JUDAGO'S SCRIPTS / CONSOLE STUFF

SO LONG.....

BATCH VARIABLE SLIP UP'S

This is a verbatim copy of a how to I posted on another site a while ago. I assure you I wrote every word, I just didn't post it here first:

Content of this page:

BATCH SCRIPT VARIABLE EXPANSION, WIN2000 AND UP.

Variable expansion in Windows 2000 and higher batch scripts.

~~A great deal of questions in the Programming forum are related to problems caused by the nature of variable expansion in batch scripts. Here I will attempt to outline some of the most common issues.~~

My variable doesn't seem to work for some reason???

Does the variable in question start with a number?

Batch variables shouldn't start with numbers, in some circumstances they will work, but it's more trouble than it's worth. Variables that start with numbers conflict with parameters, lets say we have a variable called "2var", we can normally set to it fine but when expanding "%2var%" it is interpreted as the batch parameter "%2" then the text "var%";. A delayed expansion variable !2var! will work fine in most cases.

It is also worth noting that "set /a", flatout refuses to assign to variables that start with numbers and won't perform arithmetic on them unless they are expanded.

Why doesn't [insert variable name] change in my for loop/if statement?

Firstly you must understand the manner that variables are expanded and how batch scripts are interpreted.

Batch scripts are read line-by-line into the command processor cmd.exe(or command.com for DOS and Win9x). Any variables are replaced with their values and then the line is executed; variables are updated after each line is interpreted.

Enter code blocks: A group of commands enclosed in brackets is a code block. They are most commonly seen trailing for loops or if statements but can be used on there own. Code blocks are treated differently in terms of variable expansion, they are treated as one line. This means any variables that are set inside the block don't get updated until the block has finished execution, in the case of for loops the variables aren't updated until the entire loop has finished executing.

Here's an example:

Normal code:

```
set test=Hello!
echo %test%
```

The result is:

```
Hello!
```

Inside a code block:

```
set test=Goodbye!
if 1==1 (
    set test=Hello!
    echo %test%
)
```

The result is:

```
Goodbye!
```

It may seem like an unexpected result but you must consider that %test% was expanded to what it was set to before the code block was executed because it is considered one line.

Windows 2000 and up has a way to combat this problem, delayed expansion. To use delayed expansion add this to the top of the script:

```
setlocal enabledelayedexpansion
```

With that in place any variables that need to be set to and updated within a code block should be marked with ! exclamation marks instead of % percentage signs. In the script above if delayed expansion was enabled "echo !test!" would have yielded "Hello!".

Unfortunately delayed expansion has it's problems:

1. Recursion, if a script calls itself a way to avoid executing "setlocal enabledelayedexpansion" on every recursion is important because setlocal can only nest a finite number of times. If you call setlocal too many times you may get an error along the lines of "maximum setlocal recursion level reached".

2. Where is my exclamation mark^^!, with delayed expansion enabled exclamation marks must be prefixed with two ^^ carets if they don't mark a variable. This can be a real problem when the exclamation marks are contained in a file a for /f loop is iterating over because they disappear right out of the for loop variable %%! %%! %%!. Variables that already contain ! exclamation marks before delayed expansion is enabled are unaffected, as are variables obtained using "set /p".

3. Environment variable vs local environment variable, unfortunately there is, to the best of my knowledge, no way to use delayed expansion with out using setlocal. If a script uses setlocal all variables set within it are reset to their original states on exit, often **without even calling endlocal**. This can be a mixed blessing, sometimes you don't need the variable returned, other times it can be very useful....

If there are just a few variables you want returned it is possible to use the default expansion to your advantage; you can return as many as you want in this fashion:

```
endlocal & set variable=%variable%
```

or

```
(
    endlocal
    set variable=%variable%
)
```

How do I nest variables?

Spoiler: This may take a while, the rest of this how to it nesting related.

There are actually a couple of ways to nest variables, some even work right back to DOS, others make use of delayed expansion. It's even possible to go a couple of levels deep.

All of these examples are for batch scripts as opposed to the interactive command line, nesting still works on the command line but the number of % percentage signs is different.

