How-to: Edit/Replace text within a Variable

Use the syntax below to edit and replace the characters assigned to a string variable.

```
Syntax
      %variable:StrToFind=NewStr%

      %~[param_ext]$variable:Param

Key
   StrToFind    : The characters we are looking for (not case sensitive).
   NewStr       : The chars to replace with (if any).
   variable     : The environment variable.
   param_ext    : Any filename Parameter Extension.
   Param        : A command line parameter (e.g. 1).
```

This Edit/Replace syntax can be used anywhere that you would use the %variable% such as ECHOing the variable to screen or setting one variable = another.

param_ext cannot be %* which typically represents a whole set of parameters, but this is easily worked around by setting a variable=%*

"*StrToFind*" can **begin** with an asterisk, in which case it will replace all characters to the left of "StrToFind".

NewStr can be left blank to delete characters, alternatively include ECHO: in *NewStr* if you need to generate a CR/newline in the output:

Using both an asterisk and setting *NewStr*=null will enable you to construct a left$() or right$() function using this syntax.

Examples:
The variable _test containing 12345abcabc is used for all the following examples:

```
::Replace '12345' with 'Hello '
   SET _test=12345abcABC
   SET _result=%_test:12345=Hello %
   ECHO %_result%          =Hello abcABC

::Replace the character string 'ab' with 'xy'
   SET _test=12345abcABC
   SET _result=%_test:ab=xy%
   ECHO %_result%          =12345xycxyC

::Delete the character string 'ab'
   SET _test=12345abcABC
   SET _result=%_test:ab=%
   ECHO %_result%          =12345cC

::Delete the character string 'ab' and everything before it
   SET _test=12345abcabc
   SET _result=%_test:*ab=%
   ECHO %_result%          =cabc

::Replace the character string 'ab' and everything before it with 'XY'
   SET _test=12345abcabc
   SET _result=%_test:*ab=XY%
   ECHO %_result%          =XYcabc


:: To remove characters from the right hand side of a string is
:: a two step process and requires the use of a CALL statement
:: e.g.

   SET _test=The quick brown fox jumps over the lazy dog

   :: To delete everything after the string 'brown'
   :: first delete 'brown' and everything before it
   SET _endbit=%_test:*brown=%
   Echo We dont want: [%_endbit%]

   ::Now remove this from the original string
   CALL SET _result=%%_test:%_endbit%=%%
   echo %_result%
```

The examples above assume the default Expansion of variables, if you are using DelayedExpansion then you can modify variables within a single loop/expression. Use the syntax: !_variable! instead of %_variable%

Rename a set of files (fred001.txt – fred999.txt) with a different prefix, this is similar to but more flexible than a wildcard rename, via Raymond Chen

```
Setlocal EnableDelayedExpansion
for %%i in (fred*.txt) do set "_=%%i" & ren "%%i" "!_:fred=wilma!"
```

One other advantage of DelayedExpansion is that it will allow you to replace the % character, it will still have to be escaped as %% but the replace action will then treat it like any other character:

Replace the letter P with a percent symbol:

```
Setlocal EnableDelayedExpansion
_demo=somePdemoPtextP
_demo=!_demo:P=%%!
```

Remove spaces from a text string

To delete space characters use the same syntax as above:

```
SET _no_spaces=%_some_var: =%
```

Boolean Test "does string exist ?"

To test for the existence of a value we can use a temporary variable, delete the string we are looking for (if it exists) and then compare the two variables with EQU

Example: test for the existence of the string "London" in a variable containing text (that could be in any order) "Aberdeen, London, Edinburgh"

```
Set _cities="Aberdeen, London, Edinburgh"
:: Remove London if found
Set _dummy=%_cities:London=%
IF NOT %_dummy% == %_cities% (ECHO London was found.) ELSE (ECHO London was not found.)
```

Finding items within the PATH environment variable

The %PATH% variable contains a list of folder names.

If you have a parameter containing a valid 'folder' this can be compared with the PATH variable.

This is done using the syntax: $variable:parameter

Example

%PATH% = `C:\Windows\system32;C:\Windows;C:\utils\jdk\bin`
batch parameter %1 = `C:\utils\jdk\bin`

To get the drive and Path

```
ECHO %~dp$PATH:1
```

This will either return "C:\utils\jdk\bin" or a NULL if the item is not found in the %PATH%

If the batch parameter was supplied as %2 then this would be: `ECHO %~dp$PATH:2`

This syntax can be applied where:

- The parameter is any valid parameter (%1 %2 %G) but it must contain a Full Path (not a pathname)
- The variable is %PATH% or any other variable that contains one or more Paths or pathnames separated by semicolons ;
- If nothing is found by the search, then this will return an empty string (NULL)

Be wary of using the syntax on this page to **modify** the PATH - the User path can be edited, but the System path remains read-only for most users.

Advanced Usage of %*variable*:

You can use the %*variable*: syntax and provide each of the parameters from other variables, for example if you have

%_FullString%=The ballad of John and Yoko
%_Search%=John

To remove the %_search% string from the%_FullString% you might try:

```
SET _result=%_FullString:~%_Search%=%
```

Unfortunately this will not work because the : syntax expects a value not a variable.
To work around this use the CALL command, in this case the CALL replaces the variable shown in bold with its value:

```
SET "_FullString=The ballad of John and Yoko"
SET "_Search=John"
CALL SET _result=%%_FullString:%_Search%=%%
:: If nothing was removed then the search string was not found.
If /i "%_result%"=="%_FullString%" (Echo String not found) ELSE (Echo String found)
```

"A phony smile will never replace honest integrity" ~ Bob Martinelli

Related commands:

PATH - Display or set a search path for executable files.
How-to: SUBSTRING of a variable :~
How-to: PARAMETERS - Filename Parameter Extensions.
How-to: strlen.cmd - Get string length.
How-to: ToLower.cmd - Lower case a String.
