



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 'Import-Csv'

PS:\>Get-HELP Import-Csv -Full

NAME

Import-Csv

SYNOPSIS

Creates table-like custom objects from the items in a character-separated value (CSV) file.

SYNTAX

```
Import-Csv [[-Path] <System.String[]>] [[-Delimiter] <System.Char>] [-Encoding {ASCII | BigEndianUnicode | Default | OEM | Unicode | UTF7 | UTF8 | UTF32}] [-Header
```

```
<System.String[]>] [-LiteralPath <System.String[]>] [<CommonParameters>]
```

```
Import-Csv [[-Path] <System.String[]>] [-Encoding {ASCII | BigEndianUnicode | Default | OEM | Unicode | UTF7 | UTF8 | UTF32}] [-Header <System.String[]>]  
[-LiteralPath <System.String[]>] -UseCulture [<CommonParameters>]
```

DESCRIPTION

The `Import-Csv` cmdlet creates table-like custom objects from the items in CSV files. Each column in the CSV file becomes a property of the custom object and the

items in rows become the property values. `Import-Csv` works on any CSV file, including files that are generated by the `Export-Csv` cmdlet.

You can use the parameters of the `Import-Csv` cmdlet to specify the column header row and the item delimiter, or direct `Import-Csv` to use the list separator for the current culture as the item delimiter.

You can also use the `ConvertTo-Csv` and `ConvertFrom-Csv` cmdlets to convert objects to CSV strings (and back). These cmdlets are the same as the `Export-CSV` and `Import-Csv` cmdlets, except that they do not deal with files.

If a header row entry in a CSV file contains an empty or null value, PowerShell inserts a default header row name and displays a warning message.

`Import-Csv` uses the byte-order-mark (BOM) to detect the encoding format of the file. If the file has no BOM, it assumes the encoding is UTF8.

PARAMETERS

-Delimiter <System.Char>

Specifies the delimiter that separates the property values in the CSV file. The default is a comma (',').

Enter a character, such as a colon (`:`). To specify a semicolon (`;`) enclose it in single quotation marks. To specify escaped special characters such as tab (`
`), enclose it in double quotation marks.

If you specify a character other than the actual string delimiter in the file, `Import-Csv` cannot create the objects from the CSV strings and will return the CSV strings.

Required? false

Position? 1

Default value comma (,)

Accept pipeline input? False

Accept wildcard characters? false

-Encoding <System.String>

Specifies the type of encoding for the target file. The default value is `Default`.

The acceptable values for this parameter are as follows:

- `ASCII` Uses ASCII (7-bit) character set.

- `BigEndianUnicode` Uses UTF-16 with the big-endian byte order.

- `Default` Uses the encoding that corresponds to the system's active code page (usually ANSI).

- `OEM` Uses the encoding that corresponds to the system's current OEM code page.

- `Unicode` Uses UTF-16 with the little-endian byte order.

- `UTF7` Uses UTF-7.

- `UTF8` Uses UTF-8.

- `UTF32` Uses UTF-32 with the little-endian byte order.

Required? false

Position? named

Default value Default

Accept pipeline input? False

Accept wildcard characters? false

-Header <System.String[]>

Specifies an alternate column header row for the imported file. The column header determines the property names of the objects created by `Import-Csv`.

Enter column headers as a character-separated list. Do not enclose the header string in quotation marks. Enclose each column header in single quotation marks.

If you enter fewer column headers than there are data columns, the remaining data columns are discarded. If you enter more column headers than there are data columns, the additional column headers are created with empty data columns.

When using the Header parameter, delete the original header row from the CSV file. Otherwise, `Import-Csv` creates an extra object from the items in the header row.

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

-LiteralPath <System.String[]>

Specifies the path to the CSV file to import. Unlike Path , the value of the LiteralPath parameter is used exactly as it is typed. No characters are interpreted

as wildcards. If the path includes escape characters, enclose it in single quotation marks. Single quotation marks tell PowerShell not to interpret any characters

as escape sequences.

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

-Path <System.String[]>

Specifies the path to the CSV file to import. You can also pipe a path to `Import-Csv`.

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

-UseCulture <System.Management.Automation.SwitchParameter>

Uses the list separator for the current culture as the item delimiter. To find the list separator for a culture, use the following command:

`(Get-Culture).TextInfo.ListSeparator`.

Required? true
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 a string that contains a path to this cmdlet.

OUTPUTS

System.Object

This cmdlet returns the objects described by the content in the CSV file.

NOTES

Windows PowerShell includes the following aliases for `Import-Csv`:

- `ipcsv`

Because the imported objects are CSV versions of the object type, they are not recognized and formatted by the PowerShell type formatting entries that format the non-CSV versions of the object type.

