BATCH FILE PROGRAMMING

Michael E. Valdez, Ph.D.

Table of Contents

BATCH FILE PROGRAMMING

Chapter 1.- Introduction

IBM-Type computers have a peculiar structure of the commands. In the original way, the keyboard enters commands. Each command uses one line. The computer executes when you press the Enter key. The command line procedure is complex and limited. Several alternatives have evolved. These alternatives take several names, like shell, menu, windows, etc. The programmers try to overcome the deficiencies of DOS. Microsoft DOS 5.0 comes with a Shell. This shows that even the originators of DOS recognize the limitations of the system they have developed. Microsoft also produces the Windows program, with the same purpose.

The shells, menus, or windows, make the procedure to issue commands to the computer more natural. On the other hand, the use of any of these shells, menus, or windows is complex. We replace one complexity with another. To understand this fact you need only to consider that the same programmers developed DOS and the shells or windows. The shell, menu, or window, normally analyzes your disk to find approved applications and loads their names and paths onto the shell. Even experience programmers have trouble forcing Windows or any other shell, to accept certain applications. A user without too much experience is at the mercy of the shell, menu, or window programmer. After installing one of these interfaces, he can only do what the system programmer permits him to do. This is not always what the user wants to do. The user ends up leaving the shell, menu, or window to work bare footed in the infamous command line. This is true for all IBM-type computer users, regardless of their experience. Sooner or later they need to leave the fancy shell to perform a simple function. This normally happens sooner, rather than later. This has nothing to do with the type of work you do with your computer. The same happens whether you play games, write letters, develop very complex mathematical simulations, or work with very complex graphics. Sooner that later, you will need to abandon your fancy shell for the unfriendly environment of the command line.

Every user of an IBM-type computer should learn to use the command line. The sooner it is done, the better. One easy way to learn to use the command line uses the computer itself. Get the manual that came with your version of DOS and go through it. Exercise each command, with all the switches, until you understand its operation. This proposition looks dull. As you perform these experiments, you will learn a lot. You will find that these experiments are much more interesting than what they look from here. To get as much profit as possible from these exercises, you must understand that MS DOS is a very limited system. The command line is even more limited. You do not have any chance to edit the command line. While you are typing a command, the only chance is to erase the previous letter, or the whole line. The command executes when you press the Enter key.

After the fact, you have a few more chances for editing the command line. This requires the use of the Function keys. F1 permits to copy one character from the old command line. F2 permits to copy up to the first occurrence of a given letter. F3 permits to copy the rest of the old command line. F4 permits to skip up to a given letter. F5 permits to see the old command line. DEL permits to skip one letter. INS permits to insert letters. ESC cancels all you have done. It takes a few sessions trying to work from the command line to frustrate anybody.

Most people go the route of the shells, menus, or windows. We have already seen that this is only a reprieve. Loading one of these interfaces takes a lot of your freedom. This thought should be your motivation for learning how to use your computer.

There is another path out of the unfriendly command line. This path does not reduce your freedom. You can do what you want, within the limitations of DOS. This path is Batch Processing. DOS includes a very limited batch file processor. Batch processing can enhance and simplify operating your computer. You need to learn how to use the batch processor. You need to learn its limitations.

This book tries to help you learn to use batch processing. Each chapter covers specific topics and includes many exercises you can perform. The disk includes all necessary batch files. They use the same name used in the text. If you have a hard disk, copy the whole disk to a directory. The operation will then be much faster.

I recommend that you first go through the book in sequential order, to have a good idea of what you can expect. Skip the examples and exercises. After this, take a break of a couple of days. Return to the book and go chapter by chapter. Try to understand the subjects. Examine carefully the examples. Do the exercises. When you finish you will be an expert. Later, use this book as an easy reference for your work. There is an index of commands at the end of the text. If you have any comments or suggestions, please contact me. Sorry, I cannot afford to accept collect calls or return long distance calls. Melbourne, April 1992 Michael E. Valdez P. O. Box 2382 Melbourne, FL 32902 (407) 984 0665

Chapter 2.- The DOS Command Structure

The DOS command structure is simple. Using the commands is what is complex. This simplicity of the command structure is a result of the many limitations of the system. The DOS command uses a single line. It executes when you press the Enter key.

A command has three parts: a command name, the parameters, and the switches. Some people like to call the parameters and the switches by the generic name of command tail. Some commands call a program. The tail passes to the program. The command name tells DOS what you want the computer to do. It is interesting you read this sentence again. You do not tell the computer what to do. You tell DOS to tell the computer what you want it to do. The difference is small, but important. Programming the computer, you tell it what to do. You talk directly to the computer. When you write a command line, you talk to DOS. DOS talks to the computer.

You must understand this point. Imagine you are in an office environment. You want the Secretarial Pool to perform some task. There are two chances: You talk directly with the person assigned to the job. You talk with the Pool Head. In the first case, your instructions go directly to the person performing the function. In the second case, the opinions and ideas of the Pool Head filter your instructions. This is what happens with DOS. The execution of a command is conditional to the opinions and ideas of the programmers who wrote DOS. DOS commands are not the functions the computer can perform. They are only those functions DOS programmers consider you want to perform.

Let us elaborate a little more. Any computer can perform addition of two numbers. DOS programmers consider you do not want to perform addition of two numbers. You cannot perform addition of two numbers using the command line. This expresses the basic philosophy of the command line.

The situation is somewhat similar to when you use a menu system. You can only run those programs included in the menu. In most menu systems you have the chance of adding more programs. You also can get out of the menu. When using the command line, you do not have any alternative. You could write your own system. That is another topic.

In summary, DOS gives you some commands. They are all you can use. The manual that comes with DOS has a list of all the commands and their meaning. You must type the commands exactly as they appear. You can use upper or lower case, or mixed. Do not include spaces between the letters and end the command with one space.

For example, to get a list of the files in a given directory you type dir, Dir, dIr, DiR, DIR, diR, dIR, or DiR. They are all equivalent. You cannot type d ir, or any other form like it. DOS will not understand it. Consider that DOS converts your command to upper case letters and then interprets it. All of the above forms convert into DIR. The one with the space is not.

