



### ***Windows PowerShell Get-Help on Cmdlet 'Export-Csv'***

***PS:\>Get-HELP Export-Csv -Full***

#### **NAME**

Export-Csv

#### **SYNOPSIS**

Converts objects into a series of character-separated value (CSV) strings and saves the strings to a file.

#### **SYNTAX**

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

[-Force] -InputObject <System.Management.Automation.PSObject> [-LiteralPath <System.String>] [-NoClobber] [-NoTypeInfo] [-Confirm] [-WhatIf] [<CommonParameters>]

Export-Csv [[-Path] <System.String>] [-Append] [-Encoding {ASCII | BigEndianUnicode | Default | OEM | Unicode | UTF7 | UTF8 | UTF32}] [-Force] -InputObject

<System.Management.Automation.PSObject> [-LiteralPath <System.String>] [-NoClobber] [-NoTypeInfo] [-UseCulture] [-Confirm] [-WhatIf] [<CommonParameters>]

## DESCRIPTION

The ``Export-CSV`` cmdlet creates a CSV file of the objects that you submit. Each object is a row that includes a character-separated list of the object's property

values. You can use the ``Export-CSV`` cmdlet to create spreadsheets and share data with programs that accept CSV files as input.

Do not format objects before sending them to the ``Export-CSV`` cmdlet. If ``Export-CSV`` receives formatted objects the CSV file contains the format properties rather

than the object properties. To export only selected properties of an object, use the ``Select-Object`` cmdlet.

## PARAMETERS

`-Append <System.Management.Automation.SwitchParameter>`

Use this parameter so that ``Export-CSV`` adds CSV output to the end of the specified file. Without this parameter, ``Export-CSV`` replaces the file contents without warning.

This parameter was introduced in Windows PowerShell 3.0.

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

`-Delimiter <System.Char>`

Specifies a delimiter to separate the property values. The default is a comma (`,`). Enter a character, such as a colon (`,`). To specify a semicolon (`,`), enclose it in quotation marks.

Required?	false
Position?	1
Default value	comma (`,`)

Accept pipeline input?     False

Accept wildcard characters? false

#### -Encoding <System.String>

Specifies the encoding for the exported CSV file. The default value is `ASCII`.

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            ASCII

Accept pipeline input?     False

Accept wildcard characters? false

#### -Force <System.Management.Automation.SwitchParameter>

This parameter allows `Export-Csv` to overwrite files with the Read Only attribute.

When Force and Append parameters are combined, objects that contain mismatched properties can be written to a CSV file. Only the properties that match are written to the file. The mismatched properties are discarded.

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

**-InputObject <System.Management.Automation.PSObject>**

Specifies the objects to export as CSV strings. Enter a variable that contains the objects or type a command or expression that gets the objects. You can also pipe objects to ``Export-CSV``.

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

**-LiteralPath <System.String>**

Specifies the path to the CSV output file. 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, use 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?	False
Accept wildcard characters?	false

**-NoClobber <System.Management.Automation.SwitchParameter>**

Use this parameter so that `Export-CSV` does not overwrite an existing file. By default, if the file exists in the specified path, `Export-CSV` overwrites the file without warning.

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

**-NoTypeInfo <System.Management.Automation.SwitchParameter>**

Removes the `#TYPE` information header from the output. This parameter became the default in PowerShell 6.0 and is included for backwards compatibility.

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

**-Path <System.String>**

A required parameter that specifies the location to save the CSV output file.

Required?	false
Position?	0
Default value	None
Accept pipeline input?	False
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?	false
Position?	named
Default value	False
Accept pipeline input?	False
Accept wildcard characters?	false

**-Confirm** <System.Management.Automation.SwitchParameter>

Prompts you for confirmation before running the cmdlet.

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

**-WhatIf** <System.Management.Automation.SwitchParameter>

Prevents the cmdlet from being processed or making changes. The output shows what would happen if the cmdlet were run.

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.Management.Automation.PSObject

You can pipe any object with an Extended Type System (ETS) adapter to this cmdlet.

## OUTPUTS

None

This cmdlet returns no output.

## NOTES

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

- ``epcsv``

The ``Export-CSV`` cmdlet converts the objects that you submit into a series of CSV strings and saves them in the specified text file. You can use ``Export-CSV`` to

save objects in a CSV file and then use the ``Import-Csv`` cmdlet to create objects from the CSV file.

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 using

the `ToString()` method. The strings are represented by the property value name. ``Export-CSV`` does not export the methods of the object.

The CSV strings are output as follows:

- By default the first string contains the `#TYPE` information header followed by the object type's fully qualified name.

For example, `#TYPE`

`System.Diagnostics.Process` . - If `NoTypeInfoInformation` is used the first string includes the column headers. The headers

contain the first object's property names

as a character-separated list. - The remaining strings contain character-separated lists of each object's property values.

