Introduction

First of all, I am not original the writer of this material and this is the second edition. I came across it in a text file on my computer a very very long time ago, so I decided to format it, edit it, convert it into an e-book and then publish it for free. I distributed this material amongst students in my class back in my diploma days because we had a course in Batch Programming, and the material turned out to be very useful!

This e-book, “Batch Programming”, teaches a lot about programming in MS-DOS or Batch Programming. I myself have learned a lot of new stuffs from this material, and I would recommend it for anyone who wants to know little or much about programming in MS-DOS; its application ranges from one-click disk formatting or file logging, to printing or anti-virus, etc.

From this e-book, you will learn some basic MS-DOS commands and their applications in batch programming. But you can also learn more commands if you have MS-DOS! All you need to do is open MS-DOS and type “help” then press “↵ Enter”, and all commands available to you on your computer system will be displayed. You can try them out one after another to learn how they are used.

With my little knowledge of batch programming, I was able to develop a little program to rid computers of certain viruses, one of which is very well known to remove the “Folder Options” from the Tools menu, and makes it impossible to view hidden files and folders on Microsoft Windows operating systems. I made my own antivirus! I’m sure that the knowledge from this e-book, if applied with some creativity, can yield very productive programs for you too!

Kheme

About Me

My name is Okiemute Omuta, but I'm very well known as Kheme. I am a Nigerian freelance website designer and developer specialized in PHP & MySQL. I also have a good understanding of JavaScript, DHTML and CSS. I also have a considerable applicable knowledge of Flash, Action Scripts, AJAX, C/C++ and Oracle SQL.

I have gathered experience in designing websites commercially using PHP, MySQL, (X)HTML, CSS, JavaScript and DHTML since 2003. Hence, I am able to develop both static and dynamic websites, from small scale websites through medium scale to medium-large scale websites.

Besides developing websites, I run a free website design tutorials website where I have website design related lessons. The name of the website is HTML Online (www.htmlonline.tk); I also have a few free e-books on that website, such as this one you're reading right now.

If you noticed any spelling or typographical errors, or you just like my tutorials, or you have problems or questions to ask, feel free to email me at Kheme@htmlonline.tk or visit my website at www.khemeonline.tk. Also, I consider it consolation for my efforts when my readers e-mail in to say a thing or two about my e-books, or simply just to say "Thanks!" :)

Feel free to share this e-book with you friends and those you feel could benefit from it.

Thanks for downloading one of my e-books.

Kheme

Table of Content

Introduction to Batch

Batch file programming is nothing but the Windows version of Unix Shell Programming. Let's start by understanding what happens when we give a DOS command. DOS is basically a file called “command.com”. It is this file (command.com) that handles all DOS commands that you give at the DOS prompt, such as COPY, DIR, DEL, etc. These commands are built-in with the command.com file (known as internal commands. DOS has something called external commands too such as FORMAT, UNDELETE, BACKUP, etc.

So whenever we give a DOS command either internal or external, command.com either straightaway executes the command (internal commands) or calls a separate external program which executes the command for it and then returns the result to command.com.

So why do I need Batch File Programs? Say you need to execute a set of commands over and over again to perform a routine task, say like backing up important files, Deleting temporary files (.tmp, .bak , etc) or formatting a disk, then it becomes very difficult to type the same set of commands over and over again. To perform a bulky set of same commands over and over again, batch files are used! Batch Files are to DOS what Macros are to Microsoft Office and are used to perform an automated predefined set of tasks over and over again.

So how do you create batch files? To start enjoying the use of batch files, you need to learn to create batch files. Batch files are basically plain text files containing DOS commands. So the best editor to write your commands in would be Notepad or the DOS Editor. All you need to remember is that a batch file should have the extension “.BAT”.

Executing a batch file is quite simple too. For example, if you create a batch file and save it with the filename “batch.bat” in the windows folder on drive C:, then all you need to execute the batch file is to type at DOS prompt:

```
C:\windows>batch.bat
```

So what happens when you give a batch file to the command.com to execute? Whenever command.com comes across a batch file program, it goes into batch mode. In the batch mode, it reads the commands from the batch file line by line. So basically what happens is, command.com opens the batch file and reads the first line and then it closes the batch file. It then executes the command and again reopens the batch file and reads the next line from it, and so on and so forth. Batch files are treated as internal DOS commands.

