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

PowerShell Get-Help on command 'ConvertFrom-String'

PS C:\Users\wahid> Get-Help ConvertFrom-String

NAME

ConvertFrom-String

SYNOPSIS

Extracts and parses structured properties from string content.

SYNTAX

ConvertFrom-String [-InputObject] <System.String> [-Delimiter <System.String>]
[-PropertyNames <System.String[]>] [<CommonParameters>]

ConvertFrom-String [-InputObject] <System.String> [-IncludeExtent]
[-TemplateContent <System.String[]>] [-TemplateFile <System.String[]>]
[-UpdateTemplate] [<CommonParameters>]

DESCRIPTION

The `ConvertFrom-String` cmdlet extracts and parses structured properties from string content. This cmdlet generates an object by parsing text from a traditional text stream. For each string in the pipeline, the cmdlet splits the input by either a delimiter or a parse expression, and then assigns

property names to each of the resulting split elements. You can provide these property names; if you do not, they are automatically generated for you.

The cmdlet's default parameter set, ByDelimiter , splits exactly on the regular expression delimiter. It does not perform quote matching or delimiter escaping as the `Import-Csv` cmdlet does.

The cmdlet's alternate parameter set, TemplateParsing , generates elements from the groups that are captured by a regular expression. For more information on regular expressions, see about_Regular_Expressions (../Microsoft.PowerShell.Core/About/about_Regular_Expressions.md).

This cmdlet supports two modes: basic delimited parsing, and automatically-generated, example-driven parsing.

Delimited parsing, by default, splits the input at white space, and assigns property names to the resulting groups.

You can customize the delimiter by piping the `ConvertFrom-String` results into one of the `Format- ` cmdlets, or you can use the Delimiter * parameter.

The cmdlet also supports automatically-generated, example-driven parsing based on the FlashExtract, research work by Microsoft Research (https://www.microsoft.com/research/publication/flashextract-framework-data-extraction-examples/).

PARAMETERS

-Delimiter <System.String>

Specifies a regular expression that identifies the boundary between elements. Elements that are created by the split become properties in the resulting object. The delimiter is ultimately used in a call to the Split method of the type `[System.Text.RegularExpressions.RegularExpression]`.

-IncludeExtent <System.Management.Automation.SwitchParameter>
Indicates that this cmdlet includes an extent text property that is removed by default.

-InputObject <System.String>
Specifies strings received from the pipeline, or a variable that contains a string object.

-PropertyNames <System.String[]>
Specifies an array of property names to which to assign split values in the resulting object. Every line of text that you split or parse generates elements that represent property values. If the element is the result of a capture group, and that capture group is named (for example, `(?<name>)` or `(?'name')`), then the name of that capture group is assigned to the property.

If you provide any elements in the PropertyName array, those names are assigned to properties that have not yet been named.

If you provide more property names than there are fields, PowerShell ignores the extra property names. If you do not specify enough property names to name all fields, PowerShell automatically assigns numerical property names to any properties that are not named: P1 , P2 , etc.

-TemplateContent <System.String[]>
Specifies an expression, or an expression saved as a variable, that describes the properties to which this cmdlet assigns strings. The syntax of a template field specification is the following: `{[optional-typecast]<name>:<example-value>}`.

-TemplateFile <System.String[]>
Specifies a file, as an array, that contains a template for the desired parsing of the string. In the template file, properties and their values

are enclosed in brackets, as shown below. If a property, such as the Name property and its associated other properties, appears multiple times, you can add an asterisk (`*`) to indicate that this results in multiple records. This avoids extracting multiple properties into a single record.

{Name*:David Chew}

{City:Redmond}, {State:WA}

{Name*:Evan Narvaez} {Name*:Antonio Moreno}
{City:Issaquah}, {State:WA}

-UpdateTemplate <System.Management.Automation.SwitchParameter>
Indicates that this cmdlet saves the results of a learning algorithm into a comment in the template file. This makes the algorithm learning process faster. To use this parameter, you must also specify a template file with the TemplateFile parameter.

