



QUOTE

*The only Source of Knowledge
is Experience...*

BATCH PROGRAMMING DECODED – III

[For Intermediates]

By Kvc

karanveerchouhan@gmail.com

batchprogrammers.blogspot.in

Index

1. 10 Lines About this Part
2. Loops:
 - i. The **For loop**
 - ii. **Goto** command as loop
 - iii. No **while loop** in cmd – *reason and Creation.*
3. Special operators in batch (>, <, &&, ||, &, | etc.)
4. Special Parameters in Batch (%0, %1 ..., %9)
5. The Shift Command
6. File Handling in batch (creating, modifying, deleting Files!!)
7. Operators of File-handling (in Details)
8. *Functions* in batch...
9. String manipulations in Batch...
10. Ending of this part...

10 lines about this book

As this is **part – III**, I'm considering that you had read and understand the basic concepts of batch programming written in **part – I & II** of this book's series [**Newbie & Beginners**], and you are a beginner now.

As you had a look on the Index and maybe you've noticed that this part has very few topics about batch programming than other parts. This is only because I thought that these are important topics of batch & if you understand them then you'll have better grip on batch programming. So, only few topics are there...you can understand them easily... and it'll be easier for me to explain them also.

Loops

A loop may be roughly defined as the programming statement that allows programmer to repeat a particular module (group of statements/Commands) until the required condition becomes false... Without typing the same code several times...

Cmd has only one command for Looping:

THE FOR LOOP

You can find **for loop** in almost all programming languages. Unlike all other programming languages, **for loop** is not a statement in batch... it is one of the Commands.

When I started batch, Understanding **for loop** was the most difficult thing in any batch code for me...and I think that you're experiencing the same situation also... (If you are a beginner yet)

So, let's Read about **for loop** and how we can use it in our batch Programs...

Syntax: **For [variable] in (Set) do (Commands to repeat)**

Where:

Variable: The layout of variable is different for '**for loop**', i.e. **%%[character]**
e.g. **%%a, %%b, %%A, %%B** etc.

Notice that, There is difference between **%%a** and **%%A** in case of **for loops**...

Set: It is not **set command** here... it represents the set of words... for repeating **for loop**.

Note: Current path is the same thing as **working directory**, and also notice that, if you are running **for loop** directly in Cmd console then use Single '**%**' sign and use '**%%**' before variable character if you are running **for loop** via a batch file, because while executing a batch file the console ignores the first '**%**' sign.

Like other Cmd commands, you can also Use some extra parameters with **for loop** and Change its effect in program. **For loop** can take following 4 parameters...

[Examples are after the complete description of all parameters of **for loop**...]

/D - Performs the Specified commands for each folder present in current Path.

Syntax: *for /d [variable] in (Searching filter) do (commands to repeat)*

Try in Cmd:

Try in Batch File:

Same Output as:

```
for /d %a in (*) do (echo %a)
for /d %%a in (*) do (echo %%a)
Dir /b /a:d *
```

/R - Performs the Specified task for each folder and Sub Folder present in a Specified Path. (If path is not specified then it takes current path.)

Syntax: *for /r [Path] [variable] in (Searching filter) do (commands to repeat)*

Try in Cmd:

Try in Batch File:

Same Output as:

```
for /r %a in (*) do (echo %a)
for /d %%a in (*) do (echo %%a)
Dir /s /b /a:d *
```

/L - Allows to repeat a module over the Range of specified Numbers...

Syntax: *for /l [variable] in (Start,Step,end) do (commands to repeat)*

Try in Cmd:

Try in Batch File:

Output:

```
for /l %a in (1,1,10) do (echo %a)
for /l %%a in (1,1,10) do (echo %%a)
prints 1 to 10 on console...
```

/F - Repeat specified commands through a wide variety of files, command and Strings.

Syntax: *for /f "options" [variable] in ("Set") do (commands to repeat)*

Syntax: *for /f "options" [variable] in ('command') do (commands to repeat)*

Syntax: *for /f "options" [variable] in (file-set) do (commands to repeat)*

Look in examples for more info.

Note: For a "set" ...use double quotes around the whole Set.

And for a 'command' use single quote around it...

Don't use anything to specify a filename.

Note: File name should not contain spaces; it should be only single string without carrying any space...otherwise **for loop** will not execute successfully... **If any case** filename contains spaces and you need to apply **for loop**, then you need to use 'usebackq' option in the "options" place and then you can use Double quotes around the name of the file, and **for loop** will not consider it as a **Set**, but understands that it is name of file containing spaces.

I know that it seems very uncomfortable when you read above technical description of **for loop** and its parameters for the first time.

And as I'm always saying, An Example is better than a 1000 lines of Description ... so let's take a look at my Favorite section: (and maybe yours too)

Time for Example:

This section is divided into following sub-sections...let's have a look...

Sub-Index:

1. *For simple **for loop** (without any parameter) ...*
 2. ***For loop** with **/d** parameter...*
 3. ***For loop** with **/r** parameter...*
 4. ***For loop** with **/l** parameter...*
 5. ***For loop** with **/f** parameter...*
-

1. For simple **for loop** (without any parameter) ...

WAP to run multiple commands in single Go. (Pressing Enter once in Cmd)

@echo off

Cls

Rem This is just a comment...we are using 'Rem' command here...

Rem we talked about it in part-II.

Rem we are creating a folder named 'foldername' and Pausing Console

Rem for user's keystroke...then deleting the folder. ***Only in one go.***

:: this is another method of declaring a comment...just use double colon

:: before your comment.

:: changing directory to your desktop ...so, it will be easier to find changes...

