customizable WPF-based message box.

Syntax

standard (Default)

```
New-WPFMessageBox [-Message] <String> [-Title <String>] [-Icon <String>]  
[-ButtonSet <String>] [-Background <String>] [-Quiet] [<CommonParameters>]
```

custom

```
New-WPFMessageBox [-Message] <String> [-Title <String>] [-Icon <String>]  
[-CustomButtonSet <OrderedDictionary>] [-Background <String>] [-Quiet]  
[<CommonParameters>]
```

Description

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. See examples. If you prefer to simply display the form, you can use the -Quiet parameter to suppress any output. PowerShell will block until a button is clicked or the form dismissed.

This command requires a Windows platform.

Examples

Example 1

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

Display a Yes/No message box. The value of the clicked button will be written to the pipeline. It is assumed you would use this in a script and have logic to determine what to do based on the value.

Example 2

```
PS C:\> New-WPFMessageBox -Message "Press OK when ready to continue."  
-Title "User Deletion" -Quiet -Background crimson -Icon Shield
```

Display a message box with a crimson background and using the Shield icon. No value will be written to the pipeline and PowerShell will wait until OK is clicked or the form dismissed.

Example 3

```
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})
```

Create a custom message box with a user-defined set of buttons.

Parameters

-Background

You can specify any console color or any value from <https://docs.microsoft.com/en-us/dotnet/api/system.windows.media.brushes?view=netframework-4.7.2>. You can use the name or the code. Keep in mind there are no provisions to change the font color.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: White
Accept pipeline input: False
Accept wildcard characters: False

-ButtonSet

Select a pre-defined set of buttons. Each button will close the form and write a value to the pipeline. This can serve as the "return value" of the form.

OK = 1

Cancel = 0

Yes = \$True

No = \$False

Type: `String`
Parameter Sets: standard
Aliases:
Accepted values: OK, OKCancel, YesNo

Required: False
Position: Named
Default value: OK
Accept pipeline input: False
Accept wildcard characters: False

-CustomButtonSet

You can specify your own button set defined in an ordered hashtable. Buttons will be displayed in order from left to right. You can display up to 3 buttons. The key should be the text to display and the value should be the value you expect to write to the pipeline. It is recommended that you keep the button text short. The first letter of each key will automatically be formatted as an accelerator so you should make sure each key starts with a different letter. The first key will also be set as the default.

Type: `OrderedDictionary`
Parameter Sets: custom
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Icon

Select one of the standard system icons.

Type: `String`
Parameter Sets: ([All](#))
Aliases:
Accepted values: Information, Warning, Error, Question, Shield

Required: False
Position: Named
Default value: Information
Accept pipeline input: False
Accept wildcard characters: False

-Message

Enter the text message to display.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Quiet

Suppress any pipeline output.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Title

Enter the text to be displayed in the title bar. You should keep this brief.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

System.Int32

System.Boolean

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Invoke-InputBox](#)

Open-PSScriptToolsHelp

Synopsis

Open the PSScriptTools PDF manual.

Syntax

```
Open-PSScriptToolsHelp [<CommonParameters>]
```

Description

This command will launch a PDF manual for all commands in the PSScriptTools module. It is assumed you have a default application associated with PDF files.

Examples

Example 1

```
PS C:\> Open-PSScriptToolsHelp
```

Parameters

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

None

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Help](#)

Optimize-Text

Synopsis

Clean and optimize text input.

Syntax

default (Default)

```
Optimize-Text [[-Text] <String[]>] [-Filter <Regex>] [-Ignore <String>]  
[-ToUpper] [<CommonParameters>]
```

object

```
Optimize-Text [[-Text] <String[]>] [-Filter <Regex>][-Ignore <String>]  
[-ToUpper] [-PropertyName <String>] [<CommonParameters>]
```

Description

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. See examples.

If your text file has commented lines, use the ignore parameter. As long as the character is the first non-whitespace character in the line, the line will be treated as a comment and ignored.

Finally, you can use the -Filter parameter to specify a regular expression pattern to further filter what text is written to the pipeline. The filter is applied after leading and trailing spaces have been removed and before any text is converted to upper case.

Examples

EXAMPLE 1

```
PS C:\> Get-Content c:\scripts\computers.txt
```

```
win10-ent-01
srv1
srv2
dc01

app02
```

```
PS C:\> Get-Content c:\scripts\computers.txt | Optimize-Text
```

```
win10-ent-01
srv1
quark
dc01
app02
```

The first example shows a malformed text file. In the second command, it has been optimized or normalized.

EXAMPLE 2

```
PS C:\> Get-Content c:\scripts\computers.txt |  
Optimize-Text -property computername
```

```
computername
-----
win10-ent-01
srv1
quark
dc01
app02
```

Using the same text file, the command creates a custom object using the Computername property.

EXAMPLE 3

```
PS C:\> Get-Content computers.txt | Optimize-Text -prop computername |  
Where-Object {Test-Connection $_.computername -count 1 -ea silentlycontinue} |  
Get-Service bits | Select-Object Name,Status,Machinename
```

| Name | Status | MachineName |
|------|---------|--------------|
| bits | Running | win10-ent-01 |
| bits | Running | dc01 |
| bits | Running | app02 |

Optimize the computer names in computers.txt and add a Computername property. Test each computer, ignoring those that fail, and get the Bits service on the ones that can be pinged.

EXAMPLE 4

```
PS C:\> Get-Content .\ChicagoServers.txt |  
Optimize-Text -Ignore "#" -Property ComputerName  
  
ComputerName  
-----  
chi-fp01  
chi-fp02  
chi-core01  
chi-test  
chi-dc01  
chi-dc02  
chi-dc04  
chi-db01
```

Optimize the text file ignoring any lines that start with the # character.

EXAMPLE 5

```
PS C:\> Get-Content .\ChicagoServers.txt |  
Optimize-Text -filter "dc\d{2}" -ToUpper -PropertyName Computername |  
Test-Connection -count 1  
  
Source      Destination    IPV4Address    IPV6Address    Bytes    Time(ms)  
-----      -----          -----          -----          -----  
win10-ENT-01  CHI-DC01    172.16.30.200          32        0  
win10-ENT-01  CHI-DC02    172.16.30.201          32        0  
win10-ENT-01  CHI-DC04    172.16.30.203          32        0
```

Get names from a text file that match the pattern, turn into an object with a property name, and pipe to Test-Connection.

Parameters

-Text

The text to be optimized. Typically read in from a file.

```
Type: String[]  
Parameter Sets: default  
Aliases:  
  
Required: False  
Position: 1  
Default value: None  
Accept pipeline input: True (ByValue)  
Accept wildcard characters: False
```

Type: `String[]`
Parameter Sets: object
Aliases:

`Required:` False
`Position:` 1
`Default value:` None
`Accept pipeline input:` True (`ByValue`)
`Accept wildcard characters:` False

-Filter

Use a regular expression pattern to filter. The filtering is applied after leading and trailing spaces have been trimmed and before text can be converted to upper case.

Type: `Regex`
Parameter Sets: (`All`)
Aliases:

`Required:` False
`Position:` Named
`Default value:` None
`Accept pipeline input:` False
`Accept wildcard characters:` False

-PropertyName

Assign each line of text a property name. This has the effect of turning your text file into an array of objects with a single property.

Type: `String`
Parameter Sets: object
Aliases:

`Required:` False
`Position:` Named
`Default value:` None
`Accept pipeline input:` False
`Accept wildcard characters:` False

-Ignore

Specify a character that will be interpreted as a comment character. It must be the first-word character in a line. These lines will be ignored. This parameter has an alias of 'comment'.

Type: `String`
Parameter Sets: ([All](#))
Aliases: comment

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-ToUpper

Write text output as upper case.

Type: `SwitchParameter`
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.String

Outputs

System.String

System.Management.Automation.PSObject

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

This function was originally described at <http://jdhitsolutions.com/blog/2014/09/using-optimized-text-files-in-powershell>

Related Links

[Get-Content](#)

Out-ConditionalColor

Synopsis

Display colorized pipelined output.

Syntax

property (Default)

```
Out-ConditionalColor [-PropertyConditions] <Hashtable> -Property <String>
-InputObject <PSObject[]> [<CommonParameters>]
```

conditions

```
Out-ConditionalColor [-Conditions] <OrderedDictionary>
-InputObject <PSObject[]> [<CommonParameters>]
```

Description

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 get colorized data and save the output to a variable at the same time, although you'll need to use the common OutVariable parameter (see examples).

The default behavior is to use a hash table with a property name and color. The color must be one of the standard console colors used with Write-Host.

```
$c = @{Stopped='Red';Running='Green'}
```

You can then pipe an expression to this command, specifying a property name and the hash table. If the property matches the key name, the output for that object will be colored using the corresponding hash table value.

```
Get-Service -displayname windows* | Out-ConditionalColor $c -property status
```

Or you can do more complex processing with an ordered hash table constructed using this format:

```
[ordered]@{ <comparison scriptblock> = <color>}
```

The comparison scriptblock can use \$PSItem.

```
$h=[ordered]@{  
    {$psitem.ws -gt 500mb}='red'  
    {$psitem.ws -gt 300mb}='yellow'  
    {$psitem.ws -gt 200mb}='cyan'  
}  
  
Get-Process | Out-ConditionalColor $h
```

When doing a complex comparison you must use an [ordered] hashtable as each key will be processed in order using an If/ElseIf statement.

