PSFunctionTools Help[1]

https://jdhitsolutions.github.io

v1.3.0

[1] https://github.com/jdhitsolutions/PSFunctionTools

Contents

PSFunctionTools

The commands in this module have been developed to make it easier to automate the PowerShell scripting and module development. These tools were first described in a series of blog posts.

- Exporting PowerShell Functions to Files
- Converting PowerShell Scripts to Functions
- Discovering Aliases with the PowerShell AST
- Fun with PowerShell Module Layout
- Building a PowerShell Module Inception-Style

This module has been written for **PowerShell 7.4** and later. It is most likely that the the commands will work in Windows PowerShell, but you will need to fork this module and revise as necessary. Otherwise, install this module from the PowerShell Gallery.

```
Install-Module PSFunctionTools
```

or

```
Install-PSResource PSFunctionTools
```

Commands

To see a summary of these commands at any time, run Get-PSFunctionTools

```
PS C:\> Get-PSFunctionTools

   Module: PSFunctionTools [v1.3.0]

Name                           Alias     Synopsis
----                           -----     --------
Convert-ScriptToFunction        csf      Convert a script file to a PowerShell fu...
Export-FunctionFromFile         eff      Export a PowerShell function from a scri...
Export-FunctionToFile           etf      Export a PowerShell function to a file.
Export-ModuleLayout             eml      Export a model module layout.
Format-FunctionName             ffn      Format a function name to proper case.
Get-FunctionAlias            {ga, gfal}  Get a defined function alias.
```

```
Get-FunctionAttribute        gfa     Get function attributes like cmdletbinding.
Get-FunctionName             gfn     Identify the names of PowerShell functio…
Get-FunctionProfile          gfp     Get a technical summary of a PowerShell …
Get-ModuleLayout                     Get information about a module layout file.
Get-ParameterBlock           gpb     Get a function's parameter block.
Get-PSFunctionTools                  Get a summary of PSFunctionTools commands.
Get-PSRequirements                   List PowerShell command requirements.
Import-ModuleLayout          iml     Create a module structure from a layout …
New-CommentHelp              nch     Create comment based help.
New-ModuleFromFiles                  Create a PowerShell module from a set of …
New-ModuleFromLayout                 Create a new module based on a layout.
Test-FunctionName            tfn     Test the validity of a PowerShell function…
```

Convert-ScriptToFunction

This command takes the body of a script file and wraps it in a function declaration. The command will insert missing elements like `cmdletbinding()` and comment-based help. You will most likely need to edit and clean up the result in your scripting editor. If you run this command in the PowerShell ISE or the VS Code PowerShell integrated terminal, you can use the dynamic parameter `ToEditor` to open a new file with with the output. You can edit and save the file manually.

```
Convert-ScriptToFunction c:\scripts\systemreport.ps1 -name New-SystemReport |
Out-File c:\scripts\New-SystemReport.ps1
```

It is assumed that your script file is complete and without syntax errors.

Export-FunctionFromFile

You should use `Export-FunctionFromFile` when you want to export PowerShell functions defined in in a single script file, placing each function in its own file. You might want to do this to build or restructure a PowerShell module.

You can export all functions from a file or specific functions. The default behavior is to only export functions that follow a standard verb-noun naming convention. The source must be a .ps1 or .psm1 script file.

```
Export-FunctionFromFile C:\scripts\MyInternetTools.psm1 -Name get-zipinfo `
-OutputPath c:\scripts\psinternettools\functions
```

If you run this command in the PowerShell ISE or the VS Code integrated PowerShell Terminal, you can use the dynamic parameter `Remove` to delete the function from the source file.

Export-FunctionToFile

You +can use this command to export a function which is loaded into your PowerShell session. You might need to do this when you create an ad-hoc function and want to save it to a file. This command will take the content of the function and export it to a ps1 file. The function name will be used for the file name. Although, characters like the colon will be stripped to create a filesystem-compatible filename.

You can also include `#Requires` statements.

```
Export-FunctionToFile -Name New-FileLink -Path c:\scripts -Requires
"#requires -version 5.1","#requires -RunAsAdministrator"
```

