python PowerShell FPDF Library
PDF generator

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

PowerShell Get-Help on command 'Import-PSSession'

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

NAME

   Import-PSSession

SYNOPSIS

   Imports commands from another session into the current session.

SYNTAX

   Import-PSSession [-Session] <System.Management.Automation.Runspaces.PSSession>

   [[-CommandName] <System.String[]>] [[-FormatTypeName] <System.String[]>]

   [-AllowClobber] [-ArgumentList <System.Object[]>] [-Certificate

   <System.Security.Cryptography.X509Certificates.X509Certificate2>]

   [-CommandType {Alias | Function | Filter | Cmdlet | ExternalScript |

   Application | Script | Workflow | Configuration | All}] [-DisableNameChecking]

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

   [-Module <System.String[]>] [-Prefix <System.String>] [<CommonParameters>]

DESCRIPTION

   The `Import-PSSession` cmdlet imports commands , such as cmdlets, functions,

   and aliases, from a PSSession on a local or remote computer into the current

session. You can import any command that the `Get-Command` cmdlet can find in the PSSession.

Use an `Import-PSSession` command to import commands from a customized shell, such as a Microsoft Exchange Server shell, or from a session that includes Windows PowerShell modules and snap-ins or other elements that are not in the current session.

To import commands, first use the `New-PSSession` cmdlet to create a PSSession. Then, use the `Import-PSSession` cmdlet to import the commands. By default, `Import-PSSession` imports all commands except for commands that have the same names as commands in the current session. To import all the commands, use the AllowClobber parameter.

You can use imported commands just as you would use any command in the session. When you use an imported command, the imported part of the command runs implicitly in the session from which it was imported. However, the remote operations are handled entirely by Windows PowerShell. You need not even be aware of them, except that you must keep the connection to the other session (PSSession) open. If you close it, the imported commands are no longer available.

Because imported commands might take longer to run than local commands, `Import-PSSession` adds an AsJob parameter to every imported command. This parameter allows you to run the command as a Windows PowerShell background job. For more information, see about_Jobs (../Microsoft.PowerShell.Core/about/about_Jobs.md).

When you use `Import-PSSession`, Windows PowerShell adds the imported commands to a temporary module that exists only in your session and returns an object that represents the module. To create a persistent module that you can use in future sessions, use the `Export-PSSession` cmdlet.

The `Import-PSSession` cmdlet uses the implicit remoting feature of Windows PowerShell. When you import commands into the current session, they run implicitly in the original session or in a similar session on the originating computer.

Beginning in Windows PowerShell 3.0, you can use the `Import-Module` cmdlet to import modules from a remote session into the current session. This feature uses implicit remoting. It is equivalent to using `Import-PSSession` to import selected modules from a remote session into the current session.

PARAMETERS

  -AllowClobber <System.Management.Automation.SwitchParameter>

    Indicates that this cmdlet imports the specified commands, even if they have the same names as commands in the current session.

    If you import a command with the same name as a command in the current session, the imported command hides or replaces the original commands. For more information, see about_Command_Precedence (../Microsoft.PowerShell.Core/about/about_Command_Precedence.md).

    By default, `Import-PSSession` does not import commands that have the same name as commands in the current session.

  -ArgumentList <System.Object[]>

    Specifies an array of commands that results from using the specified arguments (parameter values).

    For instance, to import the variant of the `Get-Item` command in the certificate (Cert:) drive in the PSSession in `$S`, type `Import-PSSession -Session $S -Command Get-Item -ArgumentList cert:`.

  -Certificate <System.Security.Cryptography.X509Certificates.X509Certificate2>

Specifies the client certificate that is used to sign the format files (*.Format.ps1xml) or script module files (.psm1) in the temporary module that `Import-PSSession` creates.

Enter a variable that contains a certificate or a command or expression that gets the certificate.

To find a certificate, use the `Get-PfxCertificate` cmdlet or use the `Get-ChildItem` cmdlet in the Certificate (Cert:) drive. If the certificate is not valid or does not have sufficient authority, the command fails.

-CommandName <System.String[]>

Specifies commands with the specified names or name patterns. Wildcards are permitted. Use CommandName or its alias, Name .

