Batch File Guide

Prepared By : Eng. Mohamed A. Baobeid

University of Aden - Aden- Yemen

This is an attempt at explaining the MSDOS batch language. It is not complete, just the basics. For more information on individual commands refer DOS's built-in HELP command. Some familiarity of DOS is assumed, you should already know what directories are and how to use common commands like CD, MD, DEL, RENAME etc.

Table of Contents

- Elements of the batch language
  - Variables
  - Redirection and Pipes
  - Labels, Conditions and Branching
  - Subroutines, CALL and FOR
- Launching programs
  - The path...
  - Batch for running a virus scanner
  - Launching Windows 95 programs and associated files
- Menus for programs
  - Simple example game menu
  - Using menuing systems
- Obtaining user input
  - Batch routine for entering strings
- How to set up SHELL and ANSI
- Processing Lists of Data
  - Demo batches for processing subdirectories
  - Windows 95 considerations
  - A method with does not use loadfix.com
- Creating and running other programs from batch
  - batch/qbasic string input routine
  - batch/qbasic global string change utility
  - intermixing perl and batch
- Making decisions based on the output of a program

All of the examples assume English DOS 6, some may run under DOS 5 but don't count on it. DOS 6 adds features which are very useful to batch programmers, like CHOICE and a version of FIND that returns an errorlevel.

Elements of the Batch Programming Language

The best way to learn batch is to experiment while referring to the HELP command, it explains everything. Batch programs are essentially sequences of DOS commands that 'type themselves' when the batch file is run.

Batch files consist of control elements, internal DOS commands and external commands or programs. With a little ingenuity one can duplicate most of the functions of a general purpose language, but doing any kind of math is tricky, there are no arithmetic commands at all. For the types of things batch is used for, this is not much of a problem.

Variables

Batch uses the percentage sign (%) to mark variables. The set command is used to assign and clear variables. When DOS reads the batch file, strings like `copy %var1% %var2%` will be expanded to read say `copy source.fil dest.fil` (assuming that's what they're set to) before feeding the line to the command interpreter. %% is reduced to % when interpreted, so the at-the-prompt command`for %a in (1 2 3) do echo %a` has to be written in a batch file as `for %%a in (1 2 3) do echo %%a`.

Variable names have eight significant characters (I think) and are always stored as upper case (except for the 'windir' variable added by Windows). [Testing under Win95 shows that variable names can be longer than 8 characters and all characters are significant, can't say about Dos 6.] In addition to user-set variables, the entire command line is passed to the batch as the read-only variables %0 to %9. %0 is the actual batch name *as typed*, the rest are parameters. The `shift` command moves everything down by one allowing parameters past the ninth to be retrieved (this wipes out the %0 parameter so if used for the batch to call itself it must be saved to another variable).

The following batch illustrates the use of variables...

```
@echo off
set var1=Hello
set var2=World
echo %var1% %var2%!
set var1=
set var2=
```

Explanation - the first line prevents the commands from being displayed, the @ in `@echo off` keeps *that* line from displaying. The set command is used to set two variables to "Hello" and "World". Next, `echo` is used to display the two variables to the screen followed by "!" producing the classic Hello World! display. Finally the two variables are cleared to keep them from needlessly consuming environment space.

Speaking of environment, you should specify more space than stock DOS gives you by using a CONFIG.SYS line similar to...

```
shell=c:\command.com /e:1024 /p
```

The number after /e: specifies how much space to set aside, usually 1024 is plenty. You can also execute `command /e:5000 /c progname` if you need a bunch of space. This can be included inside the batch itself. For quickies, especially if you wish to restore all variables to their original state after testing something, just enter `command /e:5000`. Type `exit` to quit the command shell and return to the previous environment.

Redirection and Pipes

Normally, input is taken from the keyboard and output goes to the console. Redirection allows input and output to refer to a file or device instead. Pipes allow the output of one program to be used as input to another program. These symbols only work with programs that read from "standard input" and write to "standard output" but fortunately this includes most DOS commands.

- The < symbol causes file to be fed to the program as input.
- The > symbol causes the program's output to be sent to the following file or device.
- The >> symbol causes the program's output to be appended to the file or device.
- The | symbol (the pipe) causes the output of the preceding program to be sent to the following program.

The following example shows how to use redirection with the FIND command...

```
@echo off
find %1<%2>nul
if not errorlevel 1 echo %2 contains %1
```

If saved as say DOESIT.BAT, entering `doesit "Word" text.fil` will print `text.fil contains "Word"` if the file TEXT.FIL contains the string "Word" (at least under DOS 6). Since we're not interested in the actual output of the FIND command, it is redirected to the nul device.