DOS might not recognize the first word in the command line. It considers it the name of a program. In this case, DOS searches your system for the place where the program might be. DOS searches first the current directory, then the root directory of the current disk, then all the places included in the PATH environmental variable. If the name does not have an extension, DOS searches first for a file with extension COM. Then searches for an extension EXE. Finally, it searches for an extension BAT.

As you see, this search can take a while in a large system. You can speed up this search. Enter the name of the program with the full path and its extension. This also avoids that DOS might find another program or batch file with the same name. For example, if you want to run a batch file called X.BAT. Imagine the path to this file is in the PATH of the system. You will gain time by calling it C:\DO\DOS\BATCH\X.BAT, instead of simply X. This is very important working with floppy disks.

The command tail has two parts, the parameters and the switches. The parameters are values or information passed to the command. With the command DIR, we can include which directory in which disk we wish to see. This is a parameter.

The second part of the tail is the switches. This part permits you to select variations of the command. Using the command DIR, you have several switches. You include the switch /w to ask for the wide printout. You include the switch /p to ask for a pause after each page. The same with the other switches.

There is one very important point to remember. Use switches only for the particular command where they appear. The switch /p means something for the command DIR and another for the command REPLACE. In general, but not always, switches and parameters can be in any order. You should remember that DOS does not have general rules. What works with one command might not work with another. A combination that works in one case, might not work in another.

This is particularly true with the other element of the command line that seldom appears as such in DOS manuals. There is another element in the command line. This is the redirection of the input to or the output from the command. You have three symbols to call for redirection:

The < shows that the command should take input from wherever follows. For example, SORT < MYFILE.TXT asks DOS to sort the input. The symbol < shows that the input must come from the file MYFILE.TXT.

The symbol > calls for redirection of the output. For example, TYPE MYFILE.TXT sends the output to the display. If your command reads TYPE MYFILE.TXT > PRN, the output of the command TYPE goes to the printer. The symbol > calls for redirecting the output to a file or a device. When used with a file, it creates the file. It destroys any other file with the same name, if it exists.

The symbol >> tests first if a file with the same name exists. If it exists, the output of the current command appends to the end of the file.

The third redirection symbol is the pipe. The pipe converts or pipes the output of one command as input of the next. The command TYPE MYFILE.TXT produces a continuous flow of output to the screen until the end of the file. On the other hand, the command TYPE MYFILE.TXT | MORE redirects the output of

TYPE as input to the filter MORE. The filter MORE sends its output to the screen in pages.

The difference between <, > and | is that the pipe | goes between commands and filters or between filters. The redirection symbols < and > go between commands and files or devices. They cannot replace the pipe |. The use of the pipe between commands produce variable results. Filters are functions like MORE, SORT, FIND. They receive an input. They perform a function. They produce an output.

It is interesting to give more time to redirection of input and output since it is such an important topic. A command normally receives input from the keyboard. Some receive no input. If you want to redirect the input to a command you need to consider where the input comes from. Use the < symbol If the input comes from a file or device. Use the pipe | if the input comes from another command or filter.

To redirect the output of a command you need to consider where the output will go. Use the symbol > if it goes to a file or another device. Use the pipe | if it goes to another command or filter.

To redirect the input of a filter you need the same considerations. Use the symbol < if the input comes from a device or file. Use the pipe | if the input comes from a command or filter. The same considerations apply when redirecting the output of a filter. Use the symbol > to redirect the output to a file or device. Use the | to redirect the output as input of a command or filter.

When making this redirection you should be careful with the organization of the names. Study the following: C > B The output of C goes to file B; C | B The output of C pipes to filter B; C < B C takes its input from file or device B. This becomes important when you want to use more that one redirection. Consider the following: SORT < C > B The filter SORT takes its input from file or device C. It sends its output to file or device B. Note that B does not take its input from C as it looks at first sight.

DOS recognizes standard input and standard output. After booting your computer, the standard input is the keyboard and the standard output is the screen. DOS programmers still call them the console, as in main frame computers. There is one important case of redirection. Every command produces another output. This is the error messages. By default, the error messages go to the screen. There is no way to redirect this output. The > and the pipe redirect the output intended for the standard output device. They do not affect the output that goes to the error output device. The standard output is device number 1. The error output device is device number 2. Again, DOS programmers consider you do not want to redirect the error output. They do not give you a command to perform this redirection. Of course, you can write a program to do it, but this is another subject.

Chapter 3.- The Batch Processor

DOS programmers recognize the limitations of the system they created. They include a Shell with the latest versions of DOS. In the same way, most versions of DOS include a Batch Processor. It was the first try to correct the limitations. Too bad the programmers did not think of the obvious way of doing it. In the lingo of the IBM-type computers, a batch file is a text file containing some DOS commands, one per line. They execute in sequence. A batch file must have an extension BAT.

There is a big difference. You prepare your batch file off line. The commands do not execute when you press the Enter key. You can read them over and over until you are sure they are exactly the way you want. After you run the batch file, the file is still there. You can run it as many times as you want, without additional work. You can correct errors. You can enhance or change its operation. In fact, your file becomes another command the computer can execute. You can create commands. A batch file can contain any command written at the DOS prompt. You can consider that the idea is to redirect the input to DOS from the batch file. DOS reads the batch file line by line. It executes them as it finds the carriage return-line feed. This is a very simplified idea, but a good starting point.

Before going further, you must remember there are three types of DOS commands. Commands used only in the CONFIG.SYS file that start the system. Commands used only inside a batch file. Commands used everywhere. There are commands used only inside a batch file. This makes some difference in the definition given above. A batch file is more that a redirection of input.

We should mention the way a batch file uses the parameters. Imagine we call a batch file with several parameters, like Y.BAT MARY LUCY SARAH From inside the batch file we can refer to the first parameter by writing %1. The second parameter is %2, and so on. This is one of the powerful characteristics of the batch file. You can develop a file to perform a function in general, and by changing the parameters you can make it apply to any condition. This fact again shows that a batch file is not only the redirection of the input to DOS. It is something more powerful than that.