NOTE: While creating a batch file, one thing that you need to keep in mind is that the filename of the batch file should not use the same name as a DOS command. For example, if you create a batch file by the name “dir.bat” and then try to execute it at the prompt, nothing will happen. This is because when command.com comes across a command, it first checks to see if it is an internal command. If it is not then command.com checks if it a .COM, .EXE or .BAT file with a matching filename. All external DOS commands use either a .COM or a .EXE extension, DOS never bothers to check if the batch program exits.

Now let's move on to your first batch program! We will unlike always first take up a simple batch file which executes or launches a “.EXE” program. Simply type the following in a blank text file and save it with a .BAT extension.

```
C:

cd windows

telnet
```

Now let's analyze the code; the first line tells command.com to go to the C: drive. Next, it tells it to change the current directory to Windows. The last line tells it to launch the telnet client. You may contradict saying that the full filename is telnet.exe. And yes you are right, but the .exe extension is automatically added by command.com. So normally, we do not need to change the drive and the directory because the Windows directory on drive C: is the default DOS folder. So instead, the bath file could simply contain `telnet` and would still work.

Now let's execute this batch file and see the results. Launch command.com and execute the batch file by typing:

`C:\WINDOWS>batch_file_name` e.g. C:\WINDOWS>scandisk

So now that you know the basic functioning of Batch files, let' move on to batch file commands.

The REM Command

The simplest basic Batch file command is the REM or the Remark command. It is used extensively by programmers to insert comments into their codes to make it more readable and understandable. This command ignores anything on that line. Anything on the line after REM is will not be displayed on the screen during execution. It is normally not used in small and easy to understand batch programs, but is very useful in huge snippets of codes with geek stuffs loaded into it. So if we add Remarks to out first batch file, it will become:

```
REM This batch file is my first batch program!

telnet
```

The only thing to keep in mind while using Remarks is to not go overboard and putting in too many of them into a single program as they tend to slow down the execution time of the batch commands.

The ECHO Command

The ECHO command is used for what the Print command is in other programming languages: To display something on the screen. It can also be used to tell the user what the bath file is currently doing. It is true that batch programs display all commands it is executing but sometimes they are not enough and it is better to also insert ECHO commands here and there to give a better description of what is presently being done. Say for example, the following batch program full of the ECHO command, deletes all files in the c:\windows\temp directory:

```
ECHO Program to delete unwanted temporary files from system

ECHO Now we go to the Windows\temp directory.

cd temp REM Because we are already in the Windows folder

ECHO Deleting unwanted temporary files...

del *.tmp

ECHO Your System is Now Clean
```

Now let's see what happens when we execute the above snippet of batch code.

```
C:\WINDOWS>batch_file_name

C:\WINDOWS>ECHO Program to delete unwanted temporary files
from system

C:\WINDOWS>ECHO Now we go to the Windows\temp directory.

Now we go to the Windows\temp directory.

C:\WINDOWS>cd temp

C:\WINDOWS\TEMP>ECHO Deleting unwanted temporary files...

Deleting unwanted temporary files...
```

```
C:\WINDOWS\TEMP>del *.tmp

C:\WINDOWS\TEMP>ECHO Your System is Now Clean

Your System is Now Clean
```

The above is a big mess! The problem is that DOS is displaying the executed command and also the statement within the ECHO command. To prevent DOS from displaying the command being executed, simply precede the batch file with the following command at the beginning of the file:

```
ECHO OFF
```

Once we add the above line to our Temporary files deleting Batch program, the output becomes:

```
C:\WINDOWS>ECHO OFF

Program to delete unwanted temporary files from system

Now we go to the temp directory.

Invalid directory

Deleting unwanted temporary files...

File not found

Your System is Now Clean
```

Hey pretty good, right? But it still shows the initial ECHO OFF command. You can prevent a particular command from showing, but still be executed by preceding the command with an @ sign. So to hide even the ECHO OFF command, simply replace the first line of the batch file with:

```
@ECHO OFF
```

You might think that to display a blank line in the output screen, you can simply type ECHO by itself, but that doesn't work. Say you have started your batch file with the command ECHO OFF and then in the later line give the ECHO command, then it will display 'ECHO is off' on the screen. You can display a blank line by giving the command:

```
ECHO.
```

Simply leaving a blank line after the dot also displays a blank line in the output.