Labels, Conditions and Branching

A label is any line that begins with a colon. Comments in batch code can be made by using a double-colon, this is better than using the REM command because labels are processed before redirection symbols. `::<remark>` causes no problems but `rem <remark>` produces errors.

The `goto` command is used to transfer control to another part of the batch file. For example...

```
:: test goto's
@echo off
goto start
*** This text can be anything ***
*** because it will never run ***
:start
```

```
echo Done.
```

The decision mechanism in batch is the `if` command. It can be used to compare one string to another, determine if a file exists or determine the errorlevel returned by a previous command. If the evaluation is true, the rest of the command is executed. The `not` modifier reverses the evaluation results. Examples...

```
if not %var%.==. goto got_var
if exist MYFILE.DAT goto got_file
if errorlevel 5 echo Errorlevel is equal or greater than 5
if not errorlevel 4 echo Errorlevel is less than 4
```

Notice the periods in the `if %var%.==.` example, they keep empty variables from producing syntax errors. '`%var%.` is equal to `.` only if the string is empty. The way the command is worded it branches to `:got_var` only if %var% is not empty. Read the wording in the `if errorlevel` command, any errorlevel equal or greater than 5 is evaluated as true.

Subroutines, CALL and FOR

Most of the elements are in place, but still no subroutines. These can be tackled either by defining a return value and GOTO'ing the routine, which GOTO's to the value of the return variable upon completion. For example...

```
set return=next1
goto subroutine
:next1
-- bla bla --
goto end
:subroutine
-- bla bla --
goto %return%
:end
```

A more elegant way is to use a branch at the top of the batch that branches to the subroutine. This can be combined with the `for` and `call` commands to perform a sequence of steps. Here is an example that prints the names of files in the current directory matching filespecs on the command line...

```
@echo off
if %1.==Sub. goto %2
for %%a in (%1 %2 %3 %4 %5) do call %0 Sub printname %%a
goto end
:printname
echo %3
:end
```

Launching programs

One of the most useful aspects of batch is its ability to set up custom environments for running programs. Here is a typical made-up scenario: Suppose you have a game that requires you to change to say the C:\GAMES\MUT then run the program MUTANT with the command line parameter /NSB. Rather than typing all that, just put it into a batch and place it in a path directory.

The path - this is a list of directories held in the %PATH% variable that DOS uses to find programs without having to specify the directory it's in. For example, the default path in a simple setup might read (from autoexec.bat)...

```
path C:\DOS;C:\WINDOWS;C:\
```

Each directory is separated by a semicolon. When a command is typed first DOS tries the current directory. If not there it begins searching each directory listed in the path line. It is convenient to place batch files in a path directory. If you haven't done so yet, make the directory C:\BATCH (for example) and add it to the path line, as in...

```
path C:\DOS;C:\WINDOWS;C:\;C:\BATCH
```

Any batch file now placed in C:\BATCH can be run from anywhere without having to change directories. Now, back to the mythical example, let's put all of the required commands into a file called say MUTANT.BAT and place it in C:\BATCH (on the path now)...

```
@echo off
c:
cd \games\mut
mutant /NSB
```

The first line turns off command echo (contained at the beginning of almost all batch files), the next line ensures the C drive is active, then it changes to the correct directory using the cd command and runs the game with the correct command line parameter.

Here is an example for a virus scanner that does not change the current directory, but instead calls a program in another directory. In this particular example, if the batch is called without parameters defaults are supplied instead...

```
@echo off
set parm=%1 %2 %3 %4 %5 %6 %7 %8 %9
if %1.==. set parm=. /nomem /sub
c:\scanners\scan\scan.exe %parm%
set parm=
```

The specific commands, directories and parameters will vary depending on the software and where it is installed. The command line is stored in the variable %parm%, if the first parameter %1 is empty, %parm% is set to a useful default, for SCAN this translates to "scan the current and all subdirectories under it, do not scan memory". If this batch was saved as say SCN.BAT into a path directory (got that

C:\BATCH directory yet?) any directory branch may be scanned by just typing SCN instead of having to type the directory\command parameters.

Launching Windows 95 programs and associated files...