As an example, let us assume we need to create a batch file to unzip any file and put the resulting file in any place. Let us say we want to call the file UZ.BAT. We write the following batch file:

```
@ECHO OFF
CD \
MD %2
CD %2
C:\ARCHIVE\PKUNZIP %1.ZIP
CD \
```

This batch file executes when typing

```
UZ A:\MYFILE C:\DO\WHERE.
```

Let us analyze how this batch file works. First it creates a directory with the name of the second parameter, if it does not exist. Then, it changes to that directory. The file assumes that the program PKUNZIP.EXE is in the directory ARCHIVE of the C drive. Then, it runs PKUNZIP with the first name as parameter. PKUNZIP unzips the file with that name onto the active directory.

A batch file can contain any of the general commands from DOS. It also can include the special commands for batch files. These commands include CALL, ECHO, FOR, GOTO, IF, PAUSE, REM, SHIFT. FOR also works in the command line. A brief description of these commands follows.
CALL is a way to execute one batch file from inside another. There are two ways of doing this. You can simple put a line with the name of the batch file and any tail, if needed. Control passes from the batch file executing to the other batch file. Say we are executing a file X.BAT and one of the lines of this file reads Y.BAT. Control passes from X.bat to Y.BAT at that point. Control never returns to X.BAT. If you insert a line that reads CALL Y.BAT, control passes to Y.BAT at that point. When Y.BAT finishes its work, control returns to X.BAT and continues execution.

ECHO has three meanings. ECHO in a line by itself, shows if the echo function is ON or OFF. Typing ECHO ON turns the function on. Typing ECHO OFF turns the echo function off. An interesting variation in the most recent versions of DOS is to type @ECHO. This means that the line where this command appears will not echo.

Produce a batch file with only one line that reads TIME. You can use the file X1.BAT from the disk. The simplest way to produce a batch file is to type at the DOS prompt

COPY CON X1.BAT.

When you press Enter the cursor disappears and the computer waits for your input. Type TIME and Enter. To end the batch file press the F6 key and the Enter key. To run the batch file you simply type X1 and Enter. You will see first a line that says C:>TIME. This is the echo of the line in the batch file. Then you see the execution of the command TIME.

Now, type another batch file. You can use the same name X1.BAT because DOS destroys the existing one. Use another name if you wish to save them. You can use X2.BAT, included in the disk. Type ECHO OFF as the first line of the batch file and TIME as the second line. Run the batch file X2 and see carefully the output on the screen. You will see a line that says C:>ECHO OFF, the echo of the first command line of the batch file. Then you will see the TIME command executing but you will not see the line C:>TIME.

If you have DOS 3.3 or later, type another batch file, or use X3.BAT from the disk. This time the first line you type will be @ECHO OFF and the second line TIME. When you run this batch file you will not see the echo of the first or the second line. You will only see the TIME command executing. This is good to avoid cluttering your screen with irrelevant information.

ECHO also displays information on the screen. This is useful to give the user instructions on what to do at each point. ECHO will show on the screen the text that is in the rest of the line. Do not use the characters <, >, or | because DOS interprets as redirection.

If you want to put a blank line, use ECHO. without any space between the ECHO and the period.

The output of ECHO normally goes to the screen, the standard output device. This output can redirect as input to a command or filter. The output also can redirect to a file, using > or >>. This permits to keep a log, for example. Once you redirect the output of ECHO, all successive input to the command comes from the ECHO. The problem appears when the command requires more that one input. ECHO can have only one line. Let us look at an example. Consider the line ECHO. > FORMAT B: in a batch file. This line asks DOS to format the disk is B. As soon as this line executes, DOS issues the normal message: Insert a disk in drive B and press any key to continue. Normally DOS waits for input from the keyboard. Now the input comes from the batch file. The ECHO produces the key press. When FORMAT finishes its task, it asks for the label you want. The ECHO does not have more output to give. DOS hangs forever waiting for input. This shows you should be careful when redirecting the input or output of a command.

FOR is a very powerful command that permits to perform loops inside a batch file. The format for this command is as follows:

```
FOR %%A IN (LIST OF WORDS WITH ANY MEANING) DO COMMAND %%A
```

that produces the effect that the command executes in turn with each of the words as a parameter. %%A represents a variable and any letter works. This is case sensitive. If you say %%B in one place and %%b in another, DOS considers them different variables.

Let us mention that when you use this command from the command line you use %a instead or %%a. One percent sign precedes the letter used for the variable in the command line. You use two in a batch file.

There are other forms we explain with examples. The one included in every book is

```
FOR %%A IN (MYFILE.TXT YOURFILE.TXT) DO TYPE %%A
```

that orders DOS to type the file MYFILE.TXT and then, to type YOURFILE.TXT. You could use the pipe to MORE to print by pages.

You also could use the form,

```
FOR %%B IN (TYPE COPY DEL) DO %%B C:\MYFILE.TXT
```

that types the file MYFILE.TXT on the screen. Then it copies it to the default directory. Finally, it removes it from where it is now.

We could also say something like this

```
FOR %%X IN (WORK DATA FILES) DO MD C:\DO\%%X
```

that creates the subdirectories WORK, DATA and FILES inside the directory C:\DO.

Here is another example of embedding the variable inside some information.

```
FOR %%G IN (LAURA MARIE LUCY) DO COPY C:\PICT\%%G.PCX B:\
```

that copies the files C:\PICT\LAURA.PCX, C:\PICT\MARIE.PCX and C:\PICT\LUCY.PCX to a disk in drive B:.

As a final example consider the following

```
FOR %%D IN (W P A O) DO DIR /%%D
```

This form shows the directory of the active directory four times, each time with a different option.

In summary, the FOR command permits to perform loops. The looping variable can be a command, a parameters, part of a parameter, or a switch. Naturally, you can run several programs, as long as none of them takes parameters, or all of them take the same parameter.

GOTO simply transfers control from the line the command appears to some other place marked with a label. A label is any word at the beginning of a line, preceded by a colon.