Cd desktop

:: we are declaring names of commands as a **set** to **for loop**

:: thus **for loop** will run for each element of the **set**.

For %%a in (**md pause rd**) do (%%a foldername)

Echo All commands applied successfully...

Pause

Exit

Rem The loop repeats 3 times as there are 3 elements in the specified **set**, and

Rem for each repetition of **for loop**, '%%a' (in brackets after **do** parameter) will take

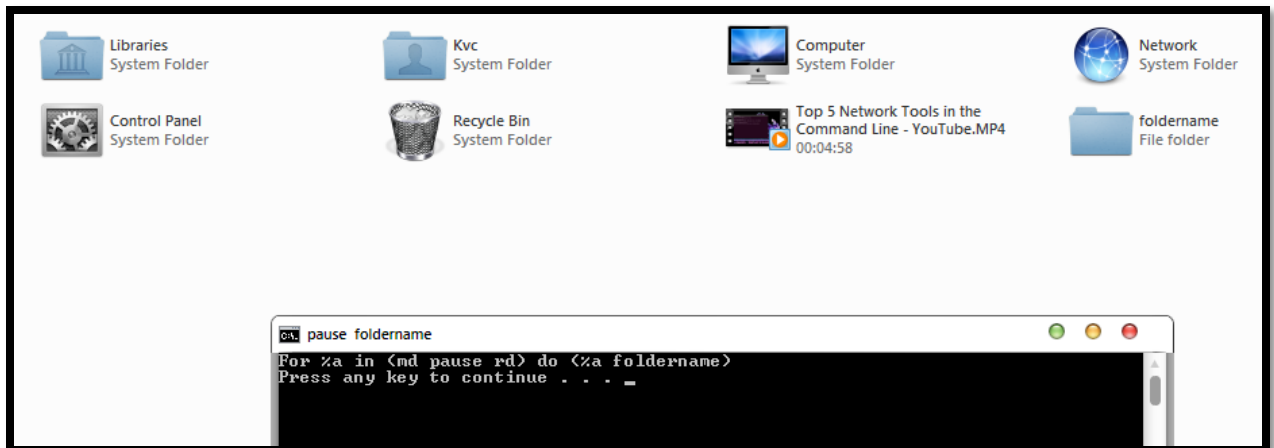
Rem firstly **md** then **pause** then **rd** as its value...thus we can run all 3 commands in single go.

Output:

Before Running Command



After running, but before pressing any key...



After running and pressing a key...



2. For loop with /d parameter...

WAP to Show all folders starting from letter 'p' in your C drive.

```
@echo off
```

```
Cls
```

```
Rem The default Working directory is your userprofile path...that is defined in  
Rem environmental variable, that maybe in your C drive...
```

```
Rem but if you are in C drive and you are trying to change directory to the root of  
Rem C drive by using simple 'Cd' command then it seems that cmd didn't execute  
Rem the command...so use the Following trick...to get required result...
```

```
:: instead of using cd /d C:, use cd /
```

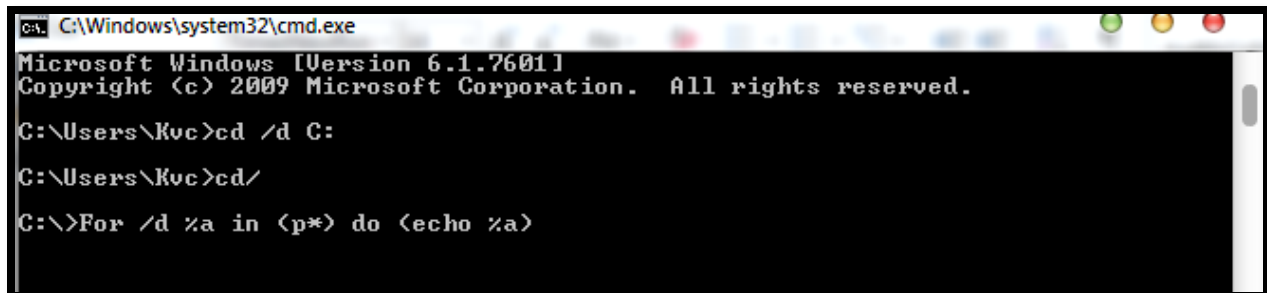
```
Cd/
```

```
For /d %%a in (p*) do (echo %%a)
```

```
Pause
```

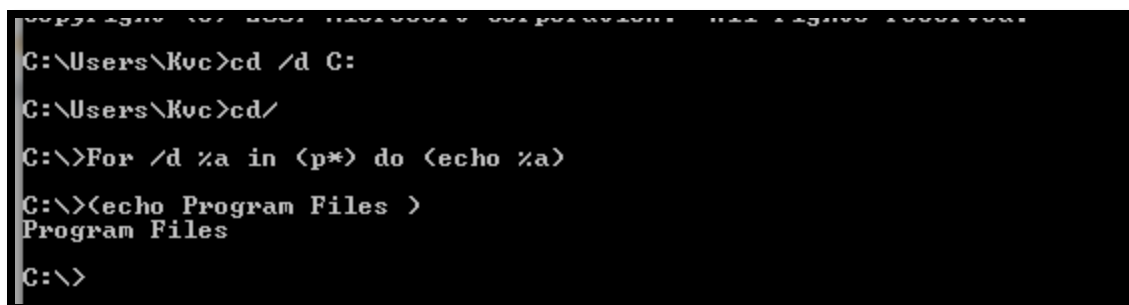
```
Exit
```

Output:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Kvc>cd /d C:
C:\Users\Kvc>cd/
C:\>For /d %a in (p*) do (echo %a)
```



```
C:\Users\Kvc>cd /d C:
C:\Users\Kvc>cd/
C:\>For /d %a in (p*) do (echo %a)
C:\>(echo Program Files )
Program Files
C:\>
```

Only one folder starting from letter 'p' is there.... i.e. program files...

3. For loop with /r parameter...

WAP to Show all folders and Sub-folders starting from letter 'w' in your C drive.

