MyWebUniversity
python
PowerShell Get-Help cmdlet
PS C:\> Get-Help
FPDF Library
PDF generator
FPDF
Full credit is given to all the above companies including the Operating System that this PDF file was generated!

Windows PowerShell Get-Help on Cmdlet 'Get-Module'

PS:\>Get-HELP Get-Module -Full

NAME

Get-Module

SYNOPSIS

List the modules imported in the current session or that can be imported from the PSModulePath.

SYNTAX

Get-Module [[-Name] <System.String[]>] [-All] [-FullyQualifiedName <Microsoft.PowerShell.Commands.ModuleSpecification[]>] [<CommonParameters>]

Get-Module [[-Name] <System.String[]>] [-All] [-FullyQualifiedName <Microsoft.PowerShell.Commands.ModuleSpecification[]>] [-ListAvailable] [-PSEdition <System.String>] [-Refresh] [<CommonParameters>]

Get-Module [[-Name] <System.String[]>] [-CimNamespace <System.String>] [-CimResourceUri <System.Uri>] -CimSession <Microsoft.Management.Infrastructure.CimSession> [-FullyQualifiedName <Microsoft.PowerShell.Commands.ModuleSpecification[]>] [-ListAvailable] [-Refresh] [<CommonParameters>]

```
    Get-Module [[-Name] <System.String[]>] [-FullyQualifiedName <Microsoft.PowerShell.Commands.ModuleSpecification[]>] [-ListAvailable] [-PSEdition <System.String>]
    -PSSession <System.Management.Automation.Runspaces.PSSession> [-Refresh] [<CommonParameters>]
```

DESCRIPTION

The `Get-Module` cmdlet lists the PowerShell modules that have been imported, or that can be imported, into a PowerShell session. Without parameters, `Get-Module` gets modules that have been imported into the current session. The ListAvailable parameter is used to list the modules that are available to be imported from the paths specified in the PSModulePath environment variable (`$env:PSModulePath`).

The module object that `Get-Module` returns contains valuable information about the module. You can also pipe the module objects to other cmdlets, such as the `Import-Module` and `Remove-Module` cmdlets.

`Get-Module` lists modules, but it does not import them. Starting in Windows PowerShell 3.0, modules are automatically imported when you use a command in the module, but a `Get-Module` command does not trigger an automatic import. You can also import the modules into your session using the `Import-Module` cmdlet.

Starting in Windows PowerShell 3.0, you can get and then, import modules from remote sessions into the local session. This strategy uses the Implicit Remoting feature of PowerShell and is equivalent to using the `Import-PSSession` cmdlet. When you use commands in modules imported from another session, the commands run implicitly in the remote session. This feature lets you manage the remote computer from the local session.

Also, starting in Windows PowerShell 3.0, you can use `Get-Module` and `Import-Module` to get and import Common Information Model (CIM) modules. CIM modules define cmdlets in Cmdlet Definition XML (CDXML) files. This feature lets you use cmdlets that are implemented in non-managed code assemblies, such as those written in C++.

Implicit remoting can be used to manage remote computers that have PowerShell remoting enabled. Create a PSSession on the remote computer and then use the PSSession

parameter of `Get-Module` to get the PowerShell modules in the remote session. When you import a module from the remote session the imported commands run in the

session on the remote computer.

You can use a similar strategy to manage computers that do not have PowerShell remoting enabled. These include computers that are not running the Windows operating

system, and computers that have PowerShell but do not have PowerShell remoting enabled.

Start by creating a CIM session on the remote computer. A CIM session is a connection to Windows Management Instrumentation (WMI) on the remote computer. Then use the

CIMSession parameter of `Get-Module` to get CIM modules from the CIM session. When you import a CIM module by using the `Import-Module` cmdlet and then run the

imported commands, the commands run implicitly on the remote computer. You can use this WMI and CIM strategy to manage the remote computer.

PARAMETERS

-All <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet gets all modules in each module folder, including nested modules, manifest (`.psd1`) files, script module (`.psm1`) files, and binary

module (`.dll`) files. Without this parameter, `Get-Module` gets only the default module in each module folder.

```
Required?                    false
Position?                    named
Default value                False
Accept pipeline input?       False
Accept wildcard characters?  false
```

-CimNamespace <System.String>

Specifies the namespace of an alternate CIM provider that exposes CIM modules. The default value is the namespace of the Module Discovery WMI provider.