IF is another very interesting command that permits to perform a function if a condition is true, The IF command accepts only three conditions: errorlevel, ==, and exist. Each of these functions inverts by using NOT.

When a program executes and ends, it produces an error level. The batch file can interpret this error level. The command line cannot test this value. The meaning of the errorlevel function changes from case to case. Usually, an error level of zero means that the program ended successfully. An errorlevel of one or above shows some kind of error. The use of this condition can be illustrated in the following example:

```
MASM %1 IF ERRORLEVEL 1 GOTO END LINK %1,%1, :END
```

In this case, the program MASM is run with the first parameter. MASM is the Microsoft Assembler. The idea is to assemble the file whose name is in the first parameter. If no error exists, the linker executes. If the assembler finds an error, the linker does not execute.

The use of IF with the == or NOT == condition is illustrated by the following example:

```
        IF %1X == X GOTO ERROR
```

```
IF NOT %1 == COPY DEL %2 GOTO END
COPY %2 B:\*.* GOTO END
:ERROR ECHO ERROR
:END
```

This batch is used to copy or delete a file ECHO xx.bat, copy or del, name of the file to copy or delete.

Let us analyze this batch file. If you run it without any parameter, it gives you an error. The condition %1X == X is only true if %1 is empty.
If you run this file with any first parameter other COPY, the batch deletes any file that matches the second parameter.
If the first parameter is COPY, it copies the file to drive B.
This batch file has one error to show you something important. This file will work correctly if everything if perfect. It is very unlikely you want to take the chance of deleting a file instead of copying if you misspell the word COPY. A better approach will be

```
IF %1 == COPY COPY %2 B:\*.*
IF %1 == DEL DEL %2
IF %1A == A ECHO ERROR
```

In this case the batch file does not do anything if you misspell COPY or DEL, or you include another word.

The final condition accepted by the IF command is EXIST, or NOT EXIST. This condition checks for files. Consider the batch file mentioned before that assembles any program. We could call this batch file with the name of a program that does not exist. The assembler will get confused, and fill the screen with error messages. A very simple change is to add the following line as the first line of the file

IF NOT EXIST %1.ASM GOTO END.

Now, if the file does not exists, the file exits peacefully. You can put a message if you want.

You also can use this command to change the COPY command of DOS. The copy command destroys any file with the same name if it exists in the destination. The following batch file will not copy a file if it exists in the destination.

```
IF EXIST %2 GOTO EXISTS
COPY %1\%2
GOTO END
:EXISTS
ECHO File %2 exists in default directory
PAUSE
```

```
    :END
```

This batch file works only when you try to copy files into the default or active directory. %1 is the path to the file to copy. %2 is the name of the file. PAUSE holds the screen until you press any key.

DOS puts a message on the screen Press any key to continue. Some times the reason for the pause is obvious. it is better to use ECHO before the PAUSE to tell the user what to do.

```
    ECHO Change disk in drive a: and
    PAUSE
```

produces an output to the screen as follows

```
    Change disk in drive a: and
    Press any key to continue....
```

REM is a command that makes DOS skip the line. You could add comments to a batch file. This is useful to explain what the batch file is all about. Although DOS do not execute the line, it reads it. This means that DOS will give an error if it finds any of the redirection symbols.

SHIFT shifts the parameters to the right. Parameter one disappears, parameter two becomes one, and so on. This is useful in batch files that need more than nine parameters.

Chapter 4.- Special Forms of Batch Files

During the years, batch files have been used to a limited extend. Most of the use of batch files have been to launch applications. Many programs include batch files that change the current or active directory and launch the program. A typical program of this kind starts by changing drives to the drive where the program has been installed, then it changes the directory to the directory where the program was installed, ending by calling the program. A well designed batch file of this type should return the active directory to where it was before. The batch file will be as follows:

```
    @ECHO OFF
    C: CD \
    CD \UTIL\DRAW
    DRAW
```

This batch file assumes we want to launch an application DRAW which is in the subdirectory DRAW of the directory UTIL in drive C:. This batch file is only required if the program DRAW needs to use other files that are stored in the same subdirectory. The basic idea is to create one batch file for each of the programs we have in our system. Most of the programs we get from others install in their own directory in drive C:. A batch file to call the program, modeled on the one above, is installed in the root of drive

C:. The root of drive C: is the boot drive for most systems. The root of drive C: is automatically in the path of the system.

Let us look at a more complex batch file. This one will install a program. We will assume the program is in drive B and should end up in its own directory in drive C. The program name is XWYTM. We will use the same name for the directory. We will assume that the distribution disk contains a batch file with that name, similar to the one above. The purpose of the batch file we will design is to create the directory and transfer the files from B: to that directory. It will also copy the file XWYTM.BAT to the root of drive C:.

```
@ECHO OFF
ECHO Creating Directory
MD C:\XWYTM
ECHO Copying Files
COPY B:\*.* C:\XWYTM\*.*
COPY B:\XWYTM.BAT C:\*.*
ECHO Installation Complete
ECHO Type XWYTM to run the program
ECHO.
```

Let us analyze the operation of this batch file. The first statement only turns off the echo to the screen. The idea is not to clutter the screen with irrelevant information. The second line tells the user we will create the directory needed in drive C:. The next line creates the directory. The next line tells the user we will copy the files. The next two lines do the actual copying. The final three lines notify the user the installation is complete and the program ready to run.

There is one slight change to this file. This change will improve its operation. The command COPY has no verify switch. We will change this command into XCOPY. This command has the switch /v, that verifies each copy. The modified batch file is:

```
@ECHO OFF
ECHO Creating Directory
MD C:\XWYTM
ECHO Copying Files
XCOPY B:\*.* C:\XWYTM\*.* /v
XCOPY B:\XWYTM.BAT C:\*.* /v
ECHO Installation Complete
ECHO Type XWYTM to run the program
ECHO.
```

You can see the changes. The two COPY commands are now XCOPY. Both have the switch /v to verify the copy. Let us complicate a little more the operation of this batch file. Consider the program we need to install has files stored in several directories of the disk. They should be copied onto subdirectories of XWYTM, with the same name. The only change is to the first XCOPY command. This line is now:

```
XCOPY B:\*.* C:\XWYTM\*.* /v /s
```

The switch /s has been added to indicate that the operation should include the subdirectories, as well. Doing this with the COPY command would be very complex.