The result of an `Import-Csv` command is a collection of strings that form a table-like custom object. Each row is a separate string, so you can use the Count

property of the object to count the table rows. The columns are the properties of the object and items in the rows are the property values.

The column header row determines the number of columns and the column names. The column names are also the names of the properties of the objects. The first row

is interpreted to be the column headers, unless you use the Header parameter to specify column headers. If any row has more values than the header row, the

additional values are ignored.

If the column header row is missing a value or contains a null or empty value, `Import-Csv` uses H followed by a number for the missing column header and property name.

In the CSV file, each object is represented by a character-separated list of the property values of the object. The property values are converted to strings by

using the ToString() method of the object, so they are represented by the name of the property value. `Export-Csv` does not export the methods of the object.

```
Get-Process | Export-Csv -Path .\Processes.csv
```

```
$P = Import-Csv -Path .\Processes.csv
```

```
$P | Get-Member
```

Type Name: 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()
BasePriority	NoteProperty	string BasePriority=8
Company	NoteProperty	string Company=Microsoft Corporation
...		

```
$P | Format-Table
```

Name	SI	Handles	VM	WS	PM	NPM	Path
ApplicationFrameHost		4	407			2199293489152	15884288
C:\WINDOWS\system32\ApplicationFrameHost.exe						15151104	23792
...							
wininit	0	157	2199112204288	4591616	1630208	10376	
winlogon	4	233	2199125549056	7659520	2826240	10992	C:\WINDOWS\System32\WinLogon.exe
WinStore.App		4	846		873435136	33652736	26607616
Files\WindowsApps\Microsoft.WindowsStore_11712.1001.13.0_x64__8weky...						55432	C:\Program
WmiPrvSE	0	201	2199100219392	8830976	3297280	10632	C:\WINDOWS\system32\wbem\wmiprvse.exe
WmiPrvSE		0	407			2199157727232	18509824
C:\WINDOWS\system32\wbem\wmiprvse.exe						12922880	16624
WUDFHost	0	834	2199310204928	51945472	87441408	24984	C:\Windows\System32\WUDFHost.exe

The `Get-Process` cmdlet sends process objects down the pipeline to the `Export-Csv`. The `Export-Csv` cmdlet converts the process objects to CSV strings and saves

the strings in the Processes.csv file. The `Import-Csv` cmdlet imports the CSV strings from the Processes.csv file. The strings are saved in the `\$P` variable. The

`\$P` variable is sent down the pipeline to the `Get-Member` cmdlet that displays the properties of the imported CSV strings. The `\$P` variable is sent down the

pipeline to the `Format-Table` cmdlet and displays the objects.

----- Example 2: Specify the delimiter -----

```
Get-Process | Export-Csv -Path .\Processes.csv -Delimiter :
```

```
$P = Import-Csv -Path .\Processes.csv -Delimiter :
```

```
$P | Format-Table
```

The `Get-Process` cmdlet sends process objects down the pipeline to `Export-Csv`. The `Export-Csv` cmdlet converts the process objects to CSV strings and saves the

strings in the Processes.csv file. The Delimiter parameter is used to specify a colon delimiter. The `Import-Csv` cmdlet imports the CSV strings from the

Processes.csv file. The strings are saved in the `\$P` variable. To `\$P` variable is sent down the pipeline to the `Format-Table` cmdlet.

--- Example 3: Specify the current culture for the delimiter ---

```
(Get-Culture).TextInfo.ListSeparator
```

```
Get-Process | Export-Csv -Path .\Processes.csv -UseCulture
```

```
Import-Csv -Path .\Processes.csv -UseCulture
```

The `Get-Culture` cmdlet uses the nested properties TextInfo and ListSeparator to get the current culture's default list separator. The `Get-Process` cmdlet sends

process objects down the pipeline to `Export-Csv`. The `Export-Csv` cmdlet converts the process objects to CSV strings and saves the strings in the Processes.csv

file. The UseCulture parameter uses the current culture's default list separator. The `Import-Csv` cmdlet imports the CSV strings from the Processes.csv file.