This command should be the last part of any pipelined expression. If you pipe to anything else, such as Sort-Object, you will lose your color formatting. Do any other sorting or filtering before piping to this command.

This command works best in the PowerShell console. It won't do anything in the PowerShell ISE.

LIMITATIONS

Due to the nature of PowerShell's formatting system, there are some limitations with this command. If the first item in your output matches one of your conditions, any text before it, such as headers, will also be colorized. This command will have no effect if the incoming object does not have a defined format view. This means you can't pipe custom objects or something using Select-Object that only includes selected properties to this command.

Examples

EXAMPLE 1

```
PS C:\> Get-Service -displayname windows* |  
Out-ConditionalColor -propertyconditions @{{Stopped='Red'}} -property Status
```

Get all services where the display name starts with windows and display stopped services in red.

EXAMPLE 2

```
PS C:\> Get-Service -displayname windows* |  
Out-ConditionalColor @{{Stopped='Red'}} status -ov winstop
```

Repeat the previous example, but also save the output to the variable winstop. When you look at \$Winstop you'll see the services, but they won't be colorized. This example uses the parameters positionally.

EXAMPLE 3

```
PS C:\> Get-EventLog system -newest 50 |  
Out-ConditionalColor @{{error='red';warning='yellow'}}  
Enter a property name: entrytype
```

Get the newest 50 entries from the System event log. Display errors in red and warnings in yellow. If you don't specify a property you will be prompted.

EXAMPLE 4

```
PS C:\> $c =[ordered]@{  
    {$psitem.length -ge 1mb}='red';  
    {$psitem.length -ge 500KB}='yellow';  
    {$psitem.length -ge 100KB}='cyan'}
```

The first command creates an ordered hashtable based on the Length property.

EXAMPLE 5

```
PS C:\> dir c:\scripts\*.doc,c:\scripts\*.pdf,c:\scripts\*.xml |  
Out-ConditionalColor $c
```

The next command uses it to get certain file types in the scripts folder and display the selected properties in color depending on the file size.

Parameters

-Conditions

Use an ordered hashtable for more complex processing. See examples.

```
Type: OrderedDictionary  
Parameter Sets: conditions  
Aliases:  
  
Required: True  
Position: 0  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-InputObject

The output from a PowerShell expression that you want to colorize.

```
Type: PSObject[]  
Parameter Sets: (All)  
Aliases:  
  
Required: True  
Position: Named  
Default value: None  
Accept pipeline input: True (ByValue)  
Accept wildcard characters: False
```

-Property

When using a simple hash table, specify the property to compare which will be done by using the -eq operator.

```
Type: String
Parameter Sets: property
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-PropertyConditions

Use a simple hashtable for basic processing or an ordered hash table for complex.

```
Type: Hashtable
Parameter Sets: property
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Management.Automation.PSObject[]

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Originally published at: <http://jdhitsolutions.com/blog/powershell/3462/friday-fun-Out-ConditionalColor/>

Related Links

[About_Hash_Tables](#)

[Show-Tree](#)

Out-Copy

Synopsis

Send command output to the pipeline and clipboard.

Syntax

```
Out-Copy [-InputObject] <Object> [-Width <Int32>] [-CommandOnly]
[<CommonParameters>]
```

Description

This command is intended for writers and those who need to document with PowerShell. You can pipe any command to this function and you will get the regular output in your PowerShell session. Simultaneously, a copy of the output will be sent to the Windows clipboard. The copied output will include a prompt constructed from the current location unless you use the CommandOnly parameter.



You can only capture what is written to the Success pipeline. This command will not copy any other streams such as Verbose, Warning, or Error.

Examples

EXAMPLE 1

```
PS C:\> Get-Process | Sort WS -Descending | Select-First 5 | Out-Copy
```

This will execute your expression and write the output to the pipeline. The output plus the command:

| | | |
|---------------------|---------------------|-----------------|
| PS C:\> Get-Process | Sort WS -Descending | Select -first 5 |
|---------------------|---------------------|-----------------|

will be copied to the clipboard. This example is using the Select-First function from the PSScriptTools module.

EXAMPLE 2

```
PS C:\> Get-ChildItem *.ps1 | Out-File c:\work\ps.txt | Out-Copy
```

Even if your command doesn't write anything to the pipeline, Out-Copy will still capture a prompt and PowerShell expression.

EXAMPLE 3

```
PS C:\> Get-CimInstance -class win32_logicaldisk -filter "drivetype = 3" |  
Out-Copy -commandonly
```

This will run the Get-CimInstance command and write results to the pipeline. But the only text that will be copied to the clipboard is:

```
Get-CimInstance -class win32_logicaldisk -filter "drivetype = 3"
```

Parameters

-InputObject

This is the piped in command.

```
Type: Object  
Parameter Sets: (All)  
Aliases:  
  
Required: True  
Position: 1  
Default value: None  
Accept pipeline input: True (ByValue)  
Accept wildcard characters: False
```

-Width

Specifies the number of characters in each line of output. Any additional characters are truncated, not wrapped.

```
Type: Int32  
Parameter Sets: (All)  
Aliases:  
  
Required: False  
Position: Named  
Default value: 80  
Accept pipeline input: False  
Accept wildcard characters: False
```

-CommandOnly

Only copy the executed command, without references to Out-Copy, to the Windows clipboard.

Type: **SwitchParameter**

Parameter Sets: (**All**)

Aliases:

Required: False

Position: Named

Default value: False

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Object

Outputs

System.Object

Notes

Learn more about PowerShell:<http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Out-String](#)

[Set-Clipboard](#)

[Tee-Object](#)

Out-More

Synopsis

Send "pages" of objects to the pipeline.

Syntax

```
Out-More [-InputObject] <Object[]> [[-Count] <Int32>] [-ClearScreen]
```

Description

This function is designed to display groups or "pages" of objects to the PowerShell pipeline. It is modeled after the legacy More.com command-line utility. By default, the command will write objects out to the pipeline in groups of 50. You will be prompted after each grouping.

Pressing M or Enter will get the next group. Pressing A will stop paging and display all of the remaining objects. Pressing N will display the next object. Press Q to stop writing anything else to the pipeline.

Note that you may encounter an error message when quitting prematurely, especially on non-Windows platforms. You can ignore these errors.

Examples

EXAMPLE 1

```
PS C:\> Get-Process | Out-More -count 10



Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	SI	ProcessName
103	9	1448	4220	67	0.02	1632	0	BtwRSupportService
80	9	3008	8588	...27	21.00	5192	1	conhost
40	5	752	2780	...82	0.00	5248	0	conhost
53	7	972	3808	...07	0.02	6876	1	conhost
482	17	1932	3692	56	0.91	708	0	csrss
520	30	2488	134628	180	31.67	784	1	csrss
408	18	6496	12436	...35	0.56	1684	0	dasHost
180	14	3348	6748	66	0.50	4688	0	devmonsrv



\[M\]ore \[A\]ll \[N\]ext \[Q\]uit


```

Display processes in groups of 10.

EXAMPLE 2

```
PS C:\> dir c:\work -file -Recurse | Out-More -ClearScreen | tee -Variable work
```

List all files in C:\Work and page them to Out-More using the default count, but after clearing the screen first.

The results are then piped to Tee-Object which saves them to a variable.

Parameters

-InputObject

```
Type: Object[]
Parameter Sets: (All)
Aliases:

Required: True
Position: 1
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Count

The number of objects to group as a page.

```
Type: Int32
Parameter Sets: (All)
Aliases: i

Required: False
Position: 2
Default value: 50
Accept pipeline input: False
Accept wildcard characters: False
```

-ClearScreen

Clear the screen before writing data to the pipeline.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: cls

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

Inputs

System.Object

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

This command was first demonstrated at <http://jdhitsolutions.com/blog/powershell/4707/a-better-powershell-more/>

Related Links

more

Out-VerboseTee

Synopsis

Write to the Verbose stream and a file.

Syntax

```
Out-VerboseTee -Value <Object> [-Path] <String> [-Encoding <Encoding>]
[-Append] [<CommonParameters>]
```

Description

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 an alias of `Tee-Verbose`.

You might use it like this in a script.

```
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.

Examples

Example 1

```
PS C:\> $VerbosePreference = "continue"
PS C:\> $log = New-CustomFileName ".\VerboseLog_%time.txt"
PS C:\> Write-Detail "This is a verbose log test" | Out-VerboseTee -path $log
PS C:\> Get-Content $log
11/29/2020 08:21:31:0704 [PROCESS] This is a verbose log test
PS C:\> $verbosePreference = "silentlyContinue"
```

Normally you would use this command inside a function or script, but you can run it from the console if you want to understand how it works.

Parameters

-Append

Append to the specified text file.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Encoding

Specify a file encoding.