```
@echo off
Cls
Rem Simple and easy to understand... 😊
Cd/
```

```
For /r %%a in (w*) do (echo %%a)
Pause
Exit
```

Output:

A lot of mess on the cmd console...and as cmd has to go through whole directory tree...
So, it consumes some time too.

4. For loop with /l parameter...

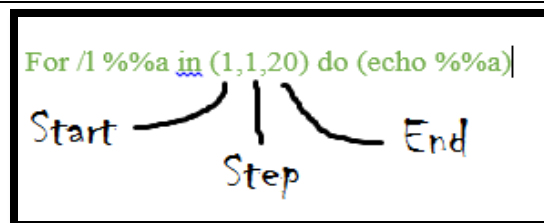
WAP to Show First 20 natural numbers...i.e. From 1 to 20 count.

```
@echo off
Cls
Rem As we know that Natural numbers starts from 1 and go up to infinity...
Rem by the difference of 1 in each number...i.e. the increment of 1 in each
Rem successive for loop
Rem but we only have to show first 20 numbers...as per our requirement...

Rem so, let's have a look at the simplest for loop syntax. That can count up to 20.
```

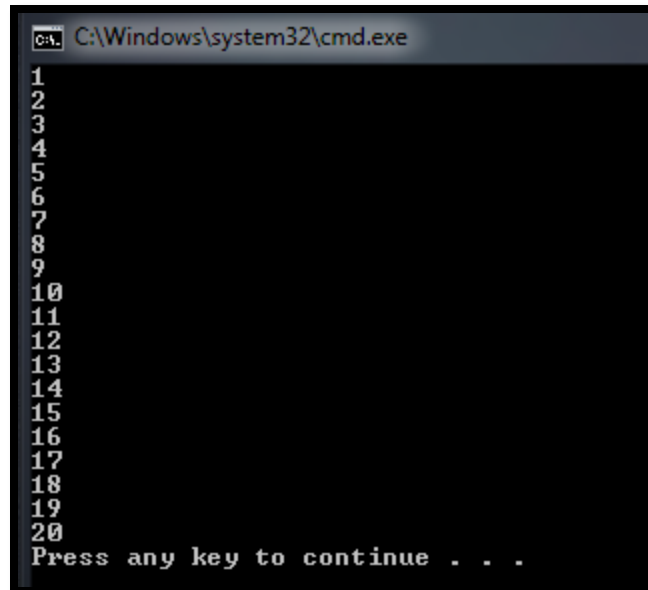
```
For /l %%a in (1,1,20) do (echo %%a)
```

```
Pause
Exit
```



This pic. May explain more...

Output:



```
C:\Windows\system32\cmd.exe
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Press any key to continue . . .
```

So Now if you understood the concept of **Start, step, end** in **/I** parameter of **for loop**, I can expect that you can easily make a batch program for printing even or odd numbers on console till 20.

Hint: Even numbers starts from **2** and have the **step** of '**2**'...in each succession...
And Odd numbers starts from **1** and they too have **step** of '**2**'... ☺
The end for both loops is obviously '**20**', as we want to print till 20...

5. For loop with **/f** parameter...

As **/f** parameter is an advanced tool for manipulating all other commands & Files...and it also takes special "**options**" to modify output of other commands/files. So, it is very difficult for me to show all possible ways of using those **options** & other manipulations in a single example. So, just showing basic example in this part...otherwise this part of book will become lengthier than 22 pages...

I'll write a separate part for the **for loop** and try to explain this topic more...about the **for loop**...

WAP to show all files present in working directory with their SIZE and File-TYPE.

@echo off

Cls

Rem As we know that Dir command will tell us about all the files and folders

Rem present in the working/current directory...but it don't tell us the file size and its type.

Rem and it also shows a lot of unwanted info. Like, Creation date, time etc.

Rem so, we need to re-format / re-arrange the output of the **DIR** command, for the

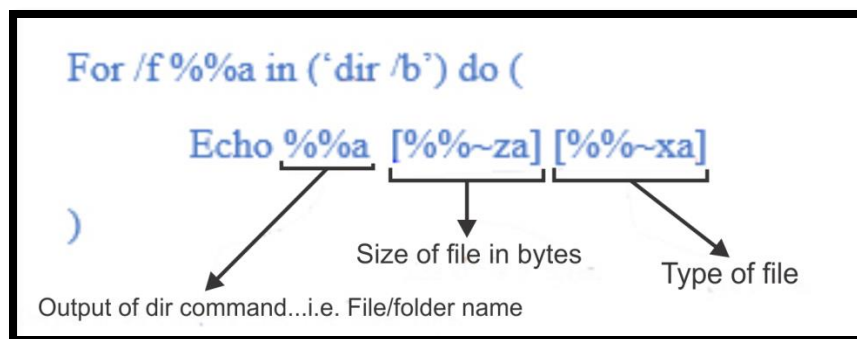
```
Rem fulfilment of our requirement...
:: Instead of running simply DIR command in console. We will firstly arrange its
:: output by using internal switches provided in the DIR command...i.e. /b /a:-d
:: try simply executing DIR command directly in cmd console...and then try DIR
:: with some switches like... DIR /b and DIR /b /a:d and DIR /b /a:-d.

