IF

Conditionally perform a command.

```
File syntax
   IF [NOT] EXIST filename command

   IF [NOT] EXIST filename (command) ELSE (command)

String syntax
   IF [/I] [NOT] item1==item2 command

   IF [/I] item1 compare-op item2 command

   IF [/I] item1 compare-op item2 (command) ELSE (command)

Error Check Syntax
   IF [NOT] DEFINED variable command

   IF [NOT] ERRORLEVEL number command

   IF CMDEXTVERSION number command

key
   item        A text string or environment variable, for more complex
               comparisons, a variable can be modified using
               either Substring or Search syntax.

   command     The command to perform.

   filename    A file to test or a wildcard pattern.

   NOT         perform the command if the condition is false.

   ==          perform the command if the two strings are equal.

   /I          Do a case Insensitive string comparison.

   compare-op  can be one of
                EQU : Equal
                NEQ : Not equal

                LSS : Less than <
                LEQ : Less than or Equal <=

                GTR : Greater than >
                GEQ : Greater than or equal >=

                This 3 digit syntax is necessary because the > and <
                symbols are recognised as redirection operators
```

IF will only parse numbers when one of (EQU, NEQ, LSS, LEQ, GTR, GEQ) is used.
The == comparison operator always results in a string comparison.

ERRORLEVEL

There are two different methods of checking an errorlevel, the first syntax (`IF ERRORLEVEL ...`) provides compatibility with ancient batch files from the days of Windows 95.
The second method is to use the `%ERRORLEVEL%` variable providing compatibility with Windows 2000 or newer.

`IF ERRORLEVEL n` statements should be read as IF *Errorlevel* >= *number*
i.e.
`IF ERRORLEVEL 0` will return TRUE whether the errorlevel is 0, 1 or 5 or 64
`IF ERRORLEVEL 1` will return TRUE whether the errorlevel is 1 or 5 or 64
`IF NOT ERRORLEVEL 1` means if ERRORLEVEL is less than 1 (Zero or negative).
This is not very readable or user friendly and does not easily account for negative error numbers.

Using the `%ERRORLEVEL%` variable is a more logical method of checking Errorlevels:

```
IF %ERRORLEVEL% NEQ 0 Echo An error was found
IF %ERRORLEVEL% EQU 0 Echo No error found

IF %ERRORLEVEL% EQU 0 (Echo No error found) ELSE (Echo An error was found)
IF %ERRORLEVEL% EQU 0 Echo No error found || Echo An error was found
```

This allows you to trap errors that can be negative numbers, you can also test for specific errors:
`IF %ERRORLEVEL% EQU 64 ...`

To deliberately raise an ERRORLEVEL in a batch script use the EXIT /B command.

It is possible (though not a good idea) to create a string variable called %ERRORLEVEL% (user variable) if present such a variable will prevent the real ERRORLEVEL (a system variable) from being used by commands such as ECHO and IF.

Test if a variable is empty

To test for the existence of a command line parameter - use empty brackets like this

```
IF [%1]==[] ECHO Value Missing
```
or
```
IF [%1] EQU [] ECHO Value Missing
```

When comparing against a variable that may be empty, we include a pair of brackets `[ ]` so that if the variable does happen to be empty the IF command still has something to compare: `IF [] EQU []` will return True.

You can in fact use almost any character for this a '~' or curly brackets, `{ }` or even the number 4, but square brackets tend to be chosen because they don't have any special meaning.
When working with filenames/paths you should always surround them with quotes, if `%_myvar%` contains "C:\Some Path" then your comparison becomes `IF ["C:\Some Path"] EQU []`
if `%_myvar%` could contain empty quotes, "" then your comparison should become `IF [%_myvar%] EQU [""]`

if `%_myvar%` will *never* contain quotes, then you can use quotes in place of the brackets `IF "%_myvar%" EQU ""`
However with this pattern if `%_myvar%` does unexpectedly contain quotes, you will get `IF ""C:\Some Path"" EQU ""` those doubled quotes, while not officially documented as an escape will still mess up the comparison.

Test if a variable is NULL

In the case of a variable that might be NULL - a null variable will remove the variable definition altogether, so testing for a NULL becomes:

```
IF NOT DEFINED _example ECHO Value Missing
```

IF DEFINED will return true if the variable contains any value (even if the value is just a space)

To test for the existence of a user variable use `SET VariableName`, or `IF DEFINED VariableName`

Test the existence of files and folders

`IF EXIST filename` Will detect the existence of a file or a folder.