Another change is to permit the user to indicate which drive has the distribution disk and which drive receives the program. The easiest way to do this is to instruct the user. This can be done in a READ.ME file. The user should call INSTALL with two parameters. The first parameter is the letter of the drive that has the distribution disk. The second parameter is the letter of the drive that receives the program. It is important to test that the user has done so. The batch file can be as follows:

```
@ECHO OFF
IF %1B == B GOTO ERROR
IF %2B == B GOTO ERROR
ECHO Program Installation Batch File
ECHO The distribution disk is in drive %1:
ECHO The Program XWYTM will install in %2:\XWYTM
ECHO.
ECHO If this is not right, press Control-C, otherwise
PAUSE
ECHO Creating Directory
MD %2:\XWYTM
ECHO Copying Files
XCOPY %1:\*.* %2:\XWYTM\*.* /v
XCOPY %1:\XWYTM.BAT %2:\*.* /v
ECHO Installation Complete
ECHO Type XWYTM to run the program
ECHO.
PAUSE
GOTO END
:ERROR ECHO This batch needs to know where is the distribution disk,
ECHO and the drive you wish to install XWYTM.
ECHO.
ECHO Please use INSTALL B C, if the distribution disk is in B
ECHO and you wish to install XWYTM in drive C. Change as needed.
ECHO.
PAUSE
:END
```

The operation of the batch file should be clear by now. I recommend you read line by line and assure yourself you understand what happens. If you have any doubt, use the hotwords to review the commands you do not remember.

We mention before that a good batch file that changes the active directory should end up returning the active directory to what it was before. This is not an easy task. DOS does not give us the tools for doing it. We need to improvise. We cannot make any assumption. We will develop the procedure to return to the active directory in general. We need to have one batch file that is run before any batch file makes changes in the active directory. We will call this batch file MARK.BAT. This batch file will create another batch file RETURN.BAT. This is the one that returns the active directory to its previous state. The operation of MARK.BAT requires we have a very small text file. We will call it CD.TXT. This file has only CD and nothing else. There is no ending carriage return line feed. We can create this file by the procedure explained before for creating batch files: COPY CON CD.TXT. We press Enter. We type CD, space, F6, Enter.

Now the file. Pay attention to all the tricks.

```
@ECHO OFF
TYPE CD.TXT > RETURN.BAT
CD >>RETURN.BAT
CD \
CD >> RETURN.BAT
```

This is all. You CALL MARK.BAT at the beginning of your batch file. You CALL RETURN.BAT before exiting your batch file. This will put the active directory to where it was. Let us analyze how MARK.BAT does it. First MARK.BAT types the file CD.TXT with the output redirected onto RETURN.BAT. This creates the file RETURN.BAT, destroying any other file with that name. Recall that CD.TXT has only the letters CD and a space. Next we execute the command CD. This command shows the current active path. The output of the command is appended to the file RETURN.BAT. Then we change to the root directory of the drive where we are. The final CD append to RETURN.BAT the drive where we are. Say, we are in drive D, directory UTIL, subdirectory TEST. We run MARK.BAT and the RETURN.BAT file produced will be like this:

```
CD D:\UTIL\TEST
D:
```

The first line changes the active directory of drive D: to UTIL\TEST. Recall that DOS maintains an active directory for each drive in our system. The second line is the one that changes the active drive to D: and achieves our purpose.

Let us consider another case. Imagine we need to perform the same operation on a number of files. The situation is like this. A friend has given us a disk with some twenty five games. Each has many files. All the files have been archived together. Each requires its own directory. Our plan is to create a directory called GAMES. Inside that directory we plan to create one directory for each game. The name of each directory will match the name of the game. Each archived file needs to be extracted to its proper directory. We need to create batch files to run the games. Think on the operations required to perform this task. You need to create the directory GAMES. You will create the 25 subdirectories for each game. You need a list of the names. You need to change the active directory to each of the subdirectories. You call the de-archiving program to extract the files. This must be repeated 25 times. Finally, you should write 25 batch files as the one shown at the beginning of the chapter. We leave as an exercise to determine how many command lines you will need to write. Determine also how many typing errors you will produce. You can simplify your life very much by using batch files.

Let us face first the task of creating the directories. We need a list of the names. We can get this by asking for a directory of drive B. We copy the names from the screen on a piece of paper. Since we will develop a batch file, let us start by getting the list of names. Imagine our batch file will be GAMES.BAT. We write on the command line:

```
DIR B:\*.* > GAMES.BAT
```

We call our text editor with GAMES.BAT. We find that the directory is actually there. Depending on what editor you use, it is easy to erase all the information except the column of names. Now we add at the beginning,

```
@ECHO OFF
C: CD \
MD GAMES
CD GAMES
FOR %%X IN (List of Names) DO CALL Z.BAT %%X
Z.BAT
```

will create the directories. We convert the list of names onto groups of names. Each name is separated by one space. Each group should have four or five names. We need several copies of the line above. We copy each group of names to each line, between the parenthesis. Each line will look like this:

```
FOR %%X IN (DEMON CRAPS SOLITAIRE BRICK CHESS) DO MD %%X
```

This line will call batch file Z.BAT with each name of the games. This ends our first batch file. We need the subdirectories created. We need the files de-archived. We need to create the batch files that will run the games. This is the job of Z.BAT. Recall that the name of the game is in parameter %1. So we write:

```
CD \
CD %1
C:\ARCHIVE\PKUNZIP B:\%1.ZIP
```

```
CD ..
ECHO
ECHO OFF > %1.BAT
ECHO C: >> %1.BAT
ECHO CD \ >> %1.BAT
ECHO CD \GAMES\%1 >> %1.BAT
ECHO %1 >> %1.BAT
```

We do not need to include ECHO OFF in this batch file because it is called from another batch file that has already turned the echo off. The echo will be off until you exit the first batch file. The first few statements are similar to the example given in a previous chapter. The CD .. simply goes down one level in the structure of the directories. The active directory is then C:\GAMES. Then we create a file %1.BAT in this directory by writing the output of a series of ECHO commands. The first command creates the batch file. The other lines append the output of the ECHO commands to the same batch file.