Use this parameter to get CIM modules from computers and devices that are not running the Windows operating system.

This parameter was introduced in Windows PowerShell 3.0.

Required?	false
Position?	named
Default value	None
Accept pipeline input?	False
Accept wildcard characters?	false

-CimResourceUri <System.Uri>

Specifies an alternate location for CIM modules. The default value is the resource URI of the Module Discovery WMI provider on the remote computer.

Use this parameter to get CIM modules from computers and devices that are not running the Windows operating system.

This parameter was introduced in Windows PowerShell 3.0.

Required?	false
Position?	named
Default value	None
Accept pipeline input?	False
Accept wildcard characters?	false

-CimSession <Microsoft.Management.Infrastructure.CimSession>

Specifies a CIM session on the remote computer. Enter a variable that contains the CIM session or a command that gets the CIM session, such as a Get-CimSession

(/powershell/module/cimcmdlets/get-cimsession)command.

`Get-Module` uses the CIM session connection to get modules from the remote computer. When you import the module by using the `Import-Module` cmdlet and use the commands from the imported module in the current session, the commands actually run on the remote computer.

You can use this parameter to get modules from computers and devices that are not running the Windows operating system, and computers that have PowerShell, but do

not have PowerShell remoting enabled.

The CimSession parameter gets all modules in the CIMSession . However, you can import only CIM-based and Cmdlet Definition XML (CDXML)-based modules.

```
Required?                    true
Position?                    named
Default value                None
Accept pipeline input?       False
Accept wildcard characters?  false
```

-FullyQualifiedName <Microsoft.PowerShell.Commands.ModuleSpecification[]>

The value can be a module name, a full module specification, or a path to a module file.

When the value is a path, the path can be fully qualified or relative. A relative path is resolved relative to the script that contains the using statement.

When the value is a name or module specification, PowerShell searches the PSModulePath for the specified module.

A module specification is a hashtable that has the following keys.

- `ModuleName` - Required Specifies the module name. - `GUID` - Optional Specifies the GUID of the module. - It's also Required to specify at least one of the

three below keys.   - `ModuleVersion` - Specifies a minimum acceptable version of the module.   - `MaximumVersion` - Specifies the maximum acceptable version of

the module.   - `RequiredVersion` - Specifies an exact, required version of the module. This can't be used with the other Version keys.

You cannot specify the FullyQualifiedName parameter in the same command as a Name parameter.

```
        Required?                    false
        Position?                    named
        Default value                None
        Accept pipeline input?       True (ByPropertyName)
        Accept wildcard characters?  false
```

-ListAvailable <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet gets all installed modules. `Get-Module` gets modules in paths listed in the PSModulePath environment variable. Without this parameter,

`Get-Module` 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.

```
        Required?                    false
        Position?                    named
        Default value                False
        Accept pipeline input?       False
        Accept wildcard characters?  false
```

-Name <System.String[]>

Specifies names or name patterns of modules that this cmdlet gets. Wildcard characters are permitted. You can also pipe the names to `Get-Module`. You cannot

specify the FullyQualifiedName parameter in the same command as a Name parameter. Name cannot accept a module GUID as a value. To return modules by specifying a

GUID, use FullyQualifiedName instead.

```
        Required?                    false
        Position?                    0
        Default value                None
        Accept pipeline input?       True (ByValue)
        Accept wildcard characters?  true
```

-PSEdition <System.String>

Gets the modules that support specified edition of PowerShell.

The acceptable values for this parameter are:

- `Desktop`

- `Core`

The `Get-Module` cmdlet checks CompatiblePSEditions property of PSModuleInfo object for the specified value and returns only those modules that have it set.

> [!NOTE] > - Desktop Edition: Built on .NET Framework and provides compatibility with scripts and modules > targeting versions of PowerShell running on full

footprint editions of Windows such as Server > Core and Windows Desktop. > - Core Edition: Built on .NET Core and provides compatibility with scripts and

modules > targeting versions of PowerShell running on reduced footprint editions of Windows such as Nano > Server and Windows IoT.

