Windows Batch File

MayBeMayBe's…

Using batch files

With batch files, which are also called batch programs or scripts, you can simplify routine or repetitive tasks. A batch file is an unformatted text file that contains one or more commands and has a .bat or .cmd file name extension. When you type the file name at the command prompt, Cmd.exe runs the commands sequentially as they appear in the file.

You can include any command in a batch file. Certain commands, such as for, goto, and if, enable you to do conditional processing of the commands in the batch file. For example, the if command carries out a command based on the results of a condition. Other commands allow you to control input and output and call other batch files.

The standard error codes that most applications return are 0 if no error occurred and 1 (or higher value) if an error occurred. Please refer to your application help documentation to determine the meaning of specific error codes.

For more information about batch file operations, see the following topics:

- Using batch parameters
- Using filters
- Using command redirection operators

For more information about commands that you can use in batch files, click a command:

- Call
- Echo
- Endlocal
- For
- Goto
- If

- Pause
- Rem
- Setlocal
- Shift

1. Using batch parameters

You can use batch parameters anywhere within a batch file to extract information about your environment settings.

Cmd.exe provides the batch parameter expansion variables %0 through %9. When you use batch parameters in a batch file, %0 is replaced by the batch file name, and %1 through %9 are replaced by the corresponding arguments that you type at the command line. To access arguments beyond %9, you need to use the shift command. For more information about the shift command, see Shift The %* batch parameter is a wildcard reference to all the arguments, not including %0, that are passed to the batch file.

For example, to copy the contents from Folder1 to Folder2, where %1 is replaced by the value Folder1 and %2 is replaced by the value Folder2, type the following in a batch file called **Mybatch.bat:**

xcopy %1₩*.* %2

To run the file, type:

mybatch.bat C:₩folder1 D:₩folder2

This has the same effect as typing the following in the batch file:

xcopy C:₩folder1 ₩*.* D:₩folder2

You can also use modifiers with batch parameters. Modifiers use current drive and directory information to expand the batch parameter as a partial or complete file or directory name. To use a modifier, type the percent (%) character followed by a tilde (~) character, and then type the appropriate modifier (that is, %~modifier).

The following table lists the modifiers you can use in expansion.

Modifier	Description
%~1	Expands %1 and removes any surrounding quotation marks ("").
%~f1	Expands %1 to a fully qualified path name.
%~d1	Expands %1 to a drive letter.
%~p1	Expands %1 to a path.
%~n1	Expands %1 to a file name.
%~x1	Expands %1 to a file extension.

Modifier	Description
%~s1	Expanded path contains short names only.
%~a1	Expands %1 to file attributes.
%~t1	Expands %1 to date and time of file.
%~z1	Expands %1 to size of file.
%~$PATH:1	Searches the directories listed in the PATH environment variable and expands %1 to the fully qualified name of the first one found. If the environment variable name is not defined or the file is not found, this modifier expands to the empty string.

The following table lists possible combinations of modifiers and qualifiers that you can use to get compound results.

Modifier	Description
%~dp1	Expands %1 to a drive letter and path.
%~nx1	Expands %1 to a file name and extension.
%~dp$PATH:1	Searches the directories listed in the PATH environment variable for %1 and expands to the drive letter and path of the first one found.
%~ftza1	Expands %1 to a dir-like output line.

Note

- In the previous examples, you can replace %1 and PATH with other batch parameter values.

The %* modifier is a unique modifier that represents all arguments passed in a batch file. You cannot use this modifier in combination with the %~ modifier. The %~ syntax must be terminated by a valid argument value.

You cannot manipulate batch parameters in the same manner that you can manipulate environment variables. You cannot search and replace values or examine substrings. However, you can assign the parameter to an environment variable, and then manipulate the environment variable.

2. Using filters

Used in conjunction with the command redirection pipe character (|), a command filter is a command within a command that reads the command's input, transforms the input, and then writes the output. Filter commands help you sort, view, and select parts of a command output.
Filter commands divide, rearrange, or extract portions of the information that passes through them. The following table lists filter commands that are available in Windows XP.

Command	Description
more	Displays the contents of a file or the output of a command in one Command Prompt window at a time.
find	Searches through files and command output for the characters you specify.
sort	Alphabetizes files and command output.

To send input from a file to a filter command, use a less than sign (<). If you want the filter command to get input from another command, use a pipe (|).

2.1 Using the more command

The **more** command displays the contents of a file or the output of a command in one Command Prompt window at a time. For example, to display the contents of a file called List.txt in one Command Prompt window at a time, type:

more < list.txt

One Command Prompt window of information appears, and then the -- More -- prompt appears at the bottom of the Command Prompt window. To continue to the next Command Prompt window, press any key on the keyboard except PAUSE. To stop the command without viewing more information, press CTRL+C.
You can use the **more** command when you work with a command that produces more than one Command Prompt window of output. For example, suppose you want to view a directory tree on your hard disk. If you have more directories than can be displayed in the Command Prompt window, you can use the **tree** command with a pipe (|) and the **more** command as follows:

tree c:₩ | more

The first Command Prompt window of output from the **tree** command appears, followed by the -- More -- prompt. Output pauses until you press any key on the keyboard, except PAUSE.

