

[SS64.com](#)[Index](#)[Latest Posts](#)[User list](#)[Search](#)[Register](#)[Login](#)

You are not logged in.

Topics: [Active](#) | [Unanswered](#)

[Index](#) » [Windows CMD Shell](#) » [Incomplete documentation for SET /A](#)

Pages: 1

npocmaka

16 Jan 2013 01:37

#1

Member

From: Bulgaria

Registered: 03 Dec 2009

Posts: 140

Just compared the SET /? and [SS64](#) page.

Nothing is mentioned about : ~ , ! (and in fact also for - as a unary operator) and , . As they are almost obvious the parentheses also are not mentioned as logical separators ..

~ is the most strange - It acts as -(x+1) :

```
C:\>set /a "i=~5"
-6
C:\>set /a "i=~(-5)"
4
```

EDIT: according to this -

[http://sourcedaddy.com/windows-7/settin ...](http://sourcedaddy.com/windows-7/settin...)

[mmand.html](#) - it's a bitwise invert (..). ! definitely does not look like bitwise not to me.

! is the most useless -In fact I don't know if it can do anything at all - it acts as every non-arithmetic symbol and used as a unary operator converts anything to 0 :

```
C:\>set /a "i=s7"
0
C:\>set /a "i=!7"
0
```

EDIT: Of course there is (as always) an old DOSTIPS thread where dbenham explains everything - <http://www.dostips.com/forum/viewtopic.php?f=3&t=2196>

! can be used against zero. And help about << and >> seems to be not correct (check the example).

I have no idea why it's included in the help (I still hope for some uber-super-duper hidden feature).

, is the most fun (I have definitions for all of them :-))
 .With this a few arithmetic variables could be initialized or an arithmetic operation could be executed without being assigned to a variable:

```
C:\>set /a "i=1,5+3"
8
C:\>set /a "i=1,2+2"
4
C:\>set /a "i=1,2+2,j=3+2"
5
C:\>echo %i% and %j%
1 and 5
C:\>set /a "2+2"
4
```

Also can break the parentheses context:

```
C:\>set /a "i=1,(2,+23)+2"
25
```

Last edited by npocmaka (16 Jan 2013 11:46)

<http://www.facebook.com/npocmaka>

Offline

npocmaka

16 Jan 2013 02:57

#2

Member

From: Bulgaria

Registered: 03 Dec 2009

Posts: 140

And three more pitfalls of set /a :

```
C:\>set /a "a=1,b=1+%a%"
Missing operand.
```

with enabled delayed expansion does not produce an error but the value of b will be 1:

```
@echo off
setlocal ENABLEDELAYEDEXPANSION
set /a "a=1,b=1+!a!"
echo !b!
endlocal
```

do not work and backward too :-)

here b will be 0 :

```
@echo off
setlocal ENABLEDELAYEDEXPANSION
set /a "a=0"
set /a "b=!!a!"
echo !b!
endlocal
```

call set /a works but not in these cases:

```
@echo off
setlocal
set /a "a=5"
call set /a "b=6^%a%"
call set /a "b=6%%a%"
endlocal
```

<http://www.facebook.com/npocmaka>

Offline

carlos

16 Jan 2013 05:58

#3

Member

Registered: 04 Nov 2008

Posts: 152

npocmaka. I write that I learn in my experience with this command. The /A in set maybe means arithmetic, and means that the expression will be evaluated. Then:

```
set "n=5"
```

is more faster than:

```
set /a "n=5"
```

because is not necessary evaluated the expression.
Always the result is saved as text in the block of environment variables.

The ! (logical negation), for work with it is necessary turn off delayed expansion. 0 is turned 1 and all other is turned 0.

The ~ (bitwise negation) is complement.

The - (sign negation) positive sign turn negative, and negative sign turn positive.

With extensions enabled you can use the other expansion:

Is not necessary turn on delayedexpansion.

You only use the name of the variable:

```
Set "number=10"  
Set /a "new=1+number"
```

Last edited by carlos (16 Jan 2013 12:20)

Offline

npocmaka

16 Jan 2013 08:05

#4

Member

From: Bulgaria

Registered: 03 Dec 2009

Posts: 140

carlos wrote:

```
Set "number=10"  
Set /a "new=1+number"
```

this was unexpected but cool ...

Turns out that consecutive sets work after all:

```
C:\>set /a "first=1,second=first+1  
2
```

works also with enabledelayedexpansion.

Last edited by npocmaka (16 Jan 2013 11:42)

<http://www.facebook.com/npocmaka>

Offline

carlos

16 Jan 2013 12:14

#5

Member

Registered: 04 Nov 2008

Posts: 152

This is a example of use of ! operator:

```
@Echo Off
SetLocal Enableextensions
Call :isLeap 2008
Echo 2008 : leap : %errorlevel%
Call :isLeap 2000
Echo 2000 : leap : %errorlevel%
Call :isLeap 1900
Echo 1999 : leap : %errorlevel%
Pause
Goto :Eof

:isLeap
::Author: Carlos
::Return 0 or 1 if is leap year
::Argument: year
SetLocal EnableExtensions Disabled
Set /A "ly=(!(%~1%%4)&!!(%~1%%100))"
Exit /B %ly%
```

Last edited by carlos (16 Jan 2013 12:16)

Offline

npocmaka

16 Jan 2013 15:47

#6

Member

From: Bulgaria

Registered: 03 Dec 2009

Posts: 140

carlos wrote:

The ! (logical negation), for work with it is necessary turn off delayed expansion. 0 is turned 1 and all other is turned 0.

