Full credit is given to the above companies including the OS that this PDF file was generated!

PowerShell Get-Help on command 'Import-Module'

PS C:\Users\wahid> Get-Help Import-Module

NAME

Import-Module

SYNOPSIS

Adds modules to the current session.

SYNTAX

Import-Module [-Assembly] <System.Reflection.Assembly[]> [-Alias
<System.String[]>] [-ArgumentList <System.Object[]>] [-AsCustomObject]
[-Cmdlet <System.String[]>] [-DisableNameChecking] [-Force] [-Function
<System.String[]>] [-Global] [-NoClobber] [-PassThru] [-Prefix
<System.String>] [-Scope {Local | Global}] [-Variable <System.String[]>]
[<CommonParameters>]

Import-Module [-Name] <System.String[]> [-Alias <System.String[]>]
[-ArgumentList <System.Object[]>] [-AsCustomObject] [-CimNamespace
<System.String>] [-CimResourceUri <System.Uri>] -CimSession
<Microsoft.Management.Infrastructure.CimSession> [-Cmdlet <System.String[]>]
[-DisableNameChecking] [-Force] [-Function <System.String[]>] [-Global]
[-MaximumVersion <System.String>] [-MinimumVersion <System.Version>]

[-NoClobber] [-PassThru] [-Prefix <System.String>] [-RequiredVersion
<System.Version>] [-Scope {Local | Global}] [-Variable <System.String[]>]
[<CommonParameters>]

Import-Module [-FullyQualifiedName]
<Microsoft.PowerShell.Commands.ModuleSpecification[]> [-Alias
<System.String[]>] [-ArgumentList <System.Object[]>] [-AsCustomObject]
[-Cmdlet <System.String[]>] [-DisableNameChecking] [-Force] [-Function
<System.String[]>] [-Global] [-NoClobber] [-PassThru] [-Prefix
<System.String>] [-Scope {Local | Global}] [-Variable <System.String[]>]
[<CommonParameters>]

Import-Module [-FullyQualifiedName]
<Microsoft.PowerShell.Commands.ModuleSpecification[]> [-Alias
<System.String[]>] [-ArgumentList <System.Object[]>] [-AsCustomObject]
[-Cmdlet <System.String[]>] [-DisableNameChecking] [-Force] [-Function
<System.String[]>] [-Global] [-NoClobber] [-PassThru] [-Prefix
<System.String>] -PSSession <System.Management.Automation.Runspaces.PSSession>
[-Scope {Local | Global}] [-Variable <System.String[]>] [<CommonParameters>]

Import-Module [-Name] <System.String[]> [-Alias <System.String[]>]
[-ArgumentList <System.Object[]>] [-AsCustomObject] [-Cmdlet
<System.String[]>] [-DisableNameChecking] [-Force] [-Function
<System.String[]>] [-Global] [-MaximumVersion <System.String>]
[-MinimumVersion <System.Version>] [-NoClobber] [-PassThru] [-Prefix
<System.String>] [-RequiredVersion <System.Version>] [-Scope {Local | Global}]
[-Variable <System.String[]>] [<CommonParameters>]

Import-Module [-Name] <System.String[]> [-Alias <System.String[]>]
[-ArgumentList <System.Object[]>] [-AsCustomObject] [-Cmdlet
<System.String[]>] [-DisableNameChecking] [-Force] [-Function
<System.String[]>] [-Global] [-MaximumVersion <System.String>]
[-MinimumVersion <System.Version>] [-NoClobber] [-PassThru] [-Prefix

<System.String>] -PSSession <System.Management.Automation.Runspaces.PSSession>
[-RequiredVersion <System.Version>] [-Scope {Local | Global}] [-Variable
<System.String[]>] [<CommonParameters>]

Import-Module [-ModuleInfo] <System.Management.Automation.PSModuleInfo[]>
[-Alias <System.String[]>] [-ArgumentList <System.Object[]>] [-AsCustomObject]
[-Cmdlet <System.String[]>] [-DisableNameChecking] [-Force] [-Function
<System.String[]>] [-Global] [-NoClobber] [-PassThru] [-Prefix
<System.String>] [-Scope {Local | Global}] [-Variable <System.String[]>]
[<CommonParameters>]

DESCRIPTION

The `Import-Module` cmdlet adds one or more modules to the current session.
Starting in PowerShell 3.0, installed modules are automatically imported to
the session when you use any commands or providers in the module. However, you
can still use the `Import-Module` command to import a module. You can disable
automatic module importing using the `$PSModuleAutoloadingPreference`
preference variable. For more information about the
`$PSModuleAutoloadingPreference` variable, see about_Preference_Variables
(About/about_Preference_Variables.md).