Rem #Extra_info.#
:: DIR /b : it shows all files/folders [names only]...excluding hidden files/folders.
:: DIR /b /a:d : it shows all folders [names only]... including hidden folders.
:: DIR /b /a:d-h : it shows all folders [names only]... excluding hidden folders.
:: DIR /b /a:-d : it shows all files [names only]... including hidden files.
:: DIR /b /a:-d-h : it shows all files [names only]... excluding hidden files.

Rem Note: all info. On console is about files/folders present in working directory.
:: So, now after a lot of comments...let's have a look at the real code.

For /f %%a in ('dir /b') do (
    Echo %%a                [%%~za bytes]                [%%~xa]
)
Pause
Exit
```

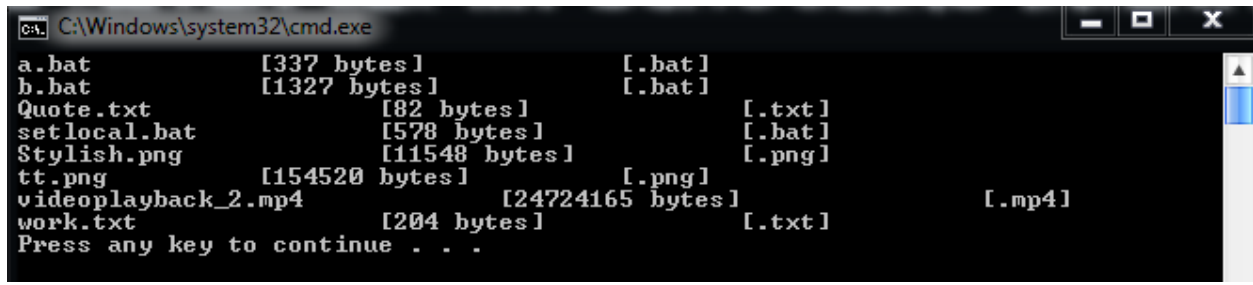
This pics. May help more ...



```
C:\Windows\system32\cmd.exe
dir /b
Contacts
Desktop
Documents
Downloads
Favorites
Links
Music
ntuser.dat
NTUSER.DAT.blues
ntuser.dat.iobit
NTUSER.DAT.iodefraq.bak
Pictures
Saved Games
Searches
Videos
```

Simple output of '**DIR /b**' ...

Output: [**Modifying DIR /b command with for loop**]



```
C:\Windows\system32\cmd.exe
a.bat           [337 bytes]           [.bat]
b.bat           [1327 bytes]          [.bat]
Quote.txt       [82 bytes]           [.txt]
setlocal.bat    [578 bytes]           [.bat]
Stylish.png     [11548 bytes]          [.png]
tt.png          [154520 bytes]         [.png]
videoplayback_2.mp4 [24724165 bytes]       [.mp4]
work.txt        [204 bytes]          [.txt]
Press any key to continue . . .
```

Goto command as loop

As already told in 1st part of this book that... this command changes/Breaks the direction of flow of cmd code compilation, which is originally by default from **top to bottom**. GOTO command only takes 1 parameter... i.e. name of the label...

So, we can use this property of **GOTO Command** to change the flow of compilation to the top of the Batch program...so that when program executes all commands of the batch program, Then the flow of compilation again goes to the top of batch program and it keeps on repeating...an example maybe a better explanation...

WAP to repeat a number of batch commands several times.

```
@echo off
Rem creating a label so, that we can change control to this label again and again...
Rem name of the label can be anything but it shouldn't contain space in its name...
Rem instead of spaces, you can use '_' (underscore) in the name...
it_is_label
cls
Echo.
Echo %date%; %time%
Timeout /t 1 >nul
Goto it_is_label
```

As the loop repeats infinitely...without any termination statement/condition...so, no need of **Exit** statement here...as the control of the code compilation will never be change to the **EXIT** statement...

Keep an eye on the time...as after every 1 second...the text at cmd console gets update...you can call it as your first **Timer program...** (Sort of)

But you can use various conditioning statements inside the repeating statement block...so that the repetition of the statements could be terminated at meeting several condition...

No while loop in cmd

[Reason and Creation.]

Frankly speaking {oh sorry...frankly writing} 😊 ☹️, I don't know about the real reason behind not having **while loop** in cmd /Batch programming... but as I know that **while loop** is nothing but the more logical version of the simple **Goto loop**...and we can also create a kind of **while loop** in batch only by using **Goto & If** command together. As while loop checks the validness of the condition for each iteration (repetition) of the specified block of the statements.

We can also check the condition for each repetition...if the condition is valid only then the next commands will execute otherwise the flow of compilation is changed to some other label... which may lead to end the program...

Let's understand the junk written above by a simple program...

I'm gonna modify the same program that you just read...so that you can easily understand the concept...

WAP to repeat a number of batch commands several times.