2.2 Using the find command

The **find** command searches files for the string or text that you specify. Cmd.exe displays every line that matches the string or text that you specify in the Command Prompt window. You can use the **find** command either as a filter command or a standard Windows XP command. For more information about using **find** as a standard command, see Find

To use **find** as a filter command, you must include a less than sign (<) and the string or text on which you want to search. By default, **find** searches are case-sensitive. For example, the following command finds occurrences of the string "Pacific Rim" in the file Trade.txt:

find "Pacific Rim" < trade.txt

The output does not include any occurrences of "pacific rim." It includes occurrences of the capitalized "Pacific Rim" only.

To save the output of the **find** command rather than display it in the Command Prompt window, type a greater than sign (>) and the name of the file where you want to store the output. For example, the following command finds occurrences of "Pacific Rim" in the Trade.txt file and saves them in Nwtrade.txt:

find "Pacific Rim" < trade.txt > nwtrade.txt

2.3 Using the sort command

The **sort** command alphabetizes a text file or the output of a command. For example, the following command sorts the contents of a file named List.txt and displays the results in the Command Prompt window:

sort < list.txt

In this example, the **sort** command sorts the lines of the List.txt file into an alphabetical list and displays the results without changing the file. To save the output of the **sort** command rather than display it, type a greater than sign (>) and a file name. For example, the following command alphabetizes the lines of the List.txt file and stores the results in the Alphlist.txt file:

sort < list.txt > alphlist.txt

To sort the output of a command, type the command, type a pipe (|), and then type **sort** (that is, *command* **| sort**). For example, the following command sorts the lines that include the string "Jones" (that is, the **find** command output) in alphabetical order:

find "Jones" maillst.txt | sort

3. Using command redirection operators

You can use redirection operators to redirect command input and output streams from the default locations to different locations. The input or output stream location is referred to as a handle

The following table lists operators that you can use to redirect command input and output streams.

Redirection OP	Description
>	Writes the command output to a file or a device, such as a printer, instead of the Command Prompt window.
<	Reads the command input from a file, instead of reading input from the keyboard.
>>	Appends the command output to the end of a file without deleting the information that is already in the file.
>&	Writes the output from one handle to the input of another handle.
<&	Reads the input from one handle and writes it to the output of another handle.
|	Reads the output from one command and writes it to the input of another command. Also known as a pipe.

By default, you send the command input (that is, the STDIN handle) from your keyboard to Cmd.exe, and then Cmd.exe sends the command output (that is, the STDOUT handle) to the Command Prompt window.

The following table lists the available handles.

Handle	Numeric equivalent of handle	Description
STDIN	0	Keyboard input
STDOUT	1	Output to the Command Prompt window
STDERR	2	Error output to the Command Prompt window
UNDEFINED	3-9	These handles are defined individually by the application and are specific to each tool.

The numbers zero through nine (that is, 0-9) represent the first 10 handles. You can use Cmd.exe to run a program and redirect any of the first 10 handles for the program. To specify which handle you want to use, type the number of the handle before the redirection operator. If you do not define a handle, the default < redirection input operator is zero (0) and the default > redirection output operator is one (1). After you type the < or > operator, you must specify where you want to read or write the data. You can specify a file name or another existing handle.

To specify redirection to existing handles, use the ampersand (&) character followed by the handle number that you want to redirect (that is, **&***handle#*). For example, the following command redirects handle 2 (that is, STDERR) into handle 1 (that is, STDOUT):

1<&2

3.1 Duplicating handles

The & redirection operator duplicates output or input from one specified handle to another specified handle. For example, to send **dir** output to File.txt and send the error output to File.txt, type:

dir>c:₩file.txt 2>&1

When you duplicate a handle, you duplicate all characteristics of the original occurrence of the handle. For example, if a handle has write-only access, all duplicates of that handle have write-only access. You cannot duplicate a handle with read-only access into a handle with write-only access.

3.2 Redirecting command input (<)

To redirect command input from the keyboard to a file or device, use the < operator. For example, to get the command input for the **sort** command from File.txt:

sort<file.txt

The contents of File.txt appear in the Command Prompt window as an alphabetized list.

The < operator opens the specified file name with read-only access. As a result, you cannot write to the file when you use this operator. For example, if you start a program with <&2, all attempts to read handle 0 fail because handle 2 is initially opened with write-only access.

Note

- Zero is the default handle for the < redirection input operator.

3.3 Redirecting command output (>)

Almost all commands send output to your Command Prompt window. Even commands that send output to a drive or printer display messages and prompts in the Command Prompt

window.

To redirect command output from the Command Prompt window to a file or device, use the > operator. You can use this operator with most commands. For example, to redirect **dir** output to Dirlist.txt:

dir>dirlist.txt

If Dirlist.txt does not exist, Cmd.exe creates it. If Dirlist.txt exists, Cmd.exe replaces the information in the file with the output from the **dir** command.

To run the **netsh routing dump** command and then send the command output to Route.cfg, type:

netsh routing dump>c:₩route.cfg

The > operator opens the specified file with write-only access. As a result, you cannot read the file when you use this operator. For example, if you start a program with redirection >&0, all attempts to write handle 1 fail because handle 0 is initially opened with read-only access.