Required?	false
Position?	named
Default value	None
Accept pipeline input?	False
Accept wildcard characters?	false

-PSSession <System.Management.Automation.Runspaces.PSSession>

Gets the modules in the specified user-managed PowerShell session (PSSession). Enter a variable that contains the session, a command that gets the session, such

as a `Get-PSSession` command, or a command that creates the session, such as a `New-PSSession` command.

When the session is connected to a remote computer, you must specify the ListAvailable parameter.

A `Get-Module` command that uses the PSSession parameter is equivalent to using the `Invoke-Command` cmdlet to run a `Get-Module -ListAvailable` command in a

PSSession .

This parameter was introduced in Windows PowerShell 3.0.

```
Required?                    true
Position?                    named
Default value                None
Accept pipeline input?       False
Accept wildcard characters?  false
```

-Refresh <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet refreshes the cache of installed commands. The command cache is created when the session starts. It enables the `Get-Command` cmdlet to

get commands from modules that are not imported into the session.

This parameter is designed for development and testing scenarios in which the contents of modules have changed since the session started.

When you specify the Refresh parameter in a command, you must specify ListAvailable .

This parameter was introduced in Windows PowerShell 3.0.

```
Required?                    false
Position?                    named
Default value                False
Accept pipeline input?       False
Accept wildcard characters?  false
```

<CommonParameters>

This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable,

OutBuffer, PipelineVariable, and OutVariable. For more information, see

about_CommonParameters (https:/go.microsoft.com/fwlink/?LinkID=113216).

INPUTS

System.String

You can pipe module names to this cmdlet.

OUTPUTS

System.Management.Automation.PSModuleInfo

This cmdlet returns objects that represent modules. When you specify the ListAvailable parameter, `Get-Module` returns a ModuleInfoGrouping object, which is a

type of PSModuleInfo object that has the same properties and methods.

NOTES

Windows PowerShell includes the following aliases for `Get-Module`:

- `gmo`

- Beginning in Windows PowerShell 3.0, the core commands that are included in PowerShell are packaged in modules. The exception is Microsoft.PowerShell.Core ,

which is a snap-in (PSSnapin). By default, only the Microsoft.PowerShell.Core snap-in is added to the session. Modules are imported automatically on first

use and you can use the `Import-Module` cmdlet to import them.

- In Windows PowerShell 2.0, and in host programs that create older-style sessions in later versions of PowerShell, the core commands are packaged in snap-ins (

PSSnapins). The exception is Microsoft.PowerShell.Core , which is always a snap-in. Also, remote sessions, such as those started by the `New-PSSession` cmdlet, are older-style sessions that include core snap-ins.

For information about the CreateDefault2 method that creates newer-style sessions with core modules, see CreateDefault2 Method
(/dotnet/api/system.management.automation.runspaces.initialsessionstate.createdefault2).

- `Get-Module` only gets modules in locations that are stored in the value of the PSModulePath environment variable (`$env:PSModulePath`). The `Import-Module`
cmdlet can import modules in other locations, but you cannot use the `Get-Module` cmdlet to get them.

- Also, starting in PowerShell 3.0, new properties have been added to the object that `Get-Module` returns that make it easier to learn about modules even
before they are imported. All properties are populated before importing. These include the ExportedCommands , ExportedCmdlets and ExportedFunctions properties
that list the commands that the module exports.

- The ListAvailable parameter gets only well-formed modules, that is, folders that contain at least one file whose base name is the same as the name of the
module folder. The base name is the name without the file name extension. Folders that contain files that have different names are considered to be
containers, but not modules.

To get modules that are implemented as DLL files, but are not enclosed in a module folder, specify both the ListAvailable and All parameters.

- To use the CIM session feature, the remote computer must have WS-Management remoting and Windows Management Instrumentation (WMI), which is the Microsoft
implementation of the Common Information Model (CIM) standard. The computer must also have the Module Discovery WMI provider or an alternate WMI provider that
has the same basic features.

You can use the CIM session feature on computers that are not running the Windows operating system and on Windows computers that have PowerShell, but do not have PowerShell remoting enabled.

You can also use the CIM parameters to get CIM modules from computers that have PowerShell remoting enabled. This includes the local computer. When you create a

CIM session on the local computer, PowerShell uses DCOM, instead of WMI, to create the session.

--- Example 1: Get modules imported into the current session ---

```
Get-Module
```

This command gets modules that have been imported into the current session.