You can turn ON the ECHO anytime by simply giving the command ECHO ON. After turning the echo on, if you give the ECHO command it will return 'ECHO is on'.

The PAUSE Command

Say you create a batch file which shows the directory listing of a particular folder before performing some other task, or before deleting all files of a folder, you need to give the user time to react and change their mind; then the PAUSE command comes in handy! The PAUSE command is used to timeout actions of a script. Now consider the following scenario:

```
REM Program to delete all .doc files from current folder
REM Program also allows user to react and abort this process
@ECHO OFF
ECHO WARNING: I'm going to delete all Word Document
ECHO Press CTRL+C to abort or simply press any key to continue
PAUSE
DEL *.doc
```

Now when you execute this batch program, you get the following output:

WARNING: I'm going to delete all Word Document

Press CTRL+C to abort or simply press a key to continue.

Press any key to continue . . .

The batch program actually asks the user if he wishes to continue and gives the user the option to abort the process. CTRL+C (or CTRL+Break) cancels the batch file program and you get the DOS prompt back.

NOTE: Say you have saved a batch file in the c:\name directory, when you launch command.com the default directory is still c:\windows and in order to execute the batch file program stored in the c:\name directory you will have to change the directory to c:\name.

The SET PATH Command

Say you have saved a batch file in the c:\name directory. Now when you launch command.com, the default directory is c:\windows and in order to execute the batch file program stored in the c:\name directory you need to change the directory and go to c:\name. This can be very irritating and time consuming. It is a good practice to store all your batch programs in the same folder. You can run a batch file stored in any folder (Say c:\name) from anywhere (even c:\windows\history) if you include the folder in which the batch file is stored (c:\name) in the AUTOEXEC.BAT file, so that DOS knows which folder to look for the batch program.

So simply open c:\autoexec.bat in Notepad and append the Path statement to the following line (where c:\name is the folder in which all your batch files are stored):

```
SET PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\name
```

Autoexec.bat runs each time at startup and DOS knows each time, in which directory to look for the batch files.

Parameters

To make batch programs really intelligent, you need to be able to provide them with parameters, which are nothing but additional valuable information which is needed to ensure that the bath program can work efficiently and flexibly. To understand how parameters work, look at the following script:

```
@ECHO OFF

ECHO First Parameter is %1

ECHO Second Parameter is %2

ECHO Third Parameter is %3
```

The script seems to be echoing messages on the screen, but what do the strange symbols %1, % 2 and %3 stand for? To find out what the strange symbols stand for, save the above script and go to DOS and execute this script by passing the below parameters:

```
C:\windows>batch_file_name abc def ghi
```

This batch file produces the following result:

First Parameter is abc

Second Parameter is def

Third Parameter is ghi

The first line in the output is produced by the code line:

```
ECHO First Parameter is %1
```

Basically what happens here is that when DOS encounters the %1 symbol, it examines the original command used to execute the bath

program and look for the first word (argument) after the batch filename and then assigns %1 the value of that word. So one can say that in the ECHO statement, %1 is replaced with the value of the first argument. In the above example the first word after the batch file name is abc, therefore %1 is assigned the value of this word.

The %2 symbol too works in the similar way, the only difference being that instead of the first argument, DOS assigns it the value of the second argument, def. Now all these symbols, %1, %2, etc. are called replaceable parameters. Actually, what happens is that %1 is not assigned the value of the first argument, but it is in fact replaced by the value of the first argument.

If the batch file command has more parameters than what the batch file is looking for, the parameters extras are ignored. For example, if while executing a batch file program, we pass four arguments, but the batch program only requires 3 parameters, then the fourth parameter will be ignored.