Note

- One is the default handle for the > redirection output operator.

3.4 Using the <& operator to redirect input and duplicate

To use the redirection input operator <&, the file you specify must already exist. If the input file exists, Cmd.exe opens it as read-only and sends the characters contained in the file as input to the command as if they were input from the keyboard. If you specify a handle, Cmd.exe duplicates the handle you specify onto the existing handle in the system.

For example, to open File.txt as input read to handle 0 (that is, STDIN), type:

<file.txt

To open File.txt, sort the contents and then send the output to the Command Prompt window (that is, STDOUT), type:

sort<file.txt

To find File.txt, and then redirect handle 1 (that is, STDOUT) and handle 2 (that is, STDERR) to the Search.txt, type:

findfile file.txt>search.txt 2<&1

To duplicate a user-defined handle 3 as input read to handle 0 (that is, STDIN), type:

<&3

3.5 Using the >& operator to redirect output and duplicate

If you redirect output to a file and you specify an existing file name, Cmd.exe opens the file as write-only and overwrites the file's contents. If you specify a handle, Cmd.exe duplicates the file onto the existing handle.

To duplicate a user-defined handle 3 into handle 1, type:

>&3

To redirect all of the output, including handle 2 (that is, STDERR), from the **ipconfig** command to handle 1 (that is, STDOUT), and then redirect the ouput to Output.log, type:

ipconfig.exe>>output.log 2>&1

3.6 Using the >> redirection operator to append output

To add the output from a command to the end of a file without losing any of the information already in the file, use two consecutive greater than signs (that is, >>). For example, the following command appends the directory list produced by the **dir** command to the Dirlist.txt file:

dir>>dirlist.txt

To append the output of the **netstat** command to the end of Tcpinfo.txt, type:

netstat>>tcpinfo.txt

3.7 Using the pipe operator (|)

The pipe operator (|) takes the output (by default, STDOUT) of one command and directs it into the input (by default, STDIN) of another command. For example, the following command sorts a directory:

dir | sort

In this example, both commands start simultaneously, but then the **sort** command pauses until it receives the **dir** command's output. The **sort** command uses the **dir** command's output as its input, and then sends its output to handle 1 (that is, STDOUT).

3.8 Combining commands with redirection operators

You can create custom commands by combining filter commands with other commands and file names. For example, you can use the following command to store the names of files that contain the string "LOG":

dir /b | find "LOG" > loglist.txt

The **dir** command's output is sent through the **find** filter command. File names that contain the string "LOG" are stored as a list of file names (for example, NetshConfig.log, Logdat.svd, and Mylog.bat) in the Loglist.txt file.

To use more than one filter in the same command, separate the filters with a pipe (|). For example, the following command searches every directory on drive C:, finds the file names that include the string "Log", and then displays them in one Command Prompt window at a time:

dir c:₩ /s /b | find "LOG" | more

By using a pipe (|), you direct Cmd.exe to send the **dir** command output through the **find** filter

command. The **find** command selects only file names that contain the string "LOG." The **more** command displays the file names that are selected by the **find** command, one Command Prompt window at a time. For more information about filter commands, see Using filters

4. Call

Calls one batch program from another without stopping the parent batch program. The **call** command accepts labels as the target of the call. **Call** has no effect at the command-line when used outside of a script or batch file.

Syntax

call [[Drive**:**][Path] FileName [BatchParameters]] [**:**label [arguments]]

Parameters

[*Drive***:][***Path***]** ***FileName*** **:** Specifies the location and name of the batch program you want to call. The *FileName* parameter must have a .bat or .cmd extension.

BatchParameters **:** Specifies any command-line information required by the batch program, including command-line options, file names, batch parameters (that is, %0 through %9), or variables (for example, *%baud%*).

:*label* **:** Specifies the label to which you want a batch program control to jump. By using **call** with this parameter, you create a new batch file context and pass control to the statement after the specified label. The first time the end of the batch file is encountered (that is, after jumping to the label), control returns to the statement after the **call** statement. The second time the end of the batch file is encountered, the batch script is exited. For a description of the **goto :eof** extension that allows you to return from a batch script, see Related Topics.

arguments **:** Specifies any command-line information that you pass to the new instance of the batch program that begins at **:***label*, including command-line options, file names, batch parameters (that is, *%1* through *%9*), or variables (for example, *%baud%*).

/? **:** Displays help at the command prompt.

Remarks

- Using batch parameters

  Batch parameters can contain any information that you can pass to a batch program, including command-line options, file names, batch parameters (that is, %0 through %9), and variables (for example, *%baud%*). For more information about batch parameters, see Related Topics.

- Using pipes and redirection symbols

  Do not use pipes and redirection symbols with **call**.

- Making a recursive call

  You can create a batch program that calls itself, however, you must provide an exit condition. Otherwise, the parent and child batch programs can loop endlessly.

- Working with command extensions

  With command extensions enabled (that is, the default), **call** accepts a *label* as the target of the call. The correct syntax is as follows:

  call :label arguments

  For more information about enabling and disabling command extensions, see **cmd** in Related Topics.