```
@echo off
```

```
Rem creating a 'counter' variable...to create a condition like...repeating the loop  
Rem until counter is less than 5 etc...i.e. loop will execute only 5 times and then  
Rem the program exits...
```

```
Set counter=0
```

```
:it_is_label
```

```
Rem checking for the valid condition...
```

```
If %counter% gtr 5 (pause & EXIT)
```

```
cls
```

```
Echo.
```

```
Echo %counter%. %date%; %time%
```

```
Timeout /t 1 >nul
```

```
:: Here...we need to change the value of variable 'counter' also, otherwise the  
:: program again behaves like the infinite loop...'cuz the IF statement's  
:: condition will never be true...if we don't change value of counter...
```

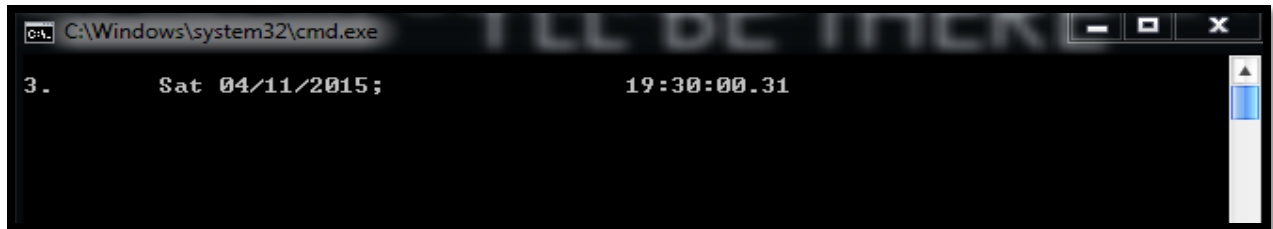
```
Set /a counter=%counter%+1
```

```
:: Or the other syntax 'Set /a counter+=1'
```

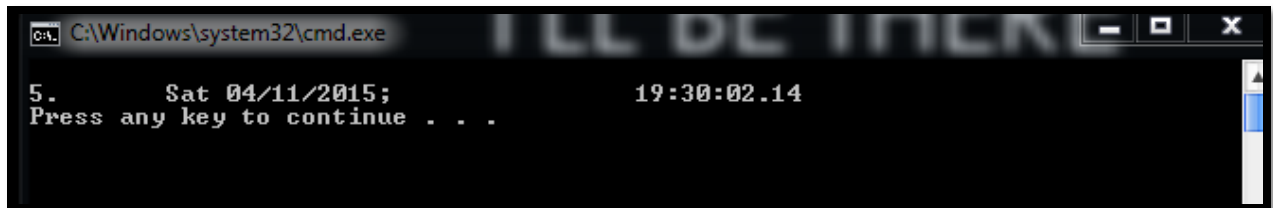
```
Goto it_is_label
```

Note: Two labels in any batch program can't have same names...as when you call any one of them...cmd console crashes down 'cuz cmd doesn't know to which label it has to change control to.

Output:



```
C:\Windows\system32\cmd.exe
3.      Sat 04/11/2015;      19:30:00.31
```



```
C:\Windows\system32\cmd.exe
5.      Sat 04/11/2015;      19:30:02.14
Press any key to continue . . .
```

Special operators in batch

There are some Special operators in the batch programming...and once you understand the nature & working of these operators in batch programming...then it will be another step on the ladder of learning Batch programming. So, without wasting any time by my foolish thoughts... let's have a look to some of them ...

I'm again going to make a table to make it easier to understand...

Operator	Working
>	<p>Causes the command's output to be sent to the specified device or file. If specified file exists...it replaces existing file having same name or creates new.</p> <div><p>E.g. Echo This is a line.>"test.txt"</p><p>A line "This is a line" is send to the file...named as "test.txt"</p></div>
>>	<p>Causes the command's output to be appended to a device or a file. It appends the output to the end of existing file or creates new file.</p> <div><p>E.g. Echo This is a line. >>"test.txt"</p><p>A line "This is a line" is send to the file...named as "test.txt"</p></div>
<	<p>Causes file to be fed to the program as input, or simply reads the command input from a file instead of reading form the keyboard.</p>

E.g. `Set /p var= <"test.txt"`

The variable 'var' has the value as the text in the 1st line of the specified file...

&

Allows to execute multiple commands in single line. Just put '&' sign between two commands and they will execute as they are in two different lines... i.e. One after the other.

E.g. `Dir /b & echo Executed Command.`

Firstly 'DIR /b' command will execute then 'Echo' command.

&&

Same working as the single '&' operator... but in this case the next command after the '&&' operator will execute only if the command before '&&' operator executed successfully... but in single '&' operator it doesn't bother about successful execution of the previous command.

E.g. `Dir /b && echo Executed Command.`

Firstly 'DIR/b' command will execute, if successful then 'Echo' command.

|

Reads the output from command written before '|' operator & writes it to the input to the command written after '|' symbol.

E.g. `Dir | find "<DIR>"`

Only those lines are printed on cmd console which contains the string "<DIR>"...the find command filter-out all other lines...to get the reversed output just put /V switch of find command.

You can get the same output by using switches of DIR command...Like

`Dir /a:d-h`

||

Used to run the second command only if the first command causes an error...you can say that it is logically opposite of the '&&' operator.

E.g. `Dir /b xx && echo Match Found!! || echo No Match Found!!`

The DIR command tries to find a file/folder named as 'xx' in the working directory...if found...It will execute the command after '&&' operator otherwise it will execute command after '||' operator.

>&

Writes output from one handle to the input of another handle...it is an advance operator and you'll come to know about its working...this time just keep it in mind that it is also there in batch

programming... so as you see any other batch file... which contains this operator... then you can understand that what it is doing in that batch file.

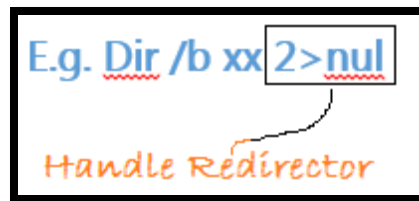
[Read about handles in next section...]

<&

Reads the input from one handle and writes it to the output of Another Handle.

This time no appropriate example for these handles redirectors, is coming to my mind... so, maybe I'll use them in any of my examples in next parts or maybe in this part...

Note: If I want to display a **file / folder** having name 'xx' in the working directory... then I only have to type 'DIR /b xx', but if there is no file having name 'xx' then it shows "File Not Found", but we only want it to show either the **file / folder** with name 'xx' or show nothing on the console... i.e. we don't want to display any error on the console... then it can be possible by using these advanced HANDLE redirectors only...



E.g. Dir /b xx 2>nul

Now it will only show the result, no error... as errors are dumped into the 'nul' [which is nothing] ...

2 is a Handle here... which stands for STDERR {standard error} ... that's why it will not show anything on cmd console if any error occurs... 'cuz all error outputs are dumped into the 'nul' ... but if there is no error then it shows result...

Here's the list of Possible HANDLES:

HANDLE	Numeric Equivalent Value	[Meaning]
STDIN	0	[Keyboard input]
STDOUT	1	[Output of command]
STDERR	2	[Error output of command]

Now try the following commands in cmd console... and you'll automatically know the difference...

- | | |
|--------------------|--|
| 1. Dir /b xx 2>nul | [No error display, only result...if any] |
| 2. Dir /b * 2>nul | [Only output result, No Error display] |
| 3. Dir /b xx 1>nul | [No Result display, Only error...if any] |
| 4. Dir /b * 1>nul | [Only output error, No result display] |

Special Parameters in Batch

After reading about 'Special Operators', let's talk about 'special parameters' in the batch programming. The knowledge of these special parameters will help you in making your own custom commands or custom functions, and you can easily pass variables from one batch program / function to another batch program / function.

For learning about special parameters, you must know that every batch program that you are creating is actually a kind of function, which may or mayn't take parameters. All programs that we've created till now, were all functions that don't take any parameter. But in case, if you need to pass a value to the function and get corresponding output according to the output, then you need to use special parameters.

There are basically only **10 Special parameters** in batch programming, which are %0, %1, %2, %3, ..., %9. Here, %0 is predefined, and it contains the name of the function (or in other words, %0 is the name of the batch file, from which it is called). The parameters can be separated by spaces, commas etc. An Example may be better Explanation:

Create a Batch file with the following code:

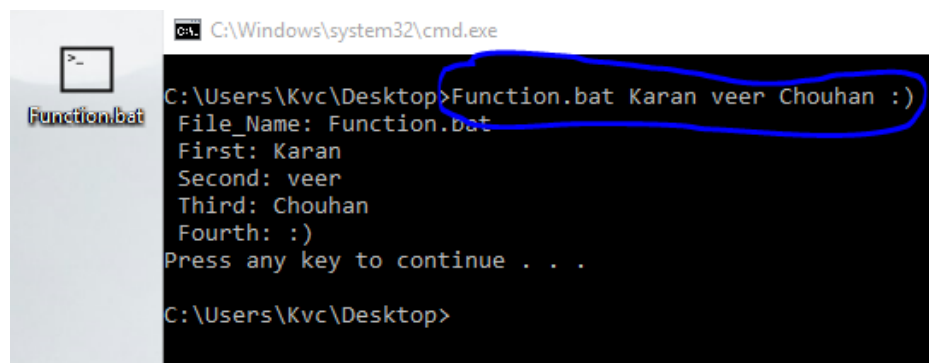
WAP to Show the Usage of Special Parameters in Batch programming.

```
@echo off
REM Displaying All parameters passed to this File, one by one...

Echo. File_Name: %0
Echo. First: %1
Echo. Second: %2
Echo. Third: %3
Echo. Fourth: %4

REM And so on...till %9...
Pause
Exit /b
```

Now, save it by name **Function.bat** At your Desktop, Now Hold **Shift** and Right click anywhere on empty space on your desktop, Click the '[Open command window here](#)' option... Now, just Type in the following text as shown in the image:



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The prompt is at 'C:\Users\Kvc\Desktop>'. The user has entered 'Function.bat Karan veer Chouhan :)' and pressed Enter. The output shows the batch file displaying the arguments: 'File_Name: Function.bat', 'First: Karan', 'Second: veer', 'Third: Chouhan', and 'Fourth: :)'. The prompt then says 'Press any key to continue . . .' and the user has pressed a key, returning the prompt to 'C:\Users\Kvc\Desktop>'. A blue oval highlights the command and its output.

```
C:\Windows\system32\cmd.exe
C:\Users\Kvc\Desktop>Function.bat Karan veer Chouhan :)
File_Name: Function.bat
First: Karan
Second: veer
Third: Chouhan
Fourth: :)
Press any key to continue . . .
C:\Users\Kvc\Desktop>
```


Now, as you've seen that, in the main code, we've not assign any value to the variables (**%0, %1...etc.**), but still on the Runtime we can assign some values which they will attain automatically... Now try the same activity, but this time with the different parameter values... Then you'll know the difference. If you have learned this topic successfully, then you are very near to be an advanced batch programmer.

The Shift Command

Shift command is very small but very much important and effective command in batch programming. As you already know that, there is a limit to the number of parameters that you can pass to a particular function because there are only 10 parameter values are available and **you can't pass more than 9** (**%0** is already defined, and you can't use this parameter's place).

But with the **Shift** Command, you can pass any number of parameters as you want. Which further means that, now you have the power to pass infinite number of parameters to a function, with almost no limitation.

What you need to do is that after using value of a parameter just Shift it with the next parameter and the value of next parameter will be saved into the same parameter number. Let's analyze an Example:

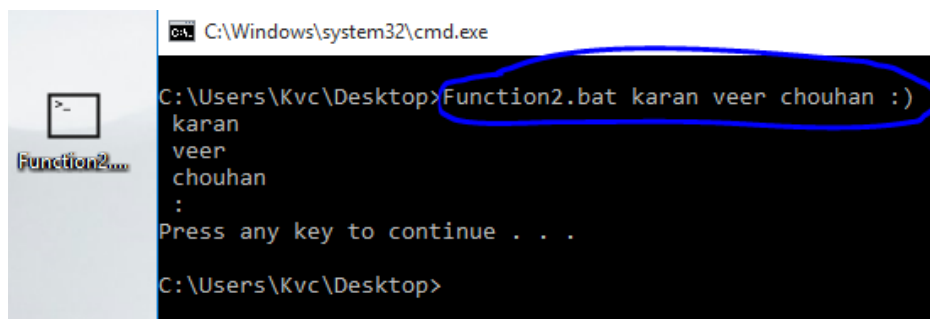
WAP to Show the Usage of Shift Command.