A module is a package that contains members that can be used in PowerShell.
Members include cmdlets, providers, scripts, functions, variables, and other
tools and files. After a module is imported, you can use the module members in
your session. For more information about modules, see about_Modules
(About/about_Modules.md).

By default, `Import-Module` imports all members that the module exports, but
you can use the Alias , Function , Cmdlet , and Variable parameters to
restrict which members are imported. The NoClobber parameter prevents
`Import-Module` from importing members that have the same names as members in
the current session.

`Import-Module` imports a module only into the current session. To import the module into every new session, add an `Import-Module` command to your PowerShell profile. For more information about profiles, see about_Profiles (About/about_Profiles.md).

You can manage remote Windows computers that have PowerShell remoting enabled by creating a PSSession on the remote computer. Then use the PSSession parameter of `Import-Module` to import the modules that are installed on the remote computer. When you use the imported commands in the current session the commands implicitly run on the remote computer.

Starting in Windows PowerShell 3.0, you can use `Import-Module` to 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++.

For remote computers that don't have PowerShell remoting enabled, including computers that aren't running the Windows operating system, you can use the CIMSession parameter of `Import-Module` to import CIM modules from the remote computer. The imported commands run implicitly on the remote computer. A CIMSession is a connection to Windows Management Instrumentation (WMI) on the remote computer.

PARAMETERS

-Alias <System.String[]>

Specifies the aliases that this cmdlet imports from the module into the current session. Enter a comma-separated list of aliases. Wildcard characters are permitted.

Some modules automatically export selected aliases into your session when you import the module. This parameter lets you select from among the

exported aliases.

-ArgumentList <System.Object[]>

Specifies an array of arguments, or parameter values, that are passed to a script module during the `Import-Module` command. This parameter is valid only when you're importing a script module.

You can also refer to the ArgumentList parameter by its alias, args . For more information about the behavior of ArgumentList , see about_Splatting (about/about_Splatting.md#splatting-with-arrays).

-AsCustomObject <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet returns a custom object with members that represent the imported module members. This parameter is valid only for script modules.

When you use the AsCustomObject parameter, `Import-Module` imports the module members into the session and then returns a PSCustomObject object instead of a PSModuleInfo object. You can save the custom object in a variable and use member-access enumeration to invoke the members.

-Assembly <System.Reflection.Assembly[]>

Specifies an array of assembly objects. This cmdlet imports the cmdlets and providers implemented in the specified assembly objects. Enter a variable that contains assembly objects or a command that creates assembly objects. You can also pipe an assembly object to `Import-Module`.

When you use this parameter, only the cmdlets and providers implemented by the specified assemblies are imported. If the module contains other files, they aren't imported, and you might be missing important members of the module. Use this parameter for debugging and testing the module, or when you're instructed to use it by the module author.

-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 import CIM modules from computers and devices that aren't running a Windows operating system.

This parameter was introduced in Windows PowerShell 3.0.

-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 import CIM modules from computers and devices that aren't running a Windows operating system.

This parameter was introduced in Windows PowerShell 3.0.

-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 (../CimCmdlets/Get-CimSession.md)command.

`Import-Module` uses the CIM session connection to import modules from the remote computer into the current session. When you use the commands from the imported module in the current session, the commands run on the remote computer.

You can use this parameter to import modules from computers and devices that aren't running the Windows operating system, and Windows computers that have PowerShell, but don't have PowerShell remoting enabled.

This parameter was introduced in Windows PowerShell 3.0.

-Cmdlet <System.String[]>

Specifies an array of cmdlets that this cmdlet imports from the module into the current session. Wildcard characters are permitted.

Some modules automatically export selected cmdlets into your session when you import the module. This parameter lets you select from among the exported cmdlets.

-DisableNameChecking <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet suppresses the message that warns you when you import a cmdlet or function whose name includes an unapproved verb or a prohibited character.

By default, when a module that you import exports cmdlets or functions that have unapproved verbs in their names, PowerShell displays the following warning message:

> WARNING: Some imported command names include unapproved verbs which might make them less > discoverable. Use the Verbose parameter for more detail or type Get-Verb to see the list of > approved verbs.

This message is only a warning. The complete module is still imported, including the non-conforming commands. Although the message is displayed to module users, the naming problem should be fixed by the module author.