Examples

To run the Checknew.bat program from another batch program, type the following command in the parent batch program:

call checknew

If the parent batch program accepts two batch parameters and you want it to pass those parameters to Checknew.bat, use the following command in the parent batch program:

call checknew %1 %2

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output

5. Echo

Turns the command-echoing feature on or off, or displays a message. Used without parameters, **echo** displays the current echo setting.

Syntax

echo [{**on**|**off**}] [*message*]

Parameters

{on|off} : Specifies whether to turn the command-echoing feature on or off.

***message* :** Specifies text you want to display on the screen.

/? : Displays help at the command prompt.

Remarks

- The **echo** *message* command is useful when echo is turned off. To display a message that is several lines long without displaying other commands, you can include several **echo** *message* commands after the **echo off** command in your batch program.
- If you use **echo off**, the command prompt does not appear on your screen. To display the command prompt, type **echo on**.
- To prevent echoing of a line, insert an at sign (@) in front of a command in a batch program.
- To echo a blank line on the screen, type:
  echo.
- To display a pipe (|) or redirection character (< or >) when you are using **echo**, use a caret character immediately before the pipe or redirection character (for example, ^>, ^<, or ^|). If you need to use the caret character (^), type two (^^).

Examples

The following example is a batch program that includes a three-line message preceded by and then followed by a blank line:

```
echo off
echo.
echo This batch program
echo formats and checks
echo new disks
echo.
```

If you want to turn echo off and you do not want to echo the **echo** command, type an at sign (@) before the command as follows:

@echo off

You can use the **if** and **echo** commands on the same command line. For example:

if exist *.rpt echo The report has arrived.

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which

6. Endlocal

Ends localization of environment changes in a batch file, restoring environment variables to their values before the matching **setlocal** command.

Syntax

endlocal

Parameters

/? : Displays help at the command prompt.

Remarks

- You must use **endlocal** in a script or batch file. If you use **endlocal** outside of a script or batch file, it has no effect.
- There is an implicit **endlocal** command at the end of a batch file.
- With command extensions enabled (that is, the default), the **endlocal** command restores the state of command extensions (that is, enabled or disabled) to what it was before the matching **setlocal** command was executed. For more information about enabling and disabling command extensions, see **cmd** in Related Topics.

Examples

You can localize environment variables in a batch file. For example:

```
@echo off
rem This program starts the superapp batch program on the network,
rem directs the output to a file, and displays the file
rem in Notepad.
setlocal
path=g:₩programs₩superapp;%path%
call superapp>c:₩superapp.out
endlocal
start notepad c:₩superapp.out
```

Formatting legend

Format	Meaning
Italic	Information that the user must supply

Format	Meaning
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output

7. For

Runs a specified command for each file in a set of files.

Syntax

for {**%**variable|**%%**variable} **in (**set**) do** command [CommandLineOptions]

Parameters

{%*variable***|%%***variable***}** : Required. Represents a replaceable parameter. Use **%***variable* to carry out **for** from the command prompt. Use **%%***variable* to carry out the **for** command within a batch file. Variables are case-sensitive and must be represented with an alpha value, such as %A, %B, or %C.

(*set***)** : Required. Specifies one or more files, directories, range of values, or text strings that you want to process with the specified command. The parentheses are required.

command : Required. Specifies the command that you want to carry out on each file, directory, range of values, or text string included in the specified **(***set***)**.

CommandLineOptions : Specifies any command-line options that you want to use with the specified command.

/?: Displays help at the command prompt.

Remarks

- Using **for**

  You can use the **for** command within a batch file or directly from the command prompt.

- Using batch parameters

  The following attributes apply to the **for** command:

  - The **for** command replaces **%***variable* or **%%***variable* with each text string in the specified *set* until the *command* processes all of the files.
  - **For** *variable* names are case-sensitive, global, and no more than 52 total can be active at any one time.
  - To avoid confusion with the batch parameters *%0* through *%9*, you can use any character for *variable* except the numerals 0 through 9. For simple batch files, a single character such as **%%f** works.
  - You can use multiple values for *variable* in complex batch files to distinguish different replaceable variables.

- Specifying a group of files

  The *set* parameter can represent a single group of files or several groups of files. You can use wildcards (that is, * and ?) to specify a file set. The following are valid file sets:

(*.doc)

(*.doc *.txt *.me)

(jan*.doc jan*.rpt feb*.doc feb*.rpt)

(ar??1991.* ap??1991.*)

When you use the **for** command, the first value in *set* replaces **%***variable* or **%%***variable*, and then the specified command processes this value. This continues until all of the files (or groups of files) that correspond to the *set* value are processed.

- Using the **in** and **do** keywords

  In and **do** are not parameters, but you must use them with **for**. If you omit either of these keywords, an error message appears.