```
Type: Encoding
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Path

The path for the output file.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Value

The message to be displayed as a verbose message and saved to the file.

Type: **Object**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: True ([ByValue](#))
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.Object

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Write-Verbose](#)

[Write-Detail](#)

Tee-Object

Remove-MergedBranch

Synopsis

Removed merged git branches.

Syntax

```
Remove-MergedBranch [-Force] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

When using git you may create multiple branches. Presumably, you merge these branches into the main or master branch. The development or patching branch remains. You can use git to remove branches. Or 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.

Examples

Example 1

```
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.

Example 2

```
PS C:\MyProject> Remove-MergedBranch -force
Deleted branch 2.1.1 (was 75f6ab8).
Deleted branch patch-254 (was 75f6ab8).
```

Remove all branches with no prompting.

Parameters

-Confirm

Prompts you for confirmation before running the cmdlet.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Force

Remove all merged branches except current and master with no prompting.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: wi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[git.exe](#)

[Get-GitSize](#)

Remove-PSAnsibleFileEntry

Synopsis

Remove a PSAnsibleFileMap entry.

Syntax

```
Remove-PSAnsibleFileEntry [-Description] <String> [-Passthru] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

Use this command to remove an entry from the global \$PSAnsibleFileMap variable. The change will not be persistent unless you export the map to a file.

Examples

Example 1

```
PS C:\> Remove-PSAnsibleFileEntry Samples
```

Remove a PSAnsibleFileMap entry with a description of 'Samples'. The change will not be persistent unless you export the map to a file.

Parameters

-Confirm

Prompts you for confirmation before running the cmdlet.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Description

Specify the description of the entry to remove.

Type: **String**

Parameter Sets: ([All](#))

Aliases:

Required: True

Position: 0

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

-Passthru

Display the updated PSAnsFileMap.

Type: **SwitchParameter**

Parameter Sets: ([All](#))

Aliases:

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

Type: **SwitchParameter**

Parameter Sets: ([All](#))

Aliases: wi

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

PSAnsiFileEntry

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Set-PSAnsiFileMapEntry](#)

[Get-PSAnsiFileMapEntry](#)

Remove-Runspace

Synopsis

Remove a runspace from your session.

Syntax

id (Default)

```
Remove-Runspace [-ID] <Int32> [-WhatIf] [-Confirm] [<CommonParameters>]
```

runspace

```
Remove-Runspace [-Runspace] <Runspace> [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

When working with PowerShell, you may discover that some commands and scripts can leave behind runspaces. 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. You cannot remove any runspace with an availability of Busy or that is already closing.

This command does not write anything to the pipeline.

Examples

Example 1

```
PS C:\> Remove-Runspace -id 18 -WhatIf
What if: Performing the operation "Remove-Runspace" on target "18 - Runspace18".
```

Show what would have happened to remove runspace with an ID of 18.

Example 2

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

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

Parameters

-Confirm

Prompts you for confirmation before running the cmdlet.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-ID

The runspace ID number.

```
Type: Int32
Parameter Sets: id
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Runspace

A runspace presumably piped into this command using Get-Runspace.

```
Type: Runspace
Parameter Sets: runspace
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

Type: **SwitchParameter**

Parameter Sets: (**All**)

Aliases: wi

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

System.Management.Automation.Runspaces.Runspace

Outputs

None

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Get-Runspace

Rename-Hashtable

Synopsis

Rename a hashtable key.

Syntax

Pipeline (Default)

```
Rename-Hashtable [-InputObject] <Object> [-Key] <String> [-NewKey] <String>
[-Passthru] [-Scope <String>] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Name

```
Rename-Hashtable [-Name] <String> [-Key] <String> [-NewKey] <String>
[-Passthru] [-Scope <String>] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

This command will rename a key in an existing hashtable or ordered dictionary. You can either pipe a hashtable object to this command or you can specify a variable name for a pre-defined hashtable. If you use this option, specify the variable name without the \$.

This command will create a temporary copy of the hashtable, create the new key, and copy the value from the old key, before removing the old key. The temporary hashtable is then set as the new value for your original variable.

This command does not write anything to the pipeline when you use a variable name unless you use -Passthru. If you pipe a hashtable to this command, the new hashtable will automatically be written to the pipeline.

You might find this command useful when building a hashtable that you intend to use with splatting where you need to align key names with parameter names.

Examples

EXAMPLE 1

```
PS C:\> Rename-Hashtable -name MyHash -key Name -newKey Computername
```

EXAMPLE 2

```
PS C:\> $newhash = Get-Service spooler |  
ConvertTo-HashTable |  
Rename-Hashtable -Key Machinename -NewKey Computername
```

This command uses the ConvertTo-Hashtable command from the PSScriptTools module to turn an object into a hashtable. The Machinename key is then renamed to Computername.

Parameters

-Name

The variable name of your hash table. DO NOT include the \$.

```
Type: String  
Parameter Sets: Name  
Aliases:  
  
Required: True  
Position: 1  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-InputObject

A piped in hashtable object

```
Type: Object  
Parameter Sets: Pipeline  
Aliases:  
  
Required: True  
Position: 1  
Default value: None  
Accept pipeline input: True (ByValue)  
Accept wildcard characters: False
```

-Key

The name of the existing hashtable key you want to rename.

```
Type: String  
Parameter Sets: (All)  
Aliases:  
  
Required: True  
Position: 2  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-NewKey

The new name of the hashtable key.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: True
Position: 3
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Passthru

Write the revised hashtable back to the pipeline. If you pipe a variable to this command, passthru will happen automatically.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-Scope

The scope where your variable is defined. The default is the global scope.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: Global
Accept pipeline input: False
Accept wildcard characters: False
```

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases: wi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Confirm

Prompts you for confirmation before running the cmdlet.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

hashtable

Outputs

None

Hashtable

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

This code was first described at <http://jdhitsolutions.com/blog/2013/01/Rename-Hashtable-key-revised>

Related Links

[About_hash_tables](#)

[ConvertTo-Hashtable](#)

[Join-Hashtable](#)

Save-GitSetup

Synopsis

Download the latest 64bit version of Git for Windows.

Syntax

```
Save-GitSetup [[-Path] <String>] [-Passthru] [<CommonParameters>]
```

Description

Non-Windows platforms have package management that make it easy to install newer versions of git. This command is for Windows platforms. You can run this command to download the latest 64bit version of Git for Windows. You will need to manually install it.

Examples

Example 1

```
C:\> Save-GitSetup -Path c:\work -Passthru

Directory: C:\work

Mode          LastWriteTime        Length Name
---          -----          ----
-a---  12/28/2020 7:29 PM    48578904 Git-2.29.2.3-64-bit.exe
```

Parameters

-Passthru

Show the downloaded file.

Type: **SwitchParameter**
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Path

Specify the location to store the downloaded file.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 0
Default value: $env:TEMP
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

None

System.IO.FileInfo

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

git.exe

Select-After

Synopsis

Select objects after a given datetime.

Syntax

```
Select-After [-InputObject <PSObject>] [-After] <DateTime> [-Property <String>]
[<CommonParameters>]
```

Description

Select-After is a simplified version of Select-Object. The premise is that you can pipe a collection of objects to this command and select objects after a given datetime, based on a property, like LastWriteTime, which is the default.

Examples

Example 1

```
PS C:\> Get-Childitem c:\work -file | Select-After "11/1/2020"
```

| Directory: C:\work | | |
|--------------------|--------------------|-----------------|
| Mode | LastWriteTime | Length Name |
| --- | ----- | ----- |
| -a-- | 11/4/2020 11:36 AM | 5008 ipperf.csv |

Select all objects that have been modified after 11/1/2020. This example is using the default -Property value of LastWriteTime.

Example 2

```
PS C:\> Get-Process | After ((Get-Date).AddMinutes(-10)) -Property StartTime
```

| NPM(K) | PM(M) | WS(M) | CPU(s) | Id | SI | ProcessName |
|--------|-------|-------|--------|-------|----|--------------------|
| 8 | 1.49 | 7.17 | 0.00 | 33248 | 0 | SearchFilterHost |
| 12 | 2.46 | 12.99 | 0.02 | 15328 | 0 | SearchProtocolHost |
| 8 | 2.60 | 8.58 | 0.03 | 9756 | 0 | svchost |
| 76 | 20.27 | 39.93 | 2.14 | 22976 | 0 | svchost |
| 8 | 1.53 | 7.29 | 0.00 | 29752 | 0 | svchost |

Get all processes where the StartTime property value is within the last 10 minutes. This example is using the "after" alias.

Parameters

-After

Enter the cutoff date.