-Force <System.Management.Automation.SwitchParameter>

This parameter causes a module to be loaded, or reloaded, over top of the current one.

-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.

-Function <System.String[]>

Specifies an array of functions that this cmdlet imports from the module into the current session. Wildcard characters are permitted. Some modules automatically export selected functions into your session when you import the module. This parameter lets you select from among the exported functions.

-Global <System.Management.Automation.SwitchParameter>

Indicates that this cmdlet imports modules into the global session state so they're available to all commands in the session.

By default, when `Import-Module` cmdlet is called from the command prompt, script file, or scriptblock, all the commands are imported into the global session state.

When invoked from another module, `Import-Module` cmdlet imports the commands in a module, including commands from nested modules, into the calling module's session state.

> [!TIP] > You should avoid calling `Import-Module` from within a module. Instead, declare the target module > as a nested module in the parent module's manifest. Declaring nested modules improves the > discoverability of dependencies.

The Global parameter is equivalent to the Scope parameter with a value of Global .

To restrict the commands that a module exports, use an `Export-ModuleMember` command in the script module.

-MaximumVersion <System.String>

Specifies a maximum version. This cmdlet imports only a version of the module that's less than or equal to the specified value. If no version qualifies, `Import-Module` returns an error.

-MinimumVersion <System.Version>

Specifies a minimum version. This cmdlet imports only a version of the module that's greater than or equal to the specified value. Use the MinimumVersion parameter name or its alias, Version . If no version qualifies, `Import-Module` generates an error.

To specify an exact version, use the RequiredVersion parameter. You can also use the Module and Version parameters of the #Requires keyword to require a specific version of a module in a script.

This parameter was introduced in Windows PowerShell 3.0.

-ModuleInfo <System.Management.Automation.PSModuleInfo[]>

Specifies an array of module objects to import. Enter a variable that contains the module objects, or a command that gets the module objects, such as the following command: `Get-Module -ListAvailable`. You can also pipe module objects to `Import-Module`.

-Name <System.String[]>

Specifies the names of the modules to import. Enter the name of the module or the name of a file in the module, such as a `.psd1`, `.psm1`, `.dll`, or `.ps1` file. File paths are optional. Wildcard characters aren't permitted. You can also pipe module names and filenames to `Import-Module`.

If you omit a path, `Import-Module` looks for the module in the paths saved in the `$env:PSModulePath` environment variable.

Specify only the module name whenever possible. When you specify a filename, only the members that are implemented in that file are imported. If the module contains other files, they aren't imported, and you might be missing important members of the module.

> [!NOTE] > While it's possible to import a script (`.ps1`) file as a module, script files are usually not > structured like script modules file (`.psm1`) file. Importing a script file doesn't guarantee > that it's usable as a module. For more information, see about_Modules (about/about_Modules.md).

-NoClobber <System.Management.Automation.SwitchParameter>

Prevents importing commands that have the same names as existing commands in the current session. By default, `Import-Module` imports all exported module commands.

Commands that have the same names can hide or replace commands in the session. To avoid command name conflicts in a session, use the Prefix or

NoClobber parameters. For more information about name conflicts and command precedence, see "Modules and Name Conflicts" in about_Modules (about/about_Modules.md)and about_Command_Precedence (about/about_Command_Precedence.md).

This parameter was introduced in Windows PowerShell 3.0.

-PassThru <System.Management.Automation.SwitchParameter>
Returns an object representing the imported module. By default, this cmdlet doesn't generate any output.

-Prefix <System.String>
Specifies a prefix that this cmdlet adds to the nouns in the names of imported module members.

Use this parameter to avoid name conflicts that might occur when different members in the session have the same name. This parameter doesn't change the module, and it doesn't affect files that the module imports for its own use. These are known as nested modules. This cmdlet affects only the names of members in the current session.

For example, if you specify the prefix UTC and then import a `Get-Date` cmdlet, the cmdlet is known in the session as `Get-UTCDate`, and it's not confused with the original `Get-Date` cmdlet.

The value of this parameter takes precedence over the DefaultCommandPrefix property of the module, which specifies the default prefix.

-PSSession <System.Management.Automation.Runspaces.PSSession>
Specifies a PowerShell user-managed session (PSSession) from which this cmdlet imports modules into the current session. Enter a variable that contains a PSSession or a command that gets a PSSession , such as a `Get-PSSession` command.

When you import a module from a different session into the current session, you can use the cmdlets from the module in the current session, just as you would use cmdlets from a local module. Commands that use the remote cmdlets run in the remote session, but the remoting details are managed in the background by PowerShell.