This is a good time for you to take a break. When you come back, review this two batch files and be sure they do as explained. If you have a disk with several archived files, try to write your own batch file to perform this function. Then, as another exercise, you can write another batch file that will remove all the files, directories and subdirectories from your hard drive.

We wish to finish this chapter that is the motivation for the next two chapters. All these batch files are small. Usually not more than 100 or less bytes. You can put a lot of them in a diskette. How about your hard disk? There is something we usually do not consider. The allocation unit is the smallest space in a disk. A small file uses one allocation unit. The allocation unit in a floppy disk is either 512 bytes or 1024 bytes, depending on the disk size. The allocation unit in a hard disk is a minimum of 2048 bytes and can be much larger. So, each batch file takes a minimum of 2K of space in your hard disk. Can we do anything about it? Keep on reading!

Chapter 5.- Multiple Batch Files

Last chapter ends with some thoughts. Let us elaborate on one of them. Imagine your system has some 250 programs. You have a batch file for running each of them. These batch files are small. Storing these batch files on floppy disk take 512 bytes per file. This assumes you use high density disks. Stored in a 40 megabyte hard disk, each file takes 2048 bytes. Imagine you have a small partition on your hard disk. You use it as boot sector and for the batch files. Each batch file takes 4096 bytes in a small partition. Your batch files take one megabyte of disk space! The space used by batch files can be large. You can reduce this storage by combining batch files. This is what we call Multiple Batch Files.

Imagine you have many batch files to run programs. Imagine now you write one batch file, RUN.BAT, to perform the function of all the small batch files. Let us analyze a particular case as an example. Say you have 3 programs: EDITOR, DRAW, and SHOW. EDITOR permits editing any text file. It requires the use of several auxiliary programs. These programs are in a directory of drive C. The directory name is UTILITY. The subdirectory name is EDITOR. DRAW is a program to draw pictures. It is in the subdirectory DRAW. The directory name is PROGRAMS. It also uses several auxiliary programs. SHOW is a program to display pictures. The program is in the root of directory UTILITY. It requires that the active directory be the subdirectory PICTURES of the directory DATA. This is where the pictures are.

You know how to write a batch file to run each of these programs. Let us see how you write one batch file RUN.BAT, to run all three. To run the program EDITOR you type RUN EDITOR. To run DRAW you type RUN DRAW. To run SHOW you type RUN SHOW. The same for any other program you wish to add. You can add all 250 programs. Keeping this in mind, the batch file RUN.BAT is like this:

```
@ECHO OFF
REM This batch file runs the supported programs.
REM First, it changes the active drive to C:.
REM and the active directory to the root of C.
REM This is only because all programs are in C:.
REM Change the drive in each part if different. C:
CD \
REM It uses the name of the program to jump.
GOTO %1
REM It displays an error message.
ECHO Program %1 not supported.
PAUSE
GOTO END
REM This part runs EDITOR.
:EDITOR
REM This line sets the active directory
REM so EDITOR can find the other programs and files.
CD \UTILITY\EDITOR
REM This line runs the program.
EDITOR.EXE
GOTO END
REM This part runs DRAW.
:DRAW
REM This line sets the active directory
REM so DRAW can find the other programs and files.
CD \PROGRAMS\DRAW
REM This line runs the program.
DRAW.EXE
GOTO END
REM This part runs SHOW.
:SHOW
REM This line sets the active directory.
REM Note that the active directory is DATA\PICTURES
REM so SHOW can find the pictures.
CD \DATA\PICTURES
REM This line runs the program. Note the full path.
REM This case shows the use of switches.
C:\PROGRAMS\SHOW.EXE /C
REM No need for GOTO END after the last one.
REM The End changes the active directory REM to where RUN.BAT is.
REM The first line is for the general case.
:END
CD \
```

The file has extensive comments so you can follow how it works. This is always a good idea. The first line disables the echo. Then we include the first set of comments. We wish to avoid echoing the comment lines. The parts that run each program are standard. The only exception is the one for SHOW. The comments explain it. We can extend this idea to any type of batch files. We pack together several batch files of any kind. We only need to append all the files one after the other. We need labels for each file. We need a GOTO %1 at the beginning. Increment the numbers used for the parameters. The name of the file is now %1. The one that was %1 is now %2.

Another variation of this concept is to have two batch files. The first one presents a menu on the screen with all the programs supported. The name of each program has a letter high lighted. You can use a different color. You can use reverse video in monochrome systems. The same batch file uses the key assignment of ANSI.SYS. It assigns each of the high lighted keys to "RUN PROGRAM." PROGRAM is the name of the program assigned to the key. The user presses the key to run the program. The batch file RUN.BAT requires a change. Disable the assignment of keys before exiting. Who said you cannot write your own menu system?

To use this procedure you need to have ANSI.SYS installed. You do this by adding a line to your CONFIG.SYS file. The line reads DEVICE=C:\DOS\ANSI.SYS Replace the path for whatever you use. Next time you boot your computer ANSI.SYS will install. You can assign a string to a key. The command is ESC[Key code;"string"p or ESC["Letter";"string"p To remove the assignment you need to issue the command again but with the same value, like: ESC[98;98p to reassign key B to B, or ESC["a";"a"p for each key you used.

You can use the PROMPT command from the keyboard. Use $e to represent the ESC key. You can use ECHO or PROMPT from a batch file. Use the ESC in either case. You also can use ECHO PROMPT from a batch file. In this case you use $e.

Finally, you can prepare a text file with all the assignments. You add a line TYPE CODES.TXT to your batch file. The same ideas work for reassigning the key.