```
Type: DateTime
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-InputObject

A piped in object.

```
Type: PSObject
Parameter Sets: (All)
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Property

Enter the property name to use for the datetime sort. It needs to be a datetime object.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: LastWriteTime
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Management.Automation.PSObject

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Select-Before](#)

[Select-Object](#)

Select-Before

Synopsis

Select objects before a given datetime.

Syntax

```
Select-Before -InputObject <PSObject> [-Before] <DateTime> [-Property <String>]
[<CommonParameters>]
```

Description

Select-Before is a simplified version of Select-Object. The premise is that you can pipe a collection of objects to this command and select objects before a given datetime, based on a property, like LastWriteTime, which is the default.

Examples

Example 1

```
PS C:\> Get-Childitem c:\work -file | Select-Before "11/1/2020"

Directory: C:\work

Mode          LastWriteTime      Length Name
----          -----          ---- 
-a--- 10/10/2020 2:09 PM        8862 Book1.xlsx
-a--- 10/30/2020 10:48 AM         0 dummy.dat
-a--- 10/13/2020 9:35 AM       447743 key1013.pdf
-a--- 10/6/2020 4:03 PM        2986 labsummary.format.ps1xml
-a--- 10/11/2020 12:33 PM       1678 prun.format.ps1xml
-a--- 10/10/2020 6:49 PM        1511 w.format.ps1xml
```

Select all objects that have been modified before 11/1/2020. This example is using the default -Property value of LastWriteTime.

Example 2

```
PS C:\> Get-Process | before (Get-Date).AddMinutes(-10) -Property StartTime
```

| NPM(K) | PM(M) | WS(M) | CPU(s) | Id | SI | ProcessName |
|--------|--------|--------|--------|-------|----|---------------------------|
| 33 | 30.21 | 46.19 | 0.81 | 9952 | 2 | ApplicationFrameHost |
| 75 | 102.42 | 126.08 | 4.89 | 16048 | 2 | Box |
| 23 | 25.27 | 33.83 | 0.33 | 5320 | 0 | Box.Desktop.UpdateService |
| 30 | 46.92 | 60.98 | 0.91 | 17384 | 2 | BoxUI |
| 31 | 39.82 | 4.34 | 0.56 | 26992 | 2 | Calculator |
| ... | | | | | | |

Get all processes where the StartTime property value is before the last 10 minutes. This example is using the "before" alias.

Parameters

-Before

Enter the cutoff date.

Type: `DateTime`
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-InputObject

A piped in object.

Type: `PSObject`
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: True (`ByValue`)
Accept wildcard characters: False

-Property

Enter the property name to use for the datetime sort. It needs to be a datetime object.

Type: **String**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: LastWritetime
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Management.Automation.PSObject

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Select-After](#)

[Select-Object](#)

Select-First

Synopsis

Select the first X number of objects.

Syntax

```
Select-First -InputObject <PSObject> [-First] <Int32> [[-Property] <String>]
[-Skip <Int32>] [-Descending] [<CommonParameters>]
```

Description

This command is intended to take pipelined input and select the first specified number of objects which are then written to the pipeline. You also have the option to sort on a specified property.

When using this command, there is a trade-off of convenience for performance. For a very large number of processed objects, use Select-Object.

Examples

EXAMPLE 1

```
PS C:\> Get-Process | Select-First 3 -property WS -descending
```

| Handles | NPM(K) | PM(K) | WS(K) | VM(M) | CPU(s) | Id | SI | ProcessName |
|---------|--------|--------|--------|-------|----------|------|----|----------------|
| 1118 | 66 | 419952 | 392396 | ...12 | 107.33 | 7312 | 1 | powershell |
| 343 | 43 | 237928 | 235508 | 1237 | 3,905.22 | 6424 | 1 | slack |
| 1051 | 88 | 231216 | 234728 | 1175 | 61.88 | 8324 | 1 | powershell_ise |

EXAMPLE 2

```
PS C:\> 1..10 | Select-First 3 -Skip 2
```

```
3
4
5
```

Select the first 3 objects after skipping 2.

Parameters

-InputObject

Pipelined input to be selected.

```
Type: PSObject
Parameter Sets: (All)
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-First

How many items do you want to select?

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: True
Position: 1
Default value: 0
Accept pipeline input: False
Accept wildcard characters: False
```

-Property

Sort first on this property then select the specified number of items.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 2
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Skip

Skip or omit the first X number of items.

Type: **Int32**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: 0
Accept pipeline input: False
Accept wildcard characters: False

-Descending

Sort the property in descending order.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

Object[]

Outputs

Object[]

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Select-Object

Select-Last

Select-Last

Synopsis

Select the last X number of objects.

Syntax

```
Select-Last -InputObject <PSObject> [-Last] <Int32> [[-Property] <String>]
[-Skip <Int32>] [-Descending] [<CommonParameters>]
```

Description

This is a modified version of Select-Object designed to select the last X number of objects. The command takes pipelined input and selects the last specified number of objects which are then written to the pipeline. You have an option to first sort on the specified property.

When using this command, there is a trade-off of convenience for performance. For a very large number of processed objects, use Select-Object.

Examples

EXAMPLE 1

```
PS C:\> dir c:\scripts\*.ps1 | last 5 -property lastwritetime

Directory: C:\scripts

Mode          LastWriteTime        Length Name
----          -----        ----
-a---  1/11/2020 7:18 PM      1818 demo-v5Classes.ps1
-a---  1/11/2020 7:20 PM      1255 demo-v5DSCClassResource.ps1
-a---  1/14/2020 12:58 PM     1967 Demo-ParamTest.ps1
-a---  1/15/2020 9:23 AM       971 Get-WorkflowVariable.ps1
-a---  1/15/2020 12:08 PM     1555 Cost.ps1
```

Get the last 5 ps1 files sorted on the LastWritetime property. This example is using the alias 'last' for Select-Last.

EXAMPLE 2

```
PS C:\> 1..10 | Select-Last 3 -skip 1

7
8
9
```

Select the last 3 items, skipping the last 1.

Parameters

-InputObject

Pipelined input to be selected.

```
Type: PSObject
Parameter Sets: (All)
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Last

How many items do you want to select?

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: True
Position: 1
Default value: 0
Accept pipeline input: False
Accept wildcard characters: False
```

-Property

Sort first on this property then select the specified number of items.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 2
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Skip

Skip or omit the last X number of items.

Type: **Int32**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: 0
Accept pipeline input: False
Accept wildcard characters: False

-Descending

Sort on the specified property in descending order.

Type: **SwitchParameter**
Parameter Sets: (**All**)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

Object[]

Outputs

Object[]

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Select-Object

Select-First

Select-Newest

Synopsis

Select the newest X number of objects after a given datetime.

Syntax

```
Select-Newest [-InputObject <PSObject>] [-Newest] <Int32> [-Property <String>]
[<CommonParameters>]
```

Description

Select-Newest is a variation on Select-Object. It is designed to make it easier to select X number of objects based on a datetime property. The default property value is LastWriteTime.

Examples

Example 1

```
PS C:\> Get-ChildItem c:\work -file | Select-Newest 1

Directory: C:\work

Mode          LastWriteTime           Length Name
----          -----           ----  --
-a--          11/4/2020 11:36 AM      5008 ipperf.csv
```

Get the newest file in the Work folder. This example is using the default -Property parameter value of LastWriteTime.

Example 2

```
PS C:\> Get-Process | newest 10 -Property starttime

  NPM(K)    PM(M)    WS(M)    CPU(s)      Id  SI ProcessName
  ----  -----  -----  -----  --  --  -----
    15      5.34    12.85     0.09  25208  0  WmiPrvSE
     7      1.31      5.95     0.02  10552  0  svchost
    35     128.28   136.05     8.62  3376  0  esrv_svc
    98      47.31     40.01     0.48  24496  2  firefox
    99     48.46    46.07     0.53  22064  2  firefox
    13      3.41     16.19     0.77  33136  2  notepad
    14      6.78     10.96     0.06  31784  0  svchost
    69     110.45   150.37     4.28  8848  2  pwsh
     5      2.52      4.52     0.02  34024  2  cmd
    10      2.06      9.00     0.12  25384  2  OpenConsole
```

Get the 10 most recent processes based on the StartTime property. This example is using the "newest" alias.

Parameters

-InputObject

A piped in object.

```
Type: PSObject
Parameter Sets: (All)
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Newest

Enter the number of newest items to select.

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Property

Enter the property name to select on. It must be a datetime object.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: LastWriteTime
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Management.Automation.PSObject

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Select-Oldest](#)

[Select-Object](#)

Select-Oldest

Synopsis

Select the oldest X number of objects before a given datetime.

Syntax

```
Select-Oldest -InputObject <PSObject> [-Oldest] <Int32> [-Property <String>]  
[<CommonParameters>]
```

Description

Select-Oldest is a variation on Select-Object. It is designed to make it easier to select X number of objects based on a datetime property. The default property value is LastWriteTime.

Examples

Example 1

```
PS C:\> Get-ChildItem c:\work -file | Select-oldest 1
```

| Mode | LastWriteTime | Length | Name |
|-------|-------------------|--------|--------------------------|
| -a--- | 10/6/2020 4:03 PM | 2986 | labsummary.format.ps1xml |

Get the oldest file in the Work folder. This example is using the default -Property parameter value of LastWriteTime.