To understand the practical usage of parameters, let's take up a real life example. Now the following script requires the user to enter the name of the files to be deleted and the folder in which they are located.

```
@ECHO OFF

CD\

CD %1

DEL %2
```

This script can be called from the DOS prompt in the following way:

```
C:\windows>batch_file_name windows\temp *.tmp
```

In a single script, we cannot use more than nine replaceable parameters. This means that a particular batch file will have replaceable parameters

from %1 to %9. In fact, there is a tenth replaceable parameter, the %0 parameter. The %0 parameter contains the name of the batch file itself.

NOTE: Say you want to execute a batch file and once the procedure of execution is complete, you want to leave DOS and return to Windows, what do you do? The EXIT command can be used in such situations. So simply end your batch file with the EXIT command:

```
EXIT
```

The Shift Command

Sometimes your batch program may need to use more than nine parameters at a time (actually you would never need to, but at least you can be sure you can handle it if you need to). To see how the SHIFT command works, look at the following code:

```
@ECHO OFF

ECHO The first Parameter is %1

ECHO.

SHIFT

ECHO The Second Parameter is %1

ECHO.

SHIFT

ECHO The Second Parameter is %1
```

Now execute this batch file from DOS and see what happens.

```
C:\windows>batch_file_name abc def ghi
```

The first Parameter is abc

The Second Parameter is def

The Second Parameter is ghi

How does it work? Well, each SHIFT command shuffles the parameters down position one. This means that after the first SHIFT %1 becomes def, %2 becomes ghi and abc is completely removed by DOS. All parameters change and move one position down.

Both normal parameters (%1, % 2, etc) and the SHIFT command can be made more efficient by grouping them with the IF conditional statement to check the parameters passed by the User.

The FOR Command

The syntax of the FOR LOOP is:

`FOR %%PARAMETER IN (set) DO command`

Most people change their mind about learning batch programming when they come across the syntax of the FOR command. I do agree that it does seem a bit weird, but it is not as difficult as it appears to be! Let's analyze the various parts of the FOR command, shall we. Before we do that look at the following example:

```
@ECHO OFF

CLS

FOR %%A  IN (abc, def, xyz) DO  ECHO  %%A
```

Basically, a FOR LOOP declares a variable (%%A) and assigns it different values as it goes through the predefined set of values (in this case: abc, def, xyz) and each time the variable is assigned a new value, the FOR loop performs a command (in this case: `ECHO %%A`).

The %%A is the variable which is assigned different values as the loop goes through the predefined set of values in the brackets. You can use any single letter character after the two % sign except numbers. We use two "%" as DOS deletes each occurrence of a single % sign in a batch file program.

The IN (abc, def, xyz) is the list through which the FOR loop goes. The variable "%%a" is assigned the various values within the brackets, as the loop moves. The items in the set can be separated with commas, colons or simply spaces.

For each item in the set, the FOR loop performs whatever command is given after the DO keyword (in this case: s).

So basically, when we execute the above batch file, the output will be:

abc

def

xyz

The FOR loop becomes very powerful if used along with replaceable parameters. Take the following example:

```
@ECHO OFF

ECHO.

ECHO I am going to delete the following files:

ECHO %1 %2

ECHO.

ECHO Press Ctrl+C to Abort process

PAUSE

FOR %%a IN (%1 %2 ) DO DEL %%a

ECHO Killed Files. Mission Accomplished.
```

At execution time, the process would be something like:

```
C:\WINDOWS>batchfilename *.tmp *.bak
```

I am going to delete the following files:

***.tmp *.bak**

Press Ctrl+C to Abort process

Press any key to continue . . .

Killed Files. Mission Accomplished.

The IF Command

The IF statement is a very useful command which allows us to make the batch files more intelligent and useful. With this command, one can make the batch programs check parameters and perform a task accordingly. Not only can the IF command check parameters, it can also checks if a particular file exists or not! On top of all this, it can also be used for the conventional checking of variables or strings.

Checking if a file exists or not

The general syntax of the IF command which checks for the existence of a file is the following:

```
IF [NOT] EXIST FILENAME Command
```

This will become clearer when we take up the following example:

```
IF EXIST c:\autoexec.bat ECHO It exists
```

This command checks to see if the file, c:\autoexec.bat exists or not. If it does then it echoes the string 'It exists'. On the other hand, if the specified file does not exist, then it does not do anything.