- Using additional forms of **for**

  If command extensions are enabled (that is, the default), the following additional forms of **for** are supported:

  - Directories only

    If *set* contains wildcards (* and ?), the specified *command* executes for each directory (instead of a set of files in a specified directory) that matches *set*. The syntax is:

    for /D {**%%** | **%**}variable **in (**set**) do** command [CommandLineOptions]

  - Recursive

    Walks the directory tree rooted at [*Drive*:]*Path*, executing the **for** statement in each directory of the tree. If no directory is specified after **/R**, the current directory is assumed. If *set* is just a single period (.), it only enumerates the directory tree. The syntax is:

    for /R [[Drive **:**]Path] {**%%** | **%**}variable **in (**set**) do** command [CommandLineOptions]

  - Iterating a range of values

    Use an iterative variable to set the starting value (*start#*) and then step through a set range of values until the value exceeds the set ending value (*end#*). **/L** will execute the iterative by comparing *start#* with *end#*. If *start#* is less than *end#* the command will execute. When the iterative variable exceeds *end#* the command shell exists the loop. You can also use a negative *step#* to step through a range in decreasing values. For example, (1,1,5) generates the sequence 1 2 3 4 5 and (5,-1,1) generates the sequence (5 4 3 2 1). The syntax is:

    for /L {**%%** | **%**}variable **in (**start#,step#,end#**) do** command [CommandLineOptions]

  - Iterating and file parsing

    Use file parsing to process command output, strings and file content. Use iterative variables to define the content or strings you want to examine and use the various *ParsingKeywords* options to further modify the parsing. Use the *ParsingKeywords* token option to specify which tokens should be passed as iterator variables. Note that when used

without the token option, **/F** will only examine the first token.

File parsing consists of reading the output, string or file content, breaking it up into individual lines of text and then parsing each line into zero or more tokens. The **for** loop is then called with the iterator variable value set to the token. By default, **/F** passes the first blank separated token from each line of each file. Blank lines are skipped. The different syntaxes are:

for /F ["ParsingKeywords"] {**%%** | **%**}variable **in** (filenameset**) do** command [CommandLineOptions]

for /F ["ParsingKeywords"] {**%%** | **%**}variable **in** ("LiteralString**") do** command [CommandLineOptions]

for /F ["ParsingKeywords"] {**%%** | **%**}variable **in** ('command'**) do** command [CommandLineOptions]

The *filenameset* argument specifies one or more file names. Each file is opened, read and processed before going on to the next file in *filenameset*. To override the default parsing behavior, specify "*ParsingKeywords*". This is a quoted string that contains one or more keywords to specify different parsing options.

If you use the usebackq option, use one of the following syntaxes:

for /F ["**usebackq**ParsingKeywords"] {**%%** | **%**}variable **in** ("filenameset"**) do** command [CommandLineOptions]

for /F ["**usebackq**ParsingKeywords"] {**%%** | **%**}variable **in** ('LiteralString'**) do** command [CommandLineOptions]

for /F ["**usebackq**ParsingKeywords"] {**%%** | **%**}variable **in** (`command`**) do** command [CommandLineOptions]

The following table lists the parsing keywords that you can use for *ParsingKeywords*.

Keyword	Description
eol=c	Specifies an end of line character (just one character).
skip=n	Specifies the number of lines to skip at the beginning of the file.
delims=xxx	Specifies a delimiter set. This replaces the default delimiter set of space and tab.
tokens=x,y,m-n	Specifies which tokens from each line are to be passed to the **for** body for each iteration. As a result, additional variable names are allocated. The *m-n* form is a range, specifying the *m*th through the *n*th tokens. If the last character in the **tokens=** string is an asterisk (*), an additional variable is allocated and receives the remaining text on the line after

	the last token that is parsed.
usebackq	Specifies that you can use quotation marks to quote file names in *filenameset*, a back quoted string is executed as a command, and a single quoted string is a literal string command.

- Variable substitution

  Substitution modifiers for **for** variable references have been enhanced. The following table lists optional syntax (for any variable **I**).

Variable with modifier	Description
%~I	Expands %I which removes any surrounding quotation marks ("").
%~fI	Expands %I to a fully qualified path name.
%~dI	Expands %I to a drive letter only.
%~pI	Expands %I to a path only.
%~nI	Expands %I to a file name only.
%~xI	Expands %I to a file extension only.
%~sI	Expands path to contain short names only.
%~aI	Expands %I to the file attributes of file.
%~tI	Expands %I to the date and time of file.
%~zI	Expands %I to the size of file.
%~$PATH:I	Searches the directories listed in the PATH environment variable and expands %I to the fully qualified name of the first one found. If the environment variable name is not defined or the file is not found by the search, this modifier expands to the empty string.

The following table lists modifier combinations that you can use to get compound results.

Variable with combined modifiers	Description
%~dpI	Expands %I to a drive letter and path only.
%~nxI	Expands %I to a file name and extension only.

%~fsI	Expands %I to a full path name with short names only.
%~dp$PATH:I	Searches the directories listed in the PATH environment variable for %I and expands to the drive letter and path of the first one found.
%~ftzaI	Expands %I to an output line that is like **dir**.

In the above examples, you can replace *%I* and PATH by other valid values. A valid **for** variable name terminates the %~ syntax.

By use uppercase variable names such as *%I*, you can make your code more readable and avoid confusion with the modifiers, which are not case-sensitive.