Windows 95 has a built-in utility for launching Windows programs and files that are associated with Windows programs directly from batch, a very useful addition! Previously this required special utilities to "bridge the gap". The basic syntax is simple...

```
start /w filename [command line arguments]
```

The full filename with directory information must be specified if not in the current directory or on the path, as usual. If filename is a long filename or contains spaces, enclose it in quotes. The /w option causes the batch to wait for the file to be closed before continuing with the batch, otherwise batch processing continues in parallel. Other start options include /m to run minimized and /max to run full screen. For example...

```
start /w /max "C:\Program Files\Myprogram\readme.txt"
```

Anything that can be double-clicked to run can be launched with start. One very useful application of this is running Windows files from inside a dos-based zip file viewer like AView that allows different viewers to be set up for various extensions. Here is a general purpose viewer for using with dos programs...

```
:: RUNAFILE.BAT (for example)
@echo off
start /w %1
```

Windows will figure out what to launched based on the selected file's extension.

Menus for programs

The CHOICE command that comes with DOS 6 is handy for creating custom menus for DOS programs as well as simple Yes/No questions. Here is a simple menu for a few games...

```
@echo off
:menu
c:
cd\games
cls
echo    GAMES MENU
echo    ==========
echo.
echo    1 - Doom
echo    2 - Raptor
echo    3 - FlightSim
echo    4 - Lotus
echo.
echo    Q - Quit
choice /c:1234Q>nul
```

```
if errorlevel 5 goto done
if errorlevel 4 goto lotus
if errorlevel 3 goto flightsim
if errorlevel 2 goto raptor
if errorlevel 1 goto doom
echo CHOICE missing
goto done

:doom
cd doom
doom
goto menu

:raptor
cd raptor
rap
goto menu

:flightsim
d:
cd\fs4
fs4
goto menu

:lotus
cd lotusdrv
lotus
goto menu

:done
```

The idea is simple, set up initial conditions, in this case change to the C: drive and the \games directory, print a menu using ECHO commands (use echo. to print a blank line) then run the CHOICE command with the parameter /c:1234Q (the valid choices) and redirect it to >nul so it doesn't mess up the screen. After the user makes a decision, the errorlevel returned depends on which one was picked, selecting Raptor for example returns errorlevel 2. They are listed backwards because "if errorlevel 2 do something" really means "if the errorlevel is 2 or greater do something".

ANSI codes can be used to spruce up menus like these, even decent interfaces can be constructed in color using nothing but batch code. Several menu utility programs are also available that are called from batch.

Using menuing systems...

Several freeware, shareware and commercial menuing programs are available that let you create nice-looking menus for your system. They're not as fun as doing it entirely in batch but they can be more productive.

A more elaborate program is something called GO-MENU from an archive called DOSVAR20 from Pinnacle Software. The shareware program was something for manipulating strings (and was never even run) but GO-MENU.EXE is great. Here is its help screen when run with "/?"...

```
      GO-MENU v1.21A -- Copyright (C) 1991,92 Pinnacle Software
(Montreal)

       Purpose:   Displays a menu of up to 10 items
       Author:    Tim Campbell (Dial 514-345-9578; GEnie
T.CAMPBELL11)
       Format:    GO-MENU [menu-file-name] [nnn] [save-file-name]
       Parms:     Specify nnn to save & read last selection
                  nnn is a save index from 0 to 255
                  save-file-name is the file that holds up to 256
saves
                  Program can continue if create or read of save file
fails
       Defaults:  menu-file-name = GO-MENU.MNU
                  save-file-name = GO-MENU.SAV
       Menu File: Line 1    Menu heading
                  Line 2+   Menu lines
                  Note:     Lines starting with ; are ignored
       Returns:   1 - 10 depending on menu selection
                  200    if user pressed Esc in menu
                  250    for help request (this display)
                  255    for program error
```

Here is an example of how I use it, extracted from my master system batch...

```
@echo off

--- stuff ---

:menu
c:
cd \
go-menu user.mnu
if errorlevel 11 goto exit
if errorlevel 10 goto boot1
if errorlevel 9 goto pic
if errorlevel 8 goto tape
if errorlevel 7 goto scanners
if errorlevel 6 goto editfiles
if errorlevel 5 goto sysinfo
if errorlevel 4 goto house
if errorlevel 3 goto setevars
if errorlevel 2 goto prompt
if errorlevel 1 goto windows
goto menu

:windows
win
goto menu

:prompt
command /e:2048
goto menu

--- more menus etc ---
```

This is the text file USER.MNU to define the menu text...

```
My Computer System
   Run Windows
   Run DOS Prompt
   Set Environment
   HouseKeeping Utilities
   System Information
   Files / Environment
   Scanners
   Backup
   PIC Menu
   Reboot
```

I have no idea if GO-MENU is freeware, shareware or isn't even supposed to be used but it does work well. You can probably find it in the SimTel archives. Look for "DOSVAR20.ZIP". Many others are available, enough to cause mental overload actually. Some of them can create very elaborate screens once you master their convulted syntax (I haven't...).