Example 2

```
PS C:\> Get-Process | where-object name -notmatch "idle|System" |
oldest 10 -Property starttime
```

| NPM(K) | PM(M) | WS(M) | CPU(s) | Id | SI | ProcessName |
|--------|-------|-------|--------|------|----|-------------|
| 16 | 8.43 | 99.83 | 2.27 | 204 | 0 | Registry |
| 3 | 1.03 | 1.12 | 0.44 | 712 | 0 | smss |
| 30 | 2.23 | 5.67 | 4.23 | 816 | 0 | csrss |
| 11 | 1.52 | 6.65 | 0.02 | 1592 | 0 | wininit |
| 11 | 6.55 | 11.63 | 25.33 | 1676 | 0 | services |
| 7 | 1.10 | 3.27 | 0.09 | 1696 | 0 | LsaIso |
| 28 | 9.81 | 24.70 | 29.61 | 1704 | 0 | lsass |
| 4 | 0.81 | 3.31 | 0.00 | 1824 | 0 | svchost |
| 26 | 13.48 | 30.38 | 22.62 | 1852 | 0 | svchost |
| 6 | 1.91 | 4.15 | 0.11 | 1876 | 0 | fontdrvhost |

Get the oldest 10 processes that don't include Idle or System. This example is using the "oldest" alias.

Parameters

-InputObject

A piped in object.

Type: **PSObject**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: True ([ByValue](#))
Accept wildcard characters: False

-Oldest

Enter the number of Oldest items to select.

Type: **Int32**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Property

Enter the property name to select on. It must be a datetime object.

Type: **String**

Parameter Sets: (**All**)

Aliases:

Required: False

Position: Named

Default value: None

Accept pipeline input: False

Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Management.Automation.PSObject

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Select-Newest

Select-Object

Set-ConsoleColor

Synopsis

Set the PowerShell console color.

Syntax

```
Set-ConsoleColor [[-Foreground] <ConsoleColor>] [[-Background] <ConsoleColor>]
[-ClearScreen] [-Passthru] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

You can use this command to modify the PowerShell console's foreground and/or background color. Note that if you are running the PSReadline module, that module has commands, like Set-PSReadLineOption, that you can use to modify your console. This command is intended for use in a traditional PowerShell console. It will not work in consoles that are part of the PowerShell ISE or Visual Studio Code.

Examples

Example 1

```
PS C:\> Set-ConsoleColor -foreground Yellow -background DarkGray -clear
```

Set the console color to yellow text and on a dark gray background.

Parameters

-Background

Specify a background console color

```
Type: ConsoleColor
Parameter Sets: (All)
Aliases: bg
Accepted values: Black, DarkBlue, DarkGreen, DarkCyan, DarkRed, DarkMagenta,
DarkYellow, Gray, DarkGray, Blue, Green, Cyan, Red, Magenta, Yellow, White

Required: False
Position: 1
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-ClearScreen

Clear the console host screen.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: cls

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Confirm

Prompts you for confirmation before running the cmdlet.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Foreground

Specify a foreground console color.

```
Type: ConsoleColor
Parameter Sets: (All)
Aliases: fg
Accepted values: Black, DarkBlue, DarkGreen, DarkCyan, DarkRed, DarkMagenta,
DarkYellow, Gray, DarkGray, Blue, Green, Cyan, Red, Magenta, Yellow, White

Required: False
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Passthru

Display foreground and background color values.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases: wi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

None

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Set-ConsoleTitle](#)

Set-ConsoleTitle

Synopsis

Set the console title text.

Syntax

```
Set-ConsoleTitle [-Title] <String> [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

Use this command to modify the text displayed in the title bar of your PowerShell console window. This command is intended for use in a traditional PowerShell console. It will not work in consoles that are part of the PowerShell ISE or Visual Studio Code. It should work in a PowerShell session running in Windows Terminal.

Examples

Example 1

```
PS C:\> Set-ConsoleTitle $env:computername
```

Set the console title to the computer name.

Example 2

```
PS C:\> if (Test-IsAdministrator) {
    Set-ConsoleTitle "Admin: PS $($PSVersionTable.PSVersion)"
}
```

Modify the console title if running as Administrator

Parameters

-Confirm

Prompts you for confirmation before running the cmdlet.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Title

Enter the title for the console window.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases: wi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable.

For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

None

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Set-ConsoleColor](#)

Set-PSAnsFileMap

Synopsis

Modify or add a PSAnsFileEntry

Syntax

```
Set-PSAnsFileMap [-Description] <String> [-Pattern <String>] [-Ansi <String>]
[-Passthru] [-WhatIf] [-Confirm] [<CommonParameters>]
```

Description

Use this command to modify an existing entry in the global \$PSAnsFileMap variable or add a new entry. If modifying, you must specify a regular expression pattern or an ANSI escape sequence. If you are adding a new entry, you need to supply both values.

Examples

Example 1

```
PS C:\> Set-PSAnsFileMap Temporary -Ansi ``e[38;5;190m"
```

Update the ANSI pattern for temporary files. This change will not persist unless you export the map.

Example 2

```
PS C:\> Set-PSAnsFileMap -Description "Config" -Pattern "\.(yml)$" -Ansi ``e[38;5;25m"ge
```

Add a new PSAnsFileMap entry. This change will not persist unless you export the map.

Parameters

-Ansi

Specify an ANSI escape sequence. You only need to define the opening sequence.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Confirm

Prompts you for confirmation before running the cmdlet.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases: cf

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Description

Specify the file map entry. If it is a new entry it will be added.

Type: **String**
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Passthru

Display the updated map.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Pattern

Specify a regular expression pattern for the file name.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: wi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

PSAnsiFileEntry

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-PSAnsFileMap](#)

[Remove-PSAnsFileEntry](#)

[Export-PSAnsFileMap](#)

Show-ANSISequence

Synopsis

Display ANSI escape sequences

Syntax

basic (Default)

```
Show-ANSISequence [-Basic] [-AsString] [<CommonParameters>]
```

foreback

```
Show-ANSISequence [-Foreground] [-Background] [-Type <String>] [-AsString]  
[<CommonParameters>]
```

RGB

```
Show-ANSISequence [-RGB <Int32[]>] [-AsString] [<CommonParameters>]
```

Description

This script is intended to make it easy to see ANSI escape sequences and how they will display in your PowerShell session. Use the -AsString parameter to write simple strings to the pipeline which makes it easier to copy items to the clipboard.

The escape character will depend on whether you are running Windows PowerShell or PowerShell 7.x. For best results, you need to run this command in a PowerShell session and host that supports ANSI escape sequences.

Examples

EXAMPLE 1

```
PS C:\> Show-ANSISequence
```

```
*****
* Basic Sequences *
*****
```

```
\e[9mCrossedOut\0m
`e[7mReverse\0m
`e[6mRapidBlink\0m
`e[5mSlowBlink\0m
`e[4mUnderline\0m
`e[3mItalic\0m
`e[2mFaint\0m
`e[1mBold\0m
```

The output will be formatted using the corresponding ANSI escape sequence as seen in PowerShell 7.x.

EXAMPLE 2

```
PS C:\>Show-ANSISequence -Foreground -Type Simple
```

```
*****
* Foreground *
*****
```

```
\e[30mHello\0m    `e[31mHello\0m    `e[32mHello\0m
`e[34mHello\0m    `e[35mHello\0m    `e[36mHello\0m
`e[90mHello\0m    `e[91mHello\0m    `e[92mHello\0m
`e[94mHello\0m    `e[95mHello\0m    `e[96mHello\0m
```

EXAMPLE 3

```
PS C:\> Show-ANSISequence -RGB 225,100,50
```

```
\e[38;2;225;100;50m256 Color (R:225)(G:100)(B:50)\0m
```

Show an RGB ANSI sequence. The output will be formatted using the sequence.

EXAMPLE 4

```
PS C:\> Show-ANSISequence -RGB 225,100,50 -AsString | Set-Clipboard
```

Repeat the previous example but write the output as a plain string and copy it to the Windows clipboard.

Parameters

-Basic

Display basic ANSI settings. This is the default output.

Type: **SwitchParameter**
Parameter Sets: basic
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-Foreground

Display foreground ANSI format settings.

Type: **SwitchParameter**
Parameter Sets: foreback
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-Background

Display background ANSI format settings.

Type: **SwitchParameter**
Parameter Sets: foreback
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-Type

You can display simple ANSI, 8-bit, or all sequences. Valid values are all,simple and 8bit.