Export-ModuleLayout

Use `Export-ModuleLayout` to export a model module directory structure to a json file. You can use `Import-ModuleLayout` to recreate the layout from the json file. The export process will include not only directories, but also text files like a readme or license file.

```
PS C:\> Export-ModuleLayout c:\work\sample -FilePath c:\work\layout.json
-Verbose
VERBOSE: Starting Export-ModuleLayout
VERBOSE: Exporting directory structure from c:\work\sample
VERBOSE: Processing .github
VERBOSE: Processing .vscode
VERBOSE: Processing docs
VERBOSE: Processing en-us
VERBOSE: Processing formats
VERBOSE: Processing functions
VERBOSE: Processing icons
VERBOSE: Processing images
VERBOSE: Processing samples
VERBOSE: Processing tests
VERBOSE: Processing types
VERBOSE: Processing changelog.md
VERBOSE: Processing License.txt
VERBOSE: Processing README.md
VERBOSE: Processing scratch-changelog.md
VERBOSE: Processing .vscode\tasks.json
VERBOSE: Processing formats\readme.txt
VERBOSE: Processing functions\private
VERBOSE: Processing functions\public
VERBOSE: Processing functions\private\readme.txt
VERBOSE: Processing functions\public\readme.txt
VERBOSE: Processing tests\readme.txt
VERBOSE: Processing types\readme.txt
```

```
VERBOSE: Exporting module layout to c:\work\layout.json.
```

Format-FunctionName

`Format-FunctionName` is intended to be used as a helper function in your scripting automation. This is a simple function that will format a verb-noun function name into proper case. It will take an input such as `test-data` and format it as `Test-Data`. It will not format as PascalCase. The command also will not verify that the verb component is acceptable. Use `Test-FunctionName` for that process.

```
PS C:\> Format-FunctionName test-data
Test-Data
```

Get-FunctionAlias

`Get-FunctionAlias` is a tool you can use in your scripting automation. It will extract function names and aliases from a PowerShell script file. The source must be a .ps1 or .psm1 file. The command will only identify aliases defined as part of the function using code like `[alias('foo')]`.

```
PS C:\> Get-FunctionAlias -Path C:\scripts\SQLBackup.psm1

Name                 Alias
----                  -----
Backup-SQLDatabase   Backup-SQL
Restore-SQLdatabase rsql
```

Get-FunctionAttribute

This command can be used to get function attributes such as cmdletbinding or alias settings.

```
Get-FunctionAttribute -path c:\scripts\PSFunctionTools\functions\public\
Get-ParameterBlock.ps1 -Name Get-ParameterBlock

Type                : cmdletbinding
NamedArguments      : {}
PositionalArguments : {}
String              : [cmdletbinding()]
Function            : Get-ParameterBlock
Path                : C:\scripts\PSFunctionTools\functions\public\
                      Get-ParameterBlock.ps1

Type                : alias
```

```
NamedArguments       : {}
PositionalArguments : {"gpb"}
String               : [alias("gpb")]
Function             : Get-ParameterBlock
Path                 : C:\scripts\PSFunctionTools\functions\public\
                       Get-ParameterBlock.ps1

Type                 : OutputType
NamedArguments       : {}
PositionalArguments : {"ParamBlockAst", "String"}
String               : [OutputType("ParamBlockAst","String")]
Function             : Get-ParameterBlock
Path                 : C:\scripts\PSFunctionTools\functions\public\
                       Get-ParameterBlock.ps1
```

Get-FunctionName

When exporting functions from files, you may only want to export specific functions, which you can do if you know the name. Use `Get-FunctionName` to identify the names of functions in a script file. The default behavior is to get names of functions that follow the verb-noun naming convention.

```
PS C:\> Get-FunctionName C:\scripts\MyInternetTools.psm1
Get-MyWhoIs
Get-GeoIP
Get-MyPublicIP
Get-MyWeather
Get-WeatherByProxy
Get-WeatherLocation
Get-QOTD
Get-ZipInfo
Get-RSSFeed
Open-URL
```

Get-ModuleLayout

This command will provide information about a module layout folder which was created using `Export-ModuleLayout`. The default output is custom object. You can elect to view the layout as a tree. This parameter requires the tree commandline utility which should be available on Windows systems by default. On non-Windows platforms, you may need to install the utility.

```
PS C:\> Get-ModuleLayout C:\scripts\simplelayout.json -AsTree
C:\<PathTo>\<MYMODULE>
```

```
|   changelog.md
|   README.md
|
+---.vscode
+---docs
+---en-us
+---formats
|       readme.txt
|
+---functions
|
+---tests
|       readme.txt
|
\---types
        readme.txt
```

Get-ParameterBlock

This command is designed to use the PowerShell AST and retrieve a function's parameter block. You might use this to build comment-based help.

```
PS C:\> Get-ParameterBlock -path c:\scripts\SimpleFunction.ps1 -name
Get-FolderData

Attributes Parameters                  Extent
---------- ----------                  ------
{}         {$Path, $Cutoff, $Filter} Param (...

PS C:\> Get-ParameterBlock  -path c:\scripts\SimpleFunction.ps1 -name
Get-FolderData -ToString
[parameter(HelpMessage = "Specify the folder to analyze")]
[string]$Path="."
[datetime]$Cutoff
[string]$Filter="*.*"
```