When you submit multiple objects to `Export-CSV`, `Export-CSV` organizes the file based on the properties of the first object that you submit. If the remaining

objects do not have one of the specified properties, the property value of that object is null, as represented by two consecutive commas. If the remaining objects

have additional properties, those property values are not included in the file.

You can use the `Import-Csv` cmdlet to recreate objects from the CSV strings in the files. The resulting objects are CSV versions of the original objects that

consist of string representations of the property values and no methods.

The `ConvertTo-Csv` and `ConvertFrom-Csv` cmdlets convert objects to CSV strings and from CSV strings. `Export-CSV` is the same as `ConvertTo-CSV`, except that it saves the CSV strings in a file.

----- Example 1: Export process properties to a CSV file -----

```
Get-Process -Name WmiPrvSE |  
    Select-Object -Property BasePriority,Id,SessionId,WorkingSet |  
    Export-Csv -Path .\WmiData.csv -NoTypeInfoInformation  
Import-Csv -Path .\WmiData.csv
```

BasePriority	Id	SessionId	WorkingSet
--------------	----	-----------	------------

8	976	0	20267008
---	-----	---	----------

8	2292	0	36786176
---	------	---	----------

8	3816	0	30351360
---	------	---	----------

8	8604	0	15011840
---	------	---	----------

8	10008	0	8830976
---	-------	---	---------

8	11764	0	14237696
---	-------	---	----------

8	54632	0	9502720
---	-------	---	---------

8	54632	0	9502720
---	-------	---	---------



The ``Get-Process`` cmdlet gets the Process objects. The Name parameter filters the output to include only the WmiPrvSE process objects. The process objects are sent

down the pipeline to the ``Select-Object`` cmdlet. ``Select-Object`` uses the Property parameter to select a subset of process object properties. The process objects are

sent down the pipeline to the ``Export-Csv`` cmdlet. ``Export-Csv`` converts the process objects to a series of CSV strings. The Path parameter specifies that the

``WmiData.csv`` file is saved in the current directory. The NoTypeInfoInformation parameter removes the #TYPE information header from the CSV output and is not required in

PowerShell 6. The ``Import-Csv`` cmdlet uses the Path parameter to display the file located in the current directory.

---- Example 2: Export processes to a comma-delimited file ----

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

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

```
"Name","SI","Handles","VM","WS","PM","NPM","Path","Parent","Company","CPU","FileVersion", ...
```

```
"ApplicationFrameHost","4","511","2203597099008","35364864","21979136","30048", ...
```

The ``Get-Process`` cmdlet gets Process objects. The process objects are sent down the pipeline to the ``Export-Csv`` cmdlet. ``Export-Csv`` converts the process objects to

a series of CSV strings. The Path parameter specifies that the ``Processes.csv`` file is saved in the current directory. The NoTypeInfoInformation parameter removes the

#TYPE information header from the CSV output and is not required in PowerShell 6. The ``Get-Content`` cmdlet uses the Path parameter to display the file located in the current directory.

-- Example 3: Export processes to a semicolon delimited file --

```
Get-Process | Export-Csv -Path .\Processes.csv -Delimiter ';' -NoTypeInfoInformation
```

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

```
"Name";"SI";"Handles";"VM";"WS";"PM";"NPM";"Path";"Parent";"Company";"CPU";"FileVersion"; ...
```

```
"ApplicationFrameHost";"4";"509";"2203595321344";"34807808";"21770240";"29504"; ...
```

The ``Get-Process`` cmdlet gets Process objects. The process objects are sent down the pipeline to the ``Export-Csv``

cmdlet. `Export-Csv` converts the process objects to

a series of CSV strings. The Path parameter specifies that the `Processes.csv` file is saved in the current directory. The Delimiter parameter specifies a semicolon

to separate the string values. The NoTypeInfoInformation parameter removes the #TYPE information header from the CSV output and is not required in PowerShell 6. The

`Get-Content` cmdlet uses the Path parameter to display the file located in the current directory.

Example 4: Export processes using the current culture's list separator

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

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

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

```
"Name","SI","Handles","VM","WS","PM","NPM","Path","Parent","Company","CPU","FileVersion", ...
```

```
"ApplicationFrameHost","4","511","2203597099008","35364864","21979136","30048", ...
```

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

gets Process objects. The process objects are sent down the pipeline to the `Export-Csv` cmdlet. `Export-Csv` converts the process objects to a series of CSV strings.

The Path parameter specifies that the `Processes.csv` file is saved in the current directory. The UseCulture parameter uses the current culture's default list

separator as the delimiter. The NoTypeInfoInformation parameter removes the #TYPE information header from the CSV output and is not required in PowerShell 6. The

`Get-Content` cmdlet uses the Path parameter to display the file located in the current directory.