- Parsing a string

  You can use the **for /F** parsing logic on an immediate string, by wrapping the *filenameset* between the parentheses in single quotation marks (that is, '*filenameset*'). *Filenameset* is treated as a single line of input from a file, and then it is parsed.

- Parsing output

  You can use the **for /F** command to parse the output of a command by making the *filenameset* between the parenthesis a back quoted string. It is treated as a command line, which is passed to a child Cmd.exe and the output is captured into memory and parsed as if it were a file.

Examples

To use **for** in a batch file, use the following syntax:

for %%variable **in (**set**) do** command [CommandLineOptions]

To display the contents of all the files in the current directory that have the extension .doc or .txt using the replaceable variable *%f*, type:

for %f in (*.doc *.txt) do type %f

In the preceding example, each file that has the .doc or .txt extension in the current directory is substituted for the *%f* variable until the contents of every file are displayed. To use this command in a batch file, replace every occurrence of *%f* with *%%f*. Otherwise, the variable is ignored and an error message is displayed.

To parse a file, ignoring commented lines, type:

for /F "eol=; tokens=2,3* delims=," %i in (myfile.txt) do @echo %i %j %k

This command parses each line in Myfile.txt, ignoring lines that begin with a semicolon and passing the second and third token from each line to the **FOR** body (tokens are delimited by commas or spaces). The body of the **FOR** statement references *%i* to get the second token, *%j* to get the third token, and *%k* to get all of the remaining tokens. If the file names that you supply

contain spaces, use quotation marks around the text (for example, "*File Name*"). To use quotation marks, you must use **usebackq**. Otherwise, the quotation marks are interpreted as defining a literal string to parse.

%i is explicitly declared in the **FOR** statement, and *%j* and *%k* are implicitly declared by using **tokens=**. You can specify up to 26 tokens using **tokens=**, provided that it does not cause an attempt to declare a variable higher than the letter 'z' or 'Z'.

To parse the output of a command by placing *filenameset* between the parentheses, type:

for /F "usebackq delims==" %i IN (`set`) DO @echo %i

This example enumerates the environment variable names in the current environment.

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output

8. Goto

Within a batch program, directs Windows XP to a line identified by a label. When the label is found, it processes the commands that begin on the next line.

Syntax

goto *label*

Parameters

label **:** Specifies the line in a batch program that you want to go to.

/? : Displays help at the command prompt.

Remarks

- Working with command extensions

  If command extensions are enabled (that is, the default) and you use the **goto** command with a target label of **:EOF**, you transfer control to the end of the current batch script file and exit the batch script file without defining a label. When you use **goto** with the **:EOF** label, you must insert a colon before the label. For example:

  goto :EOF

  For a description of extensions to the **call** command that make this feature useful, see **cmd** in Related Topics.

- Using valid *label* values

  You can use spaces in the *label* parameter, but you cannot include other separators (for example, semicolons or equal signs). The **goto** command uses only the first eight characters of a label. For example, the following labels are equivalent and resolve to **:hithere0**:

  :hithere0

  :hithere01

  :hithere02

- Matching *label* with the label in the batch program

  The *label* value you specify must match a label in the batch program. The label within the batch program must begin with a colon (:). Windows XP recognizes a batch program line beginning with a colon (:) as a label and does not process it as a command. If a line begins with a colon, any commands on that line are ignored. If your batch program does not contain the label that you specify, the batch program stops and displays the following message:

  Label not found

- Using **goto** for conditional operations

  You can use **goto** with other commands to perform conditional operations. For more

information about using **goto** for conditional operations, see **if** in Related Topics.

Examples

The following batch program formats a disk in drive A as a system disk. If the operation is successful, the **goto** command directs Windows XP to the **:end** label:

```
echo off
format a: /s
if not errorlevel 1 goto end
echo An error occurred during formatting.
:end
echo End of batch program.
```

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output

9. If

Performs conditional processing in batch programs.

Syntax

if [**not**] **errorlevel** *number command* [**else** *expression*]

if [**not**] string1**==**string2 command [**else** expression]

if [**not**] **exist** FileName command [**else** expression]

If command extensions are enabled, use the following syntax:

if [**/i**] string1 CompareOp string2 command [**else** expression]

if cmdextversion *number command* [**else** *expression*]

if defined variable command [**else** expression]

Parameters

not : Specifies that the command should be carried out only if the condition is false.

errorlevel ***number*** **:** Specifies a true condition only if the previous program run by Cmd.exe returned an exit code equal to or greater than *number*.

command **:** Specifies the command that should be carried out if the preceding condition is met.

string1**==*****string2*** **:** Specifies a true condition only if *string1* and *string2* are the same. These values can be literal strings or batch variables (for example, *%1*). You do not need to use quotation marks around literal strings.

exist ***FileName*** **:** Specifies a true condition if *FileName* exists.

CompareOp **:** Specifies a three-letter comparison operator. The following table lists valid values for *CompareOp*.