```
@echo off
REM Using only '%1' to display all parameters passed to the function...
REM Checking if '%1' is defined or exit the program...

:Top
If "%1" "NEQ" "" (
Echo. %1
Shift /1
Goto Top
)

Pause
Exit /b
```

Save the above code as Function2.bat on your Desktop, Now again open the cmd at desktop by holding shift and right clicking. Now just type the Same thing as previous example, and you'll get the similar output.



```
C:\Windows\system32\cmd.exe

C:\Users\Kvc\Desktop>Function2.bat karan veer chouhan :)
karan
veer
chouhan
:
Press any key to continue . . .

C:\Users\Kvc\Desktop>
```

As you can see that, it has the same output. Leave the problem with brackets as function is only for making you understand about the concept. Which I think It has done successfully.

File Handling in Batch

File handling is the desired operation that any user wants to do for handling its data correctly or managing a Database in files for the program. So, the knowledge of this topic will make you a better programmer in batch programming. So, let's talk about it.

Creating a file:

You can create a file via batch using the '>' parameter, either once or twice.

E.g.: If I want to save my name in a text file, then I have to write the following code:

```
Echo. Karan veer Chouhan > MyName.txt  
Echo. Karan veer Chouhan >> MyName.txt
```

Here, '>' symbol is an over-write expression, and '>>' symbol is an Addition expression. If you'll use single '>', then cmd will not care if any pre-existing file named "MyName.txt" exist or not, it will simply delete the previous data and overwrite the new data in the file.

While using Double '>>' symbol will append the data to the previous one, in the case if file is already present.

Modifying a file:

As from the above description of the '>' and '>>' symbol, you almost get an idea of how to modify the data in a file. You just need to use single '>' operator to do that. For any advance modification you need to combine **for loop** with the redirection operators. This you'll learn in next parts of this book.

Deleting a file:

Simply use the **Del** command to delete the existing files. Try 'Del /?' in cmd to get any help on it.

Operators of File Handling (in Details)

Now, you know the basics of creating and modifying a file, and we'll do a little depth analysis of the redirection operators in this section. The basic is to know that, you need to write the name of the file after redirection operator, and the path maybe either full path of the file or it may be relative path of file.

The '>' operator:

For example, if you want to save a file containing your Username on your desktop, then you may try either of the following syntax for this purpose.

```
cho. %username% > "File.txt"  
OR  
Echo. %username% > "C:\users\%username%\Desktop\File.txt"  
OR  
Echo. %Username% > "%userprofile%\Desktop\file.txt"
```

Either of the above line will have the same output, but for the first line to be successful, your working directory should be 'C:\users\[Your Username]\Desktop', otherwise the file.txt will be saved into some other location.

In case when the output of the command is multi-lined and you want to save it to a file, just use redirection operator to redirect the command's output to that file. Try the following in cmd:

```
Systeminfo > "System Details.txt"
```

The '<' Operator:

This operator has working opposite to the '>' operator, and it is used to get data from file into the variable. Try the following:

```
Set /p MyUsername=<"%userprofile%\Desktop\file.txt"
```

Now, the first line of the "file.txt" at your desktop will be saved into the variable named 'MyUsername'.

Functions in Batch

Here comes the topic of this part, which has taken 19 pages of text to reach to it. Now I'm in space problem, as I've just got only 2 pages' space now. This is because I have to complete this part in less than 22 pages. 'cuz it was my initial goal to not to exceed any part of this book more than 20 or 22 pages. So, without wasting more space, let's talk about this awesome topic.

We create functions, in order to make our work easier and less bulky. And batch also has the scope of creating functions and using them later on in any of your other batch program. As now you perfectly know about the usage of **Special parameters** in batch. That knowledge is enough to start making your own custom functions.

The **Example** you did in **Special parameters** section, was also a kind of function. So, let's start officially by doing a simple and effective example.

WAP to Display the square of a given number.