By default, `Import-PSSession` imports all commands from the session, except for commands that have the same names as commands in the current session. This prevents imported commands from hiding or replacing commands in the session. To import all commands, even those that hide or replace other commands, use the AllowClobber parameter.

If you use the CommandName parameter, the formatting files for the commands are not imported unless you use the FormatTypeName parameter. Similarly, if you use the FormatTypeName parameter, no commands are imported unless you use the CommandName parameter.

-CommandType <System.Management.Automation.CommandTypes>

Specifies the type of command objects. The default value is Cmdlet. Use CommandType or its alias, Type . The acceptable values for this parameter are:

- `Alias`: The Windows PowerShell aliases in the remote session.

- `All`: The cmdlets and functions in the remote session.

- `Application`: All the files other than Windows-PowerShell files in the

  paths that are listed in

  the Path environment variable (`$env:path`) in the remote session,

  including .txt, .exe, and .dll files. - `Cmdlet`: The cmdlets in the

  remote session. "Cmdlet" is the default.

- `ExternalScript`: The .ps1 files in the paths listed in the Path

  environment variable

  (`$env:path`) in the remote session. - `Filter` and `Function`: The

  Windows PowerShell functions in the remote session.

- `Script`: The script blocks in the remote session.

  These values are defined as a flag-based enumeration. You can combine

  multiple values together to set multiple flags using this parameter. The

  values can be passed to the CommandType parameter as an array of values or

  as a comma-separated string of those values. The cmdlet will combine the

  values using a binary-OR operation. Passing values as an array is the

  simplest option and also allows you to use tab-completion on the values.

-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, the Windows 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.

-FormatTypeName <System.String[]>
   Specifies formatting instructions for the specified Microsoft .NET
   Framework types. Enter the type names. Wildcards are permitted.

   The value of this parameter must be the name of a type that is returned by
   a `Get-FormatData` command in the session from which the commands are
   being imported. To get all of the formatting data in the remote session,
   type `*`.

   If the command does not include either the CommandName or FormatTypeName
   parameter, `Import-PSSession` imports formatting instructions for all .NET
   Framework types returned by a `Get-FormatData` command in the remote
   session.

   If you use the FormatTypeName parameter, no commands are imported unless
   you use the CommandName parameter.

   Similarly, if you use the CommandName parameter, the formatting files for
   the commands are not imported unless you use the FormatTypeName parameter.

-FullyQualifiedModule <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 can't specify the FullyQualifiedModule parameter in the same command as a Module parameter. The two parameters are mutually exclusive.

-Module <System.String[]>
    Specifies and array of commands in the Windows PowerShell snap-ins and modules. Enter the snap-in and module names. Wildcards are not permitted.

    `Import-PSSession` cannot import providers from a snap-in.

    For more information, see about_PSSnapins (../Microsoft.PowerShell.Core/About/about_PSSnapins.md)and about_Modules (../Microsoft.PowerShell.Core/About/about_Modules.md).

-Prefix <System.String>

Specifies a prefix to the nouns in the names of imported commands.

Use this parameter to avoid name conflicts that might occur when different commands in the session have the same name.

For instance, if you specify the prefix Remote and then import a `Get-Date` cmdlet, the cmdlet is known in the session as `Get-RemoteDate`, and it is not confused with the original `Get-Date` cmdlet.

-Session <System.Management.Automation.Runspaces.PSSession>
   Specifies the PSSession from which the cmdlets are imported. Enter a variable that contains a session object or a command that gets a session object, such as a `New-PSSession` or `Get-PSSession` command. You can specify only one session. This parameter is required.

<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 all commands from a PSSession -------

$S = New-PSSession -ComputerName Server01
Import-PSSession -Session $S

This command imports all commands from a PSSession on the Server01 computer into the current session, except for commands that have the same names as commands in the current session.

Because this command does not use the CommandName parameter, it also imports all of the formatting data required for the imported commands.

-- Example 2: Import commands that end with a specific string --

$S = New-PSSession https://ps.testlabs.com/powershell

Import-PSSession -Session $S -CommandName *-test -FormatTypeName *

New-Test -Name Test1

Get-Test test1 | Run-Test

These commands import the commands with names that end in "-test" from a

PSSession into the local session, and then they show how to use an imported

cmdlet.