Recall that ANSI.SYS permits to make drawings on the screen. You can have boxes of different colors. You can change the color of each name. You can change background and foreground colors. You can make a word blink. It can be a lot of fun to develop you own menu. The most important part is that you can change your menu whenever you find a limitation. You can change colors whenever you want. You can change colors every time it runs!

This requires the use of environment variables. :environment You create an environment variable with the command SET. You use SET as follows: SET Variable Name=Value The spaces and case are meaningful. They produce different variables. You find the value of an environment variable from a batch file by writing %Variable Name%. That is, the name of the variable surrounded by percent signs. Let us develop a batch file that changes colors every time it is run. The batch file is like this

```
@ECHO OFF
:AGAIN
IF %COLOR% == 34 SET COLOR=30
IF %COLOR% == 33 SET COLOR=34
IF %COLOR% == 32 SET COLOR=33
IF %COLOR% == 30 SET COLOR=32
ECHO PROMPT $e[%COLOR%m
ECHO THE COLOR NOW IS %COLOR%
ECHO IF YOU LIKE IT PRESS CONTROL-C, OTHERWISE
PAUSE
GOTO AGAIN
```

There are several points to notice. This batch file works only if the environment variable COLOR exists. You cannot define it from inside a shell. The reason is that the shell is a program. DOS ignores any definition after a program runs. It also gives an error message. Issue the command SET COLOR=30 before you use the batch file. Pay attention to the order of the numbers of the colors. Do not set the color to a number tested below. The color will change again. The batch file is a continuous loop. You need to press Control-C to stop it. You could add other colors to the list in the program. They should be in the range from 30 (black) to 37 (white). You could change the background color. Use 40 to 47. To change both, use 32;40, for example. The semicolon is necessary. No spaced allowed. Spaces change the environment variable and prevent the batch file from working.

We need to say a few words on the command PROMPT. This command sets the environment variable PROMPT. This variable determines if your prompt is C>, C:> or whatever you want. PROMPT also changes the colors of the screen. The basic command is

```
PROMPT $e[x;y;zm$p$g
```

The $e is the way to represent the ESC key from the keyboard. The [is necessary. This command has three parts: x;y;zm sets the colors of the screen. The x is the attributes; the y is the foreground color; and the z is the background color. A value does not change if omitted. The lower case m is the code. The second part is the $p which asks for the current path in the prompt. The $g simply asks that the prompt ends with >. The attributes are a number from 0 to 8, as follows:

0 All attributes off.
1 Bold on.
4 Underscore on (Monochrome only).
5 Blink on.
7 Reverse video on.
8 Concealed on.

The available colors are:

30 Black foreground	40 Black background
31 Red foreground	41 Red background
32 Green foreground	42 Green background
33 Yellow foreground	43 Yellow background
34 Blue foreground	44 Blue background
35 Magenta foreground	45 Magenta background
36 Cyan foreground	46 Cyan background
37 White foreground	47 White background

Let us find out how to use the drawing characters of IBM- type computers. Most of the set of ASCII characters from 128 to 255 are drawing characters. A few of them are shown

For boxes:

```
single line: _ 169 - 196 ¬ 170 ¦ 179 + 192 + 217
double line: + 201 - 205 + 187 ¦ 186 + 200 + 188
single tees: ¦ 180 ¤ 195 + 197 - 193 - 194
double tees: ¦ 185 ¦ 204 + 206 - 202 - 203
assorted:    _ 219 _ 220 _ 223 ¦ 221 _ 222 _ 178 _ 177 _ 176
```

Most text editors accept these codes. Hold the Alternate key. Type the three digits with the keypad. There are many more codes. You could make something like

```
ÁÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º Your Name º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

All of the above makes no sense to you? That means you do not have ANSI.SYS installed in your system. You can only do it from the CONFIG.SYS file. This Chapter should show you that you can have fun with your computer. You need to use your imagination.

Chapter 6.- Recursive Batch Files

We call recursive batch file or program one that calls itself. As you can guess, this is not easy. Consider the following batch file, we call R.BAT.

```
@ECHO OFF
CLS
ECHO Hello! This is R.BAT.
CALL R.BAT
ECHO This is the end of R.BAT
```

Let us analyze this file. First we turn the echo off. Then, we clear the screen. Then, we put the Hello message. Then, we call R.BAT. The previous operations repeat. R.BAT calls itself again. And again, and again,....... The second ECHO statement never executes. We have an infinite loop.

Let us study first why we need this structure. Consider the following practical problem. You have a subscription to a Disk Club. They send you several shareware disks every month. Every month you make copies for your brother. You also make backup copies for both of you. The programs come in 360K disks. You put them in high density disks to save space. You also compress the programs for the backup. You do this from the command line. You perform the following operations:
- create directories for each program.
- copy each program to its own directory.
- copy the programs to high density disks, for your brother.
- compress the programs.
- copy the compressed programs to high density disks.
- make another copy for your brother.
- remove all the files from your hard disk.

You dislike the many command lines you have to write. You also hate the mistakes. The computer is ideal to simplify these functions. You can recognize all them. They are very similar. We have studied all of them. They all require two batch files of the same form. Let us review this form: You need a batch file with two lines:

```
@ECHO OFF
FOR %%S IN (NAME OF THE PROGRAMS) DO CALL Z.BAT
```

The file Z.BAT performs the required function. You need to perform 6 functions of the same type. You need to write 12 batch files. One pair runs twice. You perform these operations every month. You need to save the files in your hard disk. All the files are small. They would take a lot of disk space. The first level of simplification is to recognize that one file in each pair is similar to all the others. You can put all the FOR lines in a single batch file. Now you have 7 files, instead of 12. This also prevents errors in the order of the calls. Consider you need to copy the names of the programs onto seven lines. Another chance for mistakes. The idea is to develop a single file that performs the function.

The problem gives us a matrix. This matrix has on one side several programs. The other side of the matrix has several operations. We have approached the problem from the way we execute it by hand. This is always a good starting point. This also gives the wrong solution. Computers do not work the way humans do. Let us approach the problem from another angle. Consider we have a FOR statement with all the different operations. This statement calls another batch file with the operation as parameter. The second batch file has a FOR statement with the list of program names. This line calls the batch file that performs the function. It uses the parameter for this call. It passes as parameter the name of the program. There you have another view of the same matrix. Let us study it.