```
@echo off
```

```
REM As Square is just the multiplication of same number two times...
```

```
REM Considering that number will be passed into the 1st parameter...
```

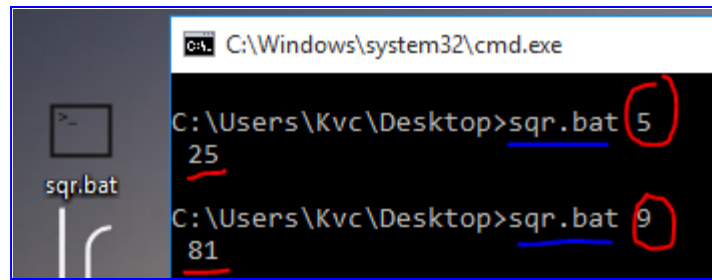
```
REM So, lets start...
```

```
Set /a Square=%1 * %1
```

```
Echo. %Square%
```

```
Exit /b
```

Just do as the same you did while using the functions created while learning the examples of the **Special parameters** in batch. Now, you just created your first official function. Save it as any name you want, I saved it as 'sqr.bat'. and here's the output on the console window:



```
C:\Windows\system32\cmd.exe
C:\Users\Kvc\Desktop>sqr.bat 5
25
C:\Users\Kvc\Desktop>sqr.bat 9
81
```

String Manipulations in Batch

Finally, the last topic of this part, and still got 1 extra page for the completion of this part. How lucky and clever I am!! Ha-ha, I know I'm talking non-sense, but ... leave it. Let's start with the topic.

Strings are the very basic and important part of any language. In order to have control over a language and its features, we must learn the methods to manipulate the string parts, then we can say that we know that language, if and only if we know the proper method to manipulate the language. So, let's learn the methods of string manipulation in batch programming.

We've Got two basic syntax for it:

1. With ~ symbol (I forgot its name... :D)
2. Without ~ symbol.

The 1st type of syntax is used to **get the sub-string** out of the main string, while the 2nd method is used to **replace a sub-string** in the main string.

For example, if I have a variable containing my name as a string in it. Now, How can I manipulate my name to get the required string?? Here's the Example:

WAP to Replace all 'a' with '-' in the string entered by User.

@echo off

REM Getting the string from user... and replacing all 'a' with '-' (Dashes)...

Set /p String=Enter Text:

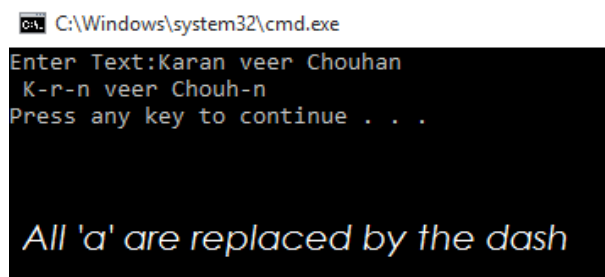
REM over-writing the same variable for reducing mess...

Set String=%String:a=-%

Echo. %String%

Pause

Exit /b



```
C:\Windows\system32\cmd.exe
Enter Text:Karan veer Chouhan
K-r-n veer Chouh-n
Press any key to continue . . .

All 'a' are replaced by the dash
```

Similarly, you can replace the whole sub-string with another sub-string. Now, let's see an example of manipulation with ~ operator.

If I want to extract first 4 characters of the string, I'll do it with using ~ operator in manipulation method.

WAP to get first 4 characters, last 2 characters and 2 characters leaving 3 characters' form start of a string.

@echo off

REM Getting the string from user... and replacing all 'a' with '-' (Dashes)...

Set /p String=Enter Text:

REM Printing all of them as per question's requirement...

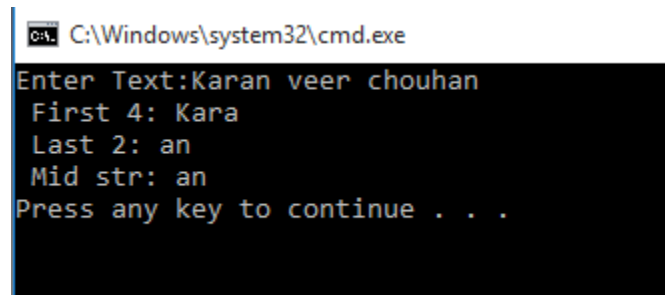
Echo. First 4: %String:~0,4%

Echo. Last 2: %String:~-2%

Echo. Mid str: %String:~3,2%

Pause

Exit /b



```
C:\Windows\system32\cmd.exe
Enter Text:Karan veer chouhan
First 4: Kara
Last 2: an
Mid str: an
Press any key to continue . . .
```

After giving this example, I learned a new thing that my name contains a lot of 'an'. In the above Syntax, If I have to write its general syntax, then it may look like:

%String:~X,Y%:	where X= The starting character of the extraction Y= The number of characters to extract
%String:~Z% :	where Z= The number of characters to extract from one end of the string, if -ve, then from last. Else from starting of string.
%String:X=Y%:	where X= The sub-string to replace, can be a character also. Y= The sub-string to replace with, can be empty also.

Ending of this part

Finally, this part ends. This part has taken very long to complete and while completing it, I had faced many tragedies. First my laptop got damaged and then it became completely a scrap. Then I got this new laptop after a long period of 6 months. More over my results made me more serious about my engineering, as from last 6 months I also have interest in chess and I spend often 2-3 hours daily on chess. So, I can give very little time to my programming life. That's why this part is so much delayed. But I tried my best to keep my blog updated by posting new and innovative programs on it.

In this part, I've separated the programs by putting them in different boxes to highlight them separately. And I've used my developed editing skills in this part, which I've learn from the experience from writhing posts on the blog and working on various document projects on industrial level while my training period as an engineer. I hope you'll like it.

At last, I just hope that you'll like my work. Please do motivate me to write more books on batch programming. As I came to completion of this part this month only because of my friend 'Pankaj'. Whenever I met him, he had one question for me, "**When your book gonna complete??**" And his continues strike made me complete it within 1 month of buying a new laptop. Thanks man!! I really appreciate your contribution.

You can also do the same, either make a comment on my blog:

<https://batchprogrammers.blogspot.com>

Or send me a mail about your views on this book, and on this part:

karanveerchouhan@gmail.com

I'll really appreciate your contribution. Thanks for giving your time for reading this stupid section.



#TheBATeam

#TheKvc