The call method:

Firstly you need to know that % single percentage signs expand variables and %% double percentage signs expand to a single percentage(it's to do with the way input is parsed). Call can work with some commands to expand/update variables multiple times on a single line, just be aware that it only works with some commands. Set and echo are two commands calling will work for, while others like if won't work. If(he he) your command won't take this abuse you can set an intermediate variable, just use set the way echo is used below.

It starts simple then turns into a headache quickly as each level reduces two %% percentage signs to one and the next level does the same.

There are probably better ways to work this out, but here is one way:

The bottom nested variable is the one you want to display/use it requires one % percentage sign on either side, double it and add one for the next level(three on each side now), the next level requires you to double this and add one(now it's seven each side), and so on. Each time you do this you must also add an extra call to the front of the command.

Here are a couple of examples:

```
set level1=Hello!
set level2=level1
call echo %%%level2%%%
```

```
set level1=Hello!
set level2=level1
set level3=level2
call call echo %%%%%%%level3%%%%%%%
```

```
set level1=Hello!
set level2=level1
set level3=level2
set level4=level3
call call call echo %%%%%%%%%%%%%%%level4%%%%%%%%%%%%%%%
```

Following this method it is possible to go through many levels (any sane person would probably find another way) so naturally I kept going, the last level that worked was level11!(on xp sp3) The line length was 4161 characters, level12 yielded "Line input too long" - Geeze I think I'm going to write a compliant or something ;)

That's great you say, but I have a variable that when combined with another will make one super variable (which I shall rule the world with! BU WA HA HA HA!) well I got you supervillains covered too.

The call method works with these too; the most common use is numbered variables so that's the example I will use here.

```
set num=1
set var=test
set test1=Hello!
call echo %%%var%%num%%%
```

I only used one call here because "var" and "num" both expand the first time and %test1% expands the second time. If you want to further nest/expand the above principles still apply, but you must remember that %test1% counts as a level, and that it still needs the % percentage signs to expand test1 from %var% and %num%.

```
set num=1
set var=test
set test1=level1
set level1=Hello!!
call call echo %%%%%%%var%%num%%%%%%%
```

Although the above can happen it's more likely that the base variable is constant.

```
set num=1
set test1=Hello!!
call echo %%test%num%%%
```

It's even more common that you have x variables that a numbered, for this I suggest the for /l loop:

```
set test0=Greetings good friend!
set test1=Hello My friend!
set test2=Um..........
set test3=How about that local sports team?
set test4=Um.......I don't like sports.
set test5=........Oh...........
set test6=How about this weather?
set test7=Yeah..........
set test8=um......goodbye.....
set test9=...oh..uh..yeah...goodbye.....
set num=9
for /l %%a in (0,1,%num%) do call echo %%test%%a%%
```

This brings me to an important point, alternatives to delayed expansion; if you don't want to/can't use it delayed expansion call can help. Just treat a variable like it's nested but don't add the extra % percentage sign because you will expand the contents as a variable instead of displaying/using the contents.

```
set test=Goodbye already!
(
    set test=Hello!!
    call echo %%test%%
    echo %test%
)
```

Your command, unlike echo or set, won't take it you say? Using set is just as useless because they are set in the same code block? for /f can help, just do your command(in place of echo) with %%~a instead of "%test%". Some special characters will have problems, but only if "test" contains any double quotes.

```
set test=Well well.....
(
    set test=Hello!
    for /f "delims=" %%a in ('call echo "%%test%%"') do echo %%~a
    echo %test%
)
```

Enter delayed expansion:

Delayed expansion makes nesting variables much easier for a single level. Simply use ! exclamation marks on the outer variable and either standard variable(s) or text and standard variable(s) within. This should cover what I was rambling on about before:

```
setlocal enabledelayedexpansion
set level1=Hello!
set level2=level1
set num=2
echo !%level2%!
echo !level%num%!
for /l %%a in (1,1,2) do echo !level%%a!
```

Now what happens if you want to combine two variables both set in the same code block?

```
setlocal enabledelayedexpansion
(
    set part1=Hello^^!
    set num=1
    for /f "delims=" %%a in ("!num!") do echo !part%%a!
)
```

~~Well that just about does it, if you want to ask a question regarding this please do NOT send me a pm, post it on the programming forum instead.~~