----- Example 5: Export processes with type information -----

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

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

```
#TYPE System.Diagnostics.Process
```

```
"Name","SI","Handles","VM","WS","PM","NPM","Path","Company","CPU","FileVersion", ...
```

```
"ApplicationFrameHost","4","507","2203595001856","35139584","20934656","29504", ...
```

The ``Get-Process`` cmdlet gets Process objects. The process objects are sent down the pipeline to the ``Export-Csv`` cmdlet. ``Export-Csv`` converts the process objects to

a series of CSV strings. The Path parameter specifies that the ``Processes.csv`` file is saved in the current directory. The ``Get-Content`` cmdlet uses the Path parameter to display the file located in the current directory.

----- Example 6: Export and append objects to a CSV file -----

```
$AppService = (Get-Service -DisplayName *Application* | Select-Object -Property DisplayName, Status)
```

```
$AppService | Export-Csv -Path .\Services.Csv -NoTypeInfoInformation
```

```
Get-Content -Path .\Services.Csv
```

```
$WinService = (Get-Service -DisplayName *Windows* | Select-Object -Property DisplayName, Status)
```

```
$WinService | Export-Csv -Path .\Services.csv -NoTypeInfoInformation -Append
```

```
Get-Content -Path .\Services.Csv
```

```
"DisplayName","Status"
```

```
"Application Layer Gateway Service","Stopped"
```

```
"Application Identity","Running"
```

```
"Windows Audio Endpoint Builder","Running"
```

```
"Windows Audio","Running"
```

```
"Windows Event Log","Running"
```

The ``Get-Service`` cmdlet gets service objects. The `DisplayName` parameter returns services that contain the word Application. The service objects are sent down the

pipeline to the ``Select-Object`` cmdlet. ``Select-Object`` uses the `Property` parameter to specify the `DisplayName` and `Status` properties. The ``$AppService`` variable stores the objects.

The ``$AppService`` objects are sent down the pipeline to the ``Export-Csv`` cmdlet. ``Export-Csv`` converts the service objects to a series of CSV strings. The Path

parameter specifies that the ``Services.csv`` file is saved in the current directory. The `NoTypeInfoInformation` parameter removes the `#TYPE` information header from the CSV

output and is not required in PowerShell 6. The ``Get-Content`` cmdlet uses the Path parameter to display the file located in the current directory.

The `Get-Service` and `Select-Object` cmdlets are repeated for services that contain the word Windows. The `\$WinService` variable stores the service objects. The

`Export-Csv` cmdlet uses the Append parameter to specify that the `\$WinService` objects are added to the existing `Services.csv` file. The `Get-Content` cmdlet is

repeated to display the updated file that includes the appended data.

Example 7: Format cmdlet within a pipeline creates unexpected results

```
Get-Date | Select-Object -Property DateTime, Day, DayOfWeek, DayOfYear |
```

```
Export-Csv -Path .\DateTime.csv -NoTypeInfoInformation
```

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

```
"DateTime","Day","DayOfWeek","DayOfYear"
```

```
"Wednesday, January 2, 2019 14:59:34","2","Wednesday","2"
```

```
Get-Date | Format-Table -Property DateTime, Day, DayOfWeek, DayOfYear |
```

```
Export-Csv -Path .\FTDateTime.csv -NoTypeInfoInformation
```

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

```
"ClassId2e4f51ef21dd47e99d3c952918aff9cd","pageHeaderEntry","pageFooterEntry","autosizeInfo", ...
```

```
"033ecb2bc07a4d43b5ef94ed5a35d280",,,, "Microsoft.PowerShell.Commands.Internal.Format. ...
```

```
"9e210fe47d09416682b841769c78b8a3",,,,,
```

```
"27c87ef9bbda4f709f6b4002fa4af63c",,,,,
```

```
"4ec4f0187cb04f4cb6973460dfe252df",,,,,
```

```
"cf522b78d86c486691226b40aa69e95c",,,,,
```

The `Get-Date` cmdlet gets the DateTime object. The object is sent down the pipeline to the `Select-Object` cmdlet. `Select-Object` uses the Property parameter to

select a subset of object properties. The object is sent down the pipeline to the `Export-Csv` cmdlet. `Export-Csv` converts the object to a CSV format. The Path

parameter specifies that the `DateTime.csv` file is saved in the current directory. The NoTypeInfoInformation parameter removes the #TYPE information header from the CSV

output and is not required in PowerShell 6. The `Get-Content` cmdlet uses the Path parameter to display the CSV file

located in the current directory.

When the ``Format-Table`` cmdlet is used within the pipeline to select properties unexpected results are received. ``Format-Table`` sends table format objects down the pipeline to the ``Export-Csv`` cmdlet rather than the `DateTime` object. ``Export-Csv`` converts the table format objects to a series of CSV strings. The ``Get-Content`` cmdlet displays the CSV file which contains the table format objects.