This parameter uses the Implicit Remoting feature of PowerShell. It's equivalent to using the `Import-PSSession` cmdlet to import particular modules from a session.

`Import-Module` can't import core PowerShell modules from another session. The core PowerShell modules have names that begin with Microsoft.PowerShell.

This parameter was introduced in Windows PowerShell 3.0.

-RequiredVersion <System.Version>

Specifies a version of the module that this cmdlet imports. If the version isn't installed, `Import-Module` generates an error.

By default, `Import-Module` imports the module without checking the version number.

To specify a minimum version, use the MinimumVersion parameter. You can also use the Module and Version parameters of the #Requires keyword to require a specific version of a module in a script.

This parameter was introduced in Windows PowerShell 3.0.

Scripts that use RequiredVersion to import modules that are included with existing releases of the Windows operating system don't automatically run in future releases of the Windows operating system. This is because

PowerShell module version numbers in future releases of the Windows operating system are higher than module version numbers in existing releases of the Windows operating system.

-Scope <System.String>

Specifies a scope to import the module in.

The acceptable values for this parameter are:

- Global . Available to all commands in the session. Equivalent to the Global parameter. - Local . Available only in the current scope.

By default, when `Import-Module` cmdlet is called from the command prompt, script file, or scriptblock, all the commands are imported into the global session state. You can use the `-Scope Local` parameter to import module content into the script or scriptblock scope.

When invoked from another module, `Import-Module` cmdlet imports the commands in a module, including commands from nested modules, into the caller's session state. Specifying `-Scope Global` or `-Global` indicates that this cmdlet imports modules into the global session state so they're available to all commands in the session.

The Global parameter is equivalent to the Scope parameter with a value of Global .

This parameter was introduced in Windows PowerShell 3.0.

-Variable <System.String[]>

Specifies an array of variables that this cmdlet imports from the module into the current session. Enter a list of variables. Wildcard characters are permitted.

Some modules automatically export selected variables into your session
when you import the module. This parameter lets you select from among the
exported variables.

<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).

Example 1: Import the members of a module into the current session

Import-Module -Name PSDiagnostics

-- Example 2: Import all modules specified by the module path --

Get-Module -ListAvailable | Import-Module

Example 3: Import the members of several modules into the current session

$m = Get-Module -ListAvailable PSDiagnostics, Dism
Import-Module -ModuleInfo $m

The `Get-Module` cmdlet gets the PSDiagnostics and Dism modules and saves the
objects in the `$m` variable. The ListAvailable parameter is required when
you're getting modules that aren't yet imported into the session.

The ModuleInfo parameter of `Import-Module` is used to import the modules into
the current session.
------ Example 4: Import all modules specified by a path ------

```
Import-Module -Name c:\ps-test\modules\test -Verbose

VERBOSE: Loading module from path 'C:\ps-test\modules\Test\Test.psm1'.
VERBOSE: Exporting function 'my-parm'.
VERBOSE: Exporting function 'Get-Parameter'.
VERBOSE: Exporting function 'Get-Specification'.
VERBOSE: Exporting function 'Get-SpecDetails'.

Using the Verbose parameter causes `Import-Module` to report progress as it
loads the module. Without the Verbose , PassThru , or AsCustomObject
parameter, `Import-Module` doesn't generate any output when it imports a
module.
-- Example 5: Restrict module members imported into a session --

Import-Module PSDiagnostics -Function Disable-PSTrace, Enable-PSTrace
(Get-Module PSDiagnostics).ExportedCommands

Key                         Value
---                         -----
Disable-PSTrace             Disable-PSTrace
Disable-PSWSManCombinedTrace Disable-PSWSManCombinedTrace
Disable-WSManTrace          Disable-WSManTrace
Enable-PSTrace              Enable-PSTrace
Enable-PSWSManCombinedTrace  Enable-PSWSManCombinedTrace
Enable-WSManTrace           Enable-WSManTrace
Get-LogProperties           Get-LogProperties
Set-LogProperties           Set-LogProperties
Start-Trace                 Start-Trace
Stop-Trace                  Stop-Trace

Get-Command -Module PSDiagnostics

CommandType     Name                Version    Source
```

```
-----------     ----                -------     ------
Function        Disable-PSTrace     6.1.0.0     PSDiagnostics
Function        Enable-PSTrace      6.1.0.0     PSDiagnostics
```

Using the Module parameter of the `Get-Command` cmdlet shows the commands that were imported from the PSDiagnostics module. The results confirm that only the `Disable-PSTrace` and `Enable-PSTrace` cmdlets were imported.

