SET

Display, set, or remove CMD environment variables. Changes made with SET will remain only for the duration of the current CMD session.

```
Syntax
      SET variable
      SET variable=string
      SET "variable=string"
      SET "variable="

      SET /A "variable=expression"
      SET /P variable=[promptString]
      SET "

Key
   variable   A new or existing environment variable name e.g. _num
   string     A text string to assign to the variable.
   /A         Arithmetic expression see full details below.
   /P         Prompt for user input.
```

Variable names are not case sensitive but the contents can be.

It is good practice to avoid using any delimiter characters (spaces, commas etc) in the variable *name*.

Delimiter characters can be used in the *value* if the complete assignment is surrounded with double quotes to prevent the delimiter being interpreted.

Any extra spaces around either the variable name or the *string*, will **not** be ignored, SET is not forgiving of extra spaces like many other scripting languages. So use SET alpha=beta, not SET alpha = beta

The first character of the name must not be numeric. It is a common practice to prefix variable names with either an underscore or a dollar sign _variable or $variable, these prefixes are not required but help to prevent any confusion with the standard built-in Windows Environment variables or any other command strings.

The CMD shell will fail to read an environment variable if it contains more than 8,191 characters.

SET is an internal command.
If Command Extensions are disabled all SET commands are disabled other than simple assignments like: _variable=MyText

Display a variable:

In most contexts, surround the variable *name* with %'s and the variable's *value* will be used
e.g. To display the value of the _department variable with the ECHO command:

```
ECHO %_department%
```

If the variable name is not found in the current environment then SET will set %ERRORLEVEL% to 1 .
This can be detected using IF ERRORLEVEL ...

Including extra characters can be useful to show any white space:

```
ECHO [%_department%]
ECHO "%_department%"
```

Type SET without parameters to display all the current environment variables.

Type SET with a variable name to display that variable

```
SET _department
```

The SET command invoked with a string (and no equal sign) will display a wildcard list of all matching variables

Display variables that begin with 'P':

```
SET p
```

Display variables that begin with an underscore

```
SET _
```

Set a variable:

Example of storing a text string:

```
C:\> SET "_dept=Sales and Marketing"
C:\> set _
_dept=Sales and Marketing
```

Set a variable that contains a redirection character, note the position of the quotes which are not saved:

```
SET "_dept=Sales & Marketing"
```

One variable can be based on another, but this is not dynamic
E.g.

```
C:\> set "xx=fish"
C:\> set "msg=%xx% chips"
C:\> set msg
msg=fish chips
```

```
C:\> set "xx=sausage"
C:\> set msg
msg=fish chips
```

```
C:\> set "msg=%xx% chips"
C:\> set msg
msg=sausage chips
```

Avoid starting variable names with a number, this will avoid the variable being mis-interpreted as a parameter
%123_myvar% < > %1 23_myvar

To display undocumented system variables:

```
SET "
```

Values with Spaces - using Double Quotes

There is no hard requirement to add quotation marks even when assigning a value that includes spaces:

```
SET _variable=one two three
ECHO %_variable%
```

Adding quotation marks is a best practise and will account for any trailing spaces and special characters like '&'.
The variable contents will **not** include the surrounding quotes:

```
SET "_variable=one & two"
ECHO "%_variable%"
```

If you place quotation marks around just the value, then those quotes **will** be stored:

```
SET _variable="one & two"
ECHO %_variable%
```

This can be used for long filenames:

```
SET _QuotedPath="H:\Config files\config 64.xml"
COPY %_QuotedPath% C:\Demo\final.xml
```

Alternatively you can add quotes at the point where they are needed:

```
SET "_Filename=config 64.xml"
COPY "H:\Config files\%_Filename%" C:\Demo\final.xml
```

Variable names with spaces

A variable can contain spaces and also the variable name itself can contain spaces, therefore the following assignment:
SET _var =MyText
will create a variable called "_var " ← note the trailing space.

Prompt for user input

The /P switch allows you to set a variable equal to a line of input entered by the user.
The Prompt string is displayed before the user input is read.

```
@echo off
Set /P _ans=Please enter Department: || Set _ans=NothingChosen
:: remove &’s and quotes from the answer (via string replace)
Set _ans=%_ans:&=%
Set _ans=%_ans:"=%
If "%_ans%"=="NothingChosen" goto sub_error
If /i "%_ans%"=="finance" goto sub_finance
If /i "%_ans%"=="hr" goto sub_hr
goto:eof

:sub_finance
echo You chose the finance dept
goto:eof

:sub_hr
echo You chose the hr dept
goto:eof

:sub_error
echo Nothing was chosen
```

Both the Prompt string and the answer provided can be left empty. If the user does not enter anything (just presses return) then the variable will be unchanged and the errorlevel will be set to 1.