Obtaining User Input

The CHOICE command is fine for asking limited-choice questions but it is no-good for getting real strings like a filename. There are two approaches that can be taken - use an external COM file like SENVAR.COM that directly sets a variable entered by the user. Or you can do it completely in batch. I've seen a couple of variations to this technique, one is contained in Dirk Van Deun's Collection of Batch Techniques file. The one I've been using originated from a computer magazine, I've modified it to put the temp files in C:\DOS to avoid accidentally overwriting 'ENTER.BAT'. You might need to change the directory, or just eliminate it altogether.

```
@echo off
:: based on batch from PC Magazine June 27, 1995 page 248
:: this version puts temps in C:\DOS dir and shortens var names
:: User input is returned in variable STR
:input
> C:\DOS\en#er.bat fc con nul /lb1 /n|date|find "    1:  "
> C:\DOS\enter.bat echo set str=
>>C:\DOS\enter.bat echo :loop
>>C:\DOS\enter.bat echo if not '%%str%%==' set str=%%str%% %%5
>>C:\DOS\enter.bat echo if '%%str%%==' set str=%%5
>>C:\DOS\enter.bat echo shift
>>C:\DOS\enter.bat echo if not '%%5==' goto loop
call en#er.bat
del C:\DOS\enter.bat
del C:\DOS\en#er.bat
```

Simply "call input.bat" (assuming that's what it's named) and the user string is returned in the %str% variable. If this routine is included in the batch program itself, set a return variable and call it like a subroutine as in...

```
echo Enter filename...
set return=here
```

```
goto input
:here
echo You entered %str%
set return=
goto done

:: above input routine
:input
:: ...
goto %return%

:done
```

A simpler way to call it is to use a universal branch at the top of the batch...

```
@echo off
if .%1==.Loop goto %2
```

then when you want input do a...

```
echo Enter filename...
call %0 Loop input
:: filename in %str%
```

When using batch input routines, do not enter redirection symbols or other stuff that messes up DOS, especially < > and |.

How to set up SHELL and ANSI

To use batch effectively you should check your CONFIG.SYS file for proper settings. To use color you need something like:

```
DEVICE=C:\DOS\ANSI.SYS
```

or if UMBs are available (have a DOS=HIGH,UMB line) use:

```
DEVICEHIGH=C:\DOS\ANSI.SYS
```

If this line is not present add it with the other DEVICE's, might help if it's first but that probably doesn't matter. When ANSI.SYS is active it interprets escape codes that set screen colors, move the cursor and all kinds of useful things. Type in HELP ANSI.SYS at a dos prompt for a detailed list of the available commands.

Among the commands is the ability to re-define any key to output an entire sequence of keys and commands upon typing the file. Files containing these sequences are known as Key Macros or ANSI Bombs, depending on the commands they contain. I use key macros to redefine my function keys to useful dos commands. I am *very* used to hitting control-x instead of typing E X I T Return. Just be aware of the potential problem and don't TYPE strange files. Use something like LIST. Other versions of

ANSI are available that do not allow key redefinition if this makes you nervous. ThunderByte's anti-virus driver will also prevent key redefinition after it's called, just define what you want defined before calling TBDRIVER.

To prevent out-of-environment errors when running batch files you should also have a SHELL statement in CONFIG.SYS to specify a larger-than-normal environment, something like:

```
shell=c:\command.com /e:1024 /p
```

Some computers have COMMAND.COM only in the DOS directory, if there is no COMMAND.COM in the root change 'c:\command.com' to 'c:\dos\command.com' or wherever it is. If a 'set comspec=c:\command.com' line is present it should match the path\filename given in the shell command.

Processing Lists of Data

A nagging problem in batch programming is how to take a list of items in a file and do anything useful with it. One solution is to use Ed Schwartz' @.COM program or a similar external utility, but there is a way to do it entirely with batch if certain precautions are taken.

The trick is to take the Dos file LOADFIX.COM and copy it to the filename ENTER.COM then use the DATE command to read the data file and output it as a batch. LOADFIX is a seldom-used command that simply loads and runs program above the first 64K of memory. It doesn't seem to have many uses as intended, but since it merely runs its parameters and it's a COM file it can be very useful for data processing from batches since it can simulate a call command without using 'CALL'. How is this useful? The DATE command outputs the string 'Enter new date (mm-dd-yy): ' then waits for input. If the input is not a valid date, it re-displays the prompt and gets more input until either a valid date or an empty line is entered. This has the effect of prepending 'Enter new date (mm-dd-yy): ' in front of each line of an input file. If LOADFIX were renamed to ENTER then each line will attempt to run NEW and whatever was in the original list shows up as parameter %3 and up. The obvious disadvantage of this technique is the input list *cannot* contain a valid date! Other than that it looks like a very promising technique.