The first command uses the `New-PSSession` cmdlet to create a PSSession. It

saves the PSSession in the `$S` variable.

The second command uses the `Import-PSSession` cmdlet to import commands from

the PSSession in `$S` into the current session. It uses the CommandName

parameter to specify commands with the Test noun and the FormatTypeName

parameter to import the formatting data for the Test commands.

The third and fourth commands use the imported commands in the current

session. Because imported commands are actually added to the current session,

you use the local syntax to run them. You do not need to use the

`Invoke-Command` cmdlet to run an imported command.

---------- Example 3: Import cmdlets from a PSSession ----------

$S1 = New-PSSession -ComputerName s1

$S2 = New-PSSession -ComputerName s2

Import-PSSession -Session s1 -Type cmdlet -Name New-Test, Get-Test

-FormatTypeName *

Import-PSSession -Session s2 -Type Cmdlet -Name Set-Test -FormatTypeName *

New-Test Test1 | Set-Test -RunType Full

This example shows that you can use imported cmdlets just as you would use

local cmdlets.

These commands import the `New-Test` and `Get-Test` cmdlets from a PSSession on the Server01 computer and the `Set-Test` cmdlet from a PSSession on the Server02 computer.

Even though the cmdlets were imported from different PSSessions, you can pipe an object from one cmdlet to another without error.

---- Example 4: Run an imported command as a background job ----

$S = New-PSSession -ComputerName Server01
Import-PSSession -Session $S -CommandName *-test* -FormatTypeName *
$batch = New-Test -Name Batch -AsJob
Receive-Job $batch

This example shows how to run an imported command as a background job.

Because imported commands might take longer to run than local commands, `Import-PSSession` adds an AsJob parameter to every imported command. The AsJob parameter lets you run the command as a background job.

The first command creates a PSSession on the Server01 computer and saves the PSSession object in the `$S` variable.

The second command uses `Import-PSSession` to import the Test cmdlets from the PSSession in `$S` into the current session.

The third command uses the AsJob parameter of the imported `New-Test` cmdlet to run a `New-Test` command as a background job. The command saves the job object that `New-Test` returns in the `$batch` variable.

The fourth command uses the `Receive-Job` cmdlet to get the results of the job in the `$batch` variable.

Example 5: Import cmdlets and functions from a Windows PowerShell module

$S = New-PSSession -ComputerName Server01

Invoke-Command -Session $S {Import-Module TestManagement}

Import-PSSession -Session $S -Module TestManagement

This example shows how to import the cmdlets and functions from a Windows PowerShell module on a remote computer into the current session.

The first command creates a PSSession on the Server01 computer and saves it in the `$S` variable.

The second command uses the `Invoke-Command` cmdlet to run an `Import-Module` command in the PSSession in `$S`.

Typically, the module would be added to all sessions by an `Import-Module` command in a Windows PowerShell profile, but profiles are not run in PSSessions.

The third command uses the Module parameter of `Import-PSSession` to import the cmdlets and functions in the module into the current session.

-------- Example 6: Create a module in a temporary file --------

PS C:\> Import-PSSession $S -CommandName Get-Date, SearchHelp -FormatTypeName * -AllowClobber

Name   : tmp_79468106-4e1d-4d90-af97-1154f9317239_tcw1zunz.ttf

Path   : C:\Users\User01\AppData\Local\Temp\tmp_79468106-4e1d-4d90-a

f97-1154f9317239_tcw1

zunz.ttf\tmp_79468106-4e1d-4d90-af97-1154f9317239_

tcw1zunz.ttf.psm1

Description   : Implicit remoting for

http://server01.corp.fabrikam.com/wsman

Guid   : 79468106-4e1d-4d90-af97-1154f9317239

Version   : 1.0

ModuleBase   : C:\Users\User01\AppData\Local\Temp\tmp_79468106-4e1d-4d90-a

f97-1154f9317239_tcw1

zunz.ttf

ModuleType   : Script

PrivateData   : {ImplicitRemoting}

AccessMode   : ReadWrite

ExportedAliases   : {}

ExportedCmdlets   : {}

ExportedFunctions : {[Get-Date, Get-Date], [SearchHelp, SearchHelp]}

ExportedVariables : {}

NestedModules   : {}

This example shows that `Import-PSSession` creates a module in a temporary

file on disk. It also shows that all commands are converted into functions

before they are imported into the current session.