Operator	Description
EQU	equal to
NEQ	not equal to
LSS	less than
LEQ	less than or equal to
GTR	greater than
GEQ	greater than or equal to

/i : Forces string comparisons to ignore case. You can use **/i** on the *string1*==*string2* form of **if**. These comparisons are generic, in that if both *string1* and *string2* are both comprised of all

numeric digits, the strings are converted to numbers and a numeric comparison is performed.

cmdextversion *number* **:** Specifies a true condition only if the internal version number associated with the Command Extensions feature of Cmd.exe is equal to or greater than *number*. The first version is 1. It is incremented by one when significant enhancements are added to the command extensions. The **cmdextversion** conditional is never true when command extensions are disabled (by default, command extensions are enabled).

defined *variable* **:** Specifies a true condition if *variable* is defined.

expression **:** Specifies a command-line command and any parameters to be passed to the command in an *else* clause.

/? : Displays help at the command prompt.

Remarks

- If the condition specified in an **if** command is true, the command that follows the condition is carried out. If the condition is false, the command in the **if** clause is ignored, and executes any command in the **else** clause, if one has been specified.
- When a program stops, it returns an exit code. You can use exit codes as conditions by using the *errorlevel* parameter.
- Using **defined** *variable*

  If you use **defined** *variable*, the following three variables are added: **%errorlevel%**, **%cmdcmdline%**, and **%cmdextversion%**.

  %errorlevel% expands into a string representation of the current value of *errorlevel*, provided that there is not already an environment variable with the name ERRORLEVEL, in which case you get the ERRORLEVEL value instead. The following example illustrates how you can use **errorlevel** after running a batch program:

  ```
  goto answer%errorlevel%
  :answer0
  echo Program had return code 0
  :answer1
  echo Program had return code 1
  goto end
  :end
  echo done!
  ```

  You can also use the *CompareOp* comparison operators as follows:

  ```
  if %errorlevel% LEQ 1 goto okay
  ```

  %cmdcmdline% expands into the original command line passed to Cmd.exe prior to any processing by Cmd.exe, provided that there is not already an environment variable with the

name **cmdcmdline**, in which case you get the **cmdcmdline** value instead.

%cmdextversion% expands into the a string representation of the current value of **cmdextversion**, provided that there is not already an environment variable with the name CMDEXTVERSION, in which case you get the CMDEXTVERSION value instead.

- Using the **else** clause

  You must use the **else** clause on the same line as the command after the **if**. For example:

  IF EXIST filename. (
  del filename.
) ELSE (
  echo filename. missing.
)

  The following code does not work because you must terminate the **del** command by a new line:

  IF EXIST filename. del filename. ELSE echo filename. missing

  The following code does not work because you must use the **else** clause on the same line as the end of the **if** command:

  IF EXIST filename. del filename.
  ELSE echo filename. missing

  If you want to format it all on a single line, use the following form of the original statement:

  IF EXIST filename. (del filename.) ELSE echo filename. missing

Examples

If the file Product.dat cannot be found, the following message appears:

if not exist product.dat echo Can't find data file

If an error occurs during the formatting of the disk in drive A, the following example displays an error message:

```
:begin
@echo off
format a: /s
if not errorlevel 1 goto end
echo An error occurred during formatting.
:end
echo End of batch program.
```

If no error occurs, the error message does not appear.

You cannot use the **if** command to test directly for a directory, but the null (NUL) device does

exist in every directory. As a result, you can test for the null device to determine whether a directory exists. The following example tests for the existence of a directory:

if exist c:₩mydir₩nul goto process

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output

10. Pause

Suspends processing of a batch program and displays a message prompting the user to press any key to continue.

Syntax

pause

Parameters

/? : Displays help at the command prompt.

Remarks

- When you run **prompt** command, the following message appears:

  Press any key to continue . . .
- If you press CTRL+C to stop a batch program, the following message appears:

  Terminate batch job (Y/N)?

  If you press Y (for yes) in response to this message, the batch program ends and control returns to the operating system. Therefore, you can insert the **pause** command before a section of the batch file you may not want to process. While **pause** suspends processing of the batch program, you can press CTRL+C and then Y to stop the batch program.

Examples

To create a batch program that prompts the user to change disks in one of the drives, type:

```
@echo off
:begin
copy a:*.*
echo Please put a new disk into drive A
pause
goto begin
```

In this example, all the files on the disk in drive A are copied to the current directory. After the displayed comment prompts you to place another disk in drive A, the **pause** command suspends processing so that you can change disks and then press any key to resume processing. This particular batch program runs in an endless loop. The **goto** BEGIN command sends the command interpreter to the begin label of the batch file. To stop this batch program, press CTRL+C and then Y.

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output

11. Rem

Enables you to include comments (remarks) in a batch file or in your configuration files.

Syntax

rem [comment]

Parameters

comment **:** Specifies any string of characters you want to include as a comment.

/? : Displays help at the command prompt.

Remarks

- Using the echo command to display comments

  The **rem** command does not display comments on the screen. You must use the **echo on** command in your batch or Config.nt file to display comments on the screen.

- Restrictions on batch file comments

  You cannot use a redirection character "(" or ")" or pipe (|) in a batch file comment.

- Using **rem** to add vertical spacing

  Although you can use **rem** without a comment to add vertical spacing to a batch file, you can also use blank lines. The blank lines are ignored when processing the batch program.