The following demo batch creates a list of subdirectories then processes the list one item at a time...

```
:: demo - process subdirectories - by Terry Newton
:: overwrites ENTER.COM DIRFILE$ DIRFILE$.BAT NEW.BAT
@echo off
:: check and branch to subroutine
if .%1==.DoDir goto DoDir
:: prepare ENTER.COM file (change c:\dos\ if different)
copy c:\dos\loadfix.com enter.com > nul
:: prepare a list of fully qualified subdirectories
dir /s /ad /b > dirfile$
```

```
:: add a blank line to the end so DATE will exit
echo.>>dirfile$
:: run each line through date to make the list-batch
type dirfile$|date|find "Enter" > dirfile$.bat
:: get rid of the original list
del dirfile$
:: prepare NEW batch to call main program for
:: each item of the list
echo @echo off > new.bat
echo %0 DoDir %%3 >> new.bat
:: now call the list-processing batch
call dirfile$.bat
:: delete the temp files and get out
del dirfile$.bat
del enter.com
del new.bat
goto done
:: This subroutine is called by NEW.BAT for each item
:: in the list plus the blank line at the end
:DoDir
if .%2==. goto done
:: here for each subdirectory with name in %2
echo Looking at subdirectory %2...
::
:done
```

Be sure that you do not have a NEW.BAT or ENTER.COM program in the current directory, they will be overwritten. In the above example, the DIR /S switch ensures that no item will be a valid date, but if /S were omitted there would be a chance a directory name might just happen to also be a date and cause your system date to be set incorrectly, so be careful when using this technique.

Examples of this technique...

RESETARC.BAT and RESETALL.BAT are a couple of batch files I made for dealing with a tape backup program that doesn't reset the archive attributes on the files it backs up. Yuk! RESETARC resets the archive bits on files in the current directory and all subdirectories below it, RESETALL resets the archive bits on every file on every drive (edit for your system). Although that's an oddball kind of function they make simple templates to use for doing other oddball functions in every directory, just substitute your commands in place of the attrib command.

DIZZY.BAT builds a file containing the contents of all FILE_ID.DIZ files in and under a specified directory. I never knew I had so much stuff!

Windows 95 considerations...

The subroutine demos will work under Windows 95 provided a compatible loadfix.com exists and can be found, and no long directory names are involved. Windows 95 leaves out many old dos commands including loadfix.com, if you haven't already, look for them at Microsoft's site or on the install CD. The main 95 difference for these batches is the /b switch of the DIR command outputs long names, an advantage or disadvantage depending on the application. Most dos commands can handle long filenames (provided they're enclosed by quotes) so if copying or other

simple tasks it's no problem. If running an old dos app that does not understand long filenames then about all that can be done is copy the target file to another file first then call the dos app on the shorter name. I have no trick long-to-short name routines but I'm sure it can be done.

The demo code conversions are not too bad, at least for the first six spaces. The main difference is as many parms possible are passed in the code that calls the subroutine, and the subroutine reconstructs the parms into a variable containing the directory name.

```
:: demo - process subdirectories - by Terry Newton
:: overwrites ENTER.COM DIRFILE$ DIRFILE$.BAT NEW.BAT
:: windows 95 compatible
@echo off
:: check and branch to subroutine
if .%1==.DoDir goto DoDir
:: prepare ENTER.COM file (looks in 2 places, hardcode if different)
if exist c:\dos\loadfix.com copy c:\dos\loadfix.com enter.com>nul
if exist c:\windows\command\loadfix.com copy
c:\windows\command\loadfix.com enter.com>nul
:: prepare a list of fully qualified subdirectories
dir /s /ad /b > dirfile$
:: add a blank line to the end so DATE will exit
echo.>>dirfile$
:: run each line through date to make the list-batch
type dirfile$|date|find "Enter" > dirfile$.bat
:: get rid of the original list
del dirfile$
:: prepare NEW batch to call main program for
:: each item of the list
echo @echo off > new.bat
echo %0 DoDir %%3 %%4 %%5 %%6 %%7 %%8 %%9>> new.bat
:: now call the list-processing batch
call dirfile$.bat
:: delete the temp files and get out
del dirfile$.bat
del enter.com
del new.bat
goto done
:: This subroutine is called by NEW.BAT for each item
:: in the list plus the blank line at the end
:DoDir
if .%2==. goto done
:: here for each subdirectory
set dirname=
:DoDirparms
set dirname=%dirname%%2
if .%3==. goto DoDirgotit
set dirname=%dirname% % nullmarker%
shift
goto DoDirparms
:DoDirgotit
echo Looking at subdirectory %dirname%...
::
:done
```

List processing without loadfix...