```
@ECHO OFF
FOR %%S IN (Z Y X W V V U) DO CALL T.BAT %%S
```

```
@ECHO OFF FOR %%Y IN (LIST OF PROGRAMS) DO CALL %1.BAT %%Y
```

Let us say the first file is F.BAT and the second G.BAT. We run F.BAT. This file has the list of functions as the name of the batch files that perform the function. F.BAT calls G.BAT seven times. Each CALL uses a different function. G.BAT receives the CALL with the function as parameter %1. It has the list of programs. It calls the batch file to perform the function. It uses the name of the program as parameter. Now you have two batch files. You need to type the name of the programs only once. This prevents mistakes. This is why we said computers do not work as humans. With this change you only have two FOR statements instead of 7.

There are other interesting characteristics. The first batch file uses no parameter. The second batch file uses one parameter. It is possible to distinguish between them. Let us put them in one file we call T.BAT.

```
@ECHO OFF
IF %1X == X GOTO MAIN
FOR %%Y IN (LIST OF PROGRAMS) DO CALL %1.BAT %%Y
GOTO END
:MAIN
FOR %%S IN (Z Y X W V V U) DO CALL T.BAT %%S
:END
```

Calling T.BAT without parameters gets you to MAIN, which calls T.BAT again. This time there is a parameter. The comparison fails. The first FOR line executes. This line calls a batch file with the program name. This is a recursive batch file. Let us analyze why it works. The reason is simple. When it calls itself it has a parameter. The original call has no parameters. The statement after the ECHO OFF uses this fact. It routes the operations one way or another. In this way, the batch file cannot call itself without control. Compare this batch file with the one given above. This is very illustrative but you still need 7 batch files.

Can we put the other batch files inside T.BAT? The answer is yes. The reason is the same. The CALL now has two parameters. Let us do it.

```
@ECHO OFF
IF %1X == X GOTO MAIN
IF %2X == X GOTO SECOND
GOTO %1
:SECOND FOR %%Y IN (LIST OF PROGRAMS) DO CALL T.BAT %1 %%Y
GOTO END
:MAIN
FOR %%S IN (Z Y X W V V U) DO CALL T.BAT %%S
GOTO END
:Z  (Perform function Z)
GOTO END
:Y (Perform function Y)
GOTO END
:X (Same for the others)
:END
```

Let us analyze how this works. The first IF statement checks for a first parameter. Actually, it tests if there is any parameter. There cannot be a second without a first. Execution goes to MAIN if there is no parameter. Main calls T.BAT with the name of the operation as parameter. This time the first IF fails. The second IF checks for a second parameter. There is none. Execution goes to SECOND. This FOR statement calls T.BAT. This CALL has two parameters. This time both IF statements fail. The execution takes the GOTO to a label equal to the first parameter. This is the function to perform. The program name is the second parameter. As an exercise it is good you write this batch file. You could use the T.BAT included with this book. The idea is to change the function performed by the small batch files. Instead of copying programs or compressing them, use a line ECHO I am in %2.BAT, working with %1 You can use any names for the list of programs. Run the batch file and you will see it works as explained. This is probably the most useful techniques for batch files.

Chapter 7.- Closing Considerations

This concludes the analysis of batch files. I am confident you can develop a batch file to perform any function you want. You need to practice thus new knowledge. This is true with any new knowledge. You have to make it set in your brain. Here you have some suggestions of how you can use this new knowledge. You set the knowledge and produce something useful. My suggestion is to start with all the programs you have in your system. Develop batch files to run them without regard of where you are. Start with the programs you use more often. Go one by one. Develop a batch file that will change the active directory and run the program. Note that you only need this if the program has some auxiliary files or programs. You could develop a batch file for running the programs that do not require auxiliary files. The batch file remembers where the program is. Which drive, directory, and subdirectory. Put these batch files in the root of C. This assumes you boot from C. Edit your AUTOEXEC.BAT file. Remove any line that sets the variable PATH. Leave in the PATH only the directory where you have DOS. This speeds up your computer.

When you finish with the small batch files, develop a single batch file as explained in Chapter 5, Multiple Batch Files. You will notice your computer is not much easier to use. I am sure that running programs is also faster. Examine your system looking for places where a batch file can simplify operations. Make a habit of writing a batch file every time you need to perform a command with more than two lines. Eventually, you might want to try to write your own menu. You will have a lot of fun.

This is a good point to think about the next step. I imagine you realize how much control you have on your computer. You did not have that control before. Imagine how much control you can have if you program your computer. Programming is not that difficult. It involves only two points:
One, you need to learn to think like the computer. You have some examples of this type.
Two, you need to learn a language the computer understands.
You already know one. You can learn another. There are many computer languages. Most of them are very easy to use. You probably know something about BASIC. This is a good start. Practice programming the computer in BASIC. A good topic, if you do not have one, is producing music. BASIC can perform very complex applications. The problem is that it is slow. You can learn other languages. Pascal is very good to learn the principles of good programming. This is the purpose of the language. The language C is very good. It is very simple to learn. You can do anything you want. Assembler Language is the most direct language to the computer. The programs are much smaller. The programs also run much faster. Learning assembler language is not as easy as learning BASIC or C. There are capabilities you only can get from assembler language. Do not try assembler language as the first one you learn.

The most important point to learn a language is to have a purpose. You can learn French very easily by going to Canada. To learn a computer language you need to have a way to use it. Think of a possible application. Say, you want to develop your own menu. Start by thinking in how it will look. Try to develop the program that produces that screen. You will realize that it is simpler than it look from the outside. It gives you the sensation of power over the computer. You do not have that sensation with DOS or any shell. I hope you learn something.

The introduction says you are welcome with comments and suggestions. This is a good point to add you are also welcome with questions, if you have any.

This page was last modified on 12/27/2001