-- Example 6: Import the members of a module and add a prefix --

```
Import-Module PSDiagnostics -Prefix x -PassThru

ModuleType Version    Name                ExportedCommands
---------- -------    ----                ----------------
Script     6.1.0.0    PSDiagnostics       {Disable-xPSTrace,
Disable-xPSWSManCombinedTrace, Disable-xW...

Get-Command -Module PSDiagnostics

CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Function        Disable-xPSTrace                    6.1.0.0    PSDiagnostics
Function        Disable-xPSWSManCombinedTrace       6.1.0.0    PSDiagnostics
Function        Disable-xWSManTrace                 6.1.0.0    PSDiagnostics
Function        Enable-xPSTrace                     6.1.0.0    PSDiagnostics
Function        Enable-xPSWSManCombinedTrace        6.1.0.0    PSDiagnostics
Function        Enable-xWSManTrace                  6.1.0.0    PSDiagnostics
Function        Get-xLogProperties                  6.1.0.0    PSDiagnostics
Function        Set-xLogProperties                  6.1.0.0    PSDiagnostics
Function        Start-xTrace                        6.1.0.0    PSDiagnostics
Function        Stop-xTrace                         6.1.0.0    PSDiagnostics
```

`Get-Command` gets the members that have been imported from the module. The output shows that the module members were correctly prefixed.

---------- Example 7: Get and use a custom object ----------

```
Get-Module -List | Format-Table -Property Name, ModuleType -AutoSize

Name          ModuleType
----          ----------
Show-Calendar     Script
BitsTransfer    Manifest
PSDiagnostics   Manifest
TestCmdlets       Script
...

$a = Import-Module -Name Show-Calendar -AsCustomObject -Passthru
$a | Get-Member

TypeName: System.Management.Automation.PSCustomObject
Name          MemberType   Definition
----          ----------   ----------
Equals        Method       bool Equals(System.Object obj)
GetHashCode   Method       int GetHashCode()
GetType       Method       type GetType()
ToString      Method       string ToString()
Show-Calendar ScriptMethod System.Object Show-Calendar();

$a."Show-Calendar"()
```

The `Show-Calendar` script module is imported using the AsCustomObject parameter to request a custom object and the PassThru parameter to return the object. The resulting custom object is saved in the `$a` variable.

The `$a` variable is piped to the `Get-Member` cmdlet to show the properties and methods of the saved object. The output shows a `Show-Calendar` script method.

To call the `Show-Calendar` script method, the method name must be enclosed in quotation marks because the name includes a hyphen.

------ Example 8: Reimport a module into the same session ------

```
Import-Module PSDiagnostics
Import-Module PSDiagnostics -Force -Prefix PS
```

The first command imports the PSDiagnostics module. The second command imports the module again, this time using the Prefix parameter.

Without the Force parameter, the session would include two copies of each PSDiagnostics cmdlet, one with the standard name and one with the prefixed name.

Example 9: Run commands that have been hidden by imported commands

```
Get-Date

Thursday, August 15, 2019 2:26:12 PM

Import-Module TestModule
Get-Date

19227

Get-Command Get-Date -All | Format-Table -Property CommandType, Name,
ModuleName -AutoSize

CommandType     Name        ModuleName
-----------     ----        ----------
Function        Get-Date    TestModule
Cmdlet          Get-Date    Microsoft.PowerShell.Utility
```

Microsoft.PowerShell.Utility\Get-Date

Thursday, August 15, 2019 2:28:31 PM

The first `Get-Date` cmdlet returns a DateTime object with the current date. After importing the TestModule module, `Get-Date` returns the year and day of the year.

Using the All parameter of `Get-Command` show all the `Get-Date` commands in the session. The results show that there are two `Get-Date` commands in the session, a function from the TestModule module and a cmdlet from the Microsoft.PowerShell.Utility module.

Because functions take precedence over cmdlets, the `Get-Date` function from the TestModule module runs, instead of the `Get-Date` cmdlet. To run the original version of `Get-Date`, you must qualify the command name with the module name.