The script empty.cmd will show if the folder is empty or not (this is not case sensitive).

Parenthesis

Parenthesis can be used to split commands across multiple lines. This enables writing more complex IF… ELSE… commands:

```
IF EXIST filename.txt (
    Echo deleting filename.txt
    Del filename.txt
 ) ELSE (
    Echo The file was not found.
 )
```

When combining an ELSE statement with parenthesis, always put the parenthesis on the same line as ELSE.
`) ELSE (` This is because CMD does a rather primitive one-line-at-a-time parsing of the command.

When using parenthesis the CMD shell will expand [read] all the variables at the beginning of the code block and use those values even if the variables value has just been changed. Turning on DelayedExpansion will force the shell to read variables at the start of every line.

Pipes

When piping commands, the expression is evaluated from left to right, so

`IF SomeCondition Command1 | Command2`is equivalent to:

`(IF SomeCondition Command1 ) | Command2`
The pipe is always created and `Command2` is always run, regardless whether `SomeCondition` is TRUE or FALSE

You can use brackets and conditionals around the *command* with this syntax:

`IF SomeCondition (Command1 | Command2)`
If the condition is met then Command1 will run, and its output will be piped to Command2.

The IF command will interpret brackets around a ***condition*** as just another character to compare (like # or @) for example:
`IF (%_var1%==(demo Echo the variable _var1 contains the text demo`

Placing an IF command on the right hand side of a pipe is also possible but the CMD shell is buggy in this area and can swallow one of the delimiter characters causing unexpected results.
A simple example that does work:

```
Echo Y | IF red==blue del *.log
```

Chaining IF commands (AND).

The only logical operator directly supported by `IF` is `NOT`, so to perform an `AND` requires chaining multiple IF statements:

```
IF SomeCondition (
   IF SomeOtherCondition (
     Command_if_both_are_true
   )
)
```

If either condition is true (OR)

This can be tested using a temporary variable:

```
Set "_tempvar="
If SomeCondition Set _tempvar=1
If SomeOtherCondition Set _tempvar=1
if %_tempvar% EQU 1 Command_to_run_if_either_is_true
```

Delimiters

If the string being compared by an IF command includes delimiters such as [Space] or [Comma], then either the delimiters must be escaped with a caret ^ or the whole string must be "quoted".
This is so that the IF statement will treat the string as a single item and not as several separate strings.

Test Numeric values

IF only parses *numbers* when one of the `compare-op` operators (`EQU, NEQ, LSS, LEQ, GTR, GEQ`) is used.
The `==` comparison operator always results in a *string* comparison.

This is an important difference because if you compare numbers as strings it can lead to unexpected results: "2" will be greater than "19" and "026" will be less than "10".

Correct numeric comparison:
`IF 2 GEQ 15 echo "bigger"`

Using parenthesis or quotes will force a string comparison:
```
IF (2) GEQ (15) echo "bigger"
IF "2" GEQ "15" echo "bigger"
```

This behaviour is exactly opposite to the SET /a command where quotes are required.

Wildcards

Wildcards are not supported by IF, so `%COMPUTERNAME%==SS6*` will not match SS64

A workaround is to retrieve the substring and compare just those characters:

```
SET _prefix=%COMPUTERNAME:~0,3%
IF %_prefix%==SS6 GOTO they_matched
```

If Command Extensions are disabled IF will only support direct comparisons: IF ==, IF EXIST, IF ERRORLEVEL also the system variable `CMDEXTVERSION` will be disabled.

Examples:

```
IF EXIST C:\logs\*.log (Echo Log file exists)

IF EXIST C:\logs\install.log (Echo Complete) ELSE (Echo failed)

IF DEFINED _department ECHO Got the _department variable

IF DEFINED _commission SET /A _salary=%_salary% + %_commission%

IF CMDEXTVERSION 1 GOTO start_process

IF %ERRORLEVEL% EQU 2 goto sub_problem2
```

IF is an internal command.

You see things; and you say 'Why?' But I dream things that never were; and I say 'why not?' ~ George Bernard Shaw

Related:

Using parenthesis to group and expand expressions.
Conditional execution syntax (AND / OR)
SET - Display or Edit environment variables
ECHO - Display message on screen
EXIT - Set a specific ERRORLEVEL
IFMEMBER - group member (Resource kit)
SC - Is a Service running (Resource kit)
Powershell: if - Conditionally perform a command
Equivalent bash command (Linux): if - Conditionally perform a command