https://riptutorial.com/batch-file/example/12859/variable-substitution

batch-file Variables in Batch Files Variable Substitution

Example#

Unlike other programming languages, in a batch file a variable is substituted by its actual value **before** the batch script is run. In other words, the substitution is made when the script is *read* into memory by the command processor, not when the script is later *run*.

This enables the use of variables as commands within the script, and as part of other variable names in the script, etc. The "script" in this context being a line - or block - of code, surrounded by round brackets: ().

But this behaviour does mean that you cannot change a variable's value inside a block!

```
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    ECHO %VAR%
    SET VAR=Goodbye
)
```

will print

```
Hello
Hello
```

since (as you see, when watching the script run in the command window) it is evaluated to:

```
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    echo Hello
    SET VAR=Goodbye
)
```

In the above example, the ECHO command is evaluated as `Hello` when the script is read into memory, so the script will echo `Hello` forever, however many passes are made through the script.

The way to achieve the more "traditional" variable behaviour (of the variable being expanded whilst the script is running) is to enable "delayed expansion". This involves adding that command into the script prior to the loop instruction (usually a FOR loop, in a batch script), and using an exclamation mark (!) instead of a percent sign (%) in the variable's name:

```
setlocal enabledelayedexpansion
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    echo !VAR!
    SET VAR=Goodbye
)
endlocal
```

will print

```
Hello
Goodbye
```

The syntax `%%a in (1,1,2)` causes the loop to run 2 times: on the first occasion, the variable bears its initial value of 'Hello', but on the second pass through the loop - having executed the second SET instruction as the last action on the 1st pass - this has changed to the revised value 'Goodbye'.

Advanced variable substitution

Now, an advanced technique. Using the CALL command allows the batch command processor to expand a variable located on the same line of the script. This can deliver multilevel expansion, by repeated CALL and modifier use.

This is useful in, for example, a FOR loop. As in the following example, where we have a numbered list of variables:

```
"c:\MyFiles\test1.txt" "c:\MyFiles\test2.txt" "c:\MyFiles\test3.txt"
```

We can achieve this using the following FOR loop:

```
setlocal enabledelayedexpansion
for %%x in (%*) do (
    set /a "i+=1"
    call set path!i!=%%~!i!
    call echo %%path!i!%%
)
endlocal
```

Output:

```
c:\MyFiles\test1.txt
c:\MyFiles\test2.txt
c:\MyFiles\test3.txt
```

Note that the variable `!i!` is first expanded to its initial value, 1, then the resulting variable, %1, is expanded to its actual value of `c:\MyFiles\test1.txt`. This is *double expansion* of the variable `i`. On the next line, `i` is again double expanded, by use of the `CALL ECHO` command together with the `%%` variable prefix, then printed to the screen (i.e. displayed on screen).

On each successive pass through the loop, the initial number is increased by 1 (due to the code `i+=1`). Thus it increases to 2 on the 2nd pass through the loop, and to 3 on the 3rd pass. Thus the string echoed to the screen alters with each pass.

PDF - Download batch-file for free