Type: **String**
Parameter Sets: foreback
Aliases:

Required: False
Position: Named
Default value: All
Accept pipeline input: False
Accept wildcard characters: False

-RGB

Display an RGB ANSI sequence. You must pass an array of values for Red,Blue, and Green. Each value must be between 0 and 255.

```
Type: Int32[]
Parameter Sets: RGB
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-AsString

Show the value as an unformatted string.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.String

Related Links

https://en.wikipedia.org/wiki/ANSI_escape_code

Show-Tree

Synopsis

Shows the specified path as a tree.

Syntax

Path (Default)

```
Show-Tree [[-Path] <String[]>] [[-Depth] <Int32>] [-IndentSize <Int32>]  
[-ShowItem] [-ShowProperty <String[]>] [-InColor] [<CommonParameters>]
```

LiteralPath

```
Show-Tree [[-LiteralPath] <String[]>] [[-Depth] <Int32>] [-IndentSize <Int32>]  
[-ShowItem] [-ShowProperty <String[]>] [-InColor] [<CommonParameters>]
```

Description

Shows the specified path as a graphical tree in the console. Show-Tree is intended as a 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. Currently, this will *not work* with any PSDrives created with the Certificate provider. It should work cross-platform.

By default, the output will only show directory or equivalent structures. But you can opt to include items well as item details by using the ShowProperty parameter. Specify a comma-separated list of properties or use * to view them all.

If the Path is a FileSystem path there is a dynamic parameter, -InColor, that will write ANSI-colored output to the pipeline. This parameter has an alias of ansi.

 This is an update to an older function in my library. I seem to recall I found the original code somewhere online, perhaps from someone like Lee Holmes. Sadly, I neglected to record the source.

Examples

EXAMPLE 1

```
PS C:\> Show-Tree C:\Work -Depth 2

C:\work
+--A
|   \--B
+--dnssuffix
|   +--docs
|   +--en-us
|   \--images
+--gpo
|   +--{65D9E940-AAD4-4508-A199-86EAE4E9E535}
|   \--{7E7F01CE-6889-44B0-9D03-818F8284EDE0}
+--installers
+--remoteop
|   \--archive
+--test files
\--tryme
    +--.vscode
    +--docs
    +--en-us
    \--test
```

Shows the directory tree structure, recursing down two levels.

EXAMPLE 2

```
PS C:\>Show-Tree HKLM:\SOFTWARE\Microsoft\.NETFramework -Depth 2 -ShowProp *
```

```
HKLM:\SOFTWARE\Microsoft\.NETFramework
+-- Enable64Bit = 1
+-- InstallRoot = C:\Windows\Microsoft.NET\Framework64\
+-- UseRyuJIT = 1
+-- Advertised
|   +--Policy
|   \--v2.0.50727
+--AssemblyFolders
|   +--ADOMD.Client 14.0
|   |   \-- (default) = C:\Program Files\Microsoft.NET\ADOMD.NET\140\
|   +--Microsoft .NET Framework 3.5 Reference Assemblies
|   |   \-- (default) = C:\Program Files\Reference Assemblies\Microsoft\Framework...
|   +--SQL Server Assemblies 140
|   |   \-- (default) = C:\Program Files\Microsoft SQL Server\140\SDK\Assemblies\
|   +--v3.0
|   |   +-- <IncludeDotNet2Assemblies> = 1
|   |   \-- All Assemblies In = C:\Program Files\Reference Assemblies\Microsoft...
|   \--v3.5
|       +-- <IncludeDotNet2Assemblies> = 1
|       \-- All Assemblies In = C:\Program Files\Reference Assemblies\Microsoft...
...
...
```

Shows the hierarchy of registry keys and values (-ShowProperty), recursing down two levels.

EXAMPLE 3

```
PS C:\> Show-Tree WSMAN: -ShowItem
```

```
WSMAN:\  
  \--localhost  
    +--MaxEnvelopeSizekb  
    +--MaxTimeoutms  
    +--MaxBatchItems  
    +--MaxProviderRequests  
    +--Client  
      |  +--NetworkDelayms  
      |  +--URLPrefix  
      |  +--AllowUnencrypted  
      |  +--Auth  
      |    |  +--Basic  
      |    |  +--Digest  
      |    |  +--Kerberos  
      |    |  +--Negotiate  
    ...
```

Shows all the containers and items in the WSMAN: drive.

Example 4

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

```
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  
...  
...
```

Show a tree listing with files including a few user-specified properties in color. This example is using parameter and command aliases.

Parameters

-Path

The path to the root of the tree that will be shown.

```
Type: String[]
Parameter Sets: Path
Aliases: FullName

Required: False
Position: 1
Default value: current location
Accept pipeline input: True (ByPropertyName, ByValue)
Accept wildcard characters: False
```

-LiteralPath

Use a literal path value.

```
Type: String[]
Parameter Sets: LiteralPath
Aliases:

Required: False
Position: 1
Default value: None
Accept pipeline input: True (ByPropertyName)
Accept wildcard characters: False
```

-Depth

Specifies how many levels of the specified path are recursed and shown.

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: False
Position: 2
Default value: 2147483647
Accept pipeline input: False
Accept wildcard characters: False
```

-IndentSize

The size of the indent per level. The default is 3. The minimum value is 1. You shouldn't have to modify this parameter.

Type: **Int32**
Parameter Sets: ([All](#))

Aliases:

Required: False
Position: Named
Default value: 3
Accept pipeline input: False
Accept wildcard characters: False

-ShowItem

Shows the items in each container or folder.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases: files

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-ShowProperty

Shows the properties on containers and items. Use * to display all properties otherwise specify a comma separated list.

Type: **String[]**
Parameter Sets: ([All](#))
Aliases: properties

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False

-InColor

Show tree and item colorized. Values are from the \$PSAnsMap variable.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases: ansi

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[tree.com](#)

[Get-ChildItem](#)

Test-EmptyFolder

Synopsis

Test if a folder is empty of files.

Syntax

```
Test-EmptyFolder [-Path] <String[]> [-Passthru] [<CommonParameters>]
```

Description

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. See examples.

Examples

Example 1

```
PS C:\> Test-EmptyFolder c:\work
False
```

Test a single folder from a parameter.

Example 2

```
PS C:\> Get-ChildItem c:\work -Directory | Test-EmptyFolder -passthru
```

| Path | Name | IsEmpty | Computername |
|---------------|-------|---------|--------------|
| C:\work\A | A | False | DESK10 |
| C:\work\alpha | alpha | False | DESK10 |
| C:\work\B | B | True | DESK10 |
| C:\work\data | data | False | DESK10 |
| C:\work\demo3 | demo3 | True | DESK10 |
| C:\work\demos | demos | False | DESK10 |
| ... | | | |

Test child folders under C:\work.

Example 3

```
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.

Parameters

-Passthru

Write a test object to the pipeline.

Type: **SwitchParameter**
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Path

Enter a file system path like C:\Scripts.

Type: **String[]**
Parameter Sets: ([All](#))
Aliases: PSPath

Required: True
Position: 0
Default value: None
Accept pipeline input: True ([ByPropertyName](#), [ByValue](#))
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String[]

Outputs

Boolean

EmptyFolder

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-FolderSizeInfo](#)

Test-Expression

Synopsis

Test a PowerShell expression over a period of time.

Syntax

Interval (Default)

```
Test-Expression [-Expression] <ScriptBlock> [-ArgumentList <Object[]>]
[-Count <Int32>] [-Interval <Double>] [-IncludeExpression] [-AsJob]
[<CommonParameters>]
```

Random

```
Test-Expression [-Expression] <ScriptBlock> [-ArgumentList <Object[]>]
[-Count <Int32>] -RandomMinimum <Double> -RandomMaximum <Double>
[-IncludeExpression] [-AsJob] [<CommonParameters>]
```

Description

This command will test a PowerShell expression or scriptblock for a specified number of times and calculate the average runtime, in milliseconds, over all the tests. The output will also show the median and trimmed values.

The median is calculated by sorting the values in ascending order and selecting the value in the center of the array. If the array has an even number of elements then the median is the average of the two values in the center. The trimmed value will toss out the lowest and highest values and average the remaining values. This may be the most accurate indication as it will eliminate any small values which might come from caching and any large values which may come a temporary shortage of resources. You will only get a value if you run more than 1 test.

Examples

Example 1

```

PS C:\> $cred = Get-credential globomantics\administrator
PS C:\> $c = "chi-dc01","chi-dc04"
PS C:\> Test-Expression {
    param ([string[]]$computer,$cred)
    get-wmiobject win32_logicaldisk -computernamespace $computer -credential $cred
} -argumentList $c,$cred

Tests      : 1
TestInterval : 0.5
AverageMS   : 1990.6779
MinimumMS   : 1990.6779
MaximumMS   : 1990.6779
MedianMS    :
TrimmedMS   :
PSVersion   : 5.1.19041.1
OS          : Microsoft Windows 10 Pro

```

Test a command once passing an argument to the scriptblock. There is no TrimmedMS value because there was only one test.

Example 2

```

PS C:\> $sb = {1..1000 | Foreach-Object {$_*2}}
PS C:\> Test-Expression $sb -count 10 -interval 2

Tests      : 10
TestInterval : 2
AverageMS   : 72.78199
MinimumMS   : 29.4449
MaximumMS   : 110.6553
MedianMS    :
TrimmedMS   :
PSVersion   : 5.1.19041.1
OS          : Microsoft Windows 10 Pro