Get-PSRequirements

As part of your scripting automation, you may want to capture requirements defined in a script file such as `# requires -version 5.1`. The command `Get-PSRequirements` will process a PowerShell script file for these type of requirements.

```
PS C:\> Get-PSRequirements -Path C:\scripts\SQLBackup.psm1
```

```
Path                   : C:\scripts\SQLBackup.psm1
RequiredApplicationId :
RequiredPSVersion      : 5.1
RequiredPSEditions     : {}
RequiredModules        : {}
RequiresPSSnapIns      : {}
RequiredAssemblies     : {}
IsElevationRequired    : True
```

Import-ModuleLayout

Use `Import-ModuleLayout` to recreate a module structure from a json file created with `Export-ModuleLayout`. Importing the json file will recreate the folders and files.

```
PS C:\> Import-ModuleLayout -Name PSDemo -ParentPath D:\scripts -Layout C:\work\layout.json

    Directory: C:\scripts\PSDemo

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          12/16/2024   9:45 AM                types
d----          12/16/2024   9:45 AM                .github
d----          12/16/2024   9:45 AM                .vscode
d----          12/16/2024   9:45 AM                docs
 ...
    Directory: D:\scripts\PSDemo\functions

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          12/16/2024   9:45 AM                public
d----          12/16/2024   9:45 AM                private

    Directory: D:\scripts\PSDemo\functions\public

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          12/16/2024   9:49 AM            276 readme.txt
 ...
```

New-CommentHelp

You can use this command in your scripting automation to generate a comment-based help block for a function. The function will use the parameter block which you can get with `Get-ParameterBlock` to define help parameters. If your parameter has a

HelpMessage defined, the value will be used in the parameter description. You can also specify a synopsis and/or description. Otherwise, you can edit the placeholders later.

```
PS C:\> Get-ParameterBlock  -path c:\scripts\SimpleFunction.ps1 -name Get-FolderData | New-
<
    .Synopsis
      Get folder details
    .Description
      <long description>
    .Parameter Path
      Specify the folder to analyze
    .Parameter Cutoff
      <enter a parameter description>
    .Parameter Filter
      <enter a parameter description>
    .Example

      <output and explanation>
    .Inputs
      <Inputs to this function (if any)>
    .Outputs
      <Output from this function (if any)>
    .Notes
      <General notes>
    .Link
      <enter a link reference>
```

New-ModuleFromFiles

`New-ModuleFromFiles` is an ***experimental*** command. It is *not* guaranteed to run without error and may change significantly between module versions. The command is designed to process a collection of PowerShell script files which contain PowerShell functions. Each function will be exported to an individual file to a location you specify.

The function relies on a module layout file to scaffold the module directory.

```
$splat = @{
    Description   = "Demo exported module"
    Files         = "c:\scripts\pstools.psm1","c:\scripts\servertools.ps1"
    Layout        = "c:\scripts\ModuleLayout.json"
    NewModuleName = "PSTools"
    ParentPath    = "c:\scripts"
    CreateHelp    = $True
    FunctionPath  = "functions\public"
    InitializeGit = $true
}
 New-ModuleFromFiles @splat
```

If you have the Platyps module installed, you can also choose to create help documentation. If you have `git` installed, you can initialize the module as a git repository. This process will also checkout a new branch.

New-ModuleFromLayout

This command is very similar to `New-ModuleFromFiles`. That function builds a module structure from existing files. This function creates a new module but without defining any commands. `New-ModuleFromLayout` will still create a module structure based on a layout and it will still create module files. Specifically,the module manifest and root module files.

```
New-ModuleFromLayout -NewModuleName PSDataResource -ParentPath c:\scripts -Description "A c
```

If `git.exe` is detected, you can use the `InitializeGit` dynamic parameter to initialize the module as a git repository.

Test-FunctionName

PowerShell function names should follow naming convention of `Verb-Noun`. The verb should be a standard verb that you see with `Get-Verb`. Use this command in your scripting automation to validate a PowerShell function name.

```
PS C:\> Test-FunctionName Test-Widget
Test-Widget
```

If the name passes validation it will be written to the pipeline. Or you can use the `-Quiet` parameter to return a traditional boolean result.

```
PS C:\> Test-FunctionName kill-system -Quiet
False
```

Get-FunctionProfile

`Get-FunctionProfile` is designed to give you a technical summary of a PowerShell function. You might use this to preview what commands a function might execute or if it supports `-WhatIf`. The function might be something someone else wrote, or perhaps you want to double-check your code.

