- Analyze DHCP server with PowerShell - Mon, Aug 10 2020
- Pooling printers to load-balance print jobs - Mon, Jul 27 2020

Contents of this article

It is important to understand that strings in PowerShell are always objects, whether you are dealing with literal strings or variables. Consequently, the methods of string objects provide most of the functions you need. As usual, you can display them with the *Get-Member* cmdlet.

```
"Hello world" | Get-Member
```

```
"Hello world" | Get-Member
```

The result list is impressive:

One Console for your virtual network: All hosts, VMs, users, sessions and processes

Download ControlUp now! Ad

Clone	CompareTo	Contains	CopyTo	EndsWith
Equals	GetEnumerator	GetHashCode	GetType	GetTypeCode

IndexOf	IndexOfAny	Insert	IsNormalized	LastIndexOf
LastIndexOfAny	Normalize	PadLeft	PadRight	Remove
Replace	Split	StartsWith	Substring	ToBoolean
ToByte	ToChar	ToCharArray	ToDateTime	ToDecimal
ToDouble	ToInt16	ToInt32	ToInt64	ToLower
ToLowerInvariant	ToSByte	ToSingle	ToString	ToType
ToUInt16	ToUInt32	ToUInt64	ToUpper	ToUpperInvar
Trim	TrimEnd	TrimStart		

Concatenating strings ^

However, some tasks require the use of operators. For example, PowerShell doesn't have a concat function. Instead, you can just concatenate two or more strings with the plus operator:

```
$h = "Hello"
$w = "world"
```

```
$h = "Hello"
$w = "world"
$h + " " + $w
```

Alternatively, you can expand the two strings within double quotes:

```
"$h $w"
```

```
"$h $w"
```

In the example, the space guarantees that the two words don't stick together. However, you could insert any other delimiter.

Concatenating strings in PowerShell

This method is also suitable for concatenating the elements of a string array:

```
$st = @("Say hello", " world")
"$st"
```

```
$st = @("Say hello", " world")
"$st"
```

PowerShell automatically inserts a space here between the linked array elements.

Another option for concatenating strings is the *join* operator. Its usage is perhaps a bit counterintuitive. Normally, you place the operator in front of the string unless you want to separate the strings with a delimiter after the merge:

```
-join($h,$w)
```

```
-join($h,$w)
```

If you don’t like the result “Helloworld” in the example and you want to add a space between the two words, you have to call *-join* this way:

```
$h,$w -join " "
```

```
$h,$w -join ″ ″
```

Splitting strings ^

For the opposite task (that is, for splitting strings), you can use either the *split* method or the *split* operator. The former is simpler and only allows for use of explicit delimiters; the operator, on the other hand, supports regular expressions.

In the following example, the string is split at the double “ll” and at the space. The delimiter itself falls by the wayside:

```
("Hello world").split("ll"" ")
```

```
(″Hello world″).split(″ll″″ ″)
```

You can specify the maximum number of substrings in which the string is split. PowerShell will then produce an array with the corresponding number of elements:

```
("Hello world").split("ll"" ", 2)
```

```
("Hello world").split("ll"" ", 2)
```

The result array in the above example has two elements.

Extracting substrings ^

The method *Substring* has a similar purpose because it can extract a part of the string. However, instead of using delimiters, you have to specify the position and length of the string that you want to extract:

```
("Hello world").Substring(2,5)
```

```
("Hello world").Substring(2,5)
```

This command results in “llo w”, which corresponds to the substring with a length of five characters that begins at the third character (counting begins at 0!).

The counterpart method to *Substring* is *Remove*. It deletes the specified range from the string:

```
("Hello world").Remove(2,3)
```

```
("Hello world").Remove(2,3)
```

In this example, “llo” is removed from “Hello world.”

To eliminate leading or trailing spaces, you can use *TrimStart*, *TrimEnd*, or *Trim*.

Searching and replacing characters ^

PowerShell knows a variety of techniques to find and replace substrings. For the more demanding tasks, regular expressions are available, which can be applied with the *-match* or *-replace* operators.

In addition, the string object offers several methods for this task. Of course, each of these methods has its specific purpose. *Replace* is the simplest of these methods and doesn't support regular expressions.

```
("Hello World").Replace("Hello","New")
```

```
("Hello World").Replace("Hello","New")
```

A counterpart to the *-match* operator doesn't exist. However, PowerShell supports several methods that specialize on a particular search type. For instance, *StartsWith* and *EndsWith* determine whether a string begins or ends with a certain character or string, respectively. Likewise, *Contains* tells you if a string contains a certain substring:

```
("Hello world").Contains("ll")
```

```
("Hello world").Contains("ll")
```

The above command results in *True*. To calculate the position of a character or substring in the string, *IndexOf* is at your disposal:

```
("Hello world").IndexOf("ll")
```

```
("Hello world").IndexOf("ll")
```

Don’t forget that counting starts at 0!

Searching and replacing characters in PowerShell

Comparing strings ^

In general, you can work with the same comparison operators as for numerical values to determine differences between strings. Primarily, this includes *-eq* and *-ne*, as well as *-like*, which supports wildcards.

String objects also offer methods for this purpose. If the first string is “bigger” than the second string (that is, if it comes first in the sort order), the cmdlet returns 1; if the first string is smaller, the result is -1.

```
("Hello world").CompareTo("Hello" + " " + "world")
```

```
("Hello world").CompareTo("Hello" + " " + "world")
```

In the above example, *CompareTo* returns 0 because the strings are identical. In contrast, the comparable call with *Equals* returns *True*:

```
("Hello world").Equals("Hello" + " " + "world")
```

```
("Hello world").Equals("Hello" + " " + "world")
```

Join the 4sysops PowerShell group!

Your question was not answered? Ask in the forum!

9+

19 Comments

1.

Great examples, I'm trying to find a way to perform string aggregation similar to the way wm_concat used to work in oracle sql. Wondering if you have any ideas.

Example:

csv input file containing the following Input:

empid,fname,lname,access_priv

123456,jane,doe,bldg1

123457,joe,smith,bldg1

123456,jane,doe,bldg2