---- Example 2: Get installed modules and available modules ----

```
Get-Module -ListAvailable
```

This command gets the modules that are installed on the computer and can be imported into the current session.

`Get-Module` looks for available modules in the path specified by the $env:PSModulePath environment variable. For more information about PSModulePath , see

about_Modules (About/about_Modules.md)and about_Environment_Variables (About/about_Environment_Variables.md).

-------------- Example 3: Get all exported files --------------

```
Get-Module -ListAvailable -All
```

This command gets all of the exported files for all available modules.

----- Example 4: Get a module by its fully qualified name -----

```
$FullyQualifiedName = @{ModuleName="Microsoft.PowerShell.Management";ModuleVersion="3.1.0.0"}
Get-Module -FullyQualifiedName $FullyQualifiedName | Format-Table -Property Name,Version

Name                          Version
----                          -------
Microsoft.PowerShell.Management  3.1.0.0
```

This example gets the Microsoft.PowerShell.Management module by specifying the fully qualified name of the module by using the FullyQualifiedName parameter. The

command then pipes the results into the `Format-Table` cmdlet to format the results as a table with Name and Version as the column headings.

In a fully qualified name for a module, the value ModuleVersion acts as minimum version. So, for this example, it matches any Microsoft.PowerShell.Management module

that is version `3.1.0.0` or higher.

------------ Example 5: Get properties of a module ------------

```
Get-Module | Get-Member -MemberType Property | Format-Table Name
```

```
Name
----
AccessMode
Author
ClrVersion
CompanyName
Copyright
Definition
Description
DotNetFrameworkVersion
ExportedAliases
ExportedCmdlets
ExportedCommands
ExportedFormatFiles
ExportedFunctions
ExportedTypeFiles
ExportedVariables
ExportedWorkflows
FileList
Guid
```

HelpInfoUri

LogPipelineExecutionDetails

ModuleBase

ModuleList

ModuleType

Name

NestedModules

OnRemove

Path

PowerShellHostName

PowerShellHostVersion

PowerShellVersion

PrivateData

ProcessorArchitecture

RequiredAssemblies

RequiredModules

RootModule

Scripts

SessionState

Version

This command gets the properties of the PSModuleInfo object that `Get-Module` returns. There is one object for each module file.

You can use the properties to format and filter the module objects. For more information about the properties, see PSModuleInfo Properties

(/dotnet/api/system.management.automation.psmoduleinfo).

The output includes the new properties, such as Author and CompanyName , that were introduced in Windows PowerShell 3.0.

------------- Example 6: Group all modules by name -------------

```
Get-Module -ListAvailable -All | Format-Table -Property Name, Moduletype, Path -Groupby Name
```

```
    Name: AppLocker


    Name      ModuleType Path
    ----      ---------- ----
    AppLocker   Manifest C:\Windows\system32\WindowsPowerShell\v1.0\Modules\AppLocker\AppLocker.psd1



       Name: Appx


    Name ModuleType Path
    ---- ---------- ----
    Appx   Manifest C:\Windows\system32\WindowsPowerShell\v1.0\Modules\Appx\en-US\Appx.psd1
    Appx   Manifest C:\Windows\system32\WindowsPowerShell\v1.0\Modules\Appx\Appx.psd1
    Appx     Script C:\Windows\system32\WindowsPowerShell\v1.0\Modules\Appx\Appx.psm1



       Name: BestPractices


    Name          ModuleType Path
    ----          ---------- ----
    BestPractices   Manifest C:\Windows\system32\WindowsPowerShell\v1.0\Modules\BestPractices\BestPractices.psd1



       Name: BitsTransfer


    Name         ModuleType Path
    ----         ---------- ----
    BitsTransfer   Manifest C:\Windows\system32\WindowsPowerShell\v1.0\Modules\BitsTransfer\BitsTransfer.psd1
```

This command gets all module files, both imported and available, and then groups them by module name. This lets you see the module files that each script is exporting.