Note that the analysis may not be 100% accurate. For example, it is difficult to distinguish between the alias `foreach` and the `foreach` enumerator.

```
PS C:\...\samples> Get-FunctionProfile -path .\SampleScript5.ps1 -name Get-Result

Name                    : Get-Result
FunctionAlias           : grx
SupportsShouldProcess   : False
ParameterSets           :
DynamicParameters       : False
RequiredVersion         : 5.1
RequiredModules         : {}
RequiresElevation       : True
Commands                : {Get-CimInstance, Get-Random, Join-Path, New-Timespan…}
ExternalCommands        : {c:\scripts\cleanup.bat, notepad.exe}
DotNet                  : {[system.datetime]::now,
                          [system.environment]::getenvironmentvariable("temp")}
Aliases                 : {gcim, tee}
Unresolved              : {w}
Path                    : C:\Scripts\PSFunctionTools\samples\SampleScript5.ps1
```

Here is a sample analysis. Commands should be PowerShell cmdlets, including resolved aliases. Detected command aliases will also be retrieved. Unresolved commands might be undefined aliases or some other command that PowerShell could not resolve.

Code Samples

This module includes a Samples folder. Here, you can find sample PowerShell scripts and functions that you can use with the commands in this module. You can use the `Open-PSFunctionToolsSamples` command to change to the samples folder and list the contents.

```
PS C:\> Open-PSFunctionToolsSamples

        Directory: C:\Program Files\PowerShell\Modules\PSFunctionTools\1.3.0\samples


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---           1/13/2025   5:25 PM           180 BuildModule.ps1
-a---           1/13/2025   5:29 PM           203 Demo-ExportFunctions.ps1
-a---           1/13/2025   6:28 PM          1005 Demo-NewModuleFromFiles.ps1
-a---           1/13/2025   5:26 PM          4146 Get-ZeroSize.ps1
-a---           1/13/2025   5:32 PM          4278 ModuleLayout.json
-a---           1/13/2025   5:36 PM          1574 POC-NewModule.ps1
-a---           1/13/2025   5:36 PM          3941 POC-NewModule2.ps1
```

```
-a---          4/18/2023   9:08 PM           1371 samplefunction.ps1
-a---          4/18/2023   9:08 PM            206 SampleScript.ps1
-a---          4/18/2023   9:08 PM            577 SampleScript2.ps1
-a---          4/18/2023   9:08 PM            503 SampleScript3.ps1
-a---          4/18/2023   9:08 PM           1371 SampleScript4.ps1
-a---          4/18/2023   9:08 PM           1536 SampleScript5.ps1
-a---          1/13/2025   6:16 PM          15148 Tools.psm1
```

Or, once you know the path, you can use the sample files to try out the module commands.

```
PS C:\ ... \PSFunctionTools\samples> Get-FunctionName .\Tools.psm1
Get-WindowsVersion
Get-WindowsVersionString
Get-OSInfo

PS C:\ ... \PSFunctionTools\samples>Get-ModuleLayout .\ModuleLayout.json -AsTree

C:\<PathTo>\<MYMODULE>
|   changelog.md
|   README.md
|
+---.vscode
+---docs
+---en-us
+---formats
|       readme.txt
|
+---functions
|   +---private
|   |       readme.txt
|   |
|   \---public
|           readme.txt
|
+---tests
|       readme.txt
|
\---types
        readme.txt

PS C:\Scripts\PSFunctionTools\samples> dir .\SampleScript* |
Get-PSRequirements | Format-Table

   Path: C:\Scripts\PSFunctionTools\samples\SampleScript.ps1
```

```
ApplicationId PSVersion PSEditions PSSnapIns Assemblies IsElevationRequired
------------- --------- ---------- --------- ---------- -------------------
                    4.0 {}         {}        {}                       False

   Path: C:\Scripts\PSFunctionTools\samples\SampleScript2.ps1

ApplicationId PSVersion PSEditions PSSnapIns Assemblies IsElevationRequired
------------- --------- ---------- --------- ---------- -------------------
                    3.0 {}         {}        {}                       False

   Path: C:\Scripts\PSFunctionTools\samples\SampleScript3.ps1

ApplicationId PSVersion PSEditions PSSnapIns Assemblies IsElevationRequired
------------- --------- ---------- --------- ---------- -------------------
                    5.0 {}         {}        {}                       True

   Path: C:\Scripts\PSFunctionTools\samples\SampleScript4.ps1
...
```

You are welcome to copy, paste, and edit these samples as much as you would like.

Bugs and Enhancements

Please use the repository's Issues section for reporting bugs and requesting new features. Remember, the commands in this module are designed for PowerShell 7.1 and later.

Related Commands and Projects

You might also be interested in the PSScriptTools module. This module has several commands that you might use in your toolmaking.

- New-PSFormatXML
- New-PSDynamicParameterForm
- New-PSDynamicParameter

Use the commands in the PSTypeExtensionTools to extend standard types as well as custom types and classes in your work.