In the above example, if the file autoexec.bat did not exist, then nothing was executed. But we can also put in the ELSE clause, that is, if the file exists, do this but if it does not exist, we make the program do something else by using the GOTO command. Let's consider the following example to make it clearer:

```
@echo off

IF EXIST C:\ankit.doc GOTO ANKIT

Goto end

:ANKIT

ECHO ANKIT

:end
```

The IF statement in this code checks to see if c:\ankit.doc exists; if it does, then DOS is branched to :ANKIT and if it does not, then DOS goes on to the next line. The next line branches DOS to :end. The :end and :ANKIT in the above example are called labels. After the branching, the respective ECHO statements take over.

NOTE: We can also check for more than one file at a time:

```
IF EXIST c:\autoexec.bat IF EXIST c:\autoexec.bak ECHO Both Exist
```

We can check to see if a file does not exist in the same way, the basic syntax then becomes:

```
IF NOT EXIST FILENAME Command
```

For Example,

```
IF NOT EXIST c:\ankit.doc ECHO It doesn't Exist
```

NOTE: How do you check for the existence of directories? Something like `IF C:\windows EXISTS ECHO Yes` will not work! In this case, we need to make use of the NULL device. The NULL device is basically nothing; it actually stands for nothing! Each directory has the NULL device present in it or at least DOS thinks so! So to check if c:\windows exits:

```
IF EXIST c:\windows\nul ECHO c:\Windows exists
```

One can also check if a drive is valid, by giving something like:

```
IF EXIST c:\io.sys ECHO Drive c: is valid
```

Comparing strings to validate parameters

The basic syntax is:

```
IF [NOT] string1==string2 Command
```

Now let's make our scripts intelligent and make them perform a task according to what parameter was passed by the user. For example:

```
@ECHO off
IF %1==cp GOTO COPY
GOTO DEL
:COPY
Copy %2  a:
GOTO :END
:DEL
Del %2
:END
```

This example too is pretty much self explanatory. The IF Statement compares the first parameter to cp, and if it matches then DOS is sent to the COPY label else to the DEL label. This example makes use of two parameters and is called by passing at least two parameters. We can edit the above example to make DOS check if a parameter was passed or not, and if not then display an error message. Just add the following lines to the beginning of the above file:

```
@ECHO OFF
IF "%1" == "" ECHO Error Message Here
```

The CHOICE Command

Before we learn how to make use of the CHOICE command, we need to know what error codes really are. Now error codes are generated by programs to provide information about the way they finished or were forced to finish their execution. For example, when we end a program by pressing CTRL+C to end a program, the error code becomes 3 and if the program closes normally, then the error code becomes 0. These numbers called error levels all by themselves are not useful, but when used with the IF ERRORLEVEL and the CHOICE command, they become very cool!

The CHOICE command takes a letter or key from the keyboard and returns the error level evaluated when the key is pressed. The general syntax of the CHOICE command is:

```
CHOICE[string][/C:keys][/S][/N][/T:key,secs]
```

The string part is nothing but the string to be displayed when the CHOICE command is run. The `/C:keys` defines the possible keys to be pressed. If options are mentioned then the default `Y/N` keys are used instead.

For example, the command,

```
CHOICE /C:A1T0
```

defines A, 1, T and O as the possible keys. During execution, if the user presses an undefined key, he will hear a beep sound and the program will continue as coded.

The `/S` flag makes the possible keys defined by the CHOICE `/c` flag case sensitive. So it means that if the `/S` flag is present then, A and a would be different.

The /N flag, if present shows the possible keys in brackets when the program is executed. If the /N flag is missing then, the possible keys are not shown in brackets. Only the value contained by STRING is shown.

/T:key,secs defines the key which is taken as the default after a certain amount of time has passed. For Example:

```
CHOICE Choose Browser /C:NI /T:I.5
```

The above command displays **Choose Browser [N,I]** and if no key is pressed for the next 5 seconds, then it chooses I.