PS C:\> $sb2 = { foreach ($i in (1..1000)) {$_*2}}
PS C:\> Test-Expression $sb2 -Count 10 -interval 2

Tests      : 10
TestInterval : 2
AverageMS   : 6.40283
MinimumMS   : 0.7466
MaximumMS   : 22.968
MedianMS    :
TrimmedMS   :
PSVersion   : 5.1.19041.1
OS          : Microsoft Windows 10 Pro

```

These examples are testing two different approaches that yield the same results over a span of 10 test runs, pausing for 2 seconds between each test. The values for Average, Minimum, and Maximum are in milliseconds.

Example 3

```
PS C:\> Test-Expression {
    Param([string]$computer)
    Get-Service bits,wuauserv,winrm -computername $computer
} -count 5 -IncludeExpression -argumentList chi-hvr2

Tests      : 5
TestInterval : 500
AverageMS   : 15.53376
MinimumMS    : 11.6745
MaximumMS    : 24.9331
MedianMS     : 13.8928
TrimmedMS    : 13.68706666666667
PSVersion    : 5.1.19041.1
OS          : Microsoft Windows 10 Pro
Expression   : Param([string]$computer) get-service bits,wuauserv,winrm -com...
Arguments    : {chi-hvr2}
```

Include the tested expression in the output.

Example 4

```
PS C:\> $j=Test-Expression { get-eventlog -list } -count 10 -Interval 5 -AsJob
PS C:\> $j | Receive-Job -keep

Tests      : 10
TestInterval : 5
AverageMS   : 2.80256
MinimumMS    : 0.7967
MaximumMS    : 14.911
MedianMS     : 1.4469
TrimmedMS    : 1.5397375
PSVersion    : 5.1.19041.1
OS          : Microsoft Windows 10 Pro
RunspaceId   : f30eb879-fe8f-4ad0-8d70-d4c8b6b4eccc
```

Run the test as a background job. When the job is complete, get the results.

Example 5

```
PS C:\>{1..1000} | Test-Expression -count 10 -RandomMinimum 1 -RandomMaximum 10

Tests      : 10
TestInterval : Random
AverageMS   : 0.63899
MinimumMS    : 0.2253
MaximumMS    : 3.9062
MedianMS     : 0.24475
TrimmedMS    : 0.2823
PSVersion    : 5.1.19041.1
OS          : Microsoft Windows 10 Pro
```

Pipe a scriptblock to be tested.

Parameters

-ArgumentList

An array of parameters to pass to the test scriptblock. Arguments are positional. If passing an array for a value enter with @().

```
Type: Object[]
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-AsJob

Run the tests as a background job.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: False
Accept pipeline input: False
Accept wildcard characters: False
```

-Count

The number of times to test the scriptblock.

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: 1
Accept pipeline input: True (ByPropertyName)
Accept wildcard characters: False
```

-Expression

The scriptblock you want to test.

```
Type: ScriptBlock
Parameter Sets: (All)
Aliases: sb

Required: True
Position: 0
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-IncludeExpression

Include the test scriptblock in the output.

```
Type: SwitchParameter
Parameter Sets: (All)
Aliases: ie

Required: False
Position: Named
Default value: False
Accept pipeline input: True (ByPropertyName)
Accept wildcard characters: False
```

-Interval

How much time to sleep in seconds between each test. The maximum value is 60. You may want to use a sleep interval to mitigate possible caching effects.

```
Type: Double
Parameter Sets: Interval
Aliases: sleep

Required: False
Position: Named
Default value: 0.5
Accept pipeline input: True (ByPropertyName)
Accept wildcard characters: False
```

-RandomMaximum

You can also specify a random interval by providing random minimum and maximum values in seconds.

```
Type: Double
Parameter Sets: Random
Aliases: max

Required: True
Position: Named
Default value: 0
Accept pipeline input: False
Accept wildcard characters: False
```

-RandomMinimum

You can also specify a random interval by providing random minimum and maximum values in seconds.

```
Type: Double  
Parameter Sets: Random  
Aliases: min  
  
Required: True  
Position: Named  
Default value: 0  
Accept pipeline input: False  
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

scriptblock

Outputs

TestResult

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/> This command was first described at <https://github.com/jdhitsolutions/Test-Expression/blob/master/docs/Test-Expression.md>)

Related Links

[Measure-Command](#)

[Test-ExpressionForm](#)

Test-ExpressionForm

Synopsis

Display a graphical test form for Test-Expression.

Syntax

```
Test-ExpressionForm [CommonParameters]
```

Description

This command will display a WPF-based form that you can use to enter in testing information. Testing intervals are in seconds. All of the values are then passed to the Test-Expression command. Results will be displayed in the form. The results only show you how long the tests took, regardless of whether or not there were errors.

When you close the form, the last result object will be passed to the pipeline, including all metadata, the scriptblock, and arguments.

This command requires a Windows platform that supports WPF.

Examples

Example 1

```
PS C:\> test-expressionform
```

Launch the form.

Parameters

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/> This command was first explained at <https://github.com/jdhitsolutions/Test-Expression/blob/master/docs/Test-ExpressionForm.md>

Related Links

[Test-Expression](#)

[Measure-Command](#)

Test-IsPSWindows

Synopsis

Test if running PowerShell on a Windows platform.

Syntax

```
Test-IsPSWindows [CommonParameters]
```

Description

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.

Examples

Example 1

```
PS C:\> Test-IsPSWindows
True
```

Parameters

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.Boolean

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Test-WithCulture

Synopsis

Test your PowerShell code using a different culture.

Syntax

scriptblock (Default)

```
Test-WithCulture [-Culture] <CultureInfo> [-Scriptblock] <ScriptBlock>
[-ArgumentList <Object[]>] [<CommonParameters>]
```

file

```
Test-WithCulture [-Culture] <CultureInfo> -FilePath <ScriptBlock>
[-ArgumentList <Object[]>] [<CommonParameters>]
```

Description

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.

Note that this command is not an absolute test. There may be commands that fail to produce the alternate culture results you expect.

Examples

Example 1

```
PS C:\> Test-WithCulture de-de -Scriptblock {($Get-Date).addDays(90)}
Montag, 14. Oktober 2020 08:59:01
```

Example2

```
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.ToString("yyyy-MM-dd") } -noElement
}

Count Name
-----
165 10/07/2020
249 11/07/2020
17 12/07/2020
16 13/07/2020
20 14/07/2020
26 15/07/2020
7 16/07/2020
```

Parameters

-ArgumentList

Specify an array of positional arguments to pass to the scriptblock for file.

Type: **Object[]**
Parameter Sets: (All)
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Culture

Enter a new culture like de-de

Type: **CultureInfo**
Parameter Sets: (All)
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-FilePath

Enter the path to a PowerShell script file to execute using the specified culture.

Type: **ScriptBlock**
Parameter Sets: file
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-Scriptblock

Enter a scriptblock to execute using the specified culture. Be aware that long or complex pipelined expressions might not give you the culture-specific results you expect.

Type: **ScriptBlock**
Parameter Sets: scriptblock
Aliases:

Required: True
Position: 1
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

None

Outputs

System.Object

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Get-Culture](#)

[Get-UICulture](#)

Trace-Message

Synopsis

Create a graphical trace window.

Syntax

message (Default)

```
Trace-Message [[-Message] <String>] [<CommonParameters>]
```

init

```
Trace-Message [-Title <String>] [-BackgroundColor <String>] [-Width <Int32>]  
[-Height <Int32>] [<CommonParameters>]
```

Description

Trace-Message is designed to be used with script or function. Its purpose is to create a graphical trace window using Windows Presentation Foundation. Inside the function or script, you can use this command to send messages to the window. When finished, you have an option to save the output to a text file.

There are 3 steps to using this function. First, in your code, you need to create a boolean global variable called `TraceEnabled`. When the value is `$True`, the `Trace-Message` command will run. When set to false, the command will be ignored. Second, you need to initialize a form, specifying the title and dimensions. The form will automatically include some pre-defined metadata. Finally, you can send trace messages to the window. All messages are prepended with a timestamp.

This command is not optimized for performance and is intended for development purposes. When your code is finished, you can set `$TraceEnabled` to `$False`. If you need to troubleshoot, you can set it to `$True`.

Examples

Example 1

```
PS C:\> Trace-Message -title "Troubleshooting Log" -width 600
```

This command will initialize a trace window with the given title and width. It is assumed you have set `$TraceEnabled` to `$True`. This is a command you would normally run in your code and not from the console.

Example 2

```
PS C:\> Trace-Message -message "Starting MyCommand"
```

This example is a continuation of the previous example. The message text will be appended to the graphical form, prepended with a timestamp.