Example 8: Using the Force parameter to overwrite read-only files

```
New-Item -Path .\ReadOnly.csv -ItemType File
Set-ItemProperty -Path .\ReadOnly.csv -Name IsReadOnly -Value $true
Get-Process | Export-Csv -Path .\ReadOnly.csv -NoTypeInfo
```

Export-Csv : Access to the path 'C:\ReadOnly.csv' is denied.

At line:1 char:15

```
+ Get-Process | Export-Csv -Path .\ReadOnly.csv -NoTypeInfo
+ ~~~~~
+ CategoryInfo          : OpenError: (:) [Export-Csv], UnauthorizedAccessException
+ FullyQualifiedErrorId : FileOpenFailure,Microsoft.PowerShell.Commands.ExportCsvCommand
```

```
Get-Process | Export-Csv -Path .\ReadOnly.csv -NoTypeInfo -Force
Get-Content -Path .\ReadOnly.csv
```

```
"Name";"SI";"Handles";"VM";"WS";"PM";"NPM";"Path";"Parent";"Company";"CPU";"FileVersion"; ...
"ApplicationFrameHost";"4";"509";"2203595321344";"34807808";"21770240";"29504"; ...
```

The ``New-Item`` cmdlet uses the `Path` and `ItemType` parameters to create the ``ReadOnly.csv`` file in the current directory. The ``Set-ItemProperty`` cmdlet uses the `Name` and `Value` parameters to change the file's `IsReadOnly` property to true. The ``Get-Process`` cmdlet gets `Process` objects. The process objects are sent down the pipeline to the ``Export-Csv`` cmdlet. ``Export-Csv`` converts the process objects to a series of CSV strings. The `Path` parameter specifies that the ``ReadOnly.csv`` file is saved

in the current directory. The `NoTypeInfo` parameter removes the `#TYPE` information header from the CSV output

and is not required in PowerShell 6. The output

shows that the file is not written because access is denied.

The Force parameter is added to the `Export-Csv` cmdlet to force the export to write to the file. The `Get-Content` cmdlet uses the Path parameter to display the file

located in the current directory.

----- Example 9: Using the Force parameter with Append -----

```
$Content = [PSCustomObject]@{Name = 'PowerShell'; Version = '7.0'}
$Content | Export-Csv -Path .\ParmFile.csv -NoTypeInfoInformation
$AdditionalContent = [PSCustomObject]@{Name = 'Windows PowerShell'; Edition = 'Desktop'}
$AdditionalContent | Export-Csv -Path .\ParmFile.csv -NoTypeInfoInformation -Append
```

Export-Csv : Cannot append CSV content to the following file: ParmFile.csv.

The appended object does not have a property that corresponds to the following column:

Version. To continue with mismatched properties, add the -Force parameter, and then retry the command.

At line:1 char:22

```
+ $AdditionalContent | Export-Csv -Path .\ParmFile.csv -NoTypeInfoInformation -Append
```

```
+ ~~~~~
```

```
+ CategoryInfo          : InvalidData: (Version:String) [Export-Csv], InvalidOperationException
```

```
+ FullyQualifiedErrorId : CannotAppendCsvWithMismatchedPropertyName,Microsoft.PowerShell. ...
```

```
$AdditionalContent | Export-Csv -Path .\ParmFile.csv -NoTypeInfoInformation -Append -Force
```

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

Name	Version
PowerShell	7.0
Windows PowerShell	

An expression creates the PSCustomObject with Name and Version properties. The values are stored in the `\$Content` variable. The `\$Content` variable is sent down the

pipeline to the `Export-Csv` cmdlet. `Export-Csv` uses the Path parameter and saves the `ParmFile.csv` file in the current directory. The NoTypeInfoInformation parameter removes the #TYPE information header from the CSV output and is not required in PowerShell 6.

Another expression creates a PSCustomObject with the Name and Edition properties. The values are stored in the `\$AdditionalContent` variable. The `\$AdditionalContent` variable is sent down the pipeline to the `Export-Csv` cmdlet. The Append parameter is used to add the data to the file. The append fails because there is a property name mismatch between Version and Edition .

The `Export-Csv` cmdlet Force parameter is used to force the export to write to the file. The Edition property is discarded. The `Import-Csv` cmdlet uses the Path parameter to display the file located in the current directory.

## RELATED LINKS

Online

Version:

[https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/export-csv?view=powershell-5.1&WT.mc\\_id=ps-gethelp](https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/export-csv?view=powershell-5.1&WT.mc_id=ps-gethelp)

ConvertFrom-Csv

ConvertTo-Csv

Format-Table

Import-Csv

Select-Object