Now to truly combine the CHOICE command with the IF ERRORLEVEL command, you need to know what the CHOICE command returns.

The CHOICE command is designed to return an error level according to the pressed key and its position in the /C flag. To understand this better, consider the following example,

```
CHOICE /C:AN12
```

Now remember that the error code value depends on the key pressed. This means that if the key A is pressed, then the error level is 1, if the key N is pressed then the error level is 2, if 1 is pressed then error level is 3 and if 2 is pressed then error level is 4.

Now let us see how the IF ERRORLEVEL command works. The general syntax of this command is:

```
IF [NOT] ERRORLEVEL number command
```

This statement evaluates the current error level number. If the condition is true then the command is executed. For Example,

```
IF ERRORLEVEL 3 ECHO Yes
```

The above statement prints Yes on the screen if the current error level is 3. The important thing to note in this statement is that the evaluation of an error level is true when the error level is equal to or higher than the number compared. For Example, in the following statement,

```
IF ERRORLEVEL 2 ECHO YES
```

The condition is true if the error level is > or = 2. Now that you know how to use the CHOICE and ERRORLEVEL IF command together, you can now easily create menu-based programs. The following is an example of such a batch file which asks the user what browser to launch.

```
@ECHO OFF

ECHO.

ECHO.

ECHO Welcome to Browser Selection Program

ECHO.

ECHO 1. Internet Explorer 5.5

ECHO 2. Mozilla 5

ECHO x. Exit Browser Selection Program

ECHO.

CHOICE "Choose Browser" /C:12x /N

IF ERRORLEVEL 3 GOTO END

IF ERRORLEVEL 2 START C:\progra~1\Netscape

IF ERRORLEVEL 1 start c:\progra~1\intern~1\iexplore.exe

:END
```

NOTE: Observe the order in which we give the IF statements.

Redirecting Output

Normally the output is sent to the screen (or standard STDOUT) and the input is read from the keyboard (or standard STDIN). This can be pretty boring, you know, but you can actually redirect both the input and output to something other than the standard I/O devices!

To send the output to somewhere other than the screen, we use the output redirection operator, > which is most commonly used to capture the results of a command in a text file. So let's say you want to read the help on how to use the net command, typing the usual Help command is not useful as the results do not fit in one screen and scrolls quickly. So instead, we use the output redirection operator to capture the results of the command onto a text file.

```
C:\windows>net > xyz.txt
```

This command will execute the net command and will store the results in the text file "xyz.txt". Whenever DOS comes by such a command, it checks if the specified file exists or not. If it does, then everything in the file is erased or lost and the results are stored in it. If no such file exists, then DOS creates a new file and stores the results in this new file.

Say, you want to store the results of more than one command in the same text file, and want to ensure that the results of no command are lost, then you make use of the double output redirection symbol, which is the >> symbol. For Example,

```
c:\windows> net >> xyz.txt
```

The above command tells DOS to execute the net command and append the output to the xyz.txt file, if it exits. DOS not only allows redirection to files, but also allows redirection to various other devices!

DEVICE NAME USED	DEVICE
AUX	Auxiliary Device (COM1)
CLOCK$	Real Time Clock
COMn	Serial Port (COM1, COM2, COM3, COM4)
CON	Console (Keyboard, Screen)
LPTn	Parallel Port (LPT1, LPT2, LPT3)
NUL	NUL Device (means Nothing)
PRN	Printer

For example, if you want to print the results of directory listings you can simply give the following command:

```
c:\windows>dir *.* > PRN
```

The NUL device is a bit difficult to understand and requires special mention. This device which is also known as the 'bit bucket' literally means nothing. Redirection to the NUL device practically has no usage but can be used to suppress the messages which DOS displays on the completion of a task. For example, when DOS has successfully copied a particular file, then it displays the message: '1 file(s) copied.' Now say you want to suppress this task completion message, you then make use of the NUL device.

```
c:\windows>copy file.txt > NUL
```

This will suppress the task completion message and not display it.