Parameters

-BackgroundColor

Specify a background color for the trace window. You can use console colors like "Cyan" or HTML color codes.

```
Type: String
Parameter Sets: init
Aliases:

Required: False
Position: Named
Default value: "#FFFFF8DC",
Accept pipeline input: False
Accept wildcard characters: False
```

-Height

Specify the Width of the trace window.

```
Type: Int32
Parameter Sets: init
Aliases:

Required: False
Position: Named
Default value: 500
Accept pipeline input: False
Accept wildcard characters: False
```

-Message

Specify a message to write to the trace window.

```
Type: String
Parameter Sets: message
Aliases:

Required: $True
Position: 0
Default value: None
Accept pipeline input: True (ByValue)
Accept wildcard characters: False
```

-Title

Specify a title for the trace window.

```
Type: String
Parameter Sets: init
Aliases:

Required: False
Position: Named
Default value: "Trace Messages"
Accept pipeline input: False
Accept wildcard characters: False
```

-Width

Specify the Width of the trace window.

```
Type: Int32
Parameter Sets: init
Aliases:

Required: False
Position: Named
Default value: 800
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.String

Outputs

None

Notes

Look at \$PSSamplePath\Get-Status.ps1 for a demonstration of this command in a function. The buttons have key accelerators of Q and S.

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[Write-Verbose](#)

Write-ANSIProgress

Synopsis

Display an ANSI progress bar.

Syntax

```
Write-ANSIProgress [-PercentComplete] <Double> [-ProgressColor <String>]
[-BarSymbol <String>] [-Position <Coordinates>] [<CommonParameters>]
```

Description

You can use this command to write an ANSI colored progress bar to the console. The output will be an array of strings. The item may be a blank line. See examples.



If you are using the Windows Terminal and are at the bottom of the screen, you may get improperly formatted results. Clear the host and try again.

Examples

Example 1

```
PS C:\> $pct = @(.10, .12, .19, .25, .43, .55, .66, .78, .90, .95, 1)
PS C:\> $pct | Write-ANSIProgress -BarSymbol Block
```

This will build a progress bar using a block symbol and the default ANSI color escape.

Example 2

```
PS C:\> $params = @{
    PercentComplete = .78
    BarSymbol = "Circle"
    "ProgressColor" = "$([char]0x1b)[92m"
}
PS C:\> Write-ANSIProgress @params
```

Create a single progress bar for 78% using the Circle symbol and a custom color.

Example 3

```
PS C:\> Get-CimInstance -ClassName win32_operatingsystem |
Select-Object -property @{N="Computername";E={$_.CSName}},
@{N="TotalMemGB";E=[Format-Value $_.TotalVisibleMemorySize -unit MB]},
@{N="FreeMemGB";E=[Format-Value $_.FreePhysicalMemory -unit MB]},
@{N="PctFree"; E={$pct=Format-Percent $_.FreePhysicalMemory / $_.TotalVisibleMemorySize}}
Write-ANSIProgress -PercentComplete ($pct/100) | Select-Last 1
}}
```

| Computername | TotalMemGB | FreeMemGB | PctFree |
|--------------|------------|-----------|---------|
| BOVINE320 | 32 | 12 | 37.87% |

Note that this example is using abbreviations in the Select-Object hashtables.

Example 4

```
PS C:\> $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)" -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
}
PS C:\> Invoke-Command -ScriptBlock $sb
```

You are most likely to use this command in a function or script. This example demonstrates using a script block.

Parameters

-BarSymbol

Specify what shape to use for the progress bar.

Type: `String`
Parameter Sets: ([All](#))
Aliases:
Accepted values: Box, Block, Circle

Required: False
Position: Named
Default value: Box
Accept pipeline input: False
Accept wildcard characters: False

-PercentComplete

Enter a percentage in decimal value like .25 up to 1.

Type: `Double`
Parameter Sets: ([All](#))
Aliases:

Required: True
Position: 0
Default value: None
Accept pipeline input: True ([ByValue](#))
Accept wildcard characters: False

-Position

Specify the cursor position or where you want to place the progress bar.

Type: `Coordinates`
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: Current position
Accept pipeline input: False
Accept wildcard characters: False

-ProgressColor

Specify an ANSI escape sequence for the progress bar color.

Type: `String`
Parameter Sets: ([All](#))
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

Inputs

System.Double

Outputs

System.String

Notes

This command will not work in the PowerShell ISE. The verbose output should only be used when troubleshooting a display problem.

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

[New-ANSIBar](#)

[New-RedGreenGradient](#)

Write-Detail

Synopsis

Write a detailed message string.

Syntax

Default (Default)

```
Write-Detail [[-Message] <String>] [-Prefix <String>] [<CommonParameters>]
```

Time

```
Write-Detail [[-Message] <String>] [-Prefix <String>] [-Time]
[<CommonParameters>]
```

Date

```
Write-Detail [[-Message] <String>] [-Prefix <String>] [-Date]
[<CommonParameters>]
```

Description

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 Begin, Process, and End scriptblocks. You can create a detailed message to indicate what part of the code is being executed. The output can include a full-time stamp, or a time string which includes a millisecond value.

In a script you might use it like this in a Begin block:

```
$pfx = "BEGIN"

Write-Detail "Starting $($MyInvocation.MyCommand)" -Prefix $pfx | Write-Verbose

Write-Detail "PS $($PSVersionTable.PSVersion)" -Prefix $pfx | Write-Verbose
```

If you don't specify a prefix, it will default to PROCESS.

Examples

EXAMPLE 1

```
PS C:\> Write-Detail "Getting file information" -Prefix Process
[PROCESS] Getting file information
```

Normally you would use this command in a function, but here is an example from the console so that you can see what to expect.

Parameters

-Message

The message to display after the time stamp and prefix.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Prefix

Indicate whether you are in the BEGIN, PROCESS, or END script block. Although you can specify any text. It will be displayed in upper case.

```
Type: String
Parameter Sets: (All)
Aliases:

Accepted values:

Required: False
Position: Named
Default value: PROCESS
Accept pipeline input: False
Accept wildcard characters: False
```

-Date

Display a date value like 9/15/2020 11:36:41.

```
Type: SwitchParameter
Parameter Sets: Date
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-Time

Display a time value with milliseconds like 11:37:01:4029.

```
Type: SwitchParameter
Parameter Sets: Time
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see about_CommonParameters (<http://go.microsoft.com/fwlink/?LinkID=113216>).

Inputs

None

Outputs

System.String

Notes

Learn more about PowerShell: <http://jdhitsolutions.com/blog/essential-powershell-resources/>

Related Links

Write-Verbose

Change Log for PSScriptTools

This is the most recent change history for this module.

v2.34.0

- Fixed typo bug in `Get-PSScriptTools` that was failing to get command aliases. (Issue #99)
- Modified `Get-PSScriptTools` to improve performance. Assuming that all exported functions use standard verbs.
- Added `Get-PSAnsifileMap`.
- Added `Set-PSAnsifileMapEntry`.
- Added `Remove-PSAnsifileMapEntry`.
- Added `Export-PSAnsifileMap`.
- Added `Show-ANSISequence`.
- Updated `filesystem.ansi.format.ps1xml` to use last matching pattern.
- Modified `Show-Tree` to better handle piped-in file and directory objects.
- Added an alias `ab` for `Add-Border`.
- Added an alias of `nab` for `New-AnsiBar`.
- Updated `README.md`.
- Help updates.

v2.33.1

- Fixed bug in `ConvertTo-WPFGrid` with refresh and timeout values. (Issue #98)
- Added missing online help links.
- Added a few related module links in `README.md`.

v2.33.0

- Added `Select-Before`, `Select-After`, `Select-Newest` and `Select-Oldest` and their respective aliases of `before`, `after`, `newest`, and `oldest`.
- Added `Get-MyCounter` and a custom format file `mycounter.format.ps1xml`.
- Added `Trace-Message` and its alias `trace`.
- Added more Verbose messages to `Get-PSScriptTools`.
- Code cleanup in `SelectFunctions.ps1`.
- Modified `Get-PSScriptTools` to let you specify a verb. Updated command help.
- Modified `ConvertTo-Markdown` to handle properties with line returns when formatting as a table. (Issue #97)
- Code cleanup in sample script files.
- Added sample file `CounterMarkdown.ps1`.
- Updated `README.md`.

v2.32.0

-
- Added `ConvertTo-ASCIIArt` and its alias `cart`.
 - Added `Get-DirectoryInfo`, its alias `dw`, and a custom formatting file, `directorystat.format.ps1xml`.
 - Modified `Open-PSScriptToolsHelp` to use `Invoke-Item` to launch the PDF file. This should work better on non-Windows platforms.
 - Modified `Get-FormatView` to accept pipeline input for the `TypeName` parameter. (Issue #95)
 - Modified `New-PSFormatXML` to use a static value width when using scriptblocks. (Issue #94)
 - Added `Out-Copy` and its alias `oc`.
 - Added `Get-CommandSyntax` and its alias `gsyn`.
 - Updated missing online help links.
 - Added a splash header to `Get-PSScriptTools`. The header writes to the host so it isn't part of the command output.
 - Updated `README.md`.

Archive

If you need to see older change history, look at the [Archive ChangeLog](#)