----- Example 7: Display the contents of a module manifest -----

```
# First command
$m = Get-Module -list -Name BitsTransfer

# Second command
Get-Content $m.Path

@ {
    GUID              = "{8FA5064B-8479-4c5c-86EA-0D311FE48875}"
    Author            = "Microsoft Corporation"
    CompanyName       = "Microsoft Corporation"
    Copyright         = "Microsoft Corporation. All rights reserved."
    ModuleVersion     = "1.0.0.0"
    Description       = "Windows PowerShell File Transfer Module"
    PowerShellVersion  = "2.0"
    CLRVersion        = "2.0"
    NestedModules     = "Microsoft.BackgroundIntelligentTransfer.Management"
    FormatsToProcess   = "FileTransfer.Format.ps1xml"
    RequiredAssemblies = Join-Path $psScriptRoot "Microsoft.BackgroundIntelligentTransfer.Management.Interop.dll"
}
```

The first command gets the PSModuleInfo object that represents BitsTransfer module. It saves the object in the `$m` variable.

The second command uses the `Get-Content` cmdlet to get the content of the manifest file in the specified path. It uses dot notation to get the path to the manifest

file, which is stored in the Path property of the object. The output shows the contents of the module manifest.

---------- Example 8: List files in module directory ----------

```
dir (Get-Module -ListAvailable FileTransfer).ModuleBase

Directory: C:\Windows\system32\WindowsPowerShell\v1.0\Modules\FileTransfer
Mode              LastWriteTime     Length Name
```

```
    ----                -------------     ------ ----
    d----        12/16/2008  12:36 PM            en-US
    -a---        11/19/2008  11:30 PM      16184 FileTransfer.Format.ps1xml
    -a---        11/20/2008  11:30 PM       1044 FileTransfer.psd1
    -a---        12/16/2008  12:20 AM     108544 Microsoft.BackgroundIntelligentTransfer.Management.Interop.dll
```

This command lists the files in the directory of the module. This is another way to determine what is in a module before you import it. Some modules might have help

files or ReadMe files that describe the module.

-------- Example 9: Get modules installed on a computer --------

```
$s = New-PSSession -ComputerName Server01

Get-Module -PSSession $s -ListAvailable
```

These commands get the modules that are installed on the Server01 computer.

The first command uses the `New-PSSession` cmdlet to create a PSSession on the Server01 computer. The command saves the PSSession in the `$s` variable.

The second command uses the PSSession and ListAvailable parameters of `Get-Module` to get the modules in the PSSession in the `$s` variable.

If you pipe modules from other sessions to the `Import-Module` cmdlet, `Import-Module` imports the module into the current session by using the implicit remoting

feature. This is equivalent to using the `Import-PSSession` cmdlet. You can use the cmdlets from the module in the current session, but commands that use these

cmdlets actually run the remote session. For more information, see `Import-Module` (Import-Module.md)and `Import-PSSession`

(../Microsoft.PowerShell.Utility/Import-PSSession.md).

Example 10: Manage a computer that does not run the Windows operating system

```
$cs = New-CimSession -ComputerName RSDGF03
```

```
    Get-Module -CimSession $cs -Name Storage | Import-Module
    Get-Command Get-Disk


    CommandType     Name                ModuleName
    -----------     ----                ----------
    Function        Get-Disk            Storage


    Get-Disk


    Number Friendly Name            OperationalStatus        Total Size Partition Style
    ------ -------------            -----------------        ---------- ---------------
    0      Virtual HD ATA Device    Online                        40 GB MBR
```

The first command uses the `New-CimSession` cmdlet to create a session on the RSDGF03 remote computer. The session connects to WMI on the remote computer. The command

saves the CIM session in the `$cs` variable.

The second command uses the CIM session in the `$cs` variable to run a `Get-Module` command on the RSDGF03 computer. The command uses the Name parameter to specify

the Storage module. The command uses a pipeline operator (`|`) to send the Storage module to the `Import-Module` cmdlet, which imports it into the local session.

The third command runs the `Get-Command` cmdlet on the `Get-Disk` command in the Storage module. When you import a CIM module into the local session, PowerShell

converts the CDXML files that represent the CIM module into PowerShell scripts, which appear as functions in the local session.

The fourth command runs the `Get-Disk` command. Although the command is typed in the local session, it runs implicitly on the remote computer from which it was

imported. The command gets objects from the remote computer and returns them to the local session.

RELATED LINKS

Online Version:

https://learn.microsoft.com/powershell/module/microsoft.powershell.core/get-module?view=powershell-5.1&WT.mc_id=ps-gethelp

    Get-CimSession

    New-CimSession

    about_Modules

    Get-PSSession

    Import-Module

    Import-PSSession

    New-PSSession

    Remove-Module