The script above strips out any '&' and " characters but may still break if the string provided contains both.
For user provided input, it is a good idea to fully sanitize any input string for potentially problematic characters (unicode/smart quotes etc).

The CHOICE command is an alternative for user input, CHOICE accepts only one character/keypress, when selecting from a limited set of options it will be faster to use.

Echo a string with no trailing CR/LF

The standard ECHO command will always add a CR/LF to the end of each string displayed, returning the cursor to the start of the next line.
SET /P does not do this, so it can be used to display a string. Feed a NUL character into SET /P like this, so it doesn’t wait for any user input:

```
Set /P _scratch="This is a message to the user " <nul
```

Place the first line of a file into a variable:

```
Set /P _MyVar=<MyFilename.txt
Echo %_MyVar%
```

The second and any subsequent lines of text in the file will be discarded.

In very early versions of CMD, any carriage returns/new lines (CR+LF) before the first line containing text were ignored.

Delete a variable

Type SET with just the variable name and an equals sign:

SET _department=

Better still, to be sure there is no trailing space after the = place the expression in parentheses or quotes:
(SET _department=)
  or
SET "_department="

Arithmetic expressions (SET /a)

Placing expressions in "quotes" is optional for simple arithmetic but required for any expression using logical operators or parentheses.
A best practice is to use quotes around all SET /A expressions, then they will always be in place when needed.

When referring to a variable in your expression, SET /A allows you to omit the %'s so _myvar instead of %_myvar%

The expression to be evaluated can include the following operators:
For the Modulus operator use (%) on the command line, or in a batch script it must be doubled up to (%%) as below.
This is to distinguish it from a FOR parameter.

```
   +   Add                 set /a "_num=_num+5"
   +=  Add variable        set /a "_num+=5"
   -   Subtract            set /a "_num=_num-5"
   -=  Subtract variable   set /a "_num-=5"
   *   Multiply            set /a "_num=_num*5"
   *=  Multiply variable   set /a "_num*=5"
   /   Divide              set /a "_num=_num/5"
   /=  Divide variable     set /a "_num/=5"
   %%  Modulus             set /a "_num=17%%5"
   %%= Modulus             set /a "_num%%=5"
   !   Logical negation  0 (FALSE) ⇨ 1 (TRUE) and any non-zero value (TRUE) ⇨ 0 (FALSE)
   ~   Bitwise invert
   &   AND                 set /a "_num=5&3"    0101 AND 0011 = 0001 (decimal 1)
   &=  AND variable        set /a "_num&=3"
   |   OR                  set /a "_num=5|3"    0101 OR 0011 = 0111 (decimal 7)
   |=  OR variable         set /a "_num|=3"
   ^   XOR                 set /a "_num=5^3"    0101 XOR 0011 = 0110 (decimal 6)
   ^=  XOR variable        set /a "_num=^3"
   <<  Left Shift.    (sign bit ⇨ 0) An arithmetic shift.
   >>  Right Shift.   (Fills in the sign bit such that a negative number always remains negative.)
                       Neither ShiftRight nor ShiftLeft will detect overflow.
   <<= Left Shift variable     set /a "_num<<=2"
   >>= Right Shift variable    set /a "_num>>=2"

  ( )  Parenthesis group expressions  set /a "_num=(2+3)*5"
   ,   Commas separate expressions    set /a "_num=2,_result=_num*5"
```

Any SET /A calculation that returns a fractional result will be rounded down to the nearest whole integer.

Floating point arithmetic is not supported but you can call PowerShell for that: powershell.exe 12.9999999 + 2105001.01
or in a batch file:
For /F %%G in ('powershell.exe 12.9999999 + 2105001.01') do Echo Result: %%G

If a variable name is specified as part of the expression, but is not defined in the current environment, then SET /a will use a value of 0.

SET /A arithmetic shift operators do not detect overflow which can cause problems for any non-trivial math, e.g. the bitwise invert often incorrectly reverses the + / - sign of the result.

See SET /a examples below and this forum thread for more.
also see SetX, VarSearch and VarSubstring for more on variable manipulation.

SET /A should work within the full range of 32 bit signed integer numbers (-2,147,483,648 through 2,147,483,647) but in practice for negative integers it will not go below -2,147,483,647 because the correct two's complement result 2,147,483,648 would cause a positive overflow.