Examples

The following example shows a batch file that uses remarks for both explanations and vertical spacing:

```
@echo off
rem This batch program formats and checks new disks.
rem It is named Checknew.bat.
rem
echo Insert new disk in drive B.
pause
format b: /v
chkdsk b:
```

Suppose you want to include in your Config.nt file an explanatory comment before the **prompt** command. To do this, add the following lines to Config.nt:

```
rem Set prompt to indicate current directory
prompt $p$g
```

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output

12. Setlocal

Starts localization of environment variables in a batch file. Localization continues until a matching **endlocal** command is encountered or the end of the batch file is reached.

Syntax

setlocal {enableextensions | disableextensions} {enabledelayedexpansion | disabledelayedexpansion}

Arguments

enableextensions : Enables the command extensions until the matching **endlocal** command is encountered, regardless of the setting prior to the **setlocal** command.

disableextensions : Disables the command extensions until the matching **endlocal** command is encountered, regardless of the setting prior to the **setlocal** command.

enabledelayedexpansion : Enables the delayed environment variable expansion until the matching **endlocal** command is encountered, regardless of the setting prior to the **setlocal** command.

disabledelayedexpansion : Disables the delayed environment variable expansion until the matching **endlocal** command is encountered, regardless of the setting prior to the **setlocal** command.

/? : Displays help at the command prompt.

Remarks

- Using setlocal

  When you use **setlocal** outside of a script or batch file, it has no effect.
- Changing environmental variables

  Use **setlocal** to change environment variables when you run a batch file. Environment changes made after you run **setlocal** are local to the batch file. Cmd.exe restores previous settings when it either encounters an **endlocal** command or reaches the end of the batch file.
- You can have more than one **setlocal** or **endlocal** command in a batch program (that is, nested commands).
- Testing for command extensions in batch files

  The **setlocal** command sets the ERRORLEVEL variable. If you pass either {**enableextensions** | **disableextensions**} or {**enabledelayedexpansion** | **disabledelayedexpansion**}, the ERRORLEVEL variable is set to zero (0). Otherwise, it is set to one (1). You can use this in batch scripts to determine whether the extensions are available, for example:

  verify other 2>nul

setlocal enableextensions

if errorlevel 1 echo Unable to enable extensions

Because **cmd** does not set the ERRORLEVEL variable when command extensions are disabled, the **verify** command initializes the ERRORLEVEL variable to a nonzero value when you use it with an invalid argument. Also, if you use the **setlocal** command with arguments {**enableextensions** | **disableextensions**} or {**enabledelayedexpansion** | **disabledelayedexpansion**} and it does not set the ERRORLEVEL variable to one (1), command extensions are not available.

For more information about enabling and disabling command extensions, see **cmd** in Related Topics.

Examples

You can localize environment variables in a batch file, as follows:

```
rem *******Begin Comment**************
  rem This program starts the superapp batch program on the network,
  rem directs the output to a file, and displays the file
  rem in Notepad.
  rem *******End Comment**************
      @echo off
      setlocal
      path=g:₩programs₩superapp;%path%
      call superapp>c:₩superapp.out
      endlocal
      start notepad c:₩superapp.out
```

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items

Format	Meaning
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output

13. Shift

Changes the position of batch parameters in a batch file.

Syntax

shift

Parameters

none

Remarks

- Using the **shift** command-line option with command extensions

  When command extensions are enabled (that is, the default), the **shift** command supports the **/n** command-line option, which tells the command to start shifting at the *n*th argument, where *n* can be a value from zero to eight. For example,

  SHIFT /2

  would shift %3 to %2, %4 to %3, and so on, and leave %0 and %1 unaffected.
- How the shift command works

  The **shift** command changes the values of the batch parameters %0 through %9 by copying each parameter into the previous one. In other words, the value of %1 is copied to %0, the value of %2 is copied to %1, and so on. This is useful for writing a batch file that performs the same operation on any number of parameters.
- Working with more than 10 batch parameters

  You can also use the **shift** command to create a batch file that can accept more than 10 batch parameters. If you specify more than 10 parameters on the command line, those that appear after the tenth (%9) will be shifted one at a time into %9.
- Using %* with **shift**

  Shift has no affect on the %* batch parameter.
- Shifting parameters back

  There is no backward **shift** command. After you carry out the **shift** command, you cannot recover the first batch parameter (%0) that existed before the shift.

Examples

The following batch file, Mycopy.bat, shows how to use **shift** with any number of batch parameters. It copies a list of files to a specific directory. The batch parameters are represented by the directory and file name arguments.

```
@echo off
rem MYCOPY.BAT copies any number of files
rem to a directory.
rem The command uses the following syntax:
rem mycopy dir file1 file2 ...
set todir=%1
:getfile
shift
if "%1"=="" goto end
copy %1 %todir%
goto getfile
:end
set todir=
echo All done
```

Formatting legend

Format	Meaning
Italic	Information that the user must supply
Bold	Elements that the user must type exactly as shown
Ellipsis (...)	Parameter that can be repeated several times in a command line
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (|). Example: {even|odd}	Set of choices from which the user must choose only one
Courier font	Code or program output