Having to locate a possibly non-existent loadfix.com is a hassle to say the least, especially these days. If the list isn't very large and processing time isn't critical, there is another way to work through a list using only common dos commands. The technique is to create a list ending with a blank line, run through date to create a temp batch, so far just like the previous method but instead of creating enter.com, create an enter.bat that sets the variable. When the temp batch runs, only the first enter line will execute, so after it returns use the find /v command to remove the last item processed, then loop until the empty line is encountered. Each item takes a bit longer to process because of the removal step, but in applications where an appreciable time is spent on each item, this doesn't matter much, better to have the simplicity.

A demo that collects and appends all .DIZ files in and below a directory into a single file...

```
:: find all .DIZ files in and below current directory
:: and append them into a single DIZFILES.TXT file.
:: coded by Terry Newton
@echo off
:: branch if need be
if .%1==.DoFile goto dofile
:: empty the output file
rem > dizfiles.txt
:: create a list of files (/-p in case pause in dircmd)
dir /s /b /-p *.DIZ > lstfile$
:: end with blank line
echo.>> lstfile$
:: run through date to make tempbat
type lstfile$ | date | find "Enter" > lstfile$.bat
:: create an enter.bat to be run by listfile$.bat
echo %0 DoFile %%4 %%5 %%6 %%7 %%8 %%9 > enter.bat
:loop
:: run temp batch
lstfile$.bat
:: ends up here with filename at parm2 and up
:dofile
:: check for exit condition
if .%2==. goto done
:: derive filename from parms
:: (only needed for long names, otherwise refer to %2)
set fname=
:doparms
set fname=%fname%%2
if .%3==. goto gotparms
set fname=%fname% % nullmarker%
shift
goto doparms
:gotparms
:: give operater something to look at
echo Working on %fname%...
:: add filename and file to output file
>> dizfiles.txt echo -------- %fname% --------
>> dizfiles.txt type "%fname%"
>> dizfiles.txt echo.
>> dizfiles.txt echo.
:: done with that file, so remove entry from lstfile$.bat
type lstfile$.bat | find /v "%fname%" > lstfile$.bat
:: and loop
goto loop
```

```
:done
:: finished, remove temp files
del enter.bat
del lstfile$.bat
del lstfile$
```

This demo incorporates Windows-95 compatibility fixes, if your application does not have filenames with spaces in them several lines of code can be saved, so if using Windows 3.1 or Dos, make the "dofile" part like this instead...

```
:dofile
:: check for exit condition
if .%2==. goto done
:: give operater something to look at
echo Working on %2...
:: add filename and file to output file
>> dizfiles.txt echo -------- %2 --------
>> dizfiles.txt type %2
>> dizfiles.txt echo.
>> dizfiles.txt echo.
:: done with that file, so remove entry from lstfile$.bat
type lstfile$.bat | find /v "%2" > lstfile$.bat
:: and loop
goto loop
```

Not totally sure (never tried it..) but the quotes around possibly long filenames will probably confuse older doses, something else to watch out for if making batch code that must run on multiple dos versions. One thing for sure, without the quotes it fails under 95 with a "too many parameters" error the first time a space is encountered in a filename.

In all of these demos, lists of filenames are used but the list does not have to be that, it can be a list of anything so long as only the last line is empty and no redirection characters (< > |) or separators (; , =) involved. Redirection not enclosed by quotes create errors and unpredictibly-named files, separators are converted to spaces. However processing lists of filenames does seem to be the obvious use.

Creating and running other programs from batch

Often times batch simply is not powerful enough or too slow to do the things that need to be done, this is when other program interpreters like QBasic and even debug can come in very handy. Using debug and assembly code in general is beyond the scope of this text and me also, I learn a few commands and work with them. QBasic isn't hard to learn though, and you do not need to learn everything to be able to use it from batch files. Reading and writing files, doing some mathematical computations (but read on), fancy screen displays and interfaces and many other cool tricks can be accomplished using batch code that writes and runs temporary basic programs. If you look around my batch site you'll find many examples.