Redirecting Input

Just as we can redirect output, we can also redirect input! This is handled by the input redirection operator, which is the < symbol. It is most commonly used to send the contents of a text file to DOS. The other common usage of this feature is the MORE command which displays a file one screen at a time, unlike the TYPE command which on execution displays the entire file (this becomes impossible to read as the file scrolls by at incredible speed). Thus, many people send the long text file to the MORE command by using the command:

```
c:\windows>more < xyz.txt
```

This command sends the contents of the xyz.txt file to the MORE command which displays the contents page by page. Once the first page is read, the MORE command displays something like the following on the screen:

```
......MORE......
```

You can also send key strokes to any DOS command which waits for user input or needs user intervention to perform a task. You can also send multiple keystrokes. For example, a typical format command requires 4 inputs, firstly pressing ↵ Enter to give the command, then disk insertion prompt, then the VOLUME label prompt and lastly the one to format another disk. So basically there are three User inputs-:

↵ ENTER, ↵ ENTER and ↵ ENTER

So you can include this in a batch file and give the format command in the following format as:

```
c:\windows>format a: < xyz.bat
```

Piping

Piping is a feature which combines both input and output Redirection. It uses the Pipe operator, which is the | symbol. This command captures the output of one command and sends it as the input of the other command. For example, when you give the command `del *.*`, you need to confirm that you mean to delete all files by pressing y. Instead, we can simply do the same without any user interaction by giving the command:

```
c:\windows> echo y | del *.*
```

This command is pretty self explanatory, y is sent to the command `del *.*` batch programming can be very easy and quite useful. The only thing that one needs to be able to become a batch file programming nerd is adequate knowledge of DOS commands. I suggest you surf the net or get a book on DOS commands and really lick the pages off the book, and only then can you become an expert!

Working with Windows Registry

There is simply no direct way of editing the windows registry through a batch file. Although there are windows registry command line options, they are not as useful as adding keys or editing keys can be! The best option we have is to create a .reg file and then execute it through a batch file. The most important thing to remember hear is the format of a .reg file and the fact that the first line of all .reg files should contain nothing but the string REGEDIT4, else Windows will not be able to recognize it as a registry file. The following is a simple example of a batch file which changes the home page of the user (if Internet Explorer is installed) to http://www.htmlonline.tk/

```
@ECHO OFF

ECHO REGEDIT4 >ankit.reg

ECHO [HKEY_CURRENT_USER\Software\Microsoft\Internet
Explorer\Main] >> ankit.reg

ECHO "Start Page"=" http://www.htmlonline.tk/" >> ankit.reg

START ankit.reg
```

Creating a .reg file is not as easy as it seems. You see, for Windows to recognize a file as a Registry file and for Windows to add the contents of the .reg file to the registry, it has to be in a particular recognizable format, or else, an error message would be displayed.

Protection against Batch Viruses

If you double-click a batch file (.bat files) it will run automatically. This can be dangerous as batch files can contain harmful commands sometimes. Worst still, if you use the single-click option, one wrong click and it’s goodbye Windows! Now most power users would like to set ‘edit’ as the default action for batch files. The best way to do that is to go to Windows Explorer's Folder Options' File View tab to modify the default action. However, to add insult to injury, when you arrive there, you will find that the Edit and Set Default buttons has been grayed out. This is a "feature" from Microsoft you might not appreciate.

To conquer our problem here, open your registry editor and go to HKEY_CLASSES_ROOT\batfile\shell\open and rename the ‘open’ key to ‘run’, thus becoming HKEY_CLASSES_ROOT\batfile\shell\run. Double-click the EditFlags binary value in HKEY_CLASSES_ROOT\batfile and enter 00 00 00 00 as the new value. Now, open Explorer, click Folder Options from the View menu and select the File Types tab, scroll down to the "MS-DOS Batch File" item, highlight it and click Edit. You'll notice that the last three buttons (Edit, Remove and Set Default) are now enabled and that you can select Edit as the default action.

Miscellaneous

Making your own Syslog Daemon

We can easily combine the power of batch file programs and the customizable Windows interface to make our own small but efficient System Logging Daemon.