<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: Generate an object with default property names --

"Hello World" | ConvertFrom-String

P1 P2
-- --
Hello World

This command generates an object with default property names, P1 and P2 .

--------- Example 1A: Get to know the generated object ---------

```
"Hello World" | ConvertFrom-String | 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()
P1          NoteProperty string P1=Hello
P2          NoteProperty string P2=World
```

Example 2: Generate an object with default property names using a delimiter

```
"Contoso\Administrator" | ConvertFrom-String -Delimiter "\\"

P1      P2
--      --
Contoso Administrator
```

Example 3: Generate an object that contains two named properties

```
$content = Get-Content C:\Windows\System32\drivers\etc\hosts
$content = $content -match "^[^#]"
$content | ConvertFrom-String -PropertyNames IP, Server
```

```
IP            Server
--            ------
192.168.7.10   W2012R2
192.168.7.20   W2016
192.168.7.101  WIN8
192.168.7.102  WIN10
```

The `Get-Content` cmdlet stores the content of a Windows hosts file in `$content`. The second command removes any comments at the beginning of the hosts file using a regular expression that matches any line that does not start with (`#`). The last command converts the remaining text into objects with Server and IP properties.

Example 4: Use an expression as the value of the TemplateContent parameter, save the results in a variable.

```
$template = @'
{Name*:Phoebe Cat}, {phone:425-123-6789}, {age:6}
{Name*:Lucky Shot}, {phone:(206) 987-4321}, {age:12}
'@

$testText = @'
Phoebe Cat, 425-123-6789, 6
Lucky Shot, (206) 987-4321, 12
Elephant Wise, 425-888-7766, 87
Wild Shrimp, (111)  222-3333, 1
'@

$PersonalData = $testText | ConvertFrom-String -TemplateContent $template
Write-output ("Pet items found: " + ($PersonalData.Count))
$PersonalData

Pet items found: 4
```

```
Name          phone          age
----          -----          ---
Phoebe Cat    425-123-6789    6
Lucky Shot    (206) 987-4321  12
Elephant Wise 425-888-7766    87
Wild Shrimp   (111)  222-3333 1
```

Each line in the input is evaluated by the sample matches. If the line matches the examples given in the pattern, values are extracted and passed to the output variable.

The sample data, `$template`, provides two different phone formats:

- `425-123-6789`

- `(206) 987-4321`

The sample data also provides two different age formats:

- `6`

- `12`

This implies that phones like `(206) 987 4321` will not be recognized, because there's no sample data that matches that pattern because there are no hyphens.

- Example 5: Specifying data types to the generated properties -

```
$template = @'
{[string]Name*:Phoebe Cat}, {[string]phone:425-123-6789}, {[int]age:6}
{[string]Name*:Lucky Shot}, {[string]phone:(206) 987-4321}, {[int]age:12}
'@
```

```
$testText = @'
Phoebe Cat, 425-123-6789, 6
Lucky Shot, (206) 987-4321, 12
Elephant Wise, 425-888-7766, 87
Wild Shrimp, (111)  222-3333, 1
'@

$PersonalData = $testText | ConvertFrom-String -TemplateContent $template
Write-output ("Pet items found: " + ($PersonalData.Count))
$PersonalData

Pet items found: 4

Name          phone           age
----          -----           ---
Phoebe Cat    425-123-6789      6
Lucky Shot    (206) 987-4321   12
Elephant Wise 425-888-7766     87
Wild Shrimp   (111)  222-3333   1

$PersonalData | 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()
age         NoteProperty int age=6
Name        NoteProperty string Name=Phoebe Cat
```

```
phone       NoteProperty string phone=425-123-6789
```

The `Get-Member` cmdlet is used to show that the age property is an integer.

REMARKS

To see the examples, type: "get-help ConvertFrom-String -examples".

For more information, type: "get-help ConvertFrom-String -detailed".

For technical information, type: "get-help ConvertFrom-String -full".

For online help, type: "get-help ConvertFrom-String -online"