Hmm.. Probably cmd.exe searches for a variable when expansion is on - and does not effect then.

carlos wrote:

With extensions enabled you can use the other expansion:

More interesting SET /A does not work with DISABLEEXTENSIONS at all :

```
@echo off
setlocal
setlocal ENABLEDELAYEDEXPANSION
set /a "test1=!0"
echo !test1!
set /a "test1=!1"
echo !test1!

endlocal
set /a "test2=!0"
echo %test2%
set /a "test2=!1"
echo %test2%

setlocal DISABLEEXTENSIONS
set /a "test5= 1 + 1"
rem will not display **2*
echo **%test5%**
endlocal
```

Last edited by npocmaka (16 Jan 2013 15:59)

<http://www.facebook.com/npocmaka>

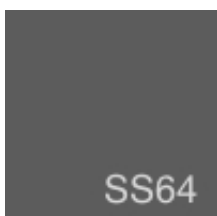
Offline

Simon Sheppard

17 Jan 2013 02:17

#7

Super Administrator



Registered: 27 Aug 2005

I've updated the SET page now, thanks for flagging this up npocmaka

Posts: 660

[Website](#)

Offline

Aacini

17 Jan 2013 03:20

#8

Member

Registered: 05 Dec 2012

Posts: 9

Comma operator allows to perform large calculations in just one line, like in this example:

```
REM CALCULATE MONTH AND DAY OF THE
SET YY=%1
SET /A A=YY%%19, B=YY/100, C=YY%%1
ECHO Easter Sunday of year %YY% is
```

You can use the assignment operator(s) at any point inside the expression, but I don't like use this feature because the result looks weird and may cause doubts in the order of evaluation:

```
SET /A D=(B=YY/100)/4, E=B%4, G=(
```

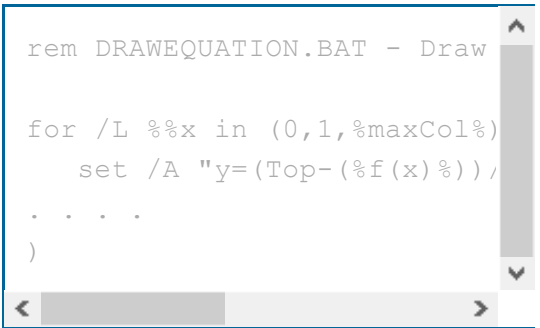
The fact that SET /A command can process *variable names* may save you from achieve an additional value expansion when array elements are processed inside a FOR loop, like in [this example](#):

```
rem Copy N random numeric elements
for /L %%i in (1,1,%N%) do (
    set /A index=(!random!*upperLim
    set /A New[!index!]=Original[!i
)
```

Note that you can not modify a variable and use its value as subscript for an array element in the same SET /A command; that is, this does NOT work:

```
for /L %%i in (1,1,%N%) do (
    set /A index=(!random!*upperLim
)
```

However, you can combine a %normal% expansion, a !delayed! expansion and the variable name expansion of SET /A command to achieve a complex calculation in a simple way, as described in [this example](#):



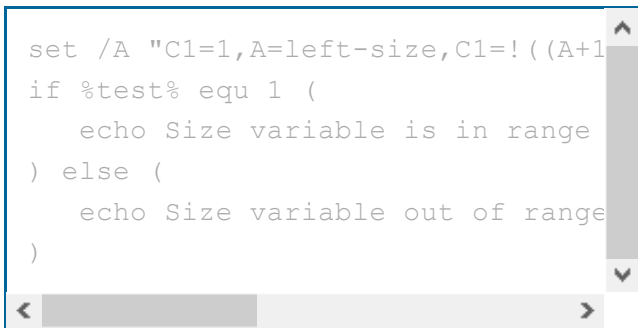
```
rem DRAWEQUATION.BAT - Draw

for /L %%x in (0,1,%maxCol%)
    set /A "y=(Top-(%f(x)%))"
    . . .
)
```

It is interesting to note that in the evaluation of previous equation, the substitution of variable values are performed *three times* in the same line:

- 0- Read the equation: $f(x)=\text{SIN}[x!]*20>>16$
- 1- Normal %variable% expansion: set /A "y=%f(x)%" becomes: set /A "y=SIN[x!]*20>>16"
- 2- Delayed !variable! expansion: set /A "y=SIN[x!]*20>>16" becomes: set /A "y=SIN[1]*20>>16" (when x=1)
- 3- Replace variable values in SET /A command: set /A "y=1144*20>>16"

Exclamation-mark (logical NOT) operator is the key to achieve conditional expressions in a way equivalent to C's conditional operator (cond)?then:else, but just for certain particular cases. For example, next line check if SIZE variable value is in range from LEFT to RIGHT (emulating an AND operation):



```
set /A "C1=1,A=left-size,C1=!((A+1
if %test% equ 1 (
    echo Size variable is in range
) else (
    echo Size variable out of range
)
```

You may review full details on the use of this technique at [this post](#).

Although SET /A documentation indicate that >> operator is right "logical shift" (RSH), it really operate as right "arithmetic shift" (RAS). If you need a pure logical shift, you must cancel the sign bit after the shift:

```
C:>set /A var=-1
-1
C:>set /A "ras=var>>1"
-1
C:>set /A "rsh=var>>1&~(1<<31)"
2147483647
```

Last line is also an example on the use of bitwise AND (&) and bitwise NOT (~) operators.

Antonio

Offline

Pages: 1

[Index](#) » [Windows CMD Shell](#)
» [Incomplete documentation for SET /A](#)

Jump to

Windows CMD Shell



Go

[Atom topic feed](#)

Powered by [FluxBB](#)