User input? The all-batch solution is ok but here's another way...

```
@echo off
echo>$inp$.bas bad$="<|>=,;":on error goto done
echo>>$inp$.bas ? "Enter something: ";
echo>>$inp$.bas line input a$:if a$="" goto wrbat
echo>>$inp$.bas for i=1 to len(bad$)
echo>>$inp$.bas if not instr(a$,mid$(bad$,i,1))=0 then a$="(error)"
echo>>$inp$.bas next i
echo>>$inp$.bas wrbat:open "$inp$.bat" for output as #1
echo>>$inp$.bas ? #1,"set input=";a$
echo>>$inp$.bas done:close #1:system
qbasic /run $inp$.bas
call $inp$.bat
del $inp$.ba?
echo You entered %input%
```

How it works... the batch code writes out a temporary basic program and runs it, the basic code prints a prompt, collects a line of user input, checks it for characters that can cause errors and replaces the input line with "(error)" if present, then writes a temporary batch that sets a variable to the input line. After the basic completes, the batch calls the temp batch to set the input variable, deletes the temp files and displays the input variable.

Ok a couple of further explorations... first off if you haven't figured it out, dos doesn't care much where redirection occurs in a command, it makes things look neater if the redirection is immediately after the echo so the remainder resembles what actually gets written, handy anytime creating files with batch but leave "echo." as "echo.>>file". Writing out strings that contain redirection symbols is not a problem, the quotes hide them from dos. The main trick is avoiding errors from < and > characters in math expressions. Fortunately just about any math equation using such characters can be rewritten with `not` and `sgn` to avoid the problem. For example, `if a > b then...` won't work, instead use something like `if sgn(a-b)=1 then...` In the input example it wasn't `if instr(...)<>0` but rather `if not instr(...)=0`.

Some more basic to batch conversions...

```
if a < 5 then...          if sgn(5 - a) = 1 then...
if a <> 5 then...         if not a = 5 then...
if a <= b then...         if not sgn(a - b) = 1 then...
```

When writing any file from batch (this applies to batch, qbasic or anything) the cardinal rules are %% becomes %, variables are substituted with their contents, and the characters < > and | must be enclosed in quotes or not used at all.

Here is another useful newly hacked-out example of using qbasic from batch...

```
:: CHSTRING.BAT - changes all occurences of one string to
:: another, if newname is not specified overwrites filename
:: under win 95 strings can contain spaces and separators
:: requires at least dos 6 and qbasic.exe
@echo off
:: verify that it has enough parms
if .%3==. echo CHSTRING "matchstring" "newstring" filename [newname]
if .%3==. goto end
:: verify that file is present
if exist %3 goto fileok
```

```
echo file %3 not found
goto end
:fileok
:: verify quotes
echo %1|find """">nul
if errorlevel 1 echo no quotes around string(s)
if errorlevel 1 goto end
echo %2|find """">nul
if errorlevel 1 echo no quotes around string(s)
if errorlevel 1 goto end
:: create a custom qbasic program
echo>chstr$$.bas :on error goto done
echo>>chstr$$.bas ls1=len(%1):open "%3" for input as #1
echo>>chstr$$.bas open "newfl$$$" for output as #2
echo>>chstr$$.bas doit:line input #1,a$:if a$="" goto wline
echo>>chstr$$.bas chline:la=len(a$):if sgn(ls1-la)=1 goto wline
echo>>chstr$$.bas b$="":c$="":p=instr(a$,%1):if p=0 goto wline
echo>>chstr$$.bas if not p=1 then b$=left$(a$,p-1)
echo>>chstr$$.bas if not p+ls1=la then c$=right$(a$,la-p-ls1+1)
echo>>chstr$$.bas a$=b$+%2+c$:goto chline
echo>>chstr$$.bas wline:print #2,a$:goto doit
echo>>chstr$$.bas done:close #1:close #2:system
:: run and delete it
qbasic /run chstr$$.bas
del chstr$$.bas
:: move output to appropriate filename
if not .%4==. move newfl$$$ %4>nul
if .%4==. move newfl$$$ %3>nul
:end
```

How it works... (I hate this part:) The initial batch lines verify the parameters for correctness and if not display various messages. Once it's satisfied with the parms it creates a temporary qbasic program that changes all occurences of "match string" to "new string", writing the results to a temp file. After running and deleting the qbasic program, it checks to see if an output filename was specified, if so it copies the temp output file to that otherwise overwrites the specified file with the changes.

The strings must be enclosed in quotations, under Windows 95 they can contain spaces and separators like commas, under Dos 6 the strings must be all one word without any strange characters, sorry... another Win95 difference, but I like this one. A related feature is specifying long filenames using quotes, many lfn-unaware batches still work if the filenames are quoted to keep the parameters intact.