The command uses the `Import-PSSession` cmdlet to import a `Get-Date` cmdlet

and a SearchHelp function into the current session.

The `Import-PSSession` cmdlet returns a PSModuleInfo object that represents

the temporary module. The value of the Path property shows that

`Import-PSSession` created a script module (.psm1) file in a temporary

location. The ExportedFunctions property shows that the `Get-Date` cmdlet and

the SearchHelp function were both imported as functions.

Example 7: Run a command that is hidden by an imported command

PS C:\> Import-PSSession $S -CommandName Get-Date -FormatTypeName *

-AllowClobber

PS C:\> Get-Command Get-Date -All

CommandType   Name   Definition

```
-----------   ----       ----------
Function     Get-Date   ...
Cmdlet       Get-Date   Get-Date [[-Date] <DateTime>] [-Year <Int32>] [-Month
<Int32>]
```

```
PS C:\> Get-Date
09074
```

```
PS C:\> (Get-Command -Type Cmdlet -Name Get-Date).PSSnapin.Name
Microsoft.PowerShell.Utility
```

```
PS C:\> Microsoft.PowerShell.Utility\Get-Date
Sunday, March 15, 2009 2:08:26 PM
```

This example shows how to run a command that is hidden by an imported command.

The first command imports a `Get-Date` cmdlet from the PSSession in the `$S` variable. Because the current session includes a `Get-Date` cmdlet, the AllowClobber parameter is required in the command.

The second command uses the All parameter of the `Get-Command` cmdlet to get all `Get-Date` commands in the current session. The output shows that the session includes the original `Get-Date` cmdlet and a `Get-Date` function. The `Get-Date` function runs the imported `Get-Date` cmdlet in the PSSession in `$S`.

The third command runs a `Get-Date` command. Because functions take precedence over cmdlets, Windows PowerShell runs the imported `Get-Date` function, which returns a Julian date.

The fourth and fifth commands show how to use a qualified name to run a command that is hidden by an imported command.

The fourth command gets the name of the Windows PowerShell snap-in that added the original `Get-Date` cmdlet to the current session.

The fifth command uses the snap-in-qualified name of the `Get-Date` cmdlet to run a `Get-Date` command.

For more information about command precedence and hidden commands, see about_Command_Precedence (../Microsoft.PowerShell.Core/about/about_Command_Precedence.md).

Example 8: Import commands that have a specific string in their names

```
PS C:\> Import-PSSession -Session $S -CommandName **Item** -AllowClobber
```

This command imports commands whose names include Item from the PSSession in `$S`. Because the command includes the CommandName parameter but not the FormatTypeData parameter, only the command is imported.

Use this command when you are using `Import-PSSession` to run a command on a remote computer and you already have the formatting data for the command in the current session.

Example 9: Use the Module parameter to discover which commands were imported into the session

```
PS C:\> $M = Import-PSSession -Session $S -CommandName *bits* -FormatTypeName *bits*
PS C:\> Get-Command -Module $M

CommandType     Name
-----------     ----
Function        Add-BitsFile
Function        Complete-BitsTransfer
Function        Get-BitsTransfer
Function        Remove-BitsTransfer
Function        Resume-BitsTransfer
```

Function   Set-BitsTransfer

Function   Start-BitsTransfer

Function   Suspend-BitsTransfer

This command shows how to use the Module parameter of `Get-Command` to find
out which commands were imported into the session by an `Import-PSSession`
command.

The first command uses the `Import-PSSession` cmdlet to import commands whose
names include "bits" from the PSSession in the `$S` variable. The
`Import-PSSession` command returns a temporary module, and the command saves
the module in the `$m` variable.

The second command uses the `Get-Command` cmdlet to get the commands that are
exported by the module in the `$M` variable.

The Module parameter takes a string value, which is designed for the module
name. However, when you submit a module object, Windows PowerShell uses the
ToString method on the module object, which returns the module name.

The `Get-Command` command is the equivalent of `Get-Command $M.Name`".

REMARKS

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

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

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

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