---- Example 4: Change property names in an imported object ----

```

Start-Job -ScriptBlock { Get-Process } | Export-Csv -Path .\Jobs.csv -NoTypeInformation

$Header = 'State', 'MoreData', 'StatusMessage', 'Location', 'Command', 'StateInfo', 'Finished',
'InstanceId', 'Id', 'Name', 'ChildJobs', 'BeginTime', 'EndTime', 'JobType', 'Output', 'Error',
'Progress', 'Verbose', 'Debug', 'Warning', 'Information'

# Delete the default header from file

$A = Get-Content -Path .\Jobs.csv

$A = $A[1..($A.Count - 1)]

$A | Out-File -FilePath .\Jobs.csv

$J = Import-Csv -Path .\Jobs.csv -Header $Header

$J

State      : Running
MoreData   : True
StatusMessage :
Location   : localhost
Command    : Get-Process
StateInfo   : Running
Finished   : System.Threading.ManualResetEvent
InstanceId  : a259eb63-6824-4b97-a033-305108ae1c2e
Id         : 1
Name       : Job1
ChildJobs  : System.Collections.Generic.List`1[System.Management.Automation.Job]
BeginTime   : 12/20/2018 18:59:57
EndTime     :
JobType    : BackgroundJob
Output     : System.Management.Automation.PSDataCollection`1[System.Management.Automation.PSObject]
Error      : System.Management.Automation.PSDataCollection`1[System.Management.Automation.ErrorRecord]
Progress   : System.Management.Automation.PSDataCollection`1[System.Management.Automation.ProgressRecord]
Verbose    : System.Management.Automation.PSDataCollection`1[System.Management.Automation.VerboseRecord]
Debug      : System.Management.Automation.PSDataCollection`1[System.Management.Automation.DebugRecord]
Warning    : System.Management.Automation.PSDataCollection`1[System.Management.Automation.WarningRecord]
Information :
System.Management.Automation.PSDataCollection`1[System.Management.Automation.InformationRecord]  Page 9/13

```

The `Start-Job` cmdlet starts a background job that runs `Get-Process`. A job object is sent down the pipeline to the `Export-Csv` cmdlet and converted to a CSV

string. The `NoTypeInformation` parameter removes the type information header from CSV output and is optional in PowerShell v6 and higher. The `'\$Header` variable

contains a custom header that replaces the following default values: `HasMoreData` , `JobStateInfo` , `PSBeginTime` , `PSEndTime` , and `PSJobTypeName` . The `'\$A` variable uses

the `Get-Content` cmdlet to get the CSV string from the `Jobs.csv` file. The `'\$A` variable is used to remove the default header from the file. The `Out-File` cmdlet

saves the new version of the `Jobs.csv` file in the `'\$A` variable. The `Import-Csv` cmdlet imports the `Jobs.csv` file and uses the `Header` parameter to apply the

`'\$Header` variable. The `'\$J` variable contains the imported `PSCustomObject` and displays the object in the PowerShell console.

----- Example 5: Create a custom object using a CSV file -----

```
Get-Content -Path .\Links.csv
```

113207,about_Aliases

113208,about_Arithmetic_Operators

113209,about_Arrays

113210,about_Assignment_Operators

113212,about_Automatic_Variables

113213,about_Break

113214,about_Command_Precedence

113215,about_Command_Syntax

144309,about_Comment_Based_Help

113216,about_CommonParameters

113217,about_Comparison_Operators

113218,about_Continue

113219,about_Core_Commands

113220,about_Data_Section

```
$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()
LinkID	NoteProperty	string LinkID=113207
TopicTitle	NoteProperty	string TopicTitle=about_Aliases

```
$A | Where-Object -Property TopicTitle -Like '*alias*'
```

```
LinkID TopicTitle
```

```
-----  
113207 about_Aliases
```

To create your Links.csv file, use the values shown in the `Get-Content` output.

The `Get-Content` cmdlet displays the Links.csv file. The `Import-Csv` cmdlet imports the Links.csv file. The Header parameter specifies the property names LinkId and

TopicTitle . The objects are stored in the `\$A` variable. The `Get-Member` cmdlet shows the property names from the Header parameter. The `Where-Object` cmdlet

selects objects with the TopicTitle property that includes alias .

```
----- Example 6: Import a CSV that is missing a value -----
```

```
Get-Content -Path .\Projects.csv
```

```
ProjectID,ProjectName,,Completed
```

```
13,Inventory,Redmond,True
```

```
440,,FarEast,True
```

```
469,Marketing,Europe,False
```

```
Import-Csv -Path .\Projects.csv
```

WARNING: One or more headers were not specified. Default names starting with "H" have been used in place of any missing headers.

```
ProjectID ProjectName H1 Completed
```

```
-----  
13 Inventory Redmond True  
440 FarEast True  
469 Marketing Europe False
```

```
(Import-Csv -Path .\Projects.csv).H1
```

WARNING: One or more headers were not specified. Default names starting with "H" have been used in place of any missing headers.

Redmond

FarEast

Europe

To create your Projects.csv file, use the values shown in the example's `Get-Content` output.

The `Get-Content` cmdlet displays the Projects.csv file. The header row is missing a value between ProjectName and Completed . The `Import-Csv` cmdlet imports the Projects.csv file and displays a warning message because H1 is a default header name. The `(Import-Csv -Path .\Projects.csv).H1` command gets the H1 property values and displays a warning.

RELATED LINKS

	Online	Version:
https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/import-csv?view=powershell-5.1&WT.mc_id=ps-gethelp		Page 12/13

ConvertFrom-Csv

ConvertTo-Csv

Export-Csv

Get-Culture