Examples

SET /A "_result=2+4"
(=6)

SET /A "_result=5"
(=5)
SET /A "_result+=5"
(=10)

SET /A "_result=2<<3"
(=16) { 2 Lsh 3 = binary 10 Lsh 3 = binary 10000 = decimal 16 }

SET /A "_result=5**%%**2"
(=1) { 5/2 = 2 + 2 remainder 1 = 1 }

SET /A "_var1=_var2=_var3=10"
(sets 3 variables to the same value - undocumented syntax.)

SET /A will treat any character string in the expression as an environment variable name. This allows you to do arithmetic with environment variables without having to type any % signs to get the values. SET /A "_result=5 + _MyVar"

Multiple calculations can be performed in one line, by separating each calculation with commas, for example:

Set "_year=1999"
Set /a "_century=_year/100, _next=_century+1"

The numbers must all be within the range of 32 bit signed integer numbers (-2,147,483,648 through 2,147,483,647) to handle larger numbers use PowerShell or VBScript.

You can also store a math expression in a variable and substitute in different values, rather like a macro.

```
SET "_math=(#+6)*5"
SET /A _result="%_math:#=4%
Echo %_result%
SET /A _result="%_math:#=10%
Echo %_result%
```

Leading Zero will specify Octal

Numeric values are decimal numbers, unless prefixed by
0x for hexadecimal numbers,
0 for octal numbers.

So 0x10 = 020 = 16 decimal

The octal notation can be confusing - all numeric values that start with zeros are treated as octal but 08 and 09 are not valid octal digits.
For example SET /a "_month=07" will return the value 7, but SET /a "_month=09" will return an error.

Permanent changes

Changes made using the SET command are NOT permanent, they apply to the current CMD prompt only and remain only until the CMD window is closed.
To permanently change a variable at the command line use SetX
or with the GUI: Control Panel ➞ System ➞ Environment ➞ System/User Variables

Changing a variable permanently with SetX will not affect any CMD prompt that is already open.
Only new CMD prompts will get the new setting.

You can of course use SetX in conjunction with SET to change both at the same time:

```
Set _Library=T:\Library\
SetX _Library T:\Library\ /m
```

Change the environment for other sessions

Neither SET nor SetX will affect other CMD sessions that are already running on the machine. This as a good thing, particularly on multi-user machines, your scripts won’t have to contend with a dynamically changing environment while they are running.

It is possible to add permanent environment variables to the registry (HKCU\Environment), but this is an undocumented (and likely unsupported) technique and still it will not take effect until the users next login.

System environment variables can be found in the registry here:
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment

CALL SET

The CALL SET syntax will expand any variables passed on the same line, which is useful if you need to set one variable based on the value of another variable. CALL SET can also evaluate a variable substring, the CALL page has more detail on this technique, though in many cases a faster to execute solution is to use Setlocal EnableDelayedExpansion.

Autoexec.bat

Any SET statement in c:\autoexec.bat will be parsed at boot time
Variables set in this way are not available to 32 bit gui programs - they won't appear in the control panel.
They will appear at the CMD prompt.

If autoexec.bat CALLS any secondary batch files, the additional batch files will NOT be parsed at boot.
This behaviour can be useful on a dual boot PC.

Errorlevels

When CMD Command Extensions are enabled (the default):

Event	Errorlevel
If the variable was successfully changed.	Errorlevel = *unchanged*,
SET No variable found or invalid name.	
SET _var=value when _var name starts with "/" and not enclosed in quotes.	1
SET /P Empty response from user.	
SET /A Unbalanced parentheses	1073750988
SET /A Missing operand	1073750989
SET /A Syntax error	1073750990
SET /A Invalid number	1073750991
SET /A Number larger than 32-bits	1073750992
SET /A Division by zero	1073750993

If the Errorlevel is *unchanged*, typically it will be 0 but if a previous command set an errorlevel, that will be preserved (this is a bug).

# I got my mind set on you
# I got my mind set on you... - Rudy Clark (James Ray/George Harrison)

Related commands

Syntax - VarSubstring Extract part of a variable (substring).
Syntax - VarSearch Search & replace part of a variable.
Syntax - Environment Variables - List of default variables.
CALL - Evaluate environment variables.
CHOICE - Accept keyboard input to a batch file.
ENDLOCAL - End localisation of environment changes, use to return values.
EXIT - Set a specific ERRORLEVEL.
PATH - Display or set a search path for executable files.
REG - Read or Set Registry values.
SETLOCAL - Begin localisation of environment variable changes.
SETX - Set an environment variable permanently.
Parameters - get a full or partial pathname from a command line variable.
StackOverflow - Storing a Newline in a variable.
Equivalent PowerShell: Set-Variable - Set a variable and a value (set/sv).
Equivalent PowerShell: Read-Host - Prompt for user input.
Equivalent bash command (Linux): env - Display, set, or remove environment variables.