For more information about command precedence in PowerShell, see about_Command_Precedence (about/about_Command_Precedence.md).
------- Example 10: Import a minimum version of a module -------

Import-Module -Name PowerShellGet -MinimumVersion 2.0.0

You can also use the RequiredVersion parameter to import a particular version of a module, or use the Module and Version parameters of the `#Requires` keyword to require a particular version of a module in a script.
------- Example 11: Import using a fully qualified name -------

PS> Get-Module -ListAvailable PowerShellGet | Select-Object Name, Version

Name Version
---- -------

PowerShellGet 2.2.1

PowerShellGet 2.1.3

PowerShellGet 2.1.2

PowerShellGet 1.0.0.1

PS> Import-Module -FullyQualifiedName @{ModuleName = 'PowerShellGet';

ModuleVersion = '2.1.3' }

------- Example 12: Import using a fully qualified path -------

PS> Get-Module -ListAvailable PowerShellGet | Select-Object Path

Path

C:\Program Files\PowerShell\Modules\PowerShellGet\2.2.1\PowerShellGet.psd1

C:\program files\powershell\6\Modules\PowerShellGet\PowerShellGet.psd1

C:\Program

Files\WindowsPowerShell\Modules\PowerShellGet\2.1.2\PowerShellGet.psd1

C:\Program

Files\WindowsPowerShell\Modules\PowerShellGet\1.0.0.1\PowerShellGet.psd1

PS> Import-Module -Name 'C:\Program

Files\PowerShell\Modules\PowerShellGet\2.2.1\PowerShellGet.psd1'

------ Example 13: Import a module from a remote computer ------

$s = New-PSSession -ComputerName Server01

Get-Module -PSSession $s -ListAvailable -Name NetSecurity

ModuleType Name ExportedCommands

---------- ---- ----------

```
Manifest   NetSecurity      {New-NetIPsecAuthProposal,
New-NetIPsecMainModeCryptoProposal, New-Ne...

Import-Module -PSSession $s -Name NetSecurity
Get-Command -Module NetSecurity -Name Get-*Firewall*

CommandType     Name                                ModuleName
-----------     ----                                ----------
Function        Get-NetFirewallAddressFilter        NetSecurity
Function        Get-NetFirewallApplicationFilter    NetSecurity
Function        Get-NetFirewallInterfaceFilter      NetSecurity
Function        Get-NetFirewallInterfaceTypeFilter  NetSecurity
Function        Get-NetFirewallPortFilter           NetSecurity
Function        Get-NetFirewallProfile              NetSecurity
Function        Get-NetFirewallRule                 NetSecurity
Function        Get-NetFirewallSecurityFilter       NetSecurity
Function        Get-NetFirewallServiceFilter        NetSecurity
Function        Get-NetFirewallSetting              NetSecurity

Get-NetFirewallRule -DisplayName "Windows Remote Management*" |
  Format-Table -Property DisplayName, Name -AutoSize

DisplayName                                 Name
-----------                                 ----
Windows Remote Management (HTTP-In)         WINRM-HTTP-In-TCP
Windows Remote Management (HTTP-In)
WINRM-HTTP-In-TCP-PUBLIC
Windows Remote Management - Compatibility Mode (HTTP-In)
WINRM-HTTP-Compat-In-TCP
```

`New-PSSession` creates a remote session (PSSession) to the `Server01` computer. The PSSession is saved in the `$s` variable.

Running `Get-Module` with the PSSession parameter shows that the NetSecurity module is installed and available on the remote computer. This command is equivalent to using the `Invoke-Command` cmdlet to run `Get-Module` command in the remote session. For example:

`Invoke-Command $s {Get-Module -ListAvailable -Name NetSecurity`

Running `Import-Module` with the PSSession parameter imports the NetSecurity module from the remote computer into the current session. The `Get-Command` cmdlet retrieves commands that begin with `Get` and include `Firewall` from the NetSecurity module. The output confirms that the module and its cmdlets were imported into the current session.

Next, the `Get-NetFirewallRule` cmdlet gets Windows Remote Management firewall rules on the `Server01` computer. This is equivalent to using the `Invoke-Command` cmdlet to run `Get-NetFirewallRule` on the remote session.
Example 14: Manage storage on a remote computer without the Windows operating system

```
$cs = New-CimSession -ComputerName RSDGF03
Import-Module -CimSession $cs -Name Storage
# Importing a CIM module, converts the CDXML files for each command into
# PowerShell scripts. These appear as functions in the local session.
Get-Command Get-Disk

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

# Use implicit remoting to query disks on the remote computer from which the
# module was imported.
Get-Disk
```

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

REMARKS

To see the examples, type: "get-help Import-Module -examples".

For more information, type: "get-help Import-Module -detailed".

For technical information, type: "get-help Import-Module -full".

For online help, type: "get-help Import-Module -online"