This is a batch guide, not a qbasic guide so I won't explain the basic part much, qbasic has an extensive on-line help facility that explains what every command does, rather I'll point out conversions needed to make it into a batch. In line 1 of the qbasic code, the ":" before "on error" is there to keep Windows 95 from interpreting the statement as "echo on" and writing "echo is on" to the file instead of the errortrap. Dos 6 doesn't have this "feature" but the fix is easy enough. In line 5, the natural form of the comparison would be "if ls1 < la" but that would create an error situation, so the equivalent comparison "if sgn(ls1-la)=1" is used instead. Line 7.. "if not p=1" instead of "if p > 1" (not equivalent but p is never <0 so it works). Line 8.. "if not p+ls1=la" instead of "if p+ls1 < la" (again not equivalent but...).

Intermixing Perl and Batch...

The following was tested using the Perl for Win32 interpreter, it should work with other Perl 5 complient interpreters if there are any others. To be useful, one usually has to make a batch file for each perl script to run, but the language definition allows for extra code before and after the script code. This makes it very easy to make a perl-to-batch "compiler" that encapsulates the script and allows it to run like a stand-alone program...

```
:: 'compiles' perl scripts into batch files
:: by Terry Newton, Feb 98
@echo off
if .%2==. echo perl2bat perlfile batfile
if .%2==. goto end
if exist %1 goto compile
echo can't find input file %1
goto end
:compile
echo compiling %1 to %2...
echo>%2 @echo off
echo>>%2 set $bat=%%0.bat
echo>>%2 if not exist %%$bat%% set $bat=%%0
echo>>%2 perl -x %%$bat%% %%1 %%2 %%3 %%4 %%5 %%6 %%7 %%8 %%9
echo>>%2 set $bat=
echo>>%2 goto p2b_end
type %1>>%2
echo>>%2 __END__
echo>>%2 :p2b_end
:end
```

Up to nine command-line parameters are passed to the script, you can modify the output to include other batch processing that may be needed before and after running the script. Since the batch execution thread never touches the script code there are no limitations on what it contains, so long as it doesn't have a line that begins with ":p2b_end" and I don't think that'll be a problem (if it is edit the two occurences to something else).

How it works... the batch must write out other batch instructions that determine the filename (varies if ran from a command line or from windows) then call the perl interpreter (modify if not "on the path") to run itself using a switch that tells the interpreter to ignore any junk at the beginning. No redirection is needed, so none of those hassles, the only conversion is all the "%" characters in the output are doubled. When written out the loader looks like...

```
@echo off
set $bat=%0.bat
if not exist %$bat% set $bat=%0
perl -x %$bat% %1 %2 %3 %4 %5 %6 %7 %8 %9
set $bat=
goto p2b_end
[simple perl script]
#!perl
print "Hello World!\n";
__END__
:p2b_end
```

Neat trick. However it doesn't run from a path directory, for that more batch processing would be necessary. If it becomes too difficult then it's easier just to run with a separate batch...

```
@echo off
perl c:\whatever\perlscr.pl %1 %2 %3 %4 %5 %6 %7 %8 %9
```

... and be done with it, even if it is yet another file.

Making decisions based on the output of a program

Often it is necessary to test the output of a program that does not return an errorlevel. The trick is to use the FIND command to search for a specific string in the program's output.

The general method for Dos 6 and above is...

```
program | find "string" > nul
if errorlevel 1 goto notfound
rem string was found
goto endfind
:notfound
rem string was not found
:endfind
```

Note - if the letter case of "string" is not known, use find /i "string" instead of find "string".

For checking for more than one string, send the program's output to a temporary file and test that...

```
program > tempfile
find "string1" < tempfile > nul
if not errorlevel 1 goto 1found
find "string2" < tempfile > nul
if not errorlevel 1 goto 2found
rem no strings found
goto endfind
:1found
rem string1 found
goto endfind
:2found
rem string2 found
:endfind
del tempfile
```

Dos 5 makes it more complicated, because FIND didn't return an errorlevel until 6. If you're not sure of the target OS a universal approach takes advantage of COPY's refusal to copy empty files...

```
program | find "string" > temp1
copy temp1 temp2 > nul
del temp1
if not exist temp2 goto notfound
```

```
del temp2
rem string found
goto endfind
:notfound
rem string not found
:endfind
```

Any command or utility that writes to standard out can be used with these methods. A typical example is determining if two files are identical (assumes English, dos 6 or better)...

```
fc file1 file2 | find "FC: no differences" > nul
if errorlevel 1 goto notequal
rem files are identical
goto donefc
:notequal
rem files are different
:donefc
```

Mohamed Baobeid - 2003