Basically, this Syslog Daemon can keep a track of the files opened (any kind of files), the time at which the files were opened and actually post the log of the user's activities on to the web, so that the system administrator can keep an eye on things! Simply follow the following steps to make the daemon-:

NOTE: In the following example, I am making a syslog daemon which keeps an eye on what text files were opened by the User. You can easily change what files you want it to keep an eye on by simply following the same steps.

1. Associating the Files to Be Monitored to the Logger

Actually this step is not the first, but being the easiest, I have to mention it earlier. The first thing to do is to associate the text files (*.txt) files to our batch file which contains the code to log the user's activities. You can of course keep an eye on other files as well, the procedure is almost similar. Anyways, we associate .txt files to our batch program so that each time a .txt file is opened, the batch file is also executed. To do this, we need to change the file associations of .txt files. For more information on Changing File Associations, refer to the Windows Help Files and simply type ‘Associations’ and search. Anyway, to change the associations of .txt files and to point it to our batch file, simply do the following:

Locate any .txt file on your system, right-click on it and click on the OPEN WITH... option will bring up OPEN WITH dialog box. Now click on the OTHER PROGRAM and locate the batch file program which contains the logging code, click on it and click OK.

Note: Make sure the “Always use the selected program to open this kind of file” check box is checked!

Now each time a .txt file is opened, the batch file is also executed, hence logging all interactions of the user with .txt files.

2. Creating the Log File

Now you need to create a text file, which actually will act like a log file and will log the activities of the user. This log file will contain the filename and the time at which the .txt file was opened. Create a new blank text file in the same directory as the batch file. Change the attributes of this log file and make it hidden with the ATTRIB command:

```
C:\windows>attrib xyz.txt +h
```

Just to ensure that a user will not know where the log file is located!

3. Coding the Logging Batch File

The coding of the actual batch file which will log the user's activities and post it on the web is quite simple. If you have read this tutorial properly till now, then you would easily be able to understand it, although I still have inserted some comments for novices.

```
Echo %1 >> xyz.txt

REM Sends the file name of the file opened to the log file

notepad %1

REM Opens notepad so the user doesn’t know something’s wrong
```

This logging file will only log the filename of the text file which was opened by the unsuspecting user, say you want to also log the time at which a particular file was opened, then you simply make use of the 'time' command. The only thing that one needs to keep in mind is that after giving the TIME command, we need to press enter too, which in turn has to be entered in the batch file too.

So you, who is the system administrator, does not have physical access or have gone on a business trip, but have access to the net and need to keep in touch with the server log file, then you easily link the log file to a HTML file and easily view it on the click of a button. You could also make this part of the site password-protected. Anyway, the linking can easily be done by creating an .htm or. html file and inserting the following snippet of code:

```
<html>

<head><title> Server Logs</title></head>

<body>

<a href="xyz.txt>Click here to read the Server Logs</a>

</body>

</html>
```

That was an example of the easiest HTML page one could create (for tutorials on HTML, visit www.htmlonline.tk). Another enhancement that one could make is to prevent the opening of a particular file. Say if you want to prevent the user from launching abc.txt then you would need to insert an IF conditional statement:

```
IF "%1" == "filename.extension" ECHO Error Message Here
```

4. Enhancing the logging Batch file to escape the eyes of the user

To enhance the functioning of our logging daemon, we need to first know its normal functioning. Normally, if you have followed the above steps properly, then each time a .txt file is opened, the batch file is launched (in a new window, which is maximized) and which in turn launches Notepad. Once the filename and time have been logged, the batch file Window does not close automatically and the User has to exit from the Window manually. So maybe someone even remotely intelligent will suspect something fishy. We can configure our batch file to work minimized and to close itself after the logging process has been completed. To do this, simply follow these steps-:

a) Right Click on the Batch File,

b) Click on properties from the Pop up menu,

c) In the Program tab click on the Close on Exit option,

d) Under the same tab, under the RUN input box, select Minimized,

e) Click on Apply and voila the batch file is now more intelligent!

This was just an example of a simple batch file program. You can easily create a more intelligent and more useful program using batch code!

That's it for batch programming... happy coding!

For more of my e-books, go to http://www.htmlonline.tk